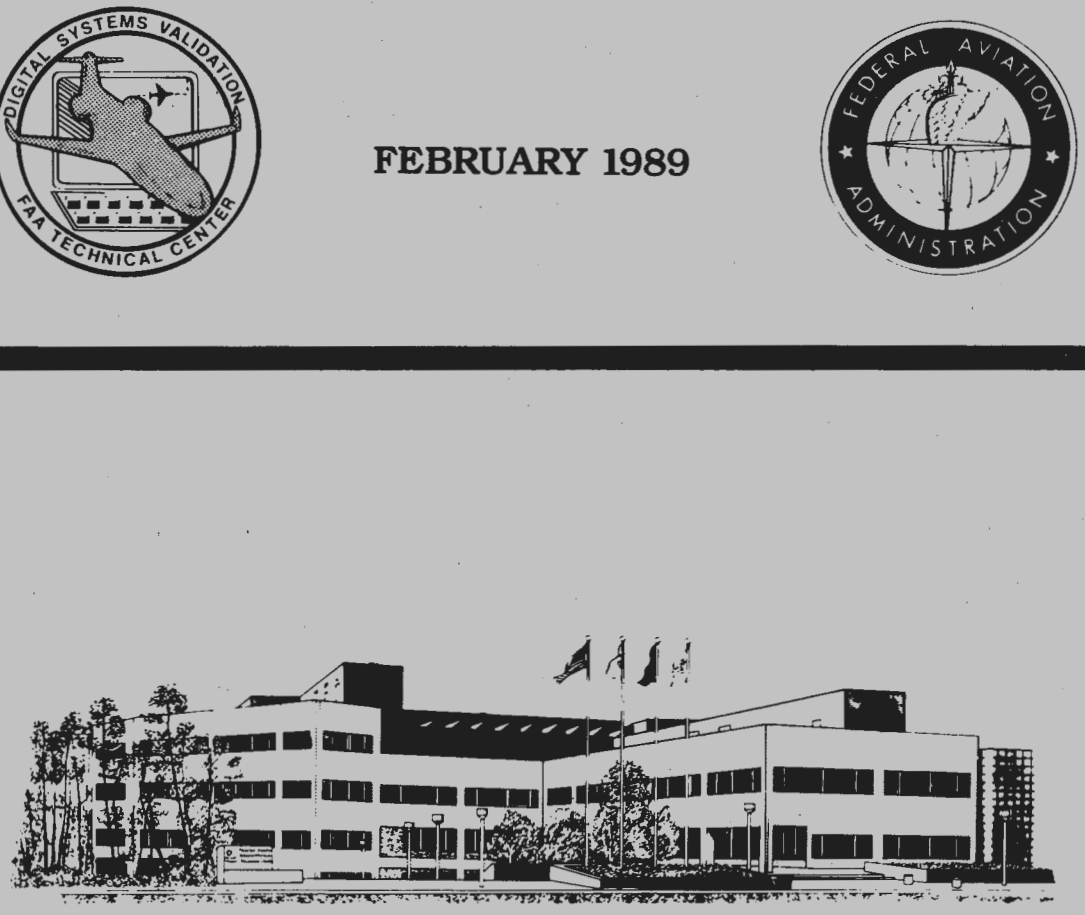
COPY 2

U.S. Department of Transportation
Federal Aviation Administration

DOT/FAA/CT-88/10

DIGITAL SYSTEMS VALIDATION HANDBOOK-VOLUME II

FEBRUARY 1989



FEDERAL AVIATION ADMINISTRATION
TECHNICAL CENTER

FAA TECHNICAL CENTER LIBRARY
00013515

NOTICE

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The United States Government assumes no liability for the contents or use thereof.

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the objective of this report.

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
DOT/FAA/CT-88/10		

4. Title and Subtitle	5. Report Date
DIGITAL SYSTEMS VALIDATION HANDBOOK - VOLUME II	FEBRUARY 1989
	6. Performing Organization Code: CRMI
7. Author(s): SEE TABLE OF CONTENTS	8. Performing Organization Report No.: DOT/FAA/CT-88/10
9. Performing Organization Name and Address: COMPUTER RESOURCE MANAGEMENT INCORPORATED, 950 HERNDON PARKWAY SUITE 360, HERNDON, VA 22070	10. Work Unit No. (TRAIS)
	11. Contract or Grant No.: DTFA03-86-C-00042
12. Sponsoring Agency Name and Address: U.S. DEPARTMENT OF TRANSPORTATION, FEDERAL AVIATION ADMINISTRATION TECHNICAL CENTER, ATLANTIC CITY INTERNATIONAL AIRPORT, NJ 08405	13. Type of Report and Period Covered: HANDBOOK
	14. Sponsoring Agency Code: ACD-230

15. Supplementary Notes

Point of Contact: Pete Saraceni, ACD-230
FAA Technical Center
Atlantic City International Airport, NJ 08405

16. Abstract

Volume II covers detailed technical topics such as latent faults; data buses; integrated assurance assessment; analytical sensor redundancy; and protection against lightning, electromagnetic interference, and high energy radio frequency fields. These topics are covered in detail to familiarize the certification engineer with the issues involved in implementing the new technologies.

Volume II covers topics that will enable the certification engineer to understand the information presented in type certification and supplemental type certification documentation, to understand variations in the implementation of technologies, and to discuss them with the design engineer.

Volume II also addresses some of the soon-to-be-available technologies in the "Advanced Validation Issues" chapter. The direction of aviation research in the United States is discussed along with challenges and problems that confront the certification engineer in certifying the new technologies.

Since the topics discussed in this Handbook are at the forefront of technological research, some of the concepts presented are subject to discussion by experts in the field. In these areas, the Handbook presents various viewpoints alerting the certification engineer to the various views so that this information will be considered in formulating decisions and developing certification criteria.

17. Key Words	18. Distribution Statement
Avionics, Digital, Validation, Certification, Reliability, Redundancy, Latent Faults, Fault Insertion, Data Buses, Lightning, Radio Frequency Fields, Transients, Verification, Composites, Modeling	This document is available to the U.S. public through the National Technical Information Service, Springfield, Virginia 22161

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	900	

U.S. Department of Transportation
Federal Aviation Administration

Technical Center

Atlantic City Int'l Airport
New Jersey 08405

May 4, 1992

Dear Colleague,

Enclosed is additional material for the Digital Systems Validation Handbook - Volume II (DOT/FAA/CT-88/10). These additions consist of a copy of Chapter 17 - Software Quality Metrics, and a revised Table of Contents, List of Authors, Glossary, and Acronyms and Abbreviations. Please add these materials to your Handbook.

If you have any questions or comments concerning the material, or if you would like to receive a copy of the Handbook, please feel free to contact me at the Technical Center.

Sincerely,

Pete Saraceni

Pete Saraceni
FAA Technical Center
ACD-230, Building 201
Atlantic City International Airport, NJ 08405

Phone: (FTS) 482-5577
(609) 484-5577

Enclosures: Chapter 17 - Software Quality Metrics
Table of Contents (revised)
List of Authors (revised)
Glossary (revised)
Acronyms and Abbreviations (revised)

DIGITAL SYSTEMS VALIDATION HANDBOOK - VOLUME II

TABLE OF CONTENTS

Chapter		Page
LIST OF AUTHORS		vii
1.	SUMMARY R. L. McDowall	1-1
2.	INTRODUCTION R. L. McDowall	2-1
3.	INTEGRATED ASSURANCE ASSESSMENT Hardy P. Curd	3-1
4.	QUADRUPLEX DIGITAL FLIGHT CONTROL SYSTEMS Lloyd N. Popish	4-1
5.	ADVANCED FAULT INSERTION AND SIMULATION METHODS William W. Cooley	5-1
6.	DIGITAL DATA BUSES FOR AVIATION APPLICATIONS Donald Eldredge and Susan Mangold	6-1
7.	ANALYTICAL SENSOR REDUNDANCY William W. Cooley and Deborah L. Shortess	7-1
8.	ESTIMATION AND MODELING FOR REAL-TIME SOFTWARE RELIABILITY MODELS Donald Eldredge and Susan Mangold	8-1
9.	FAULT TOLERANT SOFTWARE Myron J. Hecht	9-1

TABLE OF CONTENTS
(Continued)

Chapter		Page
10.	LATENT FAULTS John G. McGough	10-1
11.	AIRCRAFT ELECTROMAGNETIC COMPATIBILITY (Guidelines to Assess EMC Designs) Clifton A. Clarke and William E. Larsen	11-1
12.	FAST RISE-TIME ELECTRICAL TRANSIENTS IN AIRCRAFT Roger McConnell	12-1
13.	LIGHTNING STUDIES William W. Cooley, Barbara G. Melander, and Deborah L. Shortess	13-1
14.	HIGH ENERGY RADIO FREQUENCY FIELDS (Impact on Digital Systems) John E. Reed and Robert E. Evans	14-1
15.	ELECTROMECHANICAL ACTUATOR SYSTEMS (Electrical Systems Certification Issues) William W. Cooley	15-1
16.	ADVANCED VALIDATION ISSUES Hardy P. Curd	16-1
17.	SOFTWARE QUALITY METRICS Donald A. Elwell and Nancy J. VanSuetendael	17-1
18.	AVIONIC DATA BUS INTEGRATION TECHNOLOGY Donald A. Elwell, Lee H. Harrison, John H. Hensyl, and Nancy J. VanSuetendael	18-1
19.	PILOT-VEHICLE INTERFACE Lee H. Harrison, Joan L. Janowitz, and Maurice A. Castronuovo	19-1

TABLE OF CONTENTS
(Continued)

Chapter		Page
20.	ARTIFICIAL INTELLIGENCE WITH APPLICATIONS FOR AIRCRAFT	20-1
	Lee H. Harrison, Pamela J. Saunders, and Joan L. Janowitz	
GLOSSARY		
ACRONYMS		

LIST OF AUTHORS

Maurice A. Castronuovo
Galaxy Scientific Corp.
2500 English Creek Ave., Bldg. 11
Pleasantville, NJ 08232

Clifton A. Clarke
Boeing Commercial Airplane Company
P. O. Box 3707
Seattle, WA 98124

William W. Cooley
Science & Engineering Associates, Inc.
701 Dexter Avenue
Seattle, WA 98109

Hardy P. Curd
Computer Resource Management, Inc.
950 Herndon Parkway, Suite 360
Herndon, VA 22070

Donald Eldredge
Battelle Columbus Division
505 King Avenue
Columbus, OH 43201

Donald A. Elwell
CTA, Inc.
2500 English Creek Ave., Suite 1000
Pleasantville, NJ 08232

Robert E. Evans
FAA Technical Center
Atlantic City International Airport, NJ 08405

Lee H. Harrison
Galaxy Scientific Corp.
2500 English Creek Ave., Bldg. 11
Pleasantville, NJ 08232

Myron J. Hecht
SoHaR, Inc.
8500 Wilshire Boulevard
Beverly Hills, CA 90211

LIST OF AUTHORS
(Continued)

John H. Hensyl
Diversified International Sciences Corp.
9901-R Business Parkway
Lanham, MD 20706-1840

Joan L. Janowitz
Galaxy Scientific Corp.
2500 English Creek Ave., Bldg. 11
Pleasantville, NJ 08232

William E. Larsen
FAA Field Office
P. O. Box 25
Moffett Field, CA 94035

Susan Mangold
Battelle Columbus Division
505 King Avenue
Columbus, OH 43201

Roger McConnell
CK Consultants
5473 A Clouds Rest
Mariposa, CA 95338

R. L. McDowall
Galaxy Scientific Corp.
2500 English Creek Ave., Bldg. 11
Pleasantville, NJ 08232

John G. McGough, Consultant
150 Walnut Street
Ridgewood, NJ 07450

Barbara G. Melander
Science & Engineering Associates, Inc.
701 Dexter Avenue
Seattle, WA 98109

Lloyd N. Popish, Consultant
525 Davenport Court
Sunnyvale, CA 94087

John E. Reed
AERS/Midwest, Inc.
PO Box 965
106 Washington
Burlington, IA 52601

LIST OF AUTHORS
(Continued)

Pamela J. Saunders
Galaxy Scientific Corp.
2500 English Creek Ave., Bldg. 11
Pleasantville, NJ 08232

Deborah L. Shortess
Science & Engineering Associates, Inc.
701 Dexter Avenue
Seattle, WA 98109

Nancy J. VanSuetendael
Computer Resource Management, Inc.
950 Herndon Parkway, Suite 360
Herndon, VA 22070

GLOSSARY

α-FAULT. (10) A fault activated by the baseline program (see ß-FAULT).

A PRIORI. (20) Pertaining to deductive reasoning from assumed axioms or supposedly self-evident principles, without reference to experience.

A-SPECIFICATION. (9) The highest level specification typically produced by the contracting organization to define a system (see MIL-STD-1521).

ABSORPTION LOSS. (11) Attenuation or retention of electromagnetic energy passing through a material, a shield. Absorption loss and reflection loss contribute to total shielding effectiveness (SE).

ACCESS. (18) The process of a transmitting bus user obtaining control of a data bus in order to transmit a message.

ACTION INTEGRAL. (13) The action integral is a critical factor in the production of damage. It relates to the energy deposited or absorbed in a system. This energy cannot be defined without knowing the resistance of the system. The instantaneous power dissipated in a resistor is I^2R and is expressed in watts. For the total energy expended, the power must be integrated over time to get the total joules, watt-seconds. By specifying the integral of $i(t)^2$ over the time interval involved, a useful quantity is defined for application to any resistance value. In the case of lightning, this quantity is defined as the action integral and is specified as $i(t)^2dt$ over the time the current flows.

ACTIVE FAULT. (10) A fault that can produce an error (for some input) while executing the current program.

ACTUAL TRANSIENT LEVEL. (13) The actual transient level is the level of transients which actually appear at the system interfaces as a result of the external environment. This level may be less than or equal to the transient control level but should not be greater.

ADAPTIVE FUZZY SYSTEMS. (20) Fuzzy controllers with the ability to learn and explain their reasoning.

ADDRESSING CAPACITY. (6) The number of components addressable by the protocol used on a given data bus.

ADVISORY CIRCULAR. (18) An external FAA publication consisting of nonregulatory material of a policy, guidance, and informational nature.

AIR TRANSPORT AIRCRAFT. (18) Aircraft used in interstate, overseas, or foreign air transportation.

AIRCRAFT LIGHTNING INTERACTION. (13) An encounter with lightning that produces sufficient current within or voltages along an aircraft skin or structure to pose a threat to the aircraft electrical/electronic systems, as a result of a direct lightning attachment.

AIRWORTHINESS STANDARDS. (18) Parts 23, 25, 27, 29, and 33 of the Code of Federal Regulations, Title 14, Chapter 1, Subchapter C.

ALGORITHM. (20) A set of well-defined rules for the solution of a problem in a finite number of steps.

AMBIENT. (16) The substance which absorbs heat from the heat sink.

ANALOG. (20) Pertaining to devices, data, circuits, or systems that operate with variables that are presented by continuously measured voltages or other quantities.

ANALYTICAL REDUNDANCY. (7) The use of software algorithms which use known mathematical relationships between different sensors for sensor failure detection and replace most of additional redundant sensor hardware.

ANALYTICAL ROOT SOLUTION. (4) Information obtained from the roots of the characteristic equations of the airplane model such as short-period or phugoid frequency response.

ANGLE OF ATTACK. (4) Angle between the longitudinal axis of an aircraft and the direction of movement.

ANISOTROPIC. (19) Having different, direction dependent, physical characteristics.

ANODIZE. (11) A preparation by electrolytic process that deposits a protective oxide, insulating film on a metallic surface (aluminum). The oxide defeats electrical bonding. Alodine and iridite finishes on aluminum are conductive.

ANTAGONISTIC QUALITY FACTORS. (17) Quality Factors with conflicting attributes.

APERTURE. (11) An opening, such as a nonconductive panel joint, slot, or crack, allowing electromagnetic energy to pass through a shield.

ARC. (20) The part of a semantic network that connects one object to another.

ARCHITECTURE. (18) The design and interaction of components of a computer system.

ARTIFICIAL INTELLIGENCE. (16) The characteristics of a machine programmed to imitate human intelligence functions.

ARTIFICIAL INTELLIGENCE. (20) The subfield of computer science concerned with emulating human intelligence by use of software and hardware techniques.

ASSISTANT. (20) An Expert System that performs a technically limited subset of an expert's task.

ASSOCIATIVE MEMORY NETWORK. (20) A network where different input patterns, if sufficiently similar, become associated with one another (trigger the same response).

ASSURANCE ASSESSMENT. (4) Procedures whose purpose is to ensure that a proposed system functions according to design specifications.

ASYNCHRONOUS. (20) Operating at a speed determined by the circuit functions rather than by timing signals.

ASYNCHRONOUS MESSAGES. (6) Electronic signals with transmission times that are not known a priori. These may include priority signals requiring immediate access to the bus.

ATTACHMENT POINT. (13) A point of contact of the lightning flash with the aircraft.

ATTENSON. (19) A unique sound used on the flight deck for alerting the flightcrew of warning, caution, and advisory conditions.

ATTRACTOR. (20) The geometrical pattern toward which the trajectory of a dynamical system, represented by a curve in phase space, converges in the course of time.

AUDIO FREQUENCY (AF). (11) The spectrum (20 to 20,000 Hz) of human hearing, often defined as extending from approximately 20 Hz to 50 kHz and sometimes to 150 kHz. Audio noise is nuisance hum, static, or tones from power line 400 Hz, switching regulator and digital clock harmonics, or HF, VHF transmitter frequencies.

AUTOFEATHER. (16) To automatically and swiftly feather the propeller when the engine fails to drive it.

AUXILIARY PROGRAMS. (10) Software executed occasionally.

AVALANCHING LATENT FAULTS. (10) The successive activation of latent faults.

AVIONIC. (18) Electronic equipment used in aircraft.

AXON. (20) The pathway that carries the output of a biological neuron.

ß-FAULT. (10) A fault not activated by the baseline program (see α-FAULT).

BABBLING TRANSMITTER. (18) A bus user that transmits outside its allocated time.

BACK PROPAGATION. (20) An algorithm used to adjust weights connecting neurons in successive layers of multi-layer networks. Back propagation learns by example and repetition.

BACKSHELL. (11) Metal shell connecting circuit shields or overbraid to an electrical connector.

BACKWARD CHAINING. (20) A method of problem solving that works backward from a known conclusion in an attempt to find a path of reasoning in order to justify the conclusion.

BACKWARD RECOVERY. (9) Restoration of the system to some previous known correct state and restarting the computation from that point.

BALANCED CIRCUIT. (11) A signal, acting line-to-line, between two conductors having symmetrical voltages identical and equal in relation to other circuits and to ground. "Differential mode" is line-to-line; "common mode" is line to ground.

BALANCED CONFIGURATION. (18) A bus using the HDLC protocol that connects only primary stations.

BANDWIDTH (BW). (11) Frequencies bounded by an upper and lower limit in a given band associated with electronic devices, filters, and receivers.

BASELINE PROGRAM. (10) A set of continuously executed software modules.

BASIN OF ATTRACTION. (20) The collection of all possible initial conditions of a dynamical system for which the trajectories representing that system in phase space will converge to a particular attractor.

BENIGN FAULT. (10) A fault that cannot produce an error while executing the current program, regardless of input, but may produce an error for some other program.

BIDIRECTIONAL DATA BUS. (18) A data bus with more than one user capable of transmitting.

BINARY. (20) Composed of or characterized by two parts or elements.

BINARY SEARCH. (17) A searching algorithm in which the search population is repeatedly divided into two equal or nearly equal sections.

BIREFRINGENCE. (19) The splitting of light into two components, where each component travels at a different velocity.

BIT-ORIENTED PROTOCOL. (18) A communication protocol where message frames can vary in length, with single bit resolution.

BIT TIME. (6) The time it would take to transmit one bit. Usually this is "blank" time when nothing is being transmitted. One nth of the bus speed (i.e., on a 1 kHz bus, the bit time is 10^{-3} seconds).

BITS. (17) Binary digits.

BLACK BOX. (20) A component having known input and output, that can be readily inserted into or removed from a specific place in a larger system without knowledge of the component's detailed internal structure.

BLACKBOARD. (20) The controlling agent in a blackboard system that tracks activity between the knowledge sources.

BLACKBOARD SYSTEM. (20) An Expert System that uses multiple knowledge sources that share data.

BLOCK TRANSFER. (6) A data transfer mode allowing the transfer of variable length data blocks.

BOND, ELECTRICAL. (11) Electrical connection at two metallic surfaces securely joined to assure good conductivity often 2.5-mΩ maximum for electrical/electronic units and 1Ω for electrostatic dissipation or safety. A "faying surface" bond maintains contact between relatively large or long surfaces. Inherently bonded parts are permanently assembled and conductivity exists without special preparation: such as with welding, brazing.

BOOLEAN LOGIC. (20) The basic mathematics needed to study the logic design of digital systems. Boolean logic consists of two values and three basic operators.

BRAID, OVERBRAID. (11) Fine metallic conductors woven to form a flexible conduit or cableway and installed around insulated wires to provide protection against electric fields and radio frequencies. Best when peripherally connected to backshells. A grounding strap/jumper may be made of braid.

BREADTH FIRST. (20) The method of searching for solutions by considering all possible subgoals on one level before proceeding to the next level.

BRIDGE. (18) A BIU that is connected to more than one bus for the purpose of transferring bus messages from one bus to another, where all the buses follow the same protocol.

BROADBAND. (12) A frequency spectrum which is wide compared to the bandwidth of the device used to detect it.

BROADCAST. (4) Transmission of messages to all terminals without reference to the identification of the receiving station or terminal.

BROADCAST CAPABILITY. (6) The capacity to transmit messages to all terminals simultaneously.

BROADCAST DATA BUS. (18) A data bus where all messages are transmitted to all bus users.

BUFFER. (18) Memory used to hold segments of the data transferred between asynchronous processes.

BUS. (18) A conductor that serves as a common connection of a signal to multiple users.

BUS CONTROLLER. (18) The electronic unit that is designed to control the bus communication of all users for a centrally controlled bus.

BUS INTERFACE UNIT. (18) The electronics that interface the host CPU of an LRU to a bus medium.

BUS MESSAGE. (18) A complete set of bits that can be transferred between two bus users.

BUS NETWORK. (18) The collection of all BIUs and bus media associated with one bus.

BUS OVERLOAD. (18) The condition that exists when the time it takes to transmit outstanding messages on a bus exceeds the time allotted for those transmissions.

BUS USER. (18) Any LRU attached to a bus.

BYZANTINE RESILIENCE. (5) A fault tolerant process which is tolerant of intermittent faults that can send good information part of the time.

CABLE OR HARNESS. (11) A bundle of separate, insulated, electrical circuits, shielded or unshielded, usually long and flexible and having breakouts, terminations, overbraid, and mounting provisions completely assembled.

CABLEWAY. (11) A solid metallic housing (liner, foil, coating) surrounding and shielding insulated electrical conductors. Also called conduit, tray, or raceway. Crosswise or transverse openings or breaks in the metallic cableway cause noise voltages to be transferred to internal wire circuits.

CANARD. (16) A tail-first aerodyne, usually with auxiliary horizontal surface at the front and a vertical surface at the back.

CAT IIIa LANDING. (6) One of several landing categories defined in FAR 91. CAT IIIa implies the need for an instrument landing approach.

CENTRAL BUS CONTROL. (18) The bus control approach where a single electronic unit attached to a bus controls all the communication of the bus users.

CENTRAL CONTROL. (6) Control from one master, whether stationary or non-stationary.

CERTAINTY FACTOR. (20) A value that represents the level of belief associated with a fact or a rule.

CERTIFICATION. (18,20) The process of obtaining FAA approval for the design, manufacture, and/or sale of aircraft and associated systems, subsystems, and parts.

CHARACTER-ORIENTED PROTOCOL. (18) A communication protocol where messages can vary in length, with single character resolution.

CHARGE TRANSFER. (13) The integral of the current over its entire duration, i(t)dt, in coulombs.

CHECKSUM. (18) An error detection code produced by performing a binary addition, without carry, of all the words in a message.

CHOLESTERIC. (19) A liquid crystal phase with molecules parallel to each other but twisting slightly from layer to layer.

CHORD. (4) The straight line segment intersecting or touching an airfoil profile at two points.

CLOSED-LOOP. (18) A system where the output is a function of the input and the system's previous output.

CLOSED-LOOP. (20) A family of automatic control units linked together with a process to form an endless chain; the effects of control action are constantly measured so that if the controlled quantity departs from the norm, the control units act to bring it back.

CLUSTERING. (20) A technique to group similar objects together under the same label.

CODE. (17) The subset of software which exists for the sole purpose of being loaded into a computer to control it.

CODIFYING. (20) A method of representing knowledge or data so they can be stored in a knowledge base or database for access by a computer.

COLLEAGUE. (20) An Expert System that performs a significant subset of an expert's task.

COMBINER. (19) A screen designed to reflect selected wavelengths of light while remaining transmissive for others.

COMMAND/RESPONSE. (6) "Operation of a data bus system such that remote terminals receive and transmit data only when commanded to do so by the controller." (MIL-STD-1553 Designer's Guide, 1983, p. II-3.)

COMMAND/RESPONSE DATA BUS. (18) A data bus whose protocol initiates each data transfer with a command and terminates the transfer after a proper response is received.

COMMON MODE IMPEDANCE. (11) Impedance or resistance shared by two or more circuits so that noise voltages/currents generated by one are impressed on the others.

COMMON MODE REJECTION. (11) The ability of wiring or an electronic device to reject common mode (line-to-ground) signals and maintain fidelity of differential mode (line-to-line) signals.

COMMON MODE SIGNAL. (11) Identical and equal signals on input conductors or at the terminals of a device relative to ground.

COMPLEMENTARY QUALITY FACTORS. (17) Quality Factors with interrelated attributes.

COMPONENT DAMAGE. (13) Condition arising when the electrical characteristics of a circuit component are permanently altered beyond its specifications.

CONDUCTED EMISSION (CE) OR INTERFERENCE. (11) Voltage/current noise signals entering or leaving a unit on interface conductors. Emission is the general term, interference is undesired noise.

CONFIGURATION MANAGEMENT. (18) The precise control and documentation of the configuration of an entity at any time during its development and deployment.

CONNECTION WEIGHT. (20) A method of representing the strength of the connection between nodes.

CONTENT ADDRESSING. (6) The system of identifying message recipients based on information embedded in the message. This is in contrast to destination terminal addresses.

CONTENTION PROTOCOL. (18) A protocol that allows users to randomly access the bus at any time. When bus contention results, each user tries again to access the bus without contention.

CONTROL LAW. (7) The physical relationship between various sensors and control surfaces.

CONTROL REGISTER. (18) A register in an IC controller that receives commands from a host processor.

CONTROL STRUCTURES. (17) Programming constructs which direct the flow of control.

CONVENTIONAL PROGRAMMING. (20) The use of standard programming languages, as opposed to application development languages, financial planning languages, query languages, and report programs.

CONVERGENCE. (20) The point when the outputs of a Neural Network are no longer changing.

CORONA. (13) A luminous discharge that occurs as a result of an electrical potential difference between the aircraft and the surrounding atmosphere.

COUPLING. (11) The transfer of energy between wires or components of a circuit electrostatically, electromagnetically, or directly.

COVERAGE. (5) The conditional probability of the system successfully recovering from a component fault and continuing to perform the intended functions correctly, given the presence of the fault. Coverage is the measure of

effectiveness of a system's utilization of redundant hardware. Coverage can be qualified and applied to many different components of a system and phases of recovery process. Examples include, fault detection coverage, fault isolation coverage, latent fault coverage, sensor failure coverage, and memory failure coverage.

COVERAGE. (7) The percent confidence level of a given analytical redundancy fault detection and isolation algorithm for all types of faults.

COVERAGE. (9) The probability that when a fault occurs, it will be detected and recovery from the fault will be successful.

CRISP SET. (20) The membership concept used in traditional system modelling, where objects are either in or out of the set with no intermediate step.

CRITICAL. (13) Functions whose failure would contribute to or cause a failure condition which would prevent the continued safe flight and landing of the aircraft.

CROSS COUPLING (CROSSTALK). (11) Transfer of signals from one channel, circuit, or conductor to another as an undesired or nuisance signal or the resulting noise.

DAMAGE. (11) The irreversible failure of a component.

DATA BUS. (6,18,19) A system for transferring data between discrete pieces of equipment in the same complex.

DATA BUS PROTOCOL. (18) The set of rules that governs the transfer of data between data bus users.

DATA DRIVEN. (20) Describing the execution of a program in a data flow system, in which an instruction is carried out whenever all its input values are present.

DATA LATENCY. (6,18) The delay from the time when a piece of information becomes available at a source terminal to the time it is received at the destination.

DATA LINK ASSURANCE OF RECEIPT. (6) The guarantee of good data through the data link level.

DATA REASONABLENESS CHECK. (18) A check performed to see if a value of data is within reasonable bounds for the given context.

dBμV. (12) Decibels referred to one microvolt. Zero db represents one microvolt.

DECIBEL (dB). (11,12) Decibel expresses the ratio between two amounts of power, P1 and P2, at two separate points in a circuit. By definition, the number of dB = 10 log to the base 10 of (P1/P2). For special cases, when a standard power level P2 = 1 mW or 1 W or 1 kW, then the ratio is defined as "dBm," "dBw," or "dBKW." Because $P = V^2/R$ and also I^2R, decibels express voltage and current ratios. Ideally, the voltages and currents are measured at two points having

identical impedances. By definition, dB = 20 log V1/V2 and dB = log I1/I2. For convenience, V2 or I2 are often chosen as 1 μV or 1 μA and the ratio is defined as dB above a μV or dB above a μA when graphing emission or susceptibility limits.

DECLARATIVE. (20) Generally, the knowledge found in the knowledge base; independent knowledge that does not contain procedures or methods of handling other knowledge.

DECOUPLED MANEUVERS. (4) Changes in an aircraft's direction and attitude in one axis without affecting direction or attitude in other axes.

DEDUCTIVE. (20) Learning by logical inference or inferring from available knowledge.

DEFAULT DATA. (18) An alternative value used for a parameter whenever the normal data is not supplied.

DEFUZZIFICATION. (20) The process of translating fuzzy, or non-crisp values into crisp, clearly defined values.

DEMON. (20) A procedure activated by changing or accessing values in a database.

DENDRITE. (20) The pathway that carries the input of a biological neuron.

DEPTH FIRST. (20) A method of searching for a solution by pursuing each branch of the search tree to the end before considering another branch.

DESIGN ERROR. (4) A functional flaw resulting from a misinterpretation of the specifications of the system.

DESIGN MARGIN. (13) The difference between the equipment transient design levels and the transient control level.

DETERMINISTIC. (6) A system where all parameters are known, as opposed to a statistical system where the outcome is subject to the laws of probability.

DETERMINISTIC PROTOCOL. (18) A protocol where all parameters are known so that its various states are predictable in sequence and time.

DIAGNOSTIC FILTER. (7) An analytical algorithm which processes data from N functionally related sensors. The data are used to estimate some sensor outputs and assess the correct functioning of the sensors.

DICHROIC. (19) Having different light absorption characteristics based on incident polarization direction.

DIELECTRIC. (19) Having an insulating property with respect to an electrical field.

DIELECTRIC STRENGTH. (11) Voltage withstand capability that an insulating material sustains before destructive arcing and current flow, usually expressed

in volts per mil thickness. Dielectric withstand voltage is the voltage level at which insulation breakdown occurs.

DIFFERENTIAL MODE (DM) SIGNAL. (11) The signal in a two-wire circuit measured from line-to-line.

DIGITAL DATA BUS. (18) A data bus that uses digital electronic signals.

DIRECT EFFECTS. (13) Any physical damage to the aircraft or onboard systems due to the direct attachment of the lightning channel. This includes tearing, bending, burning, vaporization, or blasting of aircraft surfaces or structures, and damage to electrical/electronic systems.

DISSIMILAR REDUNDANCY. (18) The redundancy of systems that provide a redundancy of function, but by a different form.

DISSIMILAR SOFTWARE. (18) Redundant computer programs that provide a redundancy of function, but by a different form.

DISTRIBUTED BUS CONTROL. (18) The bus control approach where the total communication control job is distributed across the bus users, each controlling the communications during its period of responsibility.

DISTRIBUTED CONTROL. (6) Concurrent control from multiple points in the data bus system.

DOT PRODUCT. (20) The inner product of vectors $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ from n-dimensional euclidean space is the sum of $s_i y_i$, as i ranges from 1 to n.

DOUBLE FAIL-OPERATIONAL SYSTEM. (4) A quadruplex (or higher) redundant flight-control system which is designed to incur failures in two redundant lanes (or channels) before it fails.

DUAL-DUAL ARCHITECTURE. (4) Two parallel dual computers with a voting plane at the output of each dual computing lane.

DUAL FAIL-OPERATIONAL. (7) A reliability requirement placed on a system which requires the system to be operational after two failures have occurred.

DUAL GROUND. (11) Equipment case ground/return through two independent circuit paths to structure implemented in flammable zones and water leakage areas-each path meeting electrical conductivity (resistance) requirements.

ELECTRIC FIELD. (11) High-impedance, radiated voltage field, positive or negative, from a voltage source as contrasted to a low-impedance magnetic field from a current source.

ELECTROMAGNETIC COMPATIBILITY (EMC). (11) Operation within performance specification in the intended electromagnetic interference environment.

ELECTROMAGNETIC INTERFERENCE (EMI). (11) Conducted and radiated voltage/current noise signals, broadband (BB) or narrow band (NB), that degrade the specified performance of equipment.

ELECTROMIGRATION. (5) Drifting of metal atoms toward the cathode of a cathode ray tube.

ELECTROSTATIC CHARGE. (11) Electric potential energy with a surrounding electric field, uniform or nonuniform, moving or at rest, on a material.

EMISSION. (11) Voltage/current noise on a wire or in space. Broadband emission has uniform spectral energy over a wide frequency range and can be identified by the response of a measuring receiver not varying when tuned over several receiver "bandwidths." Or, energy present over a bandwidth greater than the resolution bandwidth where individual spectral components cannot be resolved. Broadband (BB) may be of two types: (1) impulse and coherent varies 20 dB per decade of bandwidth and (2) random or statistical, varies 10 dB per decade. A narrow band (NB) emission or signal, sometimes called continuous wave, occurs at a discrete frequency and does not vary with bandwidth.

EMPIRICAL MODEL. (20) The model of a system based on data and the principles of statistics. Empirical models produce relationships, not insight.

EMULATION. (18) The duplication of the behavior of a system with a different system.

ENVELOPE LIMITING. (4) General or additional limits imposed on the structural, "g" limits, speed, attitude, etc. of the aircraft. In some cases, envelope limiting imposes additional constraints on the envelope that cannot be exceeded regardless of pilot inputs.

EQUIPMENT TRANSIENT DESIGN LEVEL. (13) The level of transients which the equipment is qualified to withstand.

EQUIPMENT TRANSIENT SUSCEPTIBILITY LEVEL. (13) The transient level which will result in damage or upset to the system components. This level will be greater than the equipment transient design level.

EQUIVALENCE STATEMENT. (17) A FORTRAN statement which equates two variable names.

ERROR. (4) A mistake in specification, design, production, maintenance, or operation of a system causing undesirable performance.

ERROR. (8) A state of the system which (in the absence of any corrective action by the system) could lead to a failure that would not be attributed to any event subsequent to the error. (More accurately known as an erroneous state.)

ERROR MASKING. (18) The process of masking the presence of avionic errors, possibly by using an electronic voter to override an erroneous input with the values of substitute inputs.

EVENT. (20) A change to the blackboard of a Blackboard System.

EVENT, EXTREMELY IMPROBABLE. (4) An event with a probability of occurrence on the order of 10^{-9} or less.

EVENT, IMPROBABLE. (4) An event with a probability of occurrence on the order of 10^{-5} or less.

EVENT, PROBABLE. (4) An event with a probability of occurrence on the order of 10^{-5} or greater.

EXPERT. (20) 1. An Expert System that approaches an expert's level of performance within a given domain. 2. A person who has mastered solving specific types of problems.

EXPERT SYSTEM. (20) A computer system designed to simulate the problem solving behavior of a human expert.

EXPERT SYSTEM SHELL. (20) An Expert System development tool and run-time environment with a modifiable knowledge base.

EXTERNAL ENVIRONMENT. (13) Characterization of the natural lightning environment with idealized waveforms for engineering purposes.

FACETTE. (20) A piece of knowledge in a frame.

FAIL-OPERATIONAL. (7) A reliability requirement placed on a system which requires the system to be operational after a single failure has occurred.

FAIL-SAFE. (7) A reliability requirement placed on a system which requires that safe flight not be hindered even after a failure.

FAIL-SAFE. (18) A design philosophy that ensures that any failure in a system does not result in an unsafe condition after the failure.

FAILURE. (4) The inability of a system, subsystem, unit, or part to perform within specified limits.

FAILURE. (5) The deviation of system behavior from specifications (arithmetic failure, storage failure, flight control function failure.)

FAILURE. (8) The situation when the external behavior of a system does not conform to that prescribed by the system specification.

FAILURE, HARD. (5) Repeated use of the same input and initial conditions results in the same incorrect response.

FAILURE, HIDDEN. (4) A failure that is not manifested at the time of its occurrence.

FAILURE MECHANISM. (5) Any situation that could produce an error condition. Examples of failure mechanisms include metal migration, voltage overstress, and lack of air-conditioning.

FAILURE, PERMANENT. (5) Repeated use of the same input and initial conditions results in the same incorrect response.

FAILURE, SOFT. (5) Repeated use of the same input and initial conditions does not result in the same incorrect response.

FAILURE, TEMPORARY. (5) Repeated use of the same input and initial conditions does not result in the same incorrect response.

FAILURE, TRANSIENT. (5) Repeated use of the same input and initial conditions does not result in the same incorrect response.

FALL-TIME. (12) The time required for pulse amplitude to go from a predefined magnitude to a given level.

FALSE ALARM. (7) The declaration of a fault by a fault detection monitor or algorithm when there is no fault.

FAULT. (4) An error in the operation of a system.

FAULT. (5) The phenomenological reason for a failure (open wire, stuck-at fault, design fault, etc.). In general, any condition preventing a digital component from correctly changing state when directed to change by input parameters. For electrical components there is a one-to-one correspondence between faults and failures. The situation is not so simple with digital circuits. For if the circuit is S-A-1, any input causing a one output will be correctly processed; a little like the stopped clock that is correct twice per day. For a processor having a million or so logic gates, it is not possible to test for all the combinations of input and output states.

FAULT. (8) The adjusted cause of error.

FAULT AVOIDANCE. (9) The attempt to prevent any software faults in the final delivered product through disciplined software development practices, testing, and IV&V.

FAULT CONTAINMENT. (6,9) The capacity of a system to prohibit errors and/or failures from propagating from the source throughout the system.

FAULT CURRENT. (11) The maximum current (magnitude and duration) flowing through a fault point. This current is equal to the supply voltage divided by the dc resistance of power line leads, circuit breakers, and the current return in wire or structure.

FAULT DETECTION. (6) The capacity of a system to determine the occurrence of erroneous operation.

FAULT DETECTION. (7) The determination that a sensor is faulted by using a software algorithm.

FAULT, HARD. (4) A defect in the hardware or software of a digital control system that permanently affects some functional performance of the system.

FAULT INSERTION. (4) A testing technique used to obtain information about data latency and built-in test coverage of a digital flight-control system.

FAULT ISOLATION. (6) The capacity of a system to isolate a failure to the required level so it can reconfigure.

FAULT ISOLATION. (7) The determination that a particular sensor is faulted by using a software algorithm.

FAULT, LATENT. (5) A fault which has not yet caused a failure. (For example, a fault in a memory chip that is not being used for the foreground program or in this particular mode of the system is a latent fault.)

FAULT, SOFT. (4) A transient defect in the software of a digital flight-control system that can be overcome by error-correctable code or by recycling of power to the computer system.

FAULT, STUCK-AT. (5) A logic signal which remains at zero (S-A-0) or one (S-A-1).

FAULT TOLERANCE. (6,9) The capability to endure errors and/or failures without causing total system failure.

FAULT TOLERANCE. (7) Accommodation of sensor hardware faults based on some type of comparator scheme.

FAULT TOLERANCE. (18) The ability of a system to continue operation after a fault, possibly in a degraded condition.

FAULT TOLERANT. (4,9) Software which continues to operate satisfactorily in the presence of faults.

FAULT TOLERANT SYSTEM. (5) A system that continues to function although certain components may have faults.

FAULT TREE ANALYSIS. (4) A top-down deductive analysis that identifies the conditions and functional failures necessary to cause a defined failure condition. The fault tree can be used to establish the probability of the ultimate failure condition occurring as a function of the estimated probabilities of contributory events.

FEDERAL AVIATION REGULATIONS. (18,19) Subchapter C of the Code of Federal Regulations, Title 14, Chapter 1.

FEED FORWARD NEURAL NETWORK. (20) A Neural Network where the only connections are from a neuron in one layer to a neuron in the next layer.

FILTER. (11) Device or unit that passes or rejects a frequency band and is designed to block noise from entering or leaving a circuit or unit.

FILTER. (20) A transmission network used in electrical systems for the selective enhancement of a given class of input signals.

FINITE STATE MACHINE. (18) A state machine with a finite number of states.

FLIGHT CODE. (4) The application software of the digital flight-control system.

FLIGHT-CRITICAL. (4,7,18) A description of functions whose failure would contribute to or cause a failure condition preventing the continued safe flight and landing of the aircraft.

FLIGHT-ESSENTIAL. (4,18) A description of functions whose failure would contribute to or cause a failure condition which would significantly affect the safety of the airplane or the ability of its crew to cope with adverse operating conditions.

FLIGHT-NONESSENTIAL FUNCTION. (18) A function whose failure could not significantly degrade aircraft capability or crew ability.

FLIGHT-PHASE CRITICAL. (4) A description of functions which are critical only during certain phases of flight.

FLY-BY-GLASS. (16) Flight control system where fiber optics carry the signal.

FLY-BY-LIGHT. (4,16) Flight control system where fiber optics carry the signal.

FLY-BY-WIRE. (4,16) Flight control system with electric signaling.

FORWARD CHAINING. (20) A method of solving problems by beginning with certain data and moving down the inference chain until a solution is reached.

FORWARD RECOVERY. (9) Restoration of the system to a consistent state by compensating for inconsistencies found in the current state so that the system may continue processing.

FOURIER TRANSFORM. (12) A mathematical method for deriving the frequency spectrum from a time dependent function.

FRAME. (18) A formatted block of data words or bits that is used to construct messages.

FRAME. (20) A way of representing Expert System knowledge that consists of a set of slots that contain data.

FRAME LABEL. (20) A unique frame identification.

FUNCTIONAL PARTITIONING. (18) The partitioning of system functions by placing each group of users, which share a common function, on different data buses.

FUZZIFICATION. (20) The process of converting crisp inputs into fuzzy values.

FUZZY CONTROL VARIABLE. (20) A parameter whose value determines the action taken by fuzzy rules.

FUZZY EXPERT SYSTEM. (20) An Expert System that uses fuzzy logic techniques to solve problems.

FUZZY LOGIC. (20) The theory of representing vague or imprecise concepts to model inherent conditions.

FUZZY SET. (20) An extension of the concept of a set, in which the characteristic function which determines membership of an object in the set can take on any value between 0 and 1.

FUZZY TERMS. (20) Words used for descriptions that may not be precise in determining set memberships.

GATEWAY. (18) A bus user that is connected to more than one bus for the purpose of transferring bus messages from one bus to another, where the buses do not follow the same protocol.

GENERAL AVIATION AIRCRAFT. (18) The non-air transport civil aircraft.

GETTER. (19) A metal alloy used in a vacuum tube to absorb residual gasses.

GENERATOR POLYNOMIAL. (18) The polynomial code that is used to generate the remainder in the division of the CRC check.

GIGABIT. (16) One billion bits.

GLASS COCKPIT. (9) Advanced state-of-the-art electronic displays utilizing flat panel and/or cathode ray tube display technology for cockpit instrumentation.

GLOBAL STATE. (18) A state that represents the condition of the entire network being modeled, including senders, receivers, and the communication link.

GRACEFUL DEGRADATION. (20) A programming technique to prevent catastrophic system failure by allowing the machine to operate, though in a degraded mode, despite failure or malfunction of several integral units or subsystems.

GRAY SCALE. (19) A series of tones, varying from black to white.

GROUND. (11) A generic term having multiple meanings and indicating a circuit return path or a voltage reference: not "zero" voltage reference. Four hundred millivolts of noise voltage is common on "quiet" grounds. There are several types of returns and references.

GROUND EFFECT. (4) Increase in aircraft lift when operating near the ground.

HALF-DUPLEX. (18) Bidirectional communication between two entities on a single channel by each having a turn to control the channel.

HAMMING CODE. (18) An error detection and correction code based on the Hamming distance.

HAMMING DISTANCE. (18) The number of bit positions in which two binary words differ.

HANDSHAKING. (18) The reciprocal responses given by two electronic systems to sequence the steps of a transfer of data between them.

HARD FAILURE. (12) A failure that requires a reset of the equipment.

HARDWARE. (17) The physical components of a computer.

HARDWARE-IN-THE-LOOP SIMULATION. (18) A partial simulation of a system; part of the actual system is used in the simulation.

HAZARD FUNCTION. (8) The conditional probability that a fault is exposed in the interval t to Δt given that the fault did not occur prior to time t.

HERMETIC. (19) Having an airtight seal.

HETEROASSOCIATIVE. (20) A Neural Network where output patterns are distinct from input patterns.

HEURISTIC. (20) Any rule of thumb, strategy, or technique used to limit the time required to search for solutions in large problem spaces.

HEURISTIC REASONING. (20) Using an expert's rules of thumb, in the absence of precise control mechanisms, to reduce the space that must be searched for a solution.

HIDDEN LAYER. (20) The middle layer in a Neural Network which takes outputs from the input layer, processes them, and passes them to the output layer.

IMMUNITY. (11) Capability of a circuit or unit to operate within performance specification in a specified electromagnetic interference environment.

INDIRECT EFFECTS. (13) Voltage and/or current transients induced by lightning in aircraft electrical wiring which can produce upset and/or damage to components within electrical/electronic systems.

INDUCED VOLTAGES. (13) A voltage produced around a closed path or circuit by changing magnetic or electric fields or structural IR voltages.

INDUCTIVE. (20) Learning by information repetition.

INFERENCE CHAIN. (20) The path of reasoning that the inference engine follows to find a solution to the problem.

INFERENCE ENGINE. (20) The part of an Artificial Intelligence system that uses knowledge in the knowledge base and acquired knowledge about the problem to form an expert solution.

INFERENCE MECHANISM. (20) Controls the use of the knowledge base and databases when solving a problem.

INITIALIZATION. (6) Setting the beginning parameters and values on system power-up. For redundant systems this includes setting the initial configuration of the system.

INPUT LAYER. (20) The first layer of neurons in a Neural Network. The input layer accepts the input values and passes them to the hidden layer. The input layer does not process data.

INTERNAL ENVIRONMENT. (13) The fields and structural IR potentials produced by the external environment, along with the voltages and currents induced by them.

INTERRUPT VECTOR. (18) The address that points to the beginning of the service routine for an interrupt.

INTERRUPT VECTOR TABLE. (18) The table of interrupt vectors for all interrupts serviced by a system.

ISOLATION. (11) Electrical separation and insulation of circuits from ground and other circuits or arrangement of parts to provide protection and prevention of uncontrolled electrical contact.

ISOTROPIC. (19) Having the same physical properties in all directions.

JOULE. (12) A unit of energy equal to one watt-second.

JUMPER/STRAP. (11) A short wire, strip, strap, or braid conductor installed to make a safety ground connection, to dissipate electrostatic charge, or establish continuity around a break in a circuit.

KILOBYTE. (16) One thousand bytes.

KNOWLEDGE ACQUISITION. (20) The process of gathering the required information from various sources by a knowledge engineer.

KNOWLEDGE BASE. (20) A form of coded knowledge about a specific domain.

KNOWLEDGE-BASED SYSTEM. (20) A computer system whose usefulness derives primarily from a database containing human knowledge in a computerized format.

KNOWLEDGE ENGINEER. (20) The person responsible for extracting, organizing, and encoding the knowledge related to the problem, and importing it into a knowledge base or database.

LABEL. (20) A data item that serves to identify a data record, or a symbolic name used in a program to mark the location of a particular instruction or routine.

LABELED ADDRESSING. (6) The system of identifying message recipients based on labels. This is in contrast to destination terminal addresses.

LATENT FAULT. (10) A fault which has not yet produced a malfunction. (In the context of the single-fault model, benign and latent faults are equivalent.)

LAYER. (20) A row of neurons in a Neural Network; there are usually three layers in a network.

LIGHTNING FLASH. (13) The total lightning event in which charge is transferred from one charge center to another. It may occur within a cloud, between clouds, or between a cloud and the ground. It can consist of one or more strokes, plus intermediate or continuing currents.

LIGHTNING LEADER STROKE. (13) The leader forms an ionized path for charge to be channeled towards the opposite charge center. The stepped leader travels in a series of short, luminous steps prior to the first return stroke. The dart leader reionizes the return stroke path in one luminous step prior to each subsequent return stroke in the lightning strike.

LIGHTNING RETURN STROKE. (13) A lightning current surge that occurs when the lightning leader makes contact with the ground or an opposite charge center.

LIGHTNING STRIKE. (13) Any attachment of the lightning flash to the aircraft.

LIGHTNING STRIKE ZONES. (13) Locations on the aircraft where the lightning flash will attach or where substantial amounts of electrical current may be conducted between attachment points. The location of these zones on any aircraft is dependent on the aircraft's geometry and operational factors and often varies from one aircraft to another.

LIMITING, VOLTAGE/CURRENT. (11) Semiconductor components, diodes, Transorb, or filter designed to clip and shunt to ground an applied transient or steadystate voltage. Used to protect against noise frequencies, faults, lightning, and inductive switching transients.

LINE REPLACEABLE UNIT. (18,19) An electronics unit that is made to be replaced on the flight line, as opposed to one that requires the aircraft be taken to the shop for repair.

LINEAR BUS. (18) A bus where users are connected to the medium; one on each end, with the rest connected in between.

LOGICAL SUM. (20) The final unified result of the rule processes of a fuzzy system.

LOW-PASS FILTER. (12) An electrical circuit which allows the passage of low frequencies and prevents the passage of high frequencies.

MACHINE LEARNING. (20) The process or technique by which a device modifies its own behavior as the result of its past experience and performance.

MAGNETIC FIELD. (11) A radiated, low-impedance field having lines of "flux" or magnetomotive force associated with an electrical current.

MALFUNCTION. (11) Failure or degradation in performance that compromises flight safety.

MANCHESTER II MODULATION. (18) A non-return to zero, bipolar modulation of a voltage that encodes bits based on the zero-crossing direction of the signal.

MEAN AERODYNAMIC CHORD (also mean chord). (4) The chord of an airfoil whose length is equal to the area of the airfoil section divided by the span.

MEAN FAILURE RATE. (10) A measure of survivability defined as the reciprocal of the mean time to system failure.

MESSAGE STRUCTURE. (6) The organization of both protocol and data information in a message.

META-KNOWLEDGE. (20) Knowledge that reveals details about the system knowledge.

METRIC. (17) A measure.

MICRON. (16) One-millionth of a meter.

MISSED ALARM. (7) The failure of a fault detection monitor or algorithm to detect a fault when there is a sensor fault.

MODEL. (20) A mathematical or physical system, obeying certain specified conditions, whose behavior is used to understand a physical, biological, or social system to which it is analogous in some way.

MODELING. (18) Creating a system of mathematical equations that formulate all the significant behavior of a system.

MODULE. (17) A unit of code which implements a function.

MONITORABILITY. (6) The capacity of the protocol to be viewed passively to allow observation of the dynamics of the protocol.

MONOLITHIC. (19) A single substrate used for an integrated circuit.

MONOTONIC FUNCTION. (17) A function in which a certain change in the measure always represents a certain change in the property being measured, where either change is simply an increase or decrease in magnitude.

MULTIPLE BURST. (13) A randomly spaced series of bursts of short duration, low amplitude current pulses, with each pulse characterized by rapidly changing currents. These bursts may result from lightning leader progression or branching and may be accompanied by or superimposed on stroke or continuing currents. The multiple bursts appear to be most intense at the time of initial leader attachment to the aircraft.

MULTIPLE STRIKE. (13) Two or more lightning strikes during a single flight.

MULTIPLE STROKE. (13) Two or more return strokes occurring during a single lightning flash.

MULTIPLE TRIP MONITOR. (7) A fault detection algorithm which declares a fault after the sensor output has exceeded a predefined threshold N times.

MULTIVERSION PROGRAMMING. (18) N-version programming.

N-VERSION PROGRAMMING. (18) The independent coding of a number, N, of redundant computer programs that are run concurrently for the purpose of comparing their outputs.

NANOSECOND. (16) One-billionth of a second.

NEGATIVELY STABILIZED. (4) Aircraft design in which the point of effective lift is aft of the center of gravity.

NEMATIC. (19) A liquid crystal phase with molecules having a single optical axis in line with an applied magnetic field.

NETWORK CONTROL STRATEGY. (6) The solution proposed by the designer in addressing his specific problem (design flexibility).

NEURAL NETWORK. (20) A system modelled after the brain, used to solve problems by a mapping of input data to output data.

NEURO-FUZZY. (20) Integrating a Neural Network and fuzzy logic to solve problems.

NEURON. (20) The computational element in a Neural Network. A neuron computes the weighted sum of the inputs from other neurons and produces an output.

NODE. (20) 1. Representation of an object that is related to other objects via connection weights. 2. The part of a Neural Network that calculates and processes information.

NOISE. (11) Conducted or radiated emission causing circuit upset, performance disorder, or undesired sound.

NOISE. (20) Meaningless or erroneous bits that must be ignored or removed from a signal, especially in communication channels.

NON REAL-TIME LEARNING. (20) Learning done in a Neural Network before attempting to solve application problems.

NONLINEARITY. (20) A system in which the outputs do not correspond to the inputs in a direct or inversely proportional relationship.

NUMERICAL APERTURE. (6) The angle of acceptance of light from a light source for a given fiber optic cable.

OBJECT CODE. (17) The translation of source code that is loaded into a computer.

OBJECT-ORIENTED. (20) A representation form for knowledge in which all properties of an object are associated by the outgoing and incoming arcs at its node; the representation is geared towards manipulating objects as independent pieces of knowledge.

OBSERVER. (7) An algorithm which models physical relationships between sensor data and uses the data to provide fault detection for one or more sensors. This is also known as a Luenberger observer or a signal blender.

OPERANDS. (17) The variables or constants on which the operators act.

OPERATORS. (17) Symbols which affect the value or ordering of operands.

OPPORTUNISTIC REASONING. (20) The process of solving problems in a way that is not uniform or predictable. Opportunistic reasoning attempts to solve the problem in the best possible way for each individual situation. Rules fire when they are appropriate, without a priori ordering.

OPTIMIZING COMPILER. (17) A computer program which, while translating source code into object code, removes inefficiencies from the code.

OUTPUT LAYER. (20) The layer of neurons in a Neural Network that takes the outputs from the hidden layer, processes them, and then produces the output of the network.

OVERHEAD. (18) The message timing gaps, control bits, and error detection bits added to some data to satisfy the data bus protocol.

PARALLEL PROCESSING. (20) Performing many different calculations simultaneously to reach one conclusion or solution.

PARAMETERIZATION CAPABILITY. (6) A measure of how well the attributes of the protocol can be described by parameters.

PARITY. (18) An error detection bit added to a data word based on whether the number of "one" bits is even or odd.

PARTITIONED. (18) Colocated hardware or software functions that are designed so that adverse interactions between them cannot occur.

PEAK RATE OF RISE. (13) The maximum instantaneous slope of the waveform as it rises to its maximum value. Mathematically, the peak rate of rise of a function, $i(t)$, may be expressed as the maximum of $d[i(t)]/dt$.

PETRI NET. (18) A state analysis diagram that tracks the status of the state transition conditions of a state machine.

PHONEME. (19) The smallest elements of speech.

PHOTOLITHOGRAPHY. (19) An integrated circuit fabrication technique using photographically produced masks.

PIN LEVEL TEST. (12) An EMC test in which voltage or current is applied directly to a conductor at a connector pin.

PIXEL. (19) The smallest picture element of a display.

POINT-MASS SIMULATION. (4) Same as state variables airplane model (q.v.).

POLLING. (18) A method whereby a CPU monitors the status of a peripheral by periodically reading its status signals.

POLYNOMIAL CODE. (18) A sequence of bits that represents the coefficients of each term in a polynomial.

POSITIVELY STABILIZED AIRCRAFT. (4) Aircraft design in which the effective point of lift is forward of the center of gravity.

PRECIPITATION STATIC (P-static). (11) Electrostatic discharge, corona, arcing, and streamering, steady state or impulsive, causing circuit upset, receiver noise or component damage.

PREDICATE CALCULUS. (20) A formal language of classical logic that uses functions and predicates to describe relations between individual entities.

PREDICATE/TRANSITION NETWORK. (4) A bipartite graph (a type of linear graph) to model concurrency between redundant concurrent events. Basically a modified generalized petri net.

PRIMARY STATION. (18) An intelligent HDLC protocol user, usually used to manage the access of other bus users to the bus.

PROBABILITY. (20) A theory dealing with the uncertainty that results from random behavior. Probability is defined over the numeric range of 0 to 1.

PROCESSING ELEMENT. (20) The part of the Neural Network where the computations are performed.

PRODUCTION RULES. (20) A common technique for representing procedural knowledge in an Artificial Intelligence system.

PROGRAM. (17) A detailed set of instructions for accomplishing some purpose.

PROPAGATION DELAY. (18) The time it takes an electrical signal to travel from its source to its destination.

PROTOCOL. (18) The set of rules by which all bus users must abide to access the bus and ensure its specified operation.

Q. (12) The quality factor of a resonant circuit which is the ratio of the energy stored to the power dissipated per cycle.

QUADRUPLEX ARCHITECTURE. (4) The use of four separate lanes (or channels) of computer redundancy. Each lane can fail separately providing a fail-operational capability for the digital flight-control system.

QUALITY MEASURE. (17) A repeatable, monotonic relationship relating measures of objects (a set of numbers) to subjective qualities.

RADIATED EMISSION (RE). (11) Electromagnetic energy transmitted and propagated in space usually considered as audio frequency or radio frequency noise.

RADIO FREQUENCY (RF). (11) Frequencies in the electromagnetic spectrum used for radio communications extending from kilohertz to gigahertz.

RADIO FREQUENCY INTERFERENCE (RFI). (11) Electromagnetic interference in the radio frequency range.

RASTER. (19) A pattern of parallel electron beam scan lines used for uniform coverage of a CRT screen.

REAL-TIME LEARNING. (20) Learning done in a Neural Network while solving an application problem.

RECONFIGURATION. (6) The capacity of a system to rearrange or reconnect the system elements or functions.

RECONFIGURATION. (18) The process of a system reassigning which hardware performs a particular function.

RECOVERY BLOCK. (18) A block of code executed upon detection of a fault to recover from the erroneous condition that results.

RECOVERY CACHE. (9) The location used to preserve input values until the outputs resulting from them have been accepted.

RECURSION. (20) A technique in which an apparently circular process is used to perform an iterative process.

REDUNDANCY MANAGEMENT. (7) The computer processing which is needed to implement fault detection and isolation algorithms.

REFERENCE. (11) 1. Structure, for electronics, shields, power. 2. A grid of wires, solid sheet, or foil. 3. A wire from circuit to grounding block or case. 4. A wire from circuit to structure. 5. Shield tie. 6. Earth.

REGISTER. (18) A single word of RAM located within an IC controller that is used for transferring data and control information.

RELAXED STATIC STABILITY AIRCRAFT. (4) An aircraft whose center of gravity is behind the wing's point of effective lift.

RELIABILITY ANALYSIS. (4) A means of determining the probability of failure in a system. Military flight-critical systems typically are required to have

reliability levels of 10^{-5} to 10^{-7}, whereas civil flight-critical systems have reliability levels of 10^{-9} or less.

REMOTE TERMINAL. (18) The BIU portion of a MIL-STD-1553 bus user.

RESONANCE. (12) Resonance occurs in an electrical circuit when the energy stored in the inductance is equal to the energy stored in the capacitance.

RETURN. (11) 1. Structure, for power, fault, and "discrete" circuits. 2. A grid of wires, solid sheet, or foil. 3. A wire from circuit load back to source or to case. 4. Circuit card "ground plane," also a reference and shield.

RETURN STROKE. (13) See lightning return stroke.

REVERSION MODE. (7) The high level of redundancy in a system having different redundancy requirements for some sensors. Critical sensors may have a high level of redundancy while other sensors have low levels.

RING BUS. (18) A bus where users are connected only to the two adjacent users in a continuous ring; each connected to the next and the last one connected to the first one.

RISE-TIME. (12) The time required for a voltage pulse to reach a predefined magnitude from a given level.

ROBUSTNESS. (9) The ability of the code to perform despite some violation of the assumptions in its specifications usually via substitution of an alternate value and continuation of execution if a software fault is detected.

ROLLBACK. (9) Retrying the calculation in the event that a failure is detected, under the assumption that some external condition may have changed thereby resolving the anomaly.

RULE-BASED. (20) An Artificial Intelligence program where knowledge is represented as rules.

RULES. (20) A method of representing knowledge in the form of IF THEN statements.

SCHEDULER. (20) A part of an inference engine that determines which knowledge source should be activated, and in what order.

SEALANT. (11) An applied substance enclosing and protecting the integrity of a joint, fastener, or electrical bond from moisture, contaminants, oxidation, and acid or alkaline corrosion.

SEARCH TREE. (20) The branches of possible solutions to a problem that start with initial information and spread out following different paths.

SECONDARY STATION. (18) A simple HDLC protocol user.

SELF-SUPERVISED. (20) A method of training a Neural Network without the use of an external monitor. A feedback device that detects errors and adjusts the connection weights accordingly is used.

SEMANTIC NETWORK. (20) A method of representing knowledge using nodes and arcs.

SENSOR. (7) An instrument which measures a particular physical parameter. The data output may be digital or analog and is utilized by the flight computer.

SENSOR. (18,19) Any transducer that converts the measurement of a physical quantity to an electrical signal.

SEQUENTIAL LIKELIHOOD RATIO TEST. (7) A fault detection algorithm which is based on two hypothesized density functions of no fault or sensor fault.

SEQUENTIAL PROBABILITY RATIO TEST. (7) See sequential likelihood ratio test.

SERIAL DATA BUS. (18) A data bus capable of sending only one bit at a time, in series.

SERVICE SPECIFICATION. (18) The specification of the service provided by a protocol layer.

SHIELD. (11) A conductive material, opaque to electromagnetic energy, for confining or repelling electromagnetic fields. A structure, skin panel, case, cover, liner, foil, coating, braid, or cable-way that reduces electric and magnetic fields into or out of circuits or prevents accidental contact with hazardous voltages.

SHIELD EFFECTIVENESS (SE). (11) The ability of a shield to reject electromagnetic fields. A measure of attenuation in field strength at a point in space caused by the insertion of a shield between the source and the point.

SHIELDING. (12) Any metallic structure such as the aircraft fuselage or the woven braid on a cable that provides protection against electromagnetic fields.

SIGNAL RETURN. (11) A wire conductor between a load and the signal or driving source. Structure can be a signal and power return. Commonly, it is the low voltage side of the closed loop energy transfer circuit.

SIMULATION. (18,19) An approximated representation of the behavior of a system with a similar system.

SINGLE-ENDED CIRCUIT. (11) A circuit with source and load ends grounded to case and structure and using structure as return.

SINGLE-POINT FAILURE. (18) A failure of a component that, by itself, causes the failure of the system in which it is contained.

SINUSOID. (12) A wave form that follows the mathematical values of a sine function.

SLOTS. (20) Subdivisions making up the frame that may contain data, procedures, or pointers to other frames.

SMECTIC. (19) A liquid crystal phase with molecules arranged in layers.

SOFT FAILURE. (12) A failure which causes an alteration of data or missing data.

SOFTWARE. (17) Computer programs and the documentation associated with the programs.

SOFTWARE METRIC. (17) A measure of software objects.

SOFTWARE QUALITY FACTOR. (17) Any software attribute that contributes either directly or indirectly, positively or negatively, toward the objectives for the system in which the software resides.

SOFTWARE QUALITY METRIC. (17) (1) A measure that relates measures of the software objects (the symbols) to the software qualities (quality factors). (2) The measure of a software quality factor.

SOURCE CODE. (17) Code that can be read by people.

SPECIAL CONDITION. (18) A regulatory document that adds to, or otherwise alters, the airworthiness standards for particular aircraft.

SPURIOUS STATE. (20) A point in a Neural Network that acts as an attractor, but is not one of the stored patterns.

STABILITY. (20) The property of a system which remains under control and responds in a reasonable manner to an applied input.

STATE-VARIABLE AIRPLANE MODEL (also point-mass model). (4) Fixed aerodynamic variables are used in the solution of the equations of motion of the model instead of using look-up tables in which each derivative varies with airspeed, altitude, etc. The model performance is only accurate at or near the point in the flight envelope for which the variables are chosen.

STATIC MARGIN. (4) The degree of instability in a relaxed statically stable airplane.

STATION. (18) Bus user.

STATIONARY BUS CONTROL. (18) Bus control that is continually performed by a single bus controller, or by one of its backups.

STATUS REGISTER. (18) A register in an IC controller that holds the status of the state of certain controller functions.

STROKE. (19) A nonuniform technique of CRT screen refresh using penlike control of the electron beam.

STROUD NUMBER. (17) The total number of elementary mental discriminations that a person makes per second.

STRUCTURAL IR VOLTAGE. (13) The portion of the induced voltage resulting from the product of the distributed lightning current, I, flowing through the resistance, R, of the aircraft skin or structure.

STRUCTURE. (11) Basic members, supports, spars, stanchions, housing, skin panels, or coverings that may or may not provide conductive return paths and shields for electrical/electronic circuits.

STUB. (18) The short length of cable used to attach a single LRU to a data bus.

SUBROUTINE. (17) A self-contained body of code which can be called by other routines to perform a function.

SUPER-DIAGNOSTIC FILTER. (7) An algorithm which provides all the capabilities of a diagnostic filter. Additionally, it can isolate a specific faulted sensor. At the current time, this is the most complex technique used to implement analytical redundancy.

SUPERVISED LEARNING. (20) A method of training a Neural Network in which an external supervisor monitors the activities, and learning occurs on the basis of direct comparison of the output of the network with known correct answers.

SUSCEPTIBILITY. (11) Upset behavior or characteristic response of an equipment when subjected to specified electromagnetic energy. Identified with the point, threshold, or onset of operation outside of performance limits. Conducted Susceptibility (CS) applies to energy on interface conductors; Radiated Susceptibility (RS) to radiated fields.

SWEPT STROKE. (13) A series of successive attachments due to sweeping of the flash across the surface of the airplane by the motion of the airplane.

SYMBOLIC PROCESSING. (20) A method of processing knowledge where the relationships among the knowledge are stored using symbolic representations; thus, the system can deal freely with objects and not be concerned with their composition.

SYNAPSES. (20) Pathways in a biological neuron that transmit pulses from one neuron to another.

SYNCHRONOUS. (20) In step, or in phase, as applied to two or more circuits, devices, or machines.

SYNCHRONOUS MESSAGES. (6) Messages transmitted at a known a priori sequence and time or time interval.

SYSTEM EXPOSURE TIME. (4) The period during which a system may fail. This period extends from the last verified proper functioning to the completion of the next required performance.

SYSTEM FUNCTIONAL UPSET. (13) Impairment of system operation, whether permanent or momentary (e.g., a change of digital or analog state) which may or may not require manual reset.

SYSTEM INTEGRATOR. (18) The developer who has the responsibility to integrate the various subsystems into a working system.

SYSTEM INTEGRITY. (6) The degree to which a system is dependable.

SYSTEM RELIABILITY. (5) The probability of performing a given function from the some initial time, t=0, to time t.

TESTABILITY. (6) A measure of how well the protocol supports completeness of testing and the protocol's ability to produce repeatable or predictable results.

THRESHOLD, NOISE. (11) The lowest electromagnetic interference signal level that produces onset of susceptibility.

THROUGHPUT. (6) The productivity of a data processing system as expressed in computing work per minute or hour.

THYRISTORS. (16) Solid-state devices that convert alternating current to direct current.

TIME CONSTANT. (4) Time required to double the amplitude of the divergent real root in the pitch axis of the aircraft model.

TOKEN PASSING PROTOCOL. (18) A protocol that limits bus access to the user that has just received the token word.

TRADITIONAL PROGRAMMING. (20) The use of standard programming languages, as opposed to application development languages, financial planning languages, query languages, and report programs.

TRAINING. (20) A change in connection weight values of a Neural Network that results in capturing information that can be recalled later.

TRANSFER FUNCTION. (20) The mathematical relationship between the output of a control system and its input: for a linear system, it is the Laplace transform of the output divided by the Laplace transform of the input under conditions of zero initial-energy storage.

TRANSIENT CONTROL LEVEL. (13) The maximum allowable level of transients appearing at the systems interfaces as a result of the defined external environment.

TRANSPARENCY. (20) A characteristic of knowledge in an Expert System referring to its independence from other knowledge. There is no processing information in the transparent knowledge.

TRANSPARENT RECOVERY. (4) Correcting a soft fault without interrupting the system's intended performance.

TRANSPORT AIRCRAFT. (19) Aircraft used in interstate, overseas, or foreign air transportation.

TRIBOELECTRIC CHARGING. (13) Static electricity produced on a structure from the effects of friction.

UNACCEPTABLE RESPONSE. (11) Upset, degradation of performance, or failure, not designated a malfunction, but is detrimental or compromising to cost, schedule, comfort, or workload.

UNBALANCED CONFIGURATION. (18) A bus using the HDLC protocol that connects one primary and one or more secondary stations.

UNDESIRABLE RESPONSE. (11) Change of performance and output, not designated a malfunction or safety hazard, that is evaluated as acceptable as is because of minimum nuisance effects and excessive cost burdens to correct.

UNIDIRECTIONAL DATA BUS. (18) A data bus with only one user that is capable of transmitting.

UNSUPERVISED LEARNING. (20) A method of training a Neural Network, where no external monitor is involved; the Neural Network organizes itself by grouping and generating its own classification of inputs.

UPSET. (11) Temporary interruption of performance that is self-correcting or reversible by manual or automatic process.

UPSET. (12) A condition in which the state of a digital device is unintentionally altered, but may be restored by automatic means or by operator intervention.

UPSET. (13) See system functional upset.

VALIDATION. (4,11) Demonstration and authentication that a final product operates in all modes and performs consistently and successfully under all actual operational and environmental conditions founded upon conformance to the applicable specifications.

VALIDATION. (18) The process of evaluating whether or not items, processes, services, or documents accomplish their intended purpose in their operating environment.

VERIFICATION. (4,11) Demonstration by similarity, previous in-service experience, analysis, measurement, or operation that the performance, characteristics, or parameters of equipment and parts demonstrate accuracy, show the quality of being repeatable, and meet or are acceptable under applicable specifications.

VERIFICATION. (18) The act of reviewing, inspecting, testing, checking, auditing, or otherwise establishing and documenting whether or not items, processes, services, or documents conform to specified requirements.

VERY LARGE SCALE INTEGRATION. (20) Describing Integrated Circuits with more than 1000 elements.

VOTING PROCEDURE. (8) An algorithm included in fault tolerant software which uses the consensus recovery block method. It compares outputs of the n independent versions and determines which outputs are correct by identifying agreements among two or more versions.

WEIGHT MATRIX. (20) The collection of connection weights for an entire Neural Network.

WELL-BEHAVED FUNCTION. (17) A smooth mathematical relationship.

ACRONYMS AND ABBREVIATIONS

λ (17)	Language Level
μC	micro Controller
μm (6,20)	Micrometer
μs (18)	Microsecond
η (17)	Vocabulary of a Program
η_1 (17)	Number of Unique Operators
η_1^* (17)	Minimum Number of Unique Operators
η_2 (17)	Number of Unique Operands
η_2^* (17)	Number of Different Input and Output Parameters
ϕM (6)	Phase Modulation
$1/E_0$ (17)	Average number of discriminations a person is likely to make for each bug introduced into the code.
3-D (16)	Three-Dimensional
A/C (11)	Aircraft
A/L (3)	Approach/Land
A^3I (19)	Army-NASA Aircrew/Aircraft Integration
AAES (15)	Advanced Aircraft Electrical System
ABET (19)	Aerospace Behavioral Engineering Technology
ac (3,6,12,15,19)	Alternating Current
AC (3,5,14,17,18,19,20)	Advisory Circular
ACAP (13)	Advanced Composite Airframe Program
ACARS (11,12)	ARC Communications Addressing and Reporting System
ACES (13)	Applied Computational Electromagnetics Society
ACK (18)	Acknowledge
ACO (18,20)	Aircraft Certification Office
ACS (16)	Automatic Control System
ACT (11,12)	Active Controls Technology
ACT (17)	Analysis of Complexity Tool
AD (19)	Alerting Display
ADC (11,12)	Air Data Computer
ADF (11,12,19)	Automatic Direction Finder
ADI (3)	Automatic Direction Indicator
AE (6)	Avionics Equipment
AE4L (5,13)	SAE Subcommittee (Lightning)
AEEC (18)	Airlines Electronic Engineering Committee
AEHP (13)	Atmospheric Electricity Hazards Protection
AERA (16)	Automated En Route Air Traffic Control System
AES-S (6)	Aerospace and Electronic Systems Society
AF (11,12)	Audio Frequency
AFBW (4)	Augmented Fly-By-Wire
AFCS (11,12)	Automatic Flight Control System
AFFDL (7,8,13)	Air Force Flight Dynamics Laboratory

AFM (16)	Advanced Fuel Management
AFSC (18)	Air Force Systems Command
AFWAL (6,13)	Air Force Wright Aeronautical Laboratory
AGARD (8,20)	Advisory Group for Aerospace Research and Development
AHRS (6)	Attitude Heading Reference System
AI (16,19,20)	Artificial Intelligence
AIAA (5,6,9,15,18,20)	American Institute of Aeronautics and Astronautics
AIM (18)	Advanced Integrated MUX
AIM (20)	Airmen's Information Manual
AIR (18,19)	Aerospace Information Report
AIRLAB (5,18)	Avionics Integration Research Laboratory
AK (7)	Altitude Kinematics
ALCM (13)	Air Launched Cruise Missile
ALPA (19)	Air Line Pilot's Association
ALU (3,5,10)	Arithmetic Logic Unit
AM (19)	Active-Matrix
AM (6,14)	Amplitude Modulation
AM (5)	Amplitude Modulated
AMS (19)	Aerospace Material Specification
AMSC (5)	Document number prefix used by the Department of Defense
ANSI (11,12,20)	American National Standards Institute
AOA (4)	Angle of Attack
AP (18)	Application Processor
APU (11,12,15)	Auxiliary Power Unit
AR (7)	Analytical Redundancy
ARC (11,12)	Aeronautical Radio, Incorporated
ARIES (3)	Automated Reliability Interactive Estimation System
ARINC (3,6,18,20)	Aeronautical Radio, Incorporated
ARP (17,18,19,20)	Aerospace Recommended Practice
ARPA (20)	Advanced Research Projects Agency (formerly DARPA)
ARTERI (15)	Analytical Redundancy Technology for Engine Reliability Improvement
AS (19)	Aerospace Standard
ASCB (6,18)	Avionics Standard Communications Bus
ASDS (11,12)	Airport Surface Detection System
ASEE (5,15)	American Society of Electrical Engineers
ASIC (20)	Application Specific Integrated Circuit
ASME (5)	American Society of Mechanical Engineers
ASRS (19)	Aviation Safety Reporting System
ATC (19,20)	Air Traffic Control
ATCRBS (11,12,14)	Air Traffic Control Radar Beacon System
ATE (20)	Automatic Test Equipment
ATF (16)	Advanced Tactical Fighter
ATI (16)	Access Time Interval
ATTR (5)	Attribute
AWACS (14)	Airborne Warning and Control System

$\hat{B}$ (17)	Number of Bugs (Estimated)
B-dot (13)	Derivative of the magnetic field with respect to time
B-GLOSS (5)	Gate Logic Software Simulator developed by Bendix
BABBAGE (20)	Boeing Advanced BlackBoard Ada Generation Environment
BAC (18)	Balanced Asynchronous Configuration
BAT (17)	Battlemap Analysis Tool
BB (11,12)	Broadband
BC (18)	Bus Controller
BCAC (18)	Boeing Commercial Airplane Company
BCD (18)	Binary Coded Decimal
BCI (12)	Bulk Cable Injection
BFCS (18)	Beacon Frame Check Sequence
BGU (10)	Bus Guardian Unit
BIR (6)	Benchmark Information Rate
BIT (6,15,18,20)	Built-In Test
BITE (6,11,12,19)	Built-In Test Equipment
BIU (6,18)	Bus Interface Unit
BNR (18)	Binary
BOCP (18)	Bit-Oriented Communications Protocol
BP (18)	Basic Protocol
bps (6)	bits per second
BUSY (18)	Destination Busy
BW (11,12)	Bandwidth
C/I (5)	Communicator Interstage
C (7)	Comparator
CA (18)	Criticality Analysis
CAA (14,19)	Civil Aviation Authority
CAD (5,19)	Computer Aided Design
CALSEL (19)	Call Select
CAP (5)	Collins Application Processor
CAPS (3)	Computer Aided Production Simulator
CARE (3,5)	Computer Aided Reliability Evaluator
CARSRA (3,7)	Computer-Aided Redundant System Reliability Analysis
CAS (11,12)	Criticality Advisory System
CASSY (20)	Cockpit Assistant System
CAST (3)	Complementary Analytic Simulative Technique
CBD (5)	Commerce Business Daily
CBI (19)	Computer Based Instruction
CCITT (6)	Consultative Committee for International Telephone and Telegraph
CD (6)	Collision Detection
cdf (8)	Cumulative Density Function
CDU (11,12,19,20)	Control Display Unit
CE (11,12)	Conducted Emission
CE (17,18,19,20)	Certification Engineer
CF (20)	Certainty Factor
CFR (17,19)	Code of Federal Regulations

cm (19)	Centimeter
CM (20)	Configuration Management
CM (11,12)	Common Mode
CMC (18)	Current Mode Coupler
CMOS (5,12,20)	Complimentary Metal-Oxide Semiconductor
CMU (19)	Computer Mock-Up
CNI (20)	Communication, Navigation, and Identification
COMBIMAN (19)	Computerized Biomechanical Man-model
CONUS (14)	Contiguous United States
CP (18)	Combined Protocol
CPA (5)	Central Processor - A
CPU (5,10,18,19,20)	Central Processing Unit
CR (5)	Contractor Report
CR (6)	Command Response
CR/LF (17)	Carriage Return/Line Feed
CRC (6,18)	Cyclic Redundancy Check
CRM (19)	Cockpit Resource Management (also Crew Resource Management)
CRMI (2)	Computer Resource Management, Incorporated
CRT (11,12,16,19)	Cathode Ray Tube
CS (11,12)	Conducted Susceptibility
CSC (9)	Computer Software Component
CSCI (9,17)	Computer Software Configuration Item
CSDB (18)	Commercial Standard Data Bus
CSDL (5,10,15)	Charles Stark Draper Laboratories
CSERIAC (19)	Crew System Ergonomics Information Analysis Center
CSMA (6,16,18)	Carrier Sensed Multiple Access
CSMA/CD (6)	Carrier Sense Multiple Access/Collision Detection
CT (2,6)	Technical Center (designation used in FAA report numbering scheme)
CTA (3)	CAPS Test Adapter
CTA (5)	Collins Test Adaptor
CTS (18)	Clear To Send
CW (13)	Continuous Wave
D (17)	Program Difficulty
DADC (6)	Digital Air Data Computer
DARPA (16)	Defense Advanced Research Projects Agency (Now known as Advanced Research Projects Agency)
DATAC (6,18)	Digital Autonomous Terminal Access Communication
dB (6,12)	Decibel
dBi (14)	Decibels with respect to one milliampere
dBm (6)	Decibels with respect to one milliwatt
dc (6,12,15,19)	Direct Current
DC (18)	Display Computer
DE (5)	Diagnostic Emulation
DEFN (5)	Definition
DEP (19)	Design Eye Position
DET (18)	Driver Enable Timer
DEV (5)	Development

DF (7)	Diagnostic Filter
DFC (7)	Digital Flight Control
DFCS (3,4,7,16)	Digital Flight Control System
DFDAU (11,12)	Digital Flight Data Acquisition Unit
DFDR (11,12)	Digital Flight Data Recorder
DGAC (14)	Directorate Generale Aviation Civile
DISAC (15)	Digital Integrated Servo Actuator Controller
DITS (6,11,12,18)	Digital Information Transfer System
DM (11)	Differential Mode
DM (6)	Delay Modulation
DMA (5,6)	Direct Memory Access
DMA (18)	Direct Memory Addressing
DME (11,12,18,19)	Distance Measuring Equipment
DNA (13)	Defense Nuclear Agency
DoD (5,8,12,14,16,19,20)	Department of Defense
DOE (13)	Department of Energy
DOS (20)	Disk Operating System
DOT (2,3,6,7,8,20)	Department of Transportation
DR (17)	Direct Ratio (Average)
DRB (9)	Distributed Recovery Block
DS (17)	Direct Score
DSP (3)	Discrete Switch Panel
DSP (20)	Digital Signal Processor
DTSA (18)	Dynamic Time Slot Allocation
E/E (11,12)	Electrical/Electronic
E-FIELD (11,12)	Electric Field
E (17)	Programming Effort
E-dot (13)	Derivative of the electric field with respect to time
E^3 (11,12)	Electromagnetic Environmental Effects
EADI (11,12,19)	Electronic Attitude Director Indicator
ECAC (11,12,14)	Electromagnetic Compatibility Analysis Center
ECM (14)	Electronic Counter Measures
ECS (11,12)	Environmental Control System
EEC (5,11,12,18)	Electronic Engine Control
EED (11,12)	Electro-Explosive Device
EEPROM (20)	Electrically Erasable Programmable Read Only Memory
EES (18)	Electromagnetic Emission and Susceptibility
EFID (18)	Electronic Flight Instrument Display
EFIS (11,12,18,19)	Electronic Flight Instrument System
EFMA (3)	Executive Failure My A
EFMB (3)	Executive Failure My B
EFOA (3)	Executive Failure Other A
EFOB (3)	Executive Failure Other B
EFW (3)	Executive Failure Word
E_g (19)	Energy gap
EGT (11,12)	Exhaust Gas Temperature
EHSI (11,12)	Electronic Horizontal Situation Indicator
EIA (18)	Electronic Industries Association
EICAS (11,12)	Engine Indication and Crew Alerting System

EIU (16)	Electronic Interface Unit
EL (19)	Electroluminescent
ELS (19)	Electronic Library System
EM (5,11,12,13)	Electromagnetic
EMA (15)	Electromechanical Actuator
EMAS (2,15)	Electromechanical Actuator System
EMC (6,11,12,13,14)	Electromagnetic Compatibility
EMCad^{tm} (12)	Electromagnetic Computer aided design
EME (14)	Electromagnetic Environment
EME (11,12)	Electromagnetic Effects
EMI (5,6,11,12,13,15,16)	Electromagnetic Interference
EMIC (11,12)	Electromagnetic Interference/Compatibility
EMP (11,12,13)	Electromagnetic Pulse
EMR (5,14)	Electromagnetic Radiation
EMUX (6)	Electrical Multiplex
ENRZ (6)	Enhanced Non-Return to Zero
EOF (17)	End of File
EPES (20)	Emergency Procedures Expert System
EPR (11,12,15)	Engine Pressure Ratio
EPROM (16)	Erasable Programmable Read-Only Memory
ES (20)	Expert System
ESD (11,12)	Electrostatic Discharge
ESE (11,12)	Electric (field) Shield Effectiveness
ESS (5,9)	Electronic Switching System
ETANN (20)	Electrically Trainable Analog Neural Network
ETDL (13)	Equipment Transient Design Level
EUROCAE (14)	European Organization for Civil Aviation Electronics
EVS (19)	Enhanced Vision System
EXCHNG (5)	Exchange
EXP (5)	Experiment
F/FA (18,20)	Fault and Failure Analysis
FAA (ALL)	Federal Aviation Administration
FADEC (6,15,16)	Full Authority Digital Engine Controller
FAFTEEC (16)	Full Authority Fault Tolerant Electronic Engine Control
FAR (17,18,19,20)	Federal Aviation Regulation
FAR (3,4,6,16)	Federal Acquisition Regulation
FAS (19)	Flight Deck Alerting System
FBL (15,16)	Fly-By-Light
FBW (4,7,16)	Fly-By-Wire
FCA (20)	Fuzzy Computational Accelerator
FCC (3,4,5,6,10,11, 12,15,18,19)	Flight Control Computer
FCR (5)	Fault Containment Regions
FCS (18)	Frame Check Sequence
FCS (4,5,10,16)	Flight Control System
FD (7)	Fault Detection
FDEP (11,12)	Flight Data Entry Panel
FDFM (15)	Fault Detection and Failure Management
FDI (19)	Flight Director Indicator

FET (5,16,19)	Field Effect Transistor
FI (7)	Fault Isolation
FI (5)	Fault Insertion Circuitry
FIAT (5)	Fault Injection Automated Testing
FICA (15)	Failure Indication and Corrective Action
FIDE (20)	Fuzzy Inference Development Environment
FIIS (5,10)	Fault Insertion and Instrumentation System
FIL (20)	Fuzzy Inference Language
FIM (5)	Fault Injection Manager
FINDER (20)	Flight-Plan Interactive Negotiation and Decision-Aiding System for Enroute Rerouting
FIRE (5)	Fault Injection Receptor
FLEX (20)	FORTRAN Library for EXpert systems
FM (5)	Frequency Modulated
FM (6,14)	Frequency Modulation
FMC (11,12,19)	Flight Management Computer
FMEA (3,9,18)	Failure Mode and Effect Analysis
FMECA (5,15)	Failure Modes and Effects Criticality Analysis
FMECA (18)	Failure Mode, Effects, and Criticality Analysis
FMS (19)	Flight Management System
FOV (19)	Field of View
FP (17)	Function Point
FSM (18)	Finite State Machine
ft (14)	feet
FT (5)	Fault Tolerant
FTA (18)	Fault Tree Analysis
FTC (5)	Fault Tree Compiler
FTMP (5,10,18)	Fault-Tolerant Multiprocessor
FTP (5)	Fault Tolerant Processor
G/E (13)	Graphite Epoxy
g1 (19)	Grid 1
g2 (19)	Grid 2
g3 (19)	Grid 3
g4 (19)	Grid 4
GA (18)	General Aviation
GaAs (6,16,19,20)	Gallium Arsenide
GaAsP (19)	Gallium Arsenide Phosphide
GAMA (6,18)	General Aviation Manufacturers Association
GaP (19)	Gallium Phosphide
GCR (6)	Group Code Recording
GE (5)	General Electric
GEMACS (13)	General Electromagnetic Model for the Analysis of Complex Systems
GEN (5)	Generation
GGLOSS (5)	Generalized Gate-Level Logic System Simulator
GLOSS (5)	Gate Logic Software Simulator
GNC (16)	Guidance, Navigation, and Control
GPC (5)	General Purpose Computer
GPS (11,12)	Global-Positioning-System
GPWS (11,12)	Ground Proximity Warning System
Gr/Ep (11,12)	Graphite/Epoxy

GS (10)	Glideslope
GUI (20)	Graphical User Interface
H-FIELD (11,12)	Magnetic Field
H1 (11,12)	Fan Speed
HA (18)	Hazard Analysis
HARP (5,18)	Hybrid Automated Reliability Predictor
HDBK (5)	Handbook
HDD (19)	Head-Down Display
HDLC (6,18)	High-Level Data Link Control
HERF (5,14,16,18)	High-Energy Radio Frequency
HF (11,12,13,14,19)	High-Frequency
Hg (19)	Mercury
HIRF (15,18,19)	High-Intensity Radiated Fields
HLD (19)	Head-Level Display
HMD (19)	Helmet Mounted Displays
HMS (19)	Helmet Mounted Sight
HOL (17,20)	High Order Language
HSI (3,19)	Horizontal Situation Indicator
HSRB (6,18)	High-Speed Ring Bus
HUD (19)	Head-Up Display
HVDC (15)	High Voltage dc
HW (18)	Hardware
Hz (3,15,18,19)	Hertz
I (13)	Current
I (17)	Intelligence Content
I-dot (13)	Derivative of the current with respect to time
I/O (5,10,15,18,20)	Input/Output
IAA (3)	Integrated Assurance Assessment
IAAC (11,12)	Integrated Application of Active Controls Technology (to an Advanced Subsonic Transport Project)
IACS (18)	Integrated Avionic Computer System
IBM (5)	International Business Machines
IC (3,18,19,20)	Integrated Circuit
ICAO (14)	International Civil Aviation Organization
ICIS (5)	Intercomputer Interface Sequencer
ICS (5)	Intercomputer Sequencer
ID (5)	Identification
ID (18)	Identifier
IDG (11,12)	Integrated Drive Generator
IEEE (5,6,9,17,18,20)	Institute of Electrical and Electronics Engineers, Incorporated
I_F (19)	Forward Current
IETM (20)	Interactive Electronic Technical Manual
IFC (17)	Information Flow Complexity
IFCS (18)	Information Frame Check Sequence
IFF (14,19)	Identification - Friend or Foe
IGGLOSS (5)	Gate Logic Software Simulator (improved version developed at NASA Langley)

IJCNN (20)	International Joint Conference on Neural Networks
ILS (3,11,12,19)	Instrument Landing System
IMR (18)	Interrupt Mask Register
INS (6,11,12,19)	Inertial Navigation System
IOPA (5)	Input/Output Processor - Channel A
IOS (5)	Input/Output Subsystem
IR (19)	Infrared
IRS (11,12,19)	Inertial Reference System
ISO (17,18)	International Standards Organization
ITO (19)	Indium Tin Oxide
ITS (19)	Intelligent Tutoring System
ITT (16)	(Consultative Committee for) International Telegraphy and Telephony
IV&V (9,20)	Independent Verification and Validation
IVT (18)	Interrupt Vector Table
JPL (5)	Jet Propulsion Laboratories
k (6,20)	Thousand
kA (13,16)	Kiloampere
KBAP (20)	Knowledge-Based AutoPilot
kHz (5,6,12,14,19)	Kilohertz
km (6)	kilometer
$\hat{L}$ (17)	Estimated Program Level
L (17)	Program Level
LAN (5)	Local Area Network
LaRC (5)	Langley Research Center
LC (19)	Liquid Crystal
LCC (11,12)	Life Cycle Cost
LCD (16)	Liquid Crystal Display
LCS (19)	Liquid Crystal Shutter
LED (6,19,20)	Light Emitting Diode
LF (13)	Low Frequency
LISP (20)	LISt Processor
LOC (11,12)	Localizer
LOFT (19)	Line Oriented Flight Training
LPN (13)	Lumped Parameter Network
LRC (6)	Longitudinal Redundancy Check
LRRA (11,12,19)	Low Range Radio Altimeter
LRU (5,6,11,12,13,18,19)	Line Replaceable Unit
LSB (6,18)	Least Significant Bit
LSI (5,18,19)	Large Scale Integration
LTPB (6,18)	Linear Token Passing Bus
LTRI (13)	Lightning and Transients Research Institute
LVDT (15)	Linear Variable Differential Transformers
LVDT (3)	Linear Voltage Differential Transducer
m (18)	Original Address of Last Transmission
m (6)	meter
M (6)	Million

M (5)	Mutual (when used with RLC)
mA (3)	Milliampere
MAADS (6)	Multibus Avionic Architecture Design Study
MAC (4)	Mean Aerodynamic Chord
MAFT (16)	Multicomputer Architecture for Fault Tolerance
Mbps (6,16)	Million bytes per second
MC (18)	Mode Code
MCDP (11,12)	Maintenance Control and Display Panel
MCDU (19)	Multi-Purpose Control Display Unit
MCFCS (18)	Message Control Frame Check Sequence
MCP (11,12)	Mode Control Panel
MDICU (4,5)	Modular Digital Interface Conversion Unit
MDICU (3)	Modular Digital Interface Control Unit
MDT (6)	Mean Down Time
MEP (20)	Mission Equipment Package
MFCS (18)	Message Frame Check Sequence
Mflops (16)	Million floating-point operations per second
MFM (6)	Modified-Frequency Modulation
MFR (10)	Mean Failure Rate
MHz (5,6,12,14,18,20)	Megahertz
mil (11,12)	One thousandths of an inch (0.001)
MIL (5)	Military
MIL-HDBK (18)	Military Handbook
MIL-STD (6,18)	Military Standard
MKR (19)	Markers
ML (18)	Message Length
MLE (8)	Maximum Likelihood Estimates
MLS (11,12,19)	Microwave Landing System
mm (19,20)	Millimeter
Mn (19)	Manganese
MOS (5)	Metal-Oxide Semiconductor
MPP (16)	Massively Parallel Processor
MPSC (18)	Multi-Protocol Serial Controller
MPX (5)	Multiplex
ms (6,10,18)	Millisecond
MSB (6,18)	Most Significant Bit
MSE (11,12)	Magnetic (Field) Shielding Effectiveness
MSI (5)	Medium Scale Integration
MT (18)	Message Time
MTBCF (6)	Mean Time Between Critical Failures
MTBF (9,18)	Mean Time Between Failures
MTTF (8,10)	Mean Time to Failure
MTTR (6,18)	Mean Time To Repair
Mux (5)	Multiplexed
MUX (4,18)	Multiplexer
$\hat{N}$ (17)	Estimated Length
n (18)	Address of User Performing Computation
N (18)	Maximum Number of Users
N (17)	Implementation Length of a Program
N_1 (15)	Low Rotor Speed
N_1 (17)	Total Number of Operator Occurrences

N_1^* (17)	Minimum Number of Operators
N_2 (15)	High Rotor Speed
N2 (11,12)	Core Engine Speed
N_2 (17)	Total Number of Operand Occurrences
N_2^* (17)	Minimum Number of Operands
NA (6)	Numerical Acceptance
NA (3)	Normal Accelerometers
NADC (6,7)	Naval Air Development Center
Naecon (5)	National Avionics and Electronics Conference
NAECON (6)	National Aerospace and Electronics Conference
NAND (5)	Not AND
NAS (19)	National Airspace System
NASA (2,3,5,7,13,15,18,19,20)	National Aeronautics and Space Administration
NASC (13)	Naval Air Systems Command
NATO (14)	North Atlantic Treaty Organization
NB (11,12)	Narrow Band Signal
NCTS (18)	Not Clear To Send
ND (19)	Navigation Display
NEC (13)	Numerical Electromagnetics Code
NEMP (5,12)	Nuclear Electromagnetic Pulse
NHPP (8)	Non-Homogeneous Poisson Process
nmi (14)	nautical mile
NN (20)	Neural Network
NPRM (14)	Notice of Proposed Rulemaking
NRZ (6)	Non-return to Zero
NRZ-I (6)	Non-return to Zero Inverted
NRZ-L (6)	Non-return to Zero Dual Level
nsec (6)	Nanosecond
NSWC (13)	Naval Surface Weapons Center
NTSB (19)	National Transportation Safety Board
NVG (19)	Night Vision Goggles
NVS (8)	N-version Software
OMEGA (11,12)	Very Low Frequency Navigation
OMV (16)	Orbital Maneuvering Vehicle
ONR (20)	Office of Naval Research
OS (5)	Operating System
OSI (18)	Open Systems Interconnection
OTV (16)	Orbital Transfer Vehicle
P-Static (11,12)	Precipitation Static
P (10)	Processor
PA (19)	Public Address
PA (20)	Pilot's Associate
PAI (19)	Primary Attitude Instrument
PAL (5)	Programmable Array Logic
PAS (6)	Pilot Assist System
PAWS (5)	Padé Approximation With Scaling
PBW (15)	Power-By-Wire
PC (17)	Processing Complexity
PC (5,20)	Personal Computer
PCA (17)	Processing Complexity Adjustment

PCS (16)	Primary Control System
PCU (11,12)	Power Control Unit
pdf (8)	Probability Density Function
PE (6)	Phase Encoding
PE (20)	Processing Element
pf (12)	picofarad
PFD (19)	Primary Flight Display
PGA (20)	pin grid array
PLA (15)	Power Level Actuator
PLA (16)	Power Level Angle
PMA (18)	Parts Manufacturer Approval
PMA (20)	Portable Maintenance Aid
PMS (5)	Physical Message Switch
PNF (19)	Pilot Not Flying
PNI (19)	Primary Navigation Instrument
PRF (11,12)	Pulse Repetition Frequency
PROC (5)	Processor
PROLOG (20)	PROgramming in LOGic
PROM (3,10,17,18)	Programmable Read-Only Memory
psi (15)	pounds per square inch
PVI (16,19)	Pilot-Vehicle Interface
PWM (11,12)	Pulse Width Modulation
PZ (15)	Piezoelectric
QUAD (5)	Quadruple
R (13)	Resistance
R-C (12)	Resistor-Capacitor
RADC (8,17)	Rome Air Development Center
RAE (13)	Royal Aircraft Establishment
RAE (20)	Royal Aerospace Establishment
RAM (3,5,10,16,17,18)	Random Access Memory
RAT (6,18)	Ring Admittance Timer
RB (8)	Recovery Block
RBDCP (3)	Reliability Block Diagram Computer Program
RCA (16)	Radio Corporation of America
RCP (19)	Reconfigurable Cockpit
RCVR (5)	Receiver
RDFCS (3)	Redundant Digital Flight Control System
RDFCS (4)	Reconfigurable Digital Flight Control System (facility)
RDFCS (5)	Reconfigurable Digital Flight Control System
RDMI (11,12)	Radio Distance Magnetic Indicator
RE (11,12)	Radiated Emission
REG (5)	Register
REL (3)	Reliability
REL COMP (3)	Reliability Computers
RF (5,11,12,13,14,16,18)	Radio Frequency
RFI (11,12)	Radio Frequency Interference
RIM (18)	Ring Interface Module
RISC (20)	Reduced Instruction Set Computer
RIU (18)	Ring Interface Unit

RL (13,15)	Resistance/Inductance (Out of order in c.15)
R_L (19)	Load Resistor
RLC (5,13)	Resistance/Inductance/Capacitance
RLCM (13)	Resistance/Inductance/Capacitance/Mutual
RM (7)	Redundancy Management
RNRZ (6)	Randomized Non-return to Zero
ROM (5,10,19)	Read Only Memory
RPA (20)	Rotorcraft Pilot's Associate
RPV (16)	Remotely Piloted Vehicle
RR (18)	Read Register
RRT (18)	Ring Rotation Time
RS (18)	Recommended Standard
RS (11,12)	Radiated Susceptibility
RSS (4,7)	Relaxed Static Stability
RT (5,18)	Remote Terminal
RTCA (2,3,11,12,14, 16,17,18,20)	Radio Technical Commission for Aeronautics (now known as Requirements and Technical Concepts for Aviation)
RTE (18)	Real-Time Executive
RTI (5)	Remote Terminal Interface
RTS (18)	Request To Send
RZ (6)	Return to Zero
S (17)	Stroud Number
S/A (11,12)	Spectrum Analyzer
S-a-0 (5,10)	Stuck at Zero
S-a-1 (5,10)	Stuck at One
S-GLOSS (5)	Gate Logic Software Simulator developed by Stevens
SAE (2,5,6,13,14,16, 17,18,19,20)	Society of Automotive Engineers (now known as Engineering Society for Advanced Mobility Land Sea Air and Space)
SAI (18)	Systems Architecture and Interfaces
SAS (4,7)	Stability Augmentation System
SATCOM (19)	Satellite Communications
SATNAV (19)	Satellite Navigation
SC (18)	Special Condition
SC (20)	Special Committee
SCC (18)	System Configuration Controller
SCM (18)	Software Configuration Management
SCP (18)	Self-Checking Pair
SD (19)	System Display
SDF (7)	Super-Diagnostic Filter
SE (11,12)	Shielding Effectiveness
SELCAL (19)	Select Call
SEU (5)	Single Event Upset
SG (18)	Synchronization Gap
SHF (11,12)	Super High-Frequency
SHRD (5)	Shared
SIAM (15)	Society for Industrial and Applied Mathematics

SIF (14)	Selective Identification Facility
SIM (18)	Serial Interface Module
SIR (18)	Shared Interface RAM
SLRT (7)	Sequential Likelihood Ratio Test
SMF (18)	Self Monitor Function
SMOTEC (14)	Special Missions Operation Test and Evaluation Center
SO (17)	Second Order (Average)
SOP (19)	Standard Operating Procedure
SPRT (7)	Sequential Probability Ratio Test
SQA (18)	Software Quality Assurance
SQF (17)	Software Quality Factor
SQL (5)	Software Query Language
SQM (17)	Software Quality Metrics
SQPP (17)	Software Quality Program Plan
SRS (17)	Software Requirement Specification
SSA (18)	System Safety Assessment
SSI (5)	Small Scale Integration
SSP (3)	Servo Simulation Panel
SSPC (15)	Solid-State Power Controller
SSS (17)	System/Segment Specification
STANAG (14)	Standardization Agreement (NATO)
STC (2)	Supplemental Type Certification
STC (18,20)	Supplemental Type Certificate
STEM (5)	Scaled Taylor Expansion Matrix
STN (19)	Supertwisted Nematic
STOL (16)	Short Takeoff and Landing
str (6)	string
SURE (5)	Semi-Markov Unreliability Range Evaluator
SW (18)	Software
SYN (5)	Synch
$\hat{T}$ (17)	Estimated Programming Time
T/R (6)	Transmitter/Receiver
TACAN (14,19)	Tactical Air Navigation
TASRA (3)	Tree Aided System Reliability Analysis
T_C (18)	Count Duration
TC (2)	Type Certification
TC (18,20)	Type Certificate
TCAP (13)	Threshold Circuit Analysis Program
TCAS (11,12,16,18)	Traffic Alert and Collision Avoidance System
TCL (13)	Transient Control Level
TDM (6)	Time Division Multiplex
TDMA (18)	Time Division Multiple Access
T_F (18)	Frame Time
TEAMS (20)	Technical Expert Aircraft Maintenance System
TFCS (18)	Token Frame Check Sequence
TFEDF (18)	Token Frame Ending Delimiter Field
TFEL (19)	Thin-Film Electroluminescence
TFT (19)	Thin-Film Transistor
TG (18)	Terminal Gap
THT (6,18)	Token-Holding Timer

TI (18)	Transmit Interval
TLA (11,12)	Thrust Lever Angle
T_m (18)	Wait Time for User
TMC (11,12)	Thrust Management Computer
TMR (10)	Triple Modular Redundant
TN (19)	Twisted Nematic
TRT (18)	Token Rotation Timer
TRU (15)	Transformer Rectifier Unit
TSDF (18)	Token Starting Delimiter Field
TSO (18,19,20)	Technical Standard Order
TTL (5,11,12,13,16,19,20)	Transistor-Transistor Logic
TV (14)	Television
TWTD (13)	Thin Wire Time Domain
TX (5)	Transmit
U.K. (13,14)	United Kingdom
U.S. (14)	United States
UAC (18)	Unbalanced Asynchronous Configuration
UART (15)	Universal Asynchronous Receiver Transmitter
UHF (11,12,13,14,19)	Ultra High-Frequency
UNC (18)	Unbalanced Normal Configuration
UNIBUS (5)	Universal Bus
UPS (15)	Uninterruptible Power Supplies
USAF (16,20)	United States Air Force
USB (16)	Upper Surface Blowing
USEG (5)	Unsegmented
UUT (20)	Unit Under Test
UV (19)	Ultra-Violet
V (17)	Volume
V (19)	Volt
V/m (14)	Volt/meter
V^* (17)	Potential Volume
V&V (18,20)	Verification and Validation
V_B (19)	Breakdown voltage
V_{CC} (19)	Collector Circuit Voltage
V_F (19)	Forward Voltage
VFD (19)	Vacuum Florescent Display
VHF (11,12,13,14,19)	Very High-Frequency
VHSIC (6,16)	Very-High-Speed Integrated Circuits
VLF (11,12,19)	Very Low-Frequency
VLSI (5,6,14,18,20)	Very Large Scale Integration
VLSIC (6,16)	Very Large Scale Integrated Circuits
VOR (11,12,14,18,19)	VHF Omnidirectional Range
VORTA/VHF (11,12)	Omnirange/Tactical Air Navigation
VRC (6)	Vertical Redundancy Check
V_S (19)	Sustain-Voltage
VSI (11,12)	Vertical Speed Indicator
VSV (15)	Variable Stator Vane
VT (18)	Validation Testing
VTOL (16)	Vertical Takeoff and Landing

W/P (15)	Fuel Flow to Burner Pressure
WAI (3)	Warning Annunciation Indicator
WFM (15)	Main fuel metering valve actuator sensor
WR (18)	Write Register
WRU (11,12)	Weapons Replaceable Unit
WSO (17)	Weighted Second Order (Average)
XAB (5)	Transmit Compare A B
XMT (5)	Transmit
XMTR (3)	Transmitter
XOR (5)	Exclusive OR
ZM (6)	Zero Modulation
ZnS (19)	Zinc Sulfide
ZnSe (19)	Zinc Selenide

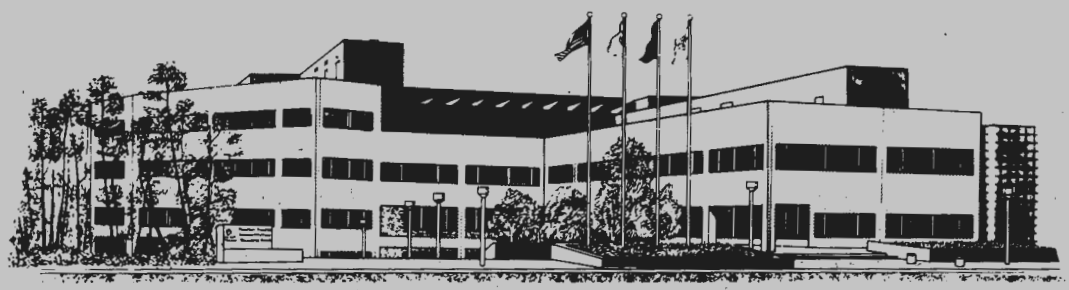
*U.S. GOVERNMENT PRINTING OFFICE: 1994-504-078-00146

U.S. Department of Transportation
Federal Aviation Administration

DOT/FAA/CT-88/10

HANDBOOK-VOLUME II
DIGITAL SYSTEMS VALIDATION

CHAPTER 1
SUMMARY



PREPARED BY:

COMPUTER RESOURCE MANAGEMENT, INC.
950 HERNDON PARKWAY, SUITE 360
HERNDON, VIRGINIA 22070

PREPARED FOR:

FEDERAL AVIATION ADMINISTRATION
TECHNICAL CENTER
ATLANTIC CITY INTERNATIONAL AIRPORT, NEW JERSEY 08405

NOTICE

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The United States Government assumes no liability for the contents or use thereof.

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the objective of this report.

EXECUTIVE SUMMARY

The Digital Systems Validation Handbook - Volume II (DOT/FAA/CT-88/10) is a followup and companion to the Handbook - Volume I Validation of Digital Systems in Avionics and Flight Control Applications (DOT/FAA/CT-82/115).

Volume I was issued in July 1983 and revised in September 1986. It deals with system overview topics such as the System Life Cycle, Mission Factors, System Architectures, Crew Workload Evaluation, Current Validation Procedures, Recommended Validation Procedures, and Recommended Configuration Management Procedures.

The emphasis in Volume I is on the validation of digital avionic systems. Validation of a system is the determination that the system is well grounded on principles or evidence. The topics presented in Volume I assist the certification engineer in understanding how to determine if the validation of a system has been correctly performed.

The emphasis in Volume II is on the certification of advanced technology digital avionic systems, as well as their validation. Volume II covers detailed technical topics such as latent faults; data busses; integrated assurance assessment; analytical sensor redundancy; and protection against lightning, electromagnetic interference, and high energy radio frequency fields. These topics are covered in detail to familiarize the certification engineer with the issues involved in implementing the new technologies.

Volume II covers topics that will enable the certification engineer to understand the information presented in type certification and supplemental type certification documentation, to understand variations in the implementation of technologies, and to discuss them with the design engineer.

Volume II also addresses some of the soon-to-be-available technologies in the "Advanced Validation Issues" chapter. The direction of aviation research in the United States is discussed along with challenges and problems that confront the certification engineer in certifying the new technologies.

Since the topics discussed in this Handbook are at the forefront of technological research, some of the concepts presented are subject to discussion by experts in the field. In these areas, the Handbook presents various viewpoints alerting the certification engineer to the various views so that this information will be considered in formulating decisions and developing certification criteria.

It is intended that this Handbook will be a living document. As technology advances, addenda to existing chapters will be issued. Where major new technologies are addressed, new chapters will be added to the Handbook. The Federal Aviation Administration (FAA), therefore, solicits your comments and

suggestions for the improvement of this Handbook. Please address any comments to:

Flight Safety Research Branch, ACD-230
Digital Systems Validation Handbook
FAA Technical Center, Bldg. 201
Atlantic City International Airport, New Jersey 08405

U.S. Department of Transportation
Federal Aviation Administration

DOT/FAA/CT-88/10

HANDBOOK-VOLUME II
DIGITAL SYSTEMS VALIDATION

CHAPTER 2
INTRODUCTION



PREPARED BY:

COMPUTER RESOURCE MANAGEMENT, INC.
950 HERNDON PARKWAY, SUITE 360
HERNDON, VIRGINIA 22070

PREPARED FOR:

FEDERAL AVIATION ADMINISTRATION
TECHNICAL CENTER
ATLANTIC CITY INTERNATIONAL AIRPORT, NEW JERSEY 08405

NOTICE

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The United States Government assumes no liability for the contents or use thereof.

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the objective of this report.

TABLE OF CONTENTS

Section		Page
1.	INTRODUCTION	2-1
	1.1. Background	2-1
	1.2. Purpose	2-2
	1.3. Scope	2-2
2.	HANDBOOK ORGANIZATION	2-3
	2.1. Handbook Format	2-3
	2.2. Chapter Formats	2-3
	2.3. Contents of Technical Chapters	2-3
	2.3.1. Chapter 3 - Integrated Assurance Assessment Activities	2-3
	2.3.2. Chapter 4 - Quadruplex Digital Flight Control System Assurance Assessment	2-3
	2.3.3. Chapter 5 - Advanced Fault Insertion & Simulation Methods	2-3
	2.3.4. Chapter 6 - Data Bus Integrity Issues	2-4
	2.3.5. Chapter 7 - Analytical Sensor Redundancy	2-4
	2.3.6. Chapter 8 - Software Reliability Assessment Methods	2-4
	2.3.7. Chapter 9 - Software Fault Tolerance	2-4
	2.3.8. Chapter 10 - Latent Faults	2-4
	2.3.9. Chapter 11 - Guidelines To Assess EMC Designs	2-5
	2.3.10. Chapter 12 - Fast Rise Time EMC Transient Testing	2-5
	2.3.11. Chapter 13 - Lightning Studies	2-5
	2.3.12. Chapter 14 - High Energy Radio Frequency Fields	2-5
	2.3.13. Chapter 15 - Electromechanical Actuator Systems (EMAS)	2-5
	2.3.14. Chapter 16 - Advanced Validation Issues	2-5
ACRONYMS AND ABBREVIATIONS		2-7

1. INTRODUCTION

1.1. Background

The rapid developments in computers and digital technology over the past 20 years have made digital systems increasingly useful in aircraft. As computers become more sophisticated, they are able to perform an increasing number of tasks which were previously performed by the pilot or by analog systems. As a result, certification engineers are seeing increasing numbers of digital systems proposed for inclusion in new generations of aircraft and for incorporation into existing aircraft designs.

The new systems pose a number of problems for the certification engineer. Error limits must be set for flight-critical, flight-essential, and non-essential systems. In setting error limits, safety is paramount, but other factors must also be taken into account. These factors include cost for incremental error reduction, weight penalties, and system interactions.

Integrating a new digital "black box" into an existing design changes many of the electromagnetic characteristics of the aircraft. Care must be taken to ensure not only that the black box performs its functions within the required error limits but also to ensure that its addition does not degrade any other systems which have already been certificated.

The design of a completely new aircraft poses its own problems in digital technology. Each individual component must meet error limits, weight limits, and size limits. Systems (composed of several components more or less collocated) must meet these requirements, plus the requirements of being able to collect systemic information and pass it on to other systems while not interfering with the operation of any of the other systems.

In the summer of 1975, the Federal Aviation Administration (FAA) and the National Aeronautics and Space Administration (NASA) began to consider and plan a joint program to assess and upgrade the technologies used to evaluate the reliability of digital flight control and avionics systems. In 1976, the FAA and NASA cosponsored a workshop on digital flight controls and avionics.

A number of reports emerged from this program. These reports were collected into the Digital Systems Validation Handbook - Volume I (DOT/FAA/CT-82/115) "Validation of Digital Systems in Avionics and Flight Control Applications." Since the 1976 workshop and the 1982 publication of Volume I, however, a number of technological advances have been made. It is time to review the issues of digital avionics and flight control systems in light of the new, emerging technologies.

There are approximately 20 areas of technological investigation that have been sponsored by the FAA and/or NASA. The investigations, the final technical

reports, and the technical results are the basis for the tutorial materials in this Handbook.

1.2. Purpose

The purpose of this Handbook is to give the certification engineer background information on the emerging technologies which will figure prominently in designs now being submitted for certification and anticipated by the end of the century. Certification engineers will then be more knowledgeable in these new areas of technology so that they may more effectively evaluate the materials submitted in support of a request for a Type Certificate (TC) or Supplemental Type Certificate (STC).

This Handbook will serve as a guide to the certification engineer. Each chapter addresses a specific area of technology. Alternate viewpoints are presented as appropriate. Worked examples guide the engineer through the application of the technology. Bibliographies in each chapter give the engineer sources of additional information about the chapter topic. Each chapter can be considered a stand-alone volume about the subject area in question and can be separated into its own volume for ease of reference.

1.3. Scope

This Handbook provides the certification engineer with information on emerging technologies. Since much of the technology in this Handbook is new, some of it is still at issue and is subject to various interpretations. For completeness, alternate viewpoints have been included wherever appropriate. This will provide the certification engineer with the widest variety of information available from which to make certification decisions.

The user should also note that although the Handbook contains the latest information available at the time each tutorial was written, some of this information may be out of date. The certification engineer should therefore use this Handbook as a guideline rather than a hardline.

2. HANDBOOK ORGANIZATION

2.1. Handbook Format

This Handbook is a dynamic document. As such, it is being issued in a loose-leaf format so that it can be expanded easily as knowledge of newer technology applications becomes available. Each chapter has the same type of tutorial/ worked example format and will be updated as a function of technological progress in the areas of civil fixed or rotary wing aircraft, electronics, and system integration. As new information is received, it will be issued either in the form of chapters to be added or revision pages for existing chapters.

The Glossary and the Acronym list for the entire book contain information from all chapters.

2.2. Chapter Formats

Each chapter contains its own Table of Contents, List of Acronyms and Abbreviations, Glossary, and References. The individual acronym lists are particularly important when dealing with such diverse topics since different subject areas may use common acronyms with different interpretations.

2.3. Contents of Technical Chapters

2.3.1. Chapter 3 - Integrated Assurance Assessment Activities

This chapter covers Integrated Assurance Assessment methods including probability analysis, fault-tree analysis, and failure modes and effects analysis. It shows how these methods may be meshed to establish the air-worthiness of a flight-critical system. It also demonstrates how these methods reinforce each other in this determination.

2.3.2. Chapter 4 - Quadruplex Digital Flight Control System Assurance Assessment

Quadruplex digital flight control systems are an evolution from the dual-dual architecture, using four parallel systems rather than two redundant systems. This chapter covers the history of the technology and identifies the functional relationships between the hardware and the software.

2.3.3. Chapter 5 - Advanced Fault Insertion & Simulation Methods

One method of testing flight-critical systems is to insert faults into the system to see if they are found by the error detection and correction mechanisms embedded in the system. There are three categories of faults: (1) faults which are inserted and found, (2) faults which are inserted and not found, and (3) no-fault situations which are identified as faults. This chapter covers the

hardware systems used in the insertion of faults and the interpretation of the results.

2.3.4. Chapter 6 - Data Bus Integrity Issues

Data buses are currently used in flight control and avionics applications to transfer data. Errors in data transmission may result in erroneous or incomprehensible data being transmitted to processors or to flight control mechanisms. This chapter covers the integrity of data buses. It contains an overview of alternative data bus architectures, including, an evaluation of their reliability. It also addresses areas of uncertainty in reliability and minimum currently accepted redundancy standards.

2.3.5. Chapter 7 - Analytical Sensor Redundancy

Analytical sensor redundancy is the use of software algorithms to substitute for hardware in providing redundant signals. If an accurate model of the airframe can be created, the software can extrapolate from prior observations and make short-term predictions about the status of the aircraft. This allows the system to compensate for hardware failure. This chapter covers the details of analytical sensor redundancy, its uses, and its limitations.

2.3.6. Chapter 8 - Software Reliability Assessment Methods

As flight functions are increasingly carried out by digital systems, the issue of the reliability of the software controlling these systems becomes more and more important to flight safety. Higher order languages, testing, debugging, and software engineering can all contribute to increased reliability. Models have been developed to assess the reliability of these systems. This chapter covers existing software reliability assessment models. It compares the models and gives an overview of their reliability as predictive tools.

2.3.7. Chapter 9 - Software Fault Tolerance

This chapter covers software methods and techniques used for detection and correction of faults. These may include checks on the reasonableness of data and comparison of information from redundant systems. This chapter also covers fault tolerant design using complexity measures and other techniques.

2.3.8. Chapter 10 - Latent Faults

Latent faults are faults which have occurred, but have not been detected because the path on which they have occurred has not been traveled. They can be both hardware and software faults. This chapter covers experiments conducted to inject faults and discover them. It discusses the prediction, triggering, identification, and correction of latent faults.

2.3.9. Chapter 11 - Guidelines To Assess EMC Designs

This chapter covers guidelines that may be used to assure electromagnetic compatibility during the design process. It also includes information on how

the certification engineer can assess how well a design or system adheres to the guidelines.

2.3.10. Chapter 12 - Fast Rise Time EMC Transient Testing

This chapter discusses EMC transient testing experiments. Presently, the chapter contains the theory for the testing only, since the testing itself has not been conducted and may not be completed in time for this handbook.

2.3.11. Chapter 13 - Lightning Studies

As more and more flight-critical systems are digitized and as composite materials become more widely used in aircraft construction, the threat from lightning increases. This chapter covers currently accepted information about lightning including estimates of currents, electric fields, and magnetic fields which aircraft might expect to encounter. It reviews methods for protecting electrical and electronic equipment from this hazard and it covers methods for isolating equipment so that potential damage may be limited.

2.3.12. Chapter 14 - High Energy Radio Frequency Fields

With changes in aircraft construction from aluminum to composites and increased criticality of software-based digital flight control and avionic systems, total aircraft RF susceptibility hardening and protection is vital. This chapter assesses the present criteria, specifications, and procedures for industry and certification authorities in the light of increasing levels of RF over a wide frequency range.

2.3.13. Chapter 15 - Electromechanical Actuator Systems (EMAS)

This chapter will cover the technical description, testing, and redundancy requirements for smart servos and actuators, primary and secondary power sources, and digital engine controllers.

2.3.14. Chapter 16 - Advanced Validation Issues

This chapter gives a brief overview of areas where advances are likely in the near future. As new information becomes available, each of these sections may be expanded to form a separate chapter.

The topics to be covered are:

- Artificial Intelligence
- Remote Processing
- Robotics (Testing Control)
- Advanced Architecture (Distributed Fault Tolerant Processing)
- Advanced Controls and Displays

- Voice Command Processing
- System Integration Technology

ACRONYMS AND ABBREVIATIONS

CRMI	Computer Resource Management, Incorporated
CT	Technical Center Identification code
DOT	Department of Transportation
EMAS	Electromechanical Actuator System
FAA	Federal Aviation Administration
NASA	National Aeronautics and Space Administration
RTCA	Radio Technical Commission for Aeronautics
SAE	Society of Automotive Engineers
STC	Supplemental Type Certification
TC	Type Certification

U.S. Department of Transportation
Federal Aviation Administration

DOT/FAA/CT-88/10

HANDBOOK-VOLUME II
DIGITAL SYSTEMS VALIDATION

CHAPTER 3
INTEGRATED ASSURANCE ASSESSMENT



PREPARED BY:

COMPUTER RESOURCE MANAGEMENT, INC.
950 HERNDON PARKWAY, SUITE 360
HERNDON, VIRGINIA 22070

PREPARED FOR:

FEDERAL AVIATION ADMINISTRATION
TECHNICAL CENTER
ATLANTIC CITY INTERNATIONAL AIRPORT, NEW JERSEY 08405

NOTICE

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The United States Government assumes no liability for the contents or use thereof.

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the objective of this report.

TABLE OF CONTENTS

Section		Page
1.	INTRODUCTION TO INTEGRATED ASSURANCE ASSESSMENT	3-1
	1.1. Justification for Integrated Assurance Assessment	3-1
	1.2. Primary Source for this Tutorial	3-2
	1.3. How to Use this Document	3-2
2.	IAA OF A RECONFIGURABLE DIGITAL FLIGHT CONTROL SYSTEM	3-5
	2.1. The Redundant Digital Flight Control System	3-5
	2.2. The IAA Study	3-11
	2.3. Conclusions and Observations	3-11
3.	CONSIDERATIONS IN THE DEVELOPMENT OF AN IAA APPROACH	3-15
	3.1. General Comments	3-15
	3.2. An IAA Approach	3-15
	3.3. Fault Tree Role in Integrated Assurance	3-16
	3.4. Failure Mode and Effect Analysis Role in Integrated Assurance	3-17
	3.5. Reliability Analysis Role in Integrated Assurance	3-18
	3.6. Fault Insertion Role in Integrated Approach	3-19
4.	FAULT TREE ANALYSIS	3-21
	4.1. General Comments	3-21
	4.2. Fault Tree Development	3-21
	4.3. Quantitative Fault Tree Analysis	3-37
5.	FAILURE MODE AND EFFECT ANALYSIS	3-43
	5.1. General Comments	3-43
	5.2. Application of FMEA to RDFCS	3-44
6.	FAULT INSERTION	3-49
	6.1. General Comments	3-49
	6.2. Application of Fault Insertion to RDFCS	3-49
	6.3. Fault Insertion Results	3-56

TABLE OF CONTENTS (Continued)

Section		Page
7.	FAILURE RATE DEVELOPMENT	3-57
	7.1. General Comments	3-57
	7.2. Rationale for Developing Failure Rates for RDFCS	3-57
8.	RELIABILITY PREDICTION USING CARSRA	3-63
	8.1. General Comments	3-63
	8.2. RDFCS Reliability - Fault Tree Analysis	3-63
	8.3. RDFCS Reliability - CARSRA	3-64
	8.4. CARSRA Application	3-64
	8.4.1. Configuration Description	3-64
	8.4.2. Markov Models	3-64
	8.4.3. Dependencies	3-67
	8.4.4. Availability	3-69
9.	SUMMARY	3-79
	9.1. Background	3-79
	9.2. Concluding Remarks	3-79
	9.3. Analysis Enhancements	3-80
APPENDICES		
	A - SUMMARY DESCRIPTION FOR REDUNDANT DIGITAL FLIGHT	3-83
	B - DESCRIPTION OF RDFCS SIMULATOR	3-85
	C - FAILURE MODES AND EFFECTS ANALYSIS RESULTS	3-89
	D - PROCESSOR SCHEMATIC DIAGRAMS	3-101
	E - RESULTS OF FAULT INSERTION	3-107
BIBLIOGRAPHY		3-123
ACRONYMS AND ABBREVIATIONS		3-125

LIST OF ILLUSTRATIONS

Figure		Page
2.1-1	RDFCS DUAL-DUAL CONFIGURATION	3-6
2.1-2	RDFCS SIMULATOR	3-7
2.1-3	CTA AND COMPUTER BREAKOUT PANEL	3-8
2.1-4	SERVO SIMULATOR PANEL	3-9
2.1-5	DISCRETE SWITCH PANEL	3-10
4.2-1	FAULT TREE TOP LEVEL	3-24
4.2-2	SENSING FUNCTION	3-25
4.2-3	NORMAL ACCELERATION SENSING	3-26
4.2-4	NO DUAL ANNUNCIATION	3-29
4.2-5	COMPUTING FUNCTION	3-31
4.2-6	CHANNEL 1A RUDDER COMMAND	3-32
4.2-7	FCC PROCESSOR CONTROL CARD	3-33
4.2-8	FCC PROCESSOR DATA PATH CARD	3-34
4.2-9	YAW AUTOPILOT SERVO COMMAND WARNING	3-35
4.2-10	SERVO FUNCTIONS	3-39
4.2-11	NO. 1 YAW AUTOPILOT SERVO	3-40
4.2-12	MULTIPLE FAILURES DURING CRUCIAL PHASE	3-41
5.1-1	ONE OUTPUT, TWO INPUT CONDITIONS	3-46
5.1-2	TWO OUTPUT, ONE INPUT CONDITION	3-47
5.2-1	PROCESSOR BLOCK DIAGRAM	3-48
6.2-1	CTA AND COMPUTER BREAKOUT PANEL	3-50
6.2-2	SERVO SIMULATOR PANEL	3-51

LIST OF ILLUSTRATIONS (Continued)

Figure		Page
6.2-3	DISCRETE SWITCH PANEL	3-52
6.2-4	FCC WITH FLIGHT CARD EXTENDED	3-54
6.2-5	FCC PROCESSOR DATA PATH CARD	3-55
8.4-1	MARKOV MODEL OF DUAL STAGE	3-66
8.4-2	MARKOV MODEL CODING FOR SENSOR STAGES	3-71
8.4-3	CARSRA INPUT (1 of 6)	3-72

LIST OF TABLES

Table		Page
1.1-1	INTEGRATED ASSURANCE ASSESSMENT MODEL	3-2
2.2-1	ASSURANCE METHOD FUNCTIONS	3-12
4.3-1	QUANTITATIVE RESULTS	3-38
7.2-1	FCC CONTROL CARD FAILURE RATE	3-59
7.2-2	PREDICTED FCC CARD FAILURE RATES	3-60
7.2-3	FAILURE RATES FOR MAJOR RDFCS COMPONENTS	3-61

1. INTRODUCTION TO INTEGRATED ASSURANCE ASSESSMENT

1.1. Justification for Integrated Assurance Assessment

The validation of flight-essential and flight-critical digital systems is of crucial importance in the determination of each system's airworthiness. This tutorial is a reference for certification engineers involved in the functional assessment of aircraft which use digital systems for avionics and flight control functions. Part 25 of the Federal Acquisition Regulation (FAR) (specifically FAR 25.1309) contains the requirements for flight-essential and flight-critical avionic systems. Federal Aviation Administration (FAA) Advisory Circular (AC) 25.1309-1 provides guidance material for demonstrating compliance with FAR 25.1309. This AC outlines the use of quantitative safety analysis techniques that may include: (1) Probability Analysis, (2) Fault Tree Analysis, (3) Failure Modes and Effects Analysis (FMEA), and (4) other comparable techniques for determining compliance with the requirements of FAR 25.1309 (b).

The use of any of these analysis techniques in isolation for an entire avionic system introduces some difficulties. To eliminate or minimize these difficulties, an Integrated Assurance Assessment (IAA) is desirable. A model for an integrated assurance approach may include a number of techniques, each contributing to the value of the analysis. For instance, fault tree analysis provides assurances that most system components have no single failure mode that results in system failure. When components have several failure modes that may have different effects on system readiness, fault tree analysis may prove cumbersome and inefficient. In this latter case, FMEA may be used to extend the examination below the component level to the failure modes of the individual piece-parts that comprise the component. Integrating fault tree analysis and FMEA permits the diagnosis of system failure from system-level functions through Integrated Circuit (IC) pins.

The above simple integration may be used to develop a probability for system failure. If the analysis contains no errors, this probability is true. If an error exists, the probability may be erroneously low and go undetected. Computation of the system failure probability by an independent analytical technique would either confirm the results of the above analysis or detect that an error was made. A computer-aided reliability analysis would provide that independent confirmation. Extending the integration to incorporate this technique means that the scope of the analysis begins at the system level and extends through the piece-part level, a system probability error may be computed, and a confirmation of that system probability error obtained.

In performing the fault tree analysis or FMEA, the effects of certain faults may not be known or may be uncertain. If this is the case, a technique for determining or verifying the effects of certain faults would improve the IAA. Fault Insertion is a technique for determining or confirming the fault effect.

Incorporating the fault insertion technique in an integrated assurance model improves the analysis that may be performed with that model.

The IAA model defined above utilizes four analysis techniques. Each of the four was incorporated in the model for specific reasons. Table 1.1-1 shows the techniques and the simplified capabilities each adds to the integrated model. Section 2 will discuss this model in greater detail.

TABLE 1.1-1. INTEGRATED ASSURANCE ASSESSMENT MODEL

Technique	System Level Capability
Fault Tree	Determine System Failure Effect of Component and Piece-Parts; Compute System Probability of Failure
Failure Mode and Effect	Determine System Failure Effect of Piece-Parts
Reliability Analysis	Confirms System Probability of Failure
Fault Insertion	Confirms System Failure Effect of Component and Piece-Parts

The evolution of an integrated assurance model is dependent on the system to be diagnosed. Simply, the capabilities of analysis, detection, annunciation, and confirmation must be provided. The specific analytic techniques that provide these capabilities may vary from model to model.

1.2. Primary Source for this Tutorial

The primary source for this tutorial is report Number DOT/FAA/CT-82/154, developed by the Lockheed-Georgia Company, entitled "IAA of a Reconfigurable Digital Flight Control System." The report documents an assurance assessment of a representative contemporary Digital Flight Control System (DFCS). The complementary use of various analytical techniques for safety analysis was stressed. Their primary objective was to explore and demonstrate the integrated application of reliability, failure effects, and system simulator methods in establishing the airworthiness of a flight-critical DFCS. The emphasis was on the mutual reinforcement of the methods.

1.3. How to Use this Document

This tutorial is divided into nine major sections:

- Section 1, "Introduction to IAA," provides an overview of IAA. It covers the Regulations and ACs to which IAA is a response. It shows four typical safety analysis techniques meshed to form an IAA model. The justification for incorporating each of the techniques is given.

- Section 2, "IAA of a Reconfigurable Digital Flight Control System," provides a discussion of the research effort introduced in Section 1. The digital system is defined and the dual-dual attribute of this system is discussed. Both the production configuration and the simulator for the subject digital system are discussed.

- Section 3, "Considerations in the Development of an IAA Model," discusses the benefits that accrue to the IAA effort for each of the safety analysis techniques incorporated. The IAA model demonstrated here was the model actually used in the subject study.

- Section 4, "Fault Tree Analysis," demonstrates the use of fault tree analysis as a top-down, deductive analysis tool to be used in the identification of conditions and functional failures that contribute to higher level component or system failure. This section contains a detailed application of fault tree analysis in the subject study.

- Section 5, "Failure Mode and Effect Analysis," demonstrates the use of FMEA as a bottom-up, inductive analysis tool to be used to determine what happens to the system or higher level components when individual parts fail. This section contains a detailed analysis of three modules of a microprocessor.

- Section 6, "Fault Insertion," demonstrates the use of fault insertion in determining and confirming fault effects by fault simulation. A detailed example is given.

- Section 7, "Failure Rate Development," demonstrates how failure rates required by the fault tree analysis and the reliability prediction tool were obtained. It does not illustrate a safety analysis technique applied in the IAA.

- Section 8, "Reliability Prediction using (Computer-Aided Redundant System Reliability Analysis) CARSRA," demonstrates the use of an analytical reliability prediction program in obtaining the probability of system failure. The details of the application of this tool to the subject study are given.

- Section 9, "Summary Statement," summarizes the tutorial. Summary commentary is given on the experience gained from performing the Redundant Digital Flight Control Analysis - relating benefits, limitations, and suggested improvements.

If the reader is interested in general knowledge, sections 1, 2, and 9 will suffice. This will provide an overview of what IAA is, when its use should be considered, and what advantages or disadvantages are likely to result from its use. Section 3 contains a detailed discussion on why the selected techniques were incorporated in the IAA model. Sections 4 through 8 contain a detailed discussion on each of the selected safety analysis techniques and their use in the subject study.

2. IAA OF A RECONFIGURABLE DIGITAL FLIGHT CONTROL SYSTEM

2.1. The Redundant Digital Flight Control System

Under report number DOT/FAA/CT-82/154, entitled "IAA of a Reconfigurable Digital Flight Control System," referenced in section 1.2, an IAA was performed on the Redundant Digital Flight Control System (RDFCS). This digital system was procured jointly by the FAA and the NASA-Ames Research Center and was a central part of the DFCS Verification Laboratory at NASA-Ames. Appendix A contains a summary description of the system architecture. While the sensors and servos are described in the production configuration, they were not actually present in the RDFCS configuration but were simulated.

In most operational modes, the system is fail-passive with a dual channel configuration. For automatic landings under Category IIIa conditions (see AC 120-28C), the system can be brought into a dual-dual fail-operational, fail-passive configuration. The classification dual-dual relates primarily to the four computer channels in the system. Each of the two Flight Control Computers (FCCs) has two channels which run frame-synchronously; each channel drives one coil of a dual-coil servo in each axis. Any indication of disagreement between the two channels in an FCC causes the servo connected to that FCC to disengage. Figure 2.1-1 summarizes the dual-dual configuration.

The RDFCS simulator is comprised primarily of a PDP 11/60 computer and the RDFCS pallet. The pallet is shown in figure 2.1-2. The PDP 11/60 computer hosts a discrete-state model of the airplane in which the RDFCS is installed. The RDFCS pallet contains the FCCs, the core memory for the FCCs, and the following devices:

- Modular Digital Interface Control Unit (MDICU) controls interface to the PDP 11/60.
- Computer Breakout Panel permits optional interruption of signals passing from MDICU to the FCC. The lower portion of figure 2.1-3 shows part of the Computer Breakout Panel.
- CAPS (Computer Aided Production Simulator) Test Adapters (CTA) permits address monitoring on the FCC processor bus. The upper portion of figure 2.1-3 shows the CTA.
- Servo Simulation Panel (SSP) simulates the dynamics of servos. The SSP is shown in figure 2.1-4.
- Discrete Switch Panel (DSP) contains switches affecting the Instrument Landing Systems (ILS). The DSP is shown in figure 2.1-5.

Appendix B contains a detailed description of the RDFCS simulator.

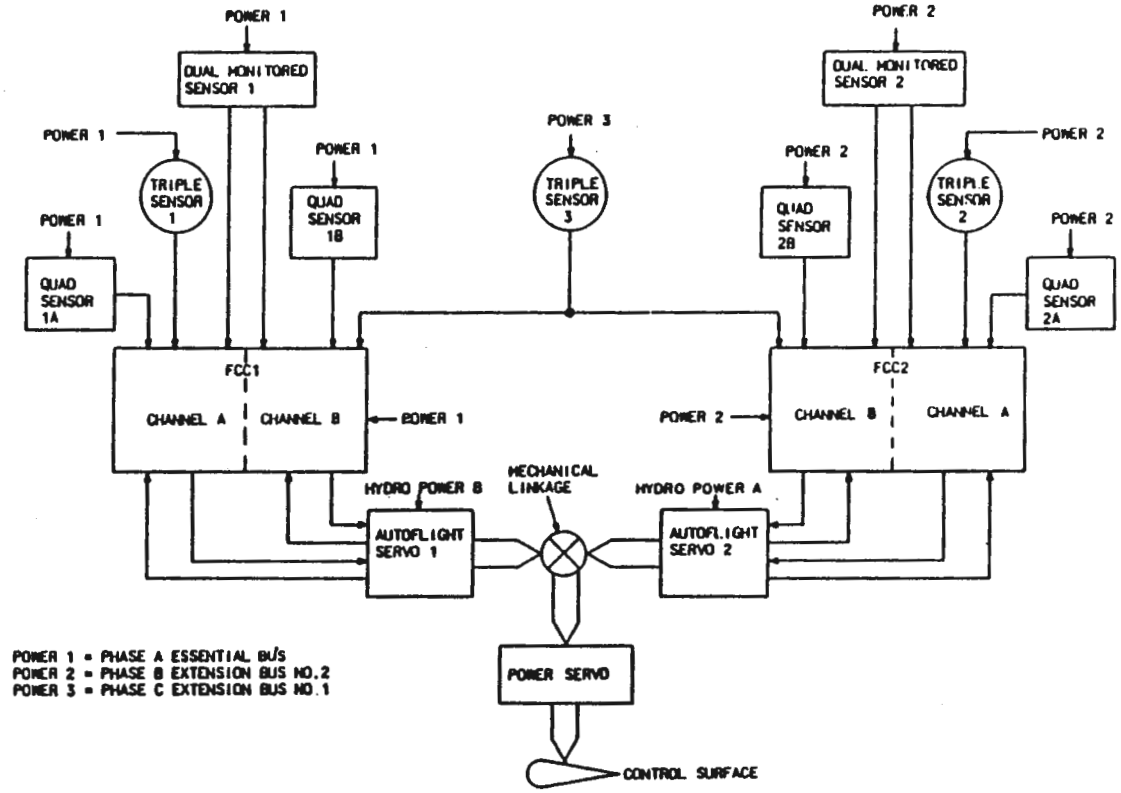


FIGURE 2.1-1. RDFCS DUAL-DUAL CONFIGURATION

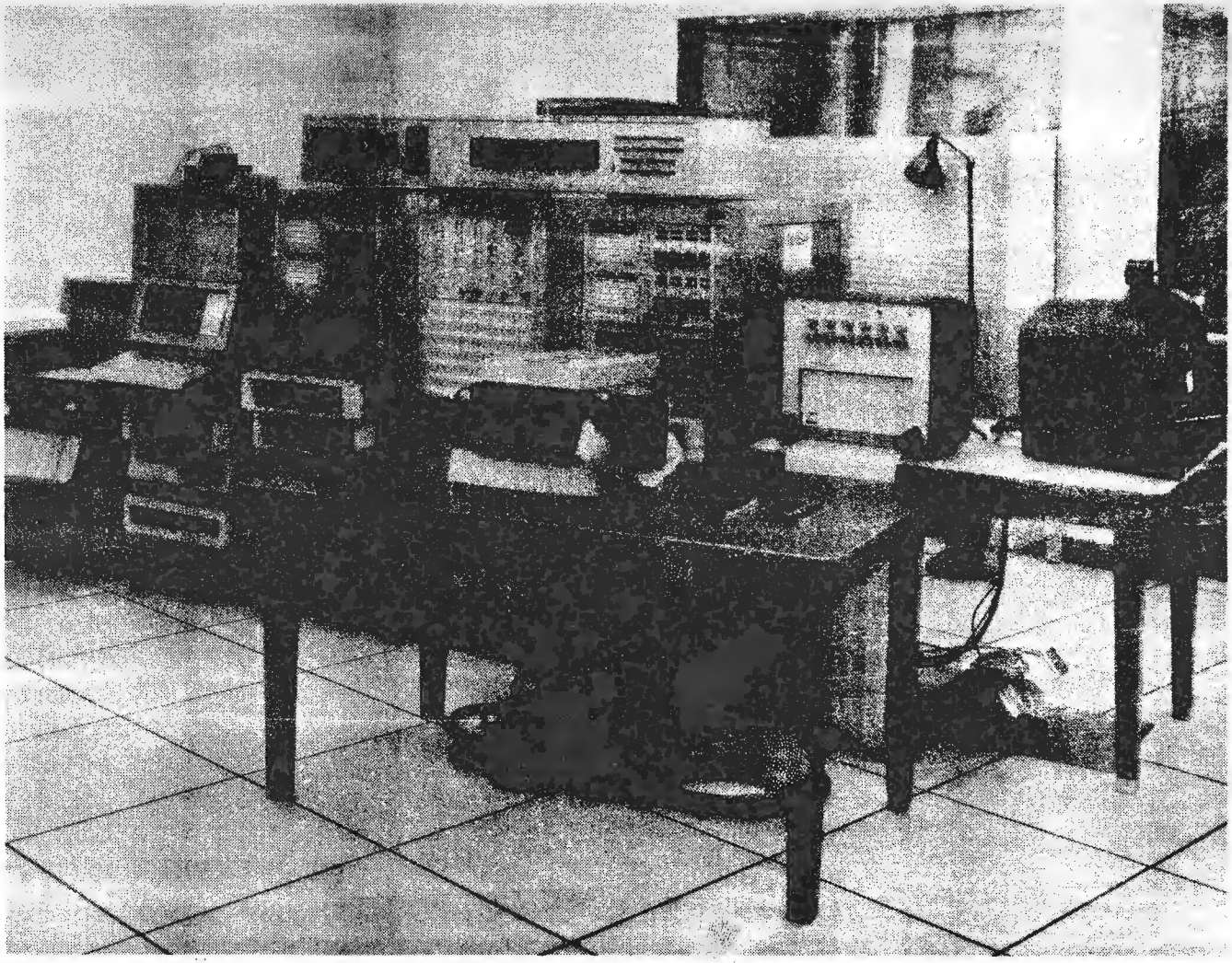


FIGURE 2.1-2. RDFCS SIMULATOR

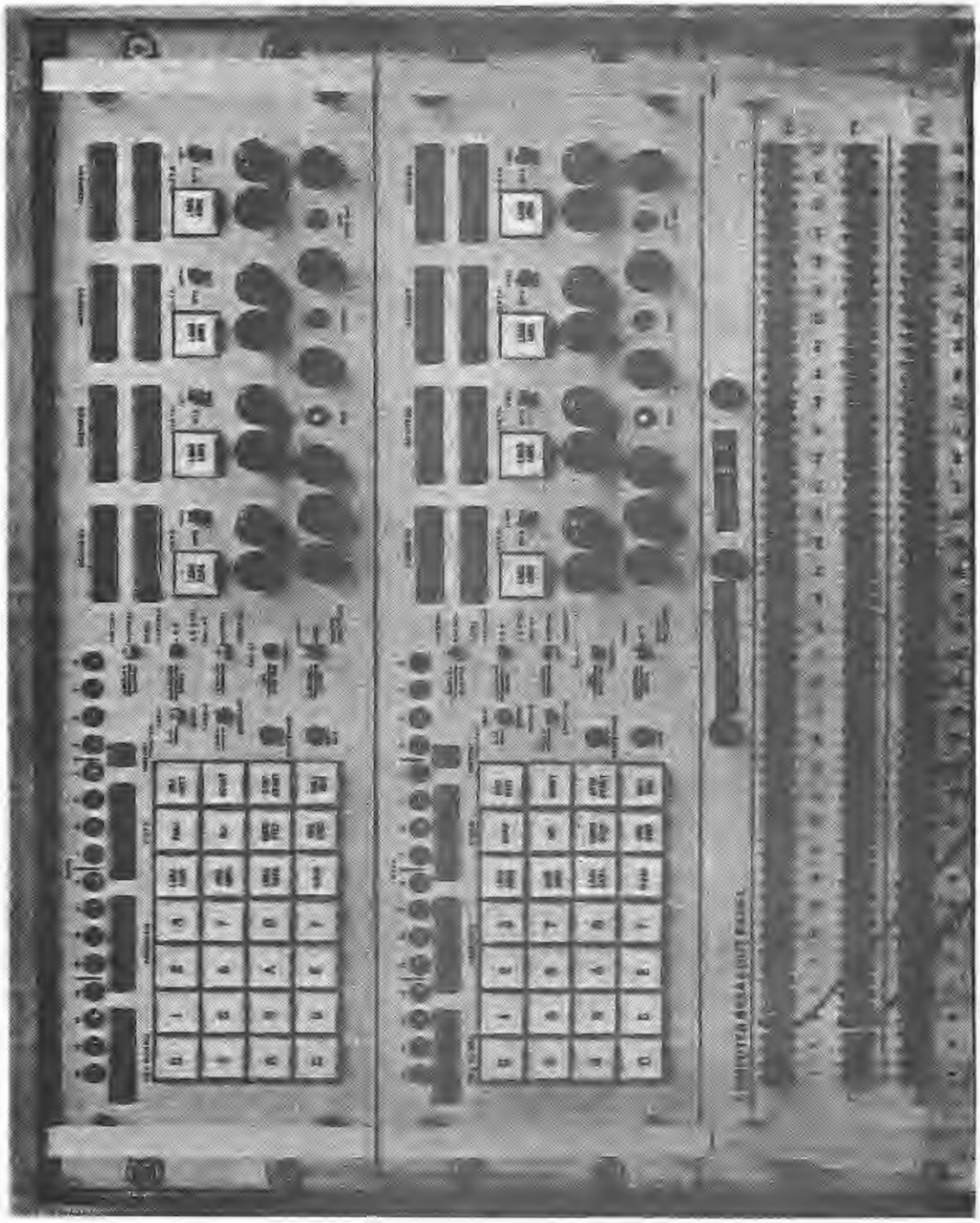


FIGURE 2.1-3. CTA AND COMPUTER BREAKOUT PANEL



FIGURE 2.1-4. SERVO SIMULATOR PANEL

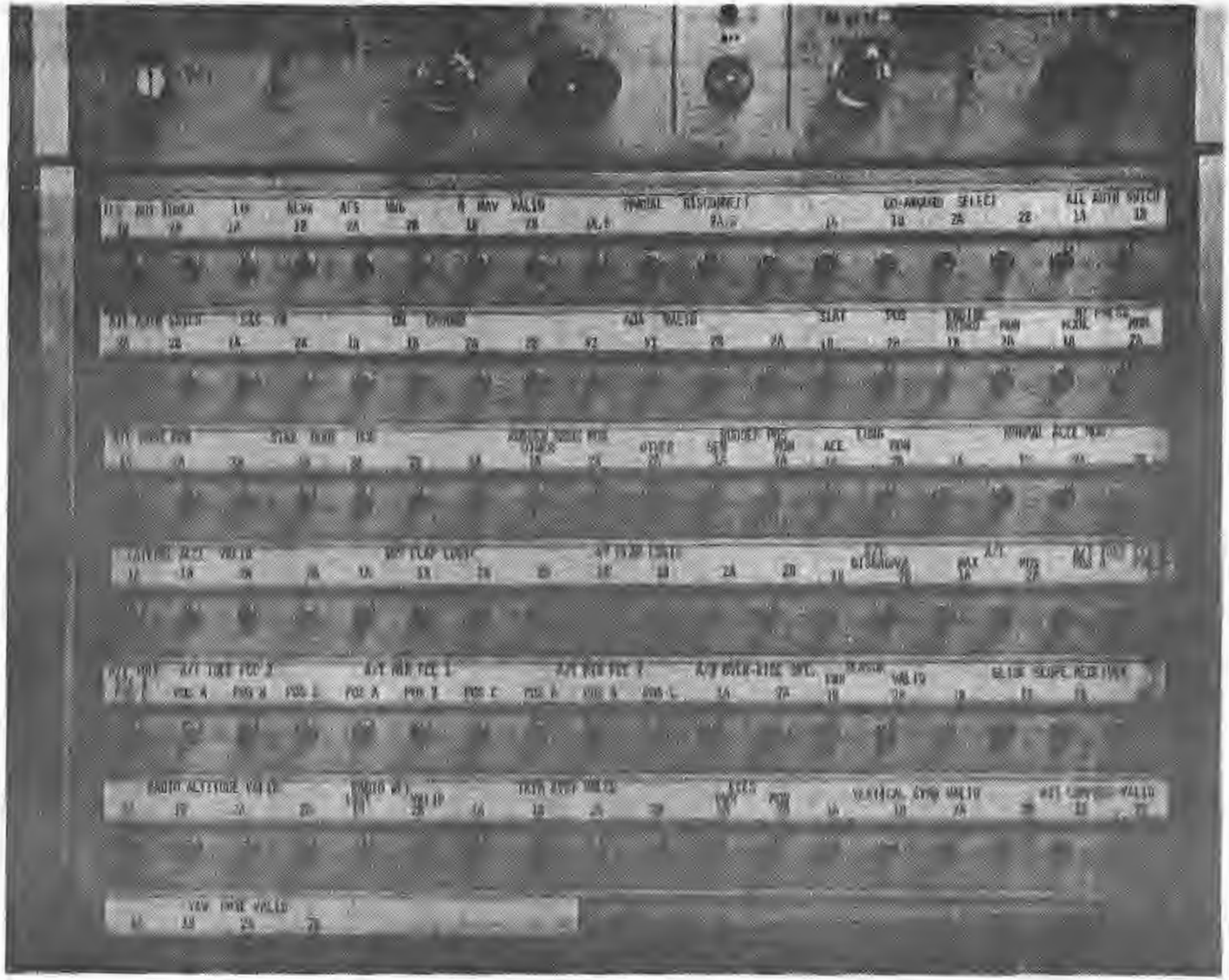


FIGURE 2.1-5. DISCRETE SWITCH PANEL

2.2. The IAA Study

During this study, the assurance methods of fault tree analysis, automated reliability prediction, FMEA, and fault insertion were applied to create a workable approach to confirming the airworthiness of the RDFCS. The scope of the effort was primarily limited to assessment of the automatic landing maneuver under Category IIIa conditions as defined in AC 120-28C. The analysis is based on the event that the airplane has an unacceptable deviation from the desired flight profile during the last 150 feet of descent while executing an automatic landing. This portion of the flight is the only flight phase during which the RDFCS performs a critical function. Category IIIa conditions imply that the human pilot cannot complete the landing using visual cues should the RDFCS fail. The application of safety analysis techniques was on a selective basis and addressed those system aspects identified in table 2.2-1.

The assessment consisted of the following major tasks:

- Application of fault tree analysis starting at the highest system functional level and proceeding to the hardware circuit card level and the module level for the processors.

- Development of a representative set of failure rates for the relevant hardware items.

- Application of an automated program, CARSRA, to the system failure modes affecting airworthiness.

- Application of FMEA to IC pin faults of three processor modules.

- Definition of faults to be inserted in the RDFCS. Faults to be inserted include faults used to determine the effect of the fault when analysis was not feasible, and faults used to confirm the manual analysis. These faults were subsequently inserted and the effects recorded.

2.3. Conclusions and Observations

The conclusions and observations resulting from this study are as follows:

- The integrated approach used in the study is capable, with diligent application, of establishing the airworthiness of a DFCS within the context of AC 25.1309-1. Specifically, this approach addresses those system aspects shown in table 2.2-1, including freedom from single-point failure modes and system failure probability.

- The IAA used in this study should be considered for use in validating other digital systems, including DFCS, in compliance with AC 25.1309-1.

TABLE 2.2-1. ASSURANCE METHOD FUNCTIONS

SYSTEM ASPECT	ASSURANCE METHOD	
	PRIMARY	CONFIRMATION
FAILURE EFFECTS		
- COMPONENT	FAULT TREE ANALYSIS	FAULT INSERTION
- DIGITAL MODULE	FAULT TREE ANALYSIS, FAILURE MODE AND EFFECT ANALYSIS	FAULT INSERTION
- DIGITAL INTEGRATED CIRCUIT	FAILURE MODE AND EFFECT ANALYSIS	FAULT INSERTION
- UNTRACTABLE CASES	FAULT INSERTION	
FAULT DETECTION/ ANNUNCIATION	FAULT TREE ANALYSIS	FAULT INSERTION
SOFTWARE FUNCTION IMPLEMENTATION	SOFTWARE TEST PROGRAM	FAULT TREE ANALYSIS
NO SINGLE-POINT FAILURE MODES	ABOVE, AS RELEVANT	ABOVE, AS RELEVANT
SYSTEM FAILURE PROBABILITY	RELIABILITY PREDICTION PROGRAM	FAULT TREE ANALYSIS QUANTITATIVE EVALUATION

- The quantitative assessment of system failure probability by two methods (fault tree analysis and analytical reliability prediction) offers increased assurance that the system meets the quantitative requirements of AC 25.1309-1. For a flight-critical system, this requirement is that the system failure probability not exceed 10^{-9} per hour of flight for each critical function the system performs.

- Fault insertion confirms that the fault detection capability and the fault tolerance capability described in the system documentation are actually implemented in the system. Since the fault tree analysis is based largely on the system response to faults as described in the system documentation, the fault insertion confirms that the fault tree analysis correctly reflects the behavior of the actual system in the presence of faults.

- The fault tree analysis generates software test requirements in terms of functions which the software must perform. These, in turn, provide a check of function criticality and of test requirements generated in accordance with Radio Technical Commission for Aeronautics (RTCA) Document DO-178.

- Fault tree analysis proved unwieldy below the circuit card level, because at lower levels many more functions are being performed than there are hardware failure modes. FMEA was accomplished successfully at the IC pin level.

- As a training facility and a Reconfigurable Test Bed, the RDFCS facility has significant and valuable capabilities for investigating assurance issues of currently definable DFCS architectures. It also has potential enhanced capability in certain areas, such as automated insertion of pin-level faults, for confirmation of analytically determined failure effects.

- The comparison of the time or cost required for the integrated approach reported here with that required for other possible assurance approaches was not specifically addressed in this study. However, the time required for the integrated approach is expected to compare favorably with that for other approaches, assuming the same depth of analysis. The cost should also compare favorably, provided a facility suitable for fault insertion is available.

3. CONSIDERATIONS IN THE DEVELOPMENT OF AN IAA APPROACH

3.1. General Comments

As a first step, a preliminary hazard analysis should be performed on the aircraft model. This analysis should identify those system functions that are critical and the aircraft systems that must operate correctly in order for the function to be performed correctly. According to AC 25.1309-1, a critical function is one "whose failure would contribute to or cause a failure condition which would prevent the continued safe flight and landing" of the aircraft. Essential system functions should also be identified. Essential functions are those "whose failure would contribute to or cause a failure condition which would significantly impact the safety of the airplane or the ability of the flight crew to cope with adverse operating conditions." A complete quantitative system analysis is normally required for critical systems but is more discretionary for essential systems. The following list illustrates the types of functions that may be critical:

- The primary flight control system.
- Hydraulic power for airplanes with powered flight control systems and no manual reversion.
- Secondary flight control systems, if failure of these systems can result in uncontrolled flight.
- Engine control system elements that affect all engines simultaneously.

Since the RDFCS performs critical system functions, a rigorous assurance assessment is justified. AC 25.1309-1 suggests that failure conditions for critical system functions be improbable. This requires a failure probability of less than 1.0×10^{-9}. An IAA incorporating both quantitative and qualitative analyses offers advantages over the application of a single methodology. The analytical techniques used here to perform the integrated assessment are in harmony with AC 25.1309-1.

3.2. An IAA Approach

The steps required in performing a typical integrated assessment of the RDFCS are as follows:

- System Description: Define a baseline configuration of the RDFCS and prepare a corresponding analytical description. This description may include those additional components, beyond the RDFCS, whose exclusion would make the analysis unrealistic.

- Fault Tree Analysis: Perform a fault tree analysis beginning at the system level. To be meaningful, this analysis should extend beyond the component level to the piece-part level. This will be the case when digital modules are present with a large number of different failure modes. To perform this total analysis, both the fault tree and the FMEA tools should be used.

- Failure Rate Generation: Develop a set of representative failure rates for the components and parts of the RDFCS that will permit the evaluation of the fault tree for failure probability.

- Fault Simulation Review: Define the simulated fault conditions that are to be inserted into the RDFCS simulator. These faults shall be inserted for two purposes: to confirm the assumptions underlying the fault tree analysis, and to resolve uncertainty of the effect of the fault when analysis is not easily controlled.

- Flight Case Review: Review those flight cases available on the RDFCS simulator. If they are inadequate, augment them. Assure that all needed transitions between cases are possible.

- Reliability Analysis: Determine the tool to be used for analytic computation of system reliability. The CARSRA reliability program was used in the subject study, and is discussed in this tutorial.

3.3. Fault Tree Role in Integrated Assurance

The IAA of the RDFCS begins with a fault tree analysis of the system function. As shown in table 2.2-1, the fault tree analysis has several functions. The first function is to assure that no system component has any failure mode which can result in system failure. Most of the components, such as the sensors and servos, have only a few failure modes which can be observed at the interfaces with the rest of the system. For these components, the fault tree analysis provides assurance that no failure modes can cause system failure. The assurance is obtained by reviewing the completed tree and determining that system failure can only occur as a result of multiple failures.

In general, digital modules (and therefore digital components) can have a substantial number of different failure modes. In such cases, it becomes quite laborious to continue the fault tree development to a level of detail sufficient to confirm that none of those failure modes can cause system failure. The second function of fault tree analysis is to identify which digital modules are involved in performing critical functions. The task of assuring that no single module level failure can cause system failure is performed with FMEA.

A major benefit of fault tree analysis is that it focuses on the functions performed by the system elements, including those system elements involved in detecting faults and providing appropriate annunciation to the flight crew. Consequently, the third function of fault tree analysis is to confirm the adequacy of monitoring (i.e., fault detection and annunciation) in the system.

Fault tree analysis is also used to identify specific software functions required for system operation, including fault monitoring implemented in

software. The software test requirements for these functions are then specifically reviewed to confirm that these requirements are adequate. This fourth function of fault trees is illustrated and discussed in greater detail as the tree for the RDFCS is developed.

The fifth function of fault tree analysis is to provide an alternate means of computing the probability of system failure. This provides a check of the probability obtained from the reliability analysis. In the subject study, the fault tree analysis verified the probability obtained from the CARSRA program to ensure that the CARSRA input does not have errors which would produce a false low probability of system failure.

3.4. Failure Mode and Effect Analysis Role in Integrated Assurance

Fault tree analysis provides assurance that most system components, such as analog sensors and servos, have no single failure mode which produces system failure. This is because such components have only a few possible failure modes, and it frequently is not necessary to distinguish in the fault tree among these modes. When it is necessary to distinguish among modes, it is usually fairly simple to identify the modes which are relevant in the branch of the tree being developed. The analysis can often be extended below the component level to the failure modes of the individual piece-parts which comprise the component. Analysis to this very detailed level is sometimes necessary to ascertain that a component has no failure modes which could remain undetected until a second failure occurs elsewhere in the system.

However, fault tree analysis is cumbersome and inefficient if extended from system level to the IC pin level in the processor of a digital system. Essentially, this is a result of two basic characteristics of digital systems:

- Functions which are described very simply at a higher level (e.g., sensor monitoring) require a myriad of sequential operations at the IC level. These operations are required to obtain the proper data, route it to the proper registers within the Arithmetic Logic Unit (ALU) where arithmetic and logic operations are actually performed, and route the results to the proper storage register or output port. Many different ICs are involved in each of these operations.

- Many interfaces between ICs involve several pins, and it is the combination of pin states (electrically high or low) which is significant. That is, each combination of pin states represents a different data value or instruction, and the effect of a single pin being in the wrong (faulted) state depends on the state of the other (non-faulted) pins.

The net result of these characteristics of digital hardware is that there are many more integrated-circuit-level operations performed in executing the flight software than there are pin-level failure modes. In extending a fault tree analysis from failure of system-level functions to failure of IC pins, all of these detailed operations must be included and accounted for-an extremely inefficient process. Once the fault tree had been fully developed, another extremely laborious task would remain: reviewing the tree to make certain (1) that all of the failure modes of the ICs had been accounted for, and that no

failure mode could remain undetected until a second failure occurred, with the combined effect of both faults producing a hazardous condition; and (2) that no failure mode could by itself produce a hazardous condition.

FMEA provides a means of systematically examining all of the potential failure modes of the ICs to confirm that none of them could cause a hazard directly or remain latent and subsequently cause a hazard in conjunction with a second failure.

3.5. Reliability Analysis Role in Integrated Assurance

CARSRA is an analytical reliability prediction program used in this IAA to obtain the probability of system failure (Bjurman, et al., 1976). The use of CARSRA, along with the quantitative assessment produced by evaluating the fault tree analysis, provides two independent computations of system failure probability. This reduces the risk of a false, low probability of failure being produced by a single method and the error remaining undetected.

Although CARSRA is identified specifically in the IAA used in this analysis, some other method (except fault tree analysis) could be used. If an alternate method is used, it should have sufficient configuration adaptability to produce the predicted probability of system failure without requiring simplifying assumptions which would produce a false, low prediction. According to Hitt and Eldredge, the particular fault-tolerant system attributes that reliability models should be capable of handling are the following:

- Computer design (e.g., failure modes, loss of subsets of functions)
- Dependencies between subsystems
- Number of redundant components
- Redundancy management (e.g., active or standby spares, voting algorithms)
- Permanent, latent, and transient faults
- Fault recovery (i.e., coverage)
- Data transmission schemes and bus architectures
- Partitioning of processing functions

Some of the earlier reliability analysis models were adequate for analysis of certain types of systems. Examples of these are Tree Aided System Reliability Analysis (TASRA), Reliability Block Diagram Computer Program (RBDCP), Reliability (REL 70), and Reliability Computers (REL COMP) models that were primarily developed in the early 1970s. Later models such as, Automated Reliability Interactive Estimation System (ARIES) (Ng and Avizienis, 1978), Computer Aided Reliability Evaluator (CARE II) (NASA, NASI-12668, March 1974), CARSRA (Bjurman, et al., 1976), and Complimentary Analytic Simulative Technique (CAST) (Conn et al., 1977) offered expanded capabilities that were more adaptable to systems

having more components and greater complexity. CARE III and an improved ARIES are examples of the continuing evolution.

The certification engineer must choose the appropriate reliability analysis model based on the ability of that model to represent the system to be analyzed. In the event that a suitable match cannot be made, manual analysis may be the only alternative.

3.6. Fault Insertion Role in Integrated Approach

Fault insertion is used in this IAA for three purposes as shown in table 2.2-1.

- Faults are inserted, on a sampling basis, to confirm the fault effects reflected in the fault tree analysis and fault effects determined during FMEA. This includes faults of components (sensors and servos in this study) and faults of ICs (pin-level faults in the digital processor).

- Faults are inserted, also on a sampling basis, to confirm fault detection and annunciation functions implemented in the system. Many of these are also inserted to confirm fault effects.

- Faults are inserted to determine the effect when the analysis is intractable or when there is some uncertainty in the analysis result.

4. FAULT TREE ANALYSIS

4.1. General Comments

The role of fault tree analysis in the IAA was specified in section 3.3. Appendix 1 of AC 25.1309-1 stipulates that for the purpose of conducting failure analyses, "the format of quantitative analyses, which use NUREG-0492 as a guide, will be acceptable to the FAA." NUREG-0492 is a textbook for fault tree analysis published for those unfamiliar with the concepts of systems analysis. It is titled "Fault Tree Handbook," and contains an excellent bibliography on the entire subject of reliability.

4.2. Fault Tree Development

AC 25.1309-1 defines fault tree analysis as "a top down deductive analysis identifying the conditions and functional failures necessary to cause a defined failure condition." The fault tree, when defined for the entire system failure state, may be used to mathematically determine a system failure probability by proper consideration of the failure probabilities of all contributing events.

Fault tree analysis is a structured method of "exploding" a top-level fault into all external events, conditions, or lower-level faults that contribute to it. The top-level fault may be a system failure or a component failure. In the former case, the lower-level faults are component faults; in the latter, piece-part faults. If they are not elementary, component faults may then be recursively exploded into their respective causative elements. The topology of the resulting graph (fault tree) permits the evaluation of the combinations of external events, conditions, or faults that cause the top-level fault.

In complex systems, the graphs become so cumbersome that user aids are necessary. One such aid is the "cut set." The "cut set" is the combination of lower-level events that result in a certain higher-level failure. The procedure for obtaining cut sets may either be manual or automated. The length of a cut set portrays the levels of redundancy in the system design. Minimal cut sets are important in that they show the minimum levels of redundancy. Minimal cut sets simplify the computation of the probability of failure of the respective top-level event.

The fault tree analysis contained in this section is based on the RDFCS being in the failure condition stipulated in section 2.2: simply, during a crucial flight landing phase, when the human pilot may not complete the landing, the RDFCS fails. In the following analysis this time period will be referred to as the "crucial phase."

The analysis begins with the RDFCS in the dual-dual configuration. The system is placed in this configuration by depressing the ILS push-button. The use of the ILS push-button results in the selection of the Approach/Land (A/L) mode.

After this switch has been momentarily depressed, the A/L mode is transmitted to the FCCs and latched in. The switch is no longer needed, and therefore does not enter into the analysis.

Figures 4.2-1 through 4.2-12 contain a partial fault tree analysis of the RDFCS during the crucial phase. The numbers in engineering notation appearing at nodal points are failure probabilities and will be used in sections 4.3 and 7.2.

The top event of figure 4.2-1, Unacceptable Deviation from Path/Altitude/ Speed Profile during Crucial Flight Phase, may be considered as equivalent to RDFCS failure. This event can be caused by any of three conditions or subevents. For convenience, these can be referred to as Level-2 events with the top event considered to be at Level-1. The Level-2 events are shown as the middle row in figure 4.2-1. The first of these is that the system design is in some manner deficient for the environmental conditions encountered. This includes the possibility that the conditions encountered are outside of the system design requirements. It also includes the possibility that the control laws are deficient for some conditions which may be expected. This possibility is outside the scope of this tutorial and is not pursued here. Mulcare et al., (1979) and DOT/FAA/CT-82/140 (1982) address this subject. In particular, section 3.3.1.3. of Mulcare et al., (1979), discusses establishing an upper bound on the probability of a deficient control law by statistical methods.

The second of the Level-2 events occurs if the airplane enters the crucial phase with the RDFCS not fail-operational, and then a component failure occurs which prevents the system from completing the landing.

The third of the Level-2 events is that the crucial phase is entered with a fail-operational RDFCS, but multiple component failures occur before the end of the phase, and these failures result in RDFCS system failure.

The second of the Level-2 events, that the crucial phase is initialized without fail-operational capability, is expanded into three relevant functional areas or Level-3 events: sensing aircraft altitude and position, computing required outputs, and servo response to computed commands. The first of these, the sensing function, is expanded in figure 4.2-2 into the various parameters needed by the FCCs in the automatic landing control laws. At this and higher levels, the fault tree is functionally oriented: failures are in terms of loss of function rather than loss of hardware.

The fault tree stub of figure 4.2-3 extends the sensing function for normal acceleration to the individual hardware elements used to measure the acceleration and transmit it to the computers. The failure of that normal acceleration signal No. 1 to be present in all computer channels can be caused by loss of the sensor itself, associated wiring, or one of the circuit cards involved in receiving the signal and transmitting it to all channels. The A24 Autoland Sensor Input and A27 Discrete Input Cards are both involved. The A24 card handles the analog acceleration signal and the A27 card handles the validity discrete signal. The processor itself is not involved in the data acquisition process and so is not shown. At this level, the transition has been made from required functions to the hardware which perform these functions.

Failure of the system to provide a NO DUAL annunciation is shown in figure 4.2-4. This figure is of particular interest because of the explicit software function identified. A failure rate of zero is assigned to failure of this function, because it can be explicitly and exhaustively tested. Once it has been so tested, the probability of both NO DUAL annunciations failing because of a generic software error is taken to be zero. A generic software error is a discrepancy in the software that will cause all computer channels which use that software to produce the same, but wrong, result. Multiple computer channels do not provide redundancy with respect to generic software errors as long as the same software is used in all channels (as it is in most contemporary systems) including the RDFCS. DOT/FAA/CT-82/140 may be consulted for a discussion of software errors, and RTCA Document DO-178 should be consulted for a discussion of software test requirements.

Fault tree stubs similar to that shown in figure 4.2-3 were developed for the other sensors of figure 4.2-2. These are very much like the stub shown in figure 4.2-3 and so are not included in this tutorial.

The second of the Level-3 events of figure 4.2-1 is initiation of the crucial flight phase without fail-operational computing capability, and that an additional component failure causes system failure before the phase is complete. This is shown in figure 4.2-5 as four Level-4 events. The first of these, that channel A of FCC No. 1 fails above alert height, can be caused by either channel of the FCC failing to produce a required output, as shown by the eight events at the lowest level (Level-5) in figure 4.2-5.

Figure 4.2-6 continues the development of the fault tree for one of the Level-5 events of figure 4.2-5. This event, failure of the A channel of FCC No. 1 to produce a rudder command, can be caused by failure of any one of several cards within the channel. In this study, the two cards which make up the processor were considered in more depth than the others. These two, the A13 Control Card and the A14 Data Path Card, are shown in figures 4.2-7 and 4.2-8, respectively. Also shown in each of figures 4.2-7 and 4.2-8 is a subevent for failure of a miscellaneous part, such as the circuit board, the edge connector, or other part which is not included in one of the modules named in the other blocks.

Theoretically, the fault tree analysis of the failure of the processor to compute the rudder command can be continued below the module level to the individual IC pins or discrete piece-parts. The desirability of doing this is questionable, however, because of the nature of the processor. The processor is not designed to perform a single specific function, such as computing rudder commands. It is designed to efficiently perform a number of simple functions, such as addition, multiplication, and logic operations. A suitable sequence of such operations (i.e., the flight software) is used to make the processor generate the rudder command, the aileron command, and so forth. It is much easier to relate the modules and ICs to the simple functions (addition, multiplication) than to relate them to the more complicated functions of computing the command for a particular servo.

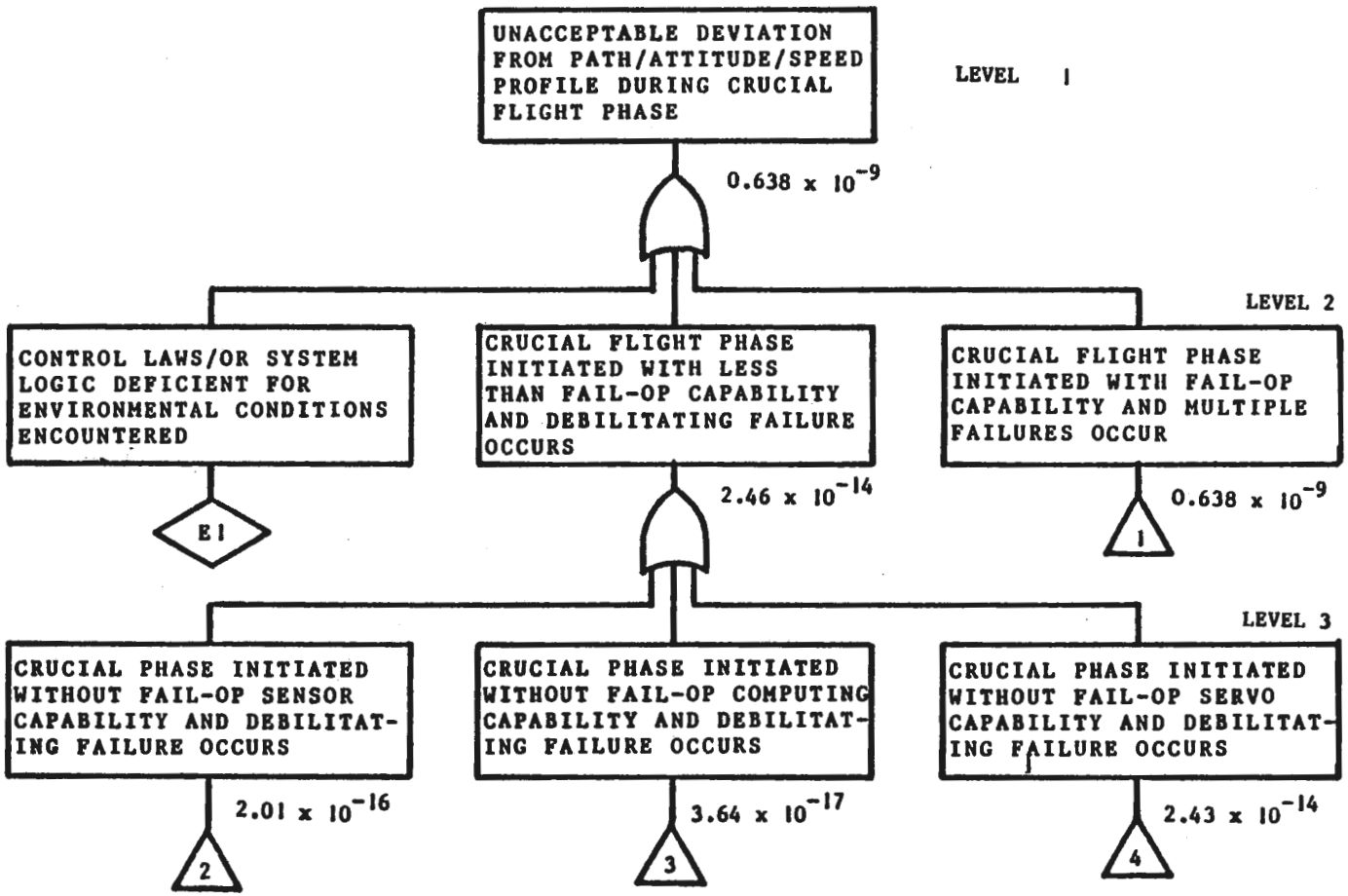


FIGURE 4.2-1. FAULT TREE TOP LEVEL

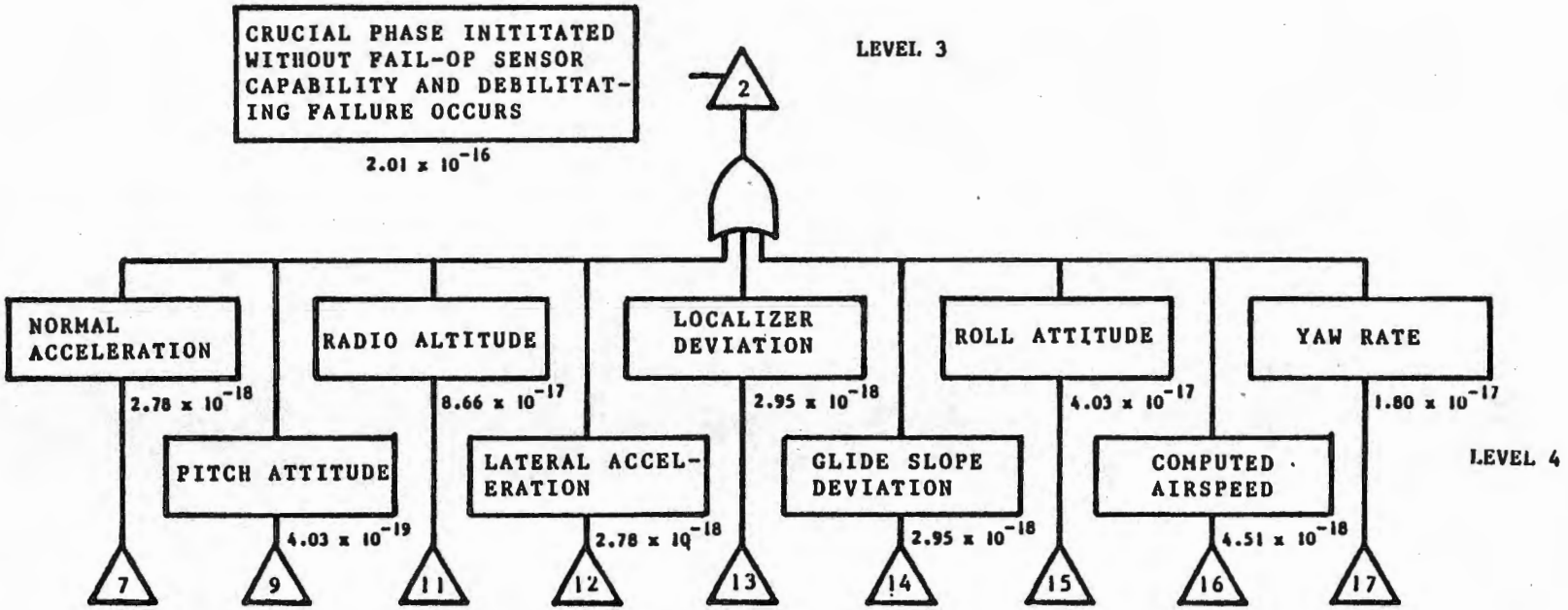


FIGURE 4.2-2. SENSING FUNCTION

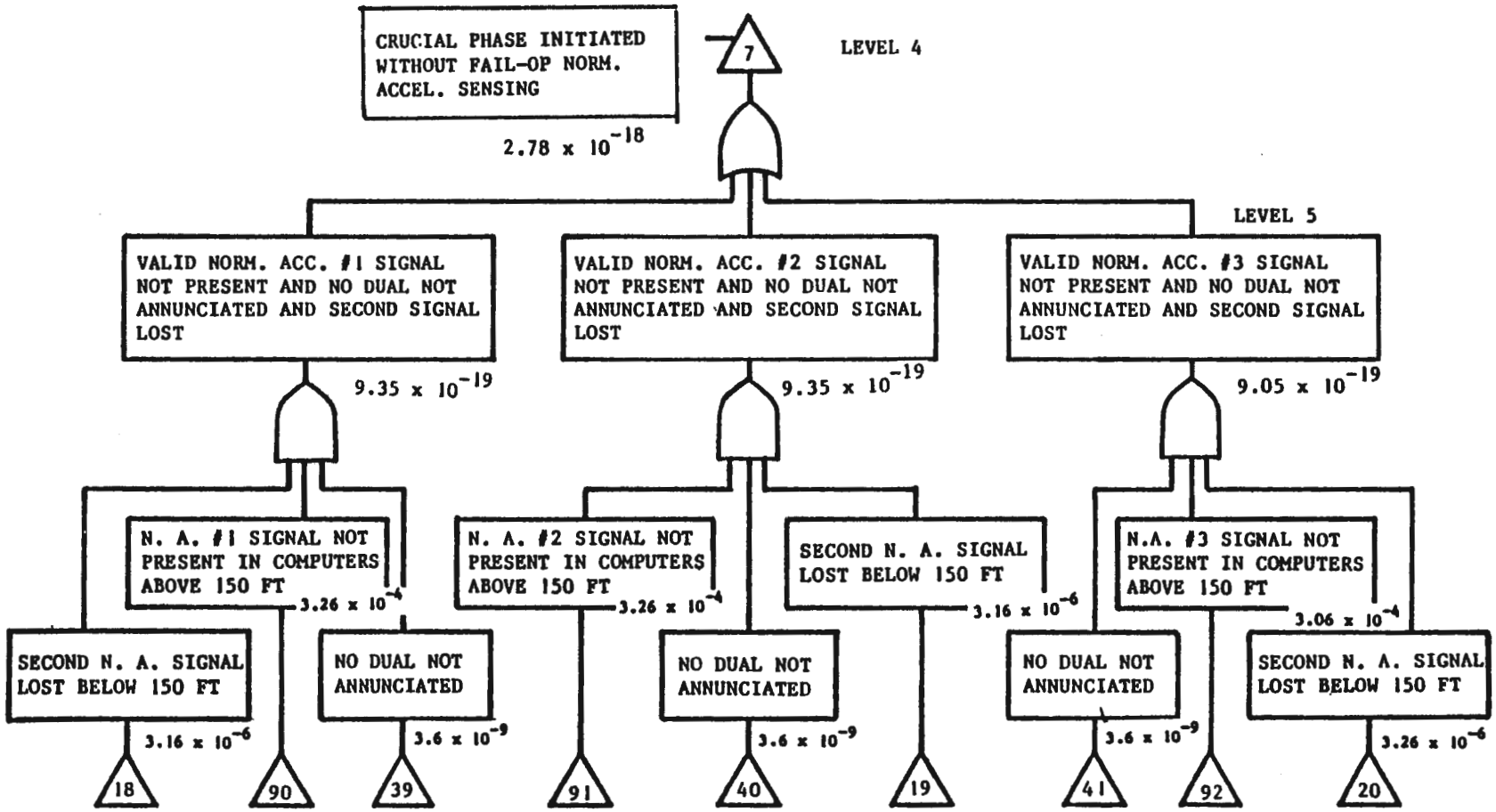


FIGURE 4.2-3. NORMAL ACCELERATION SENSING (1 of 3)

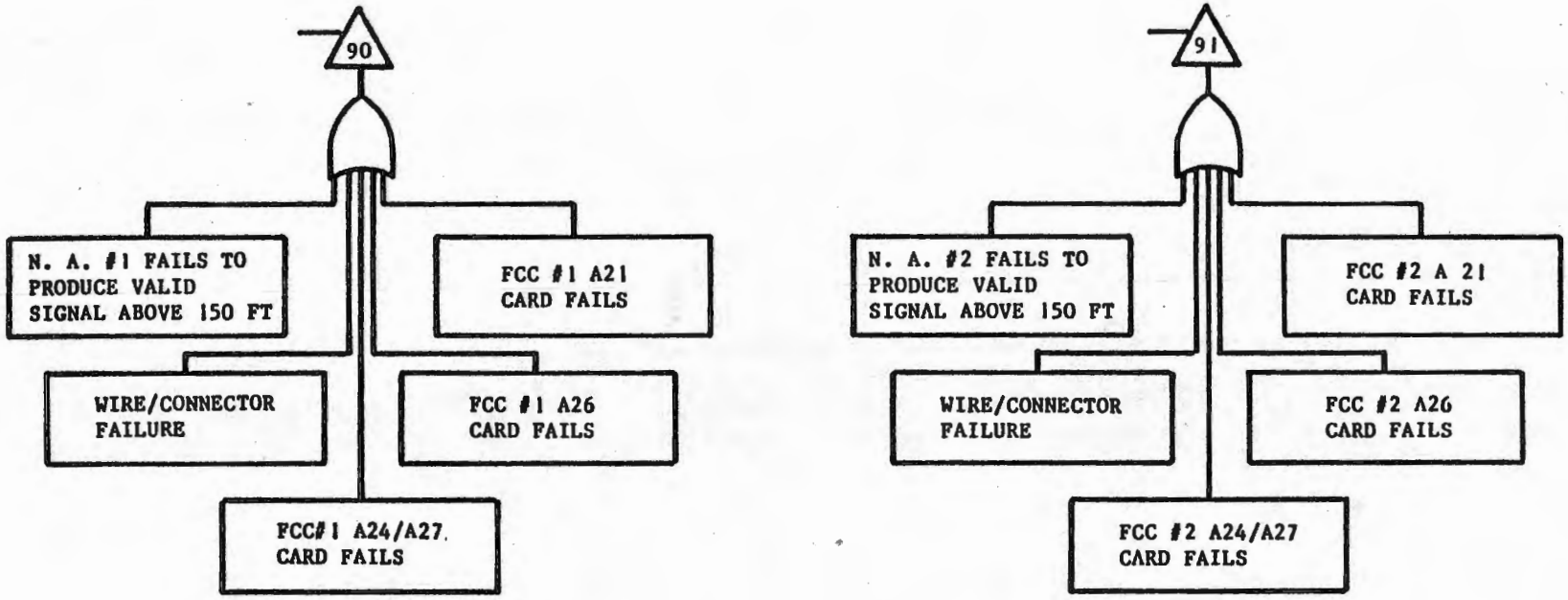


FIGURE 4.2-3. NORMAL ACCELERATION SENSING (2 of 3)

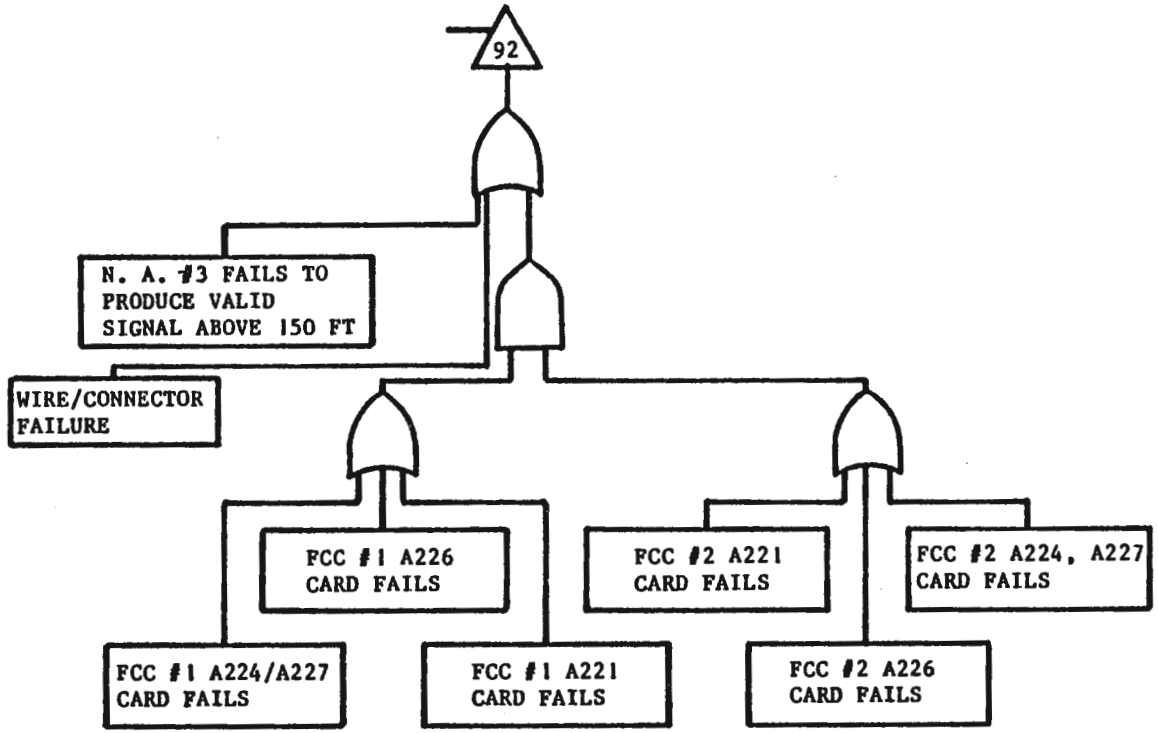


FIGURE 4.2-3. NORMAL ACCELERATION SENSING (3 of 3)

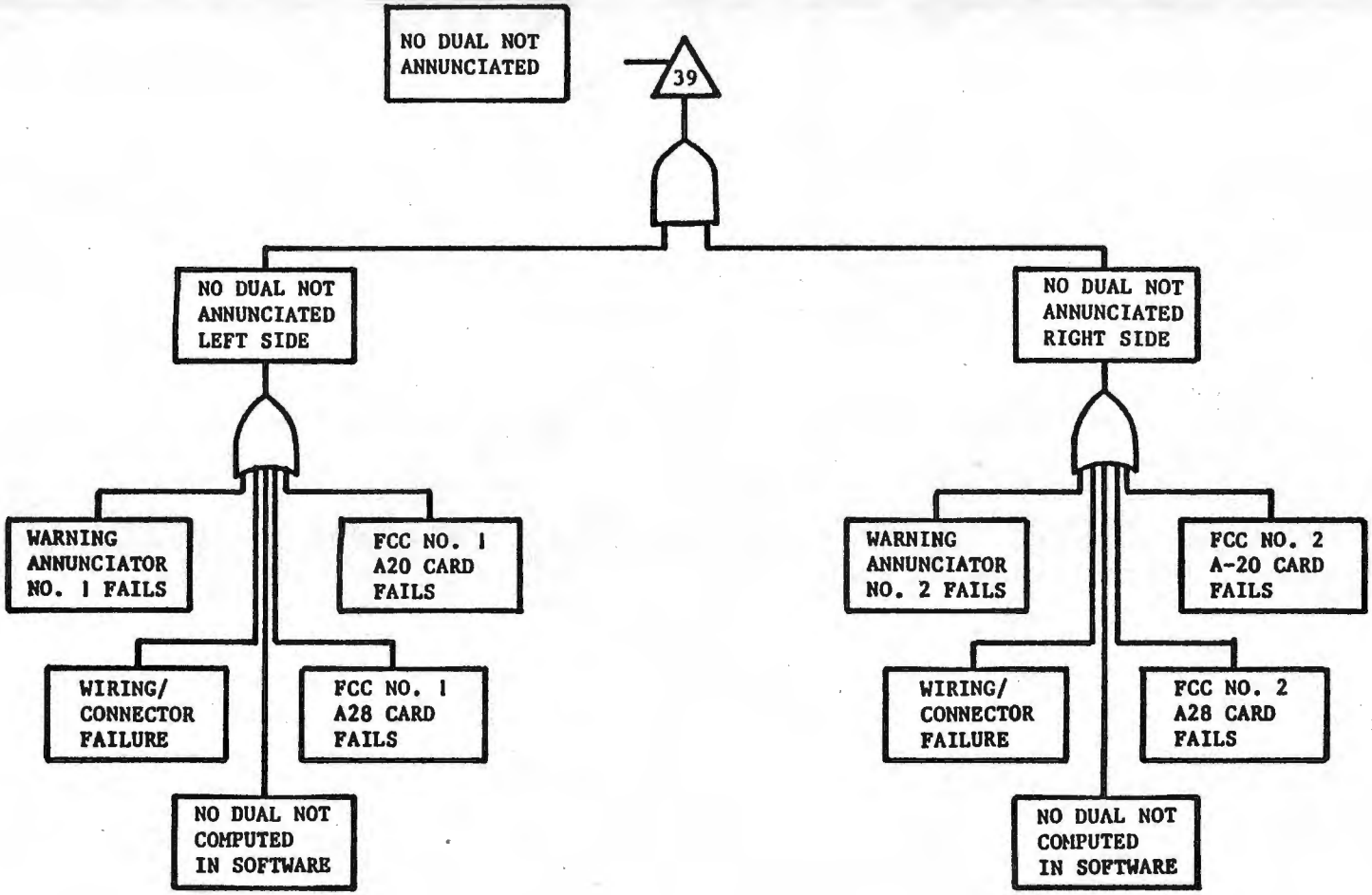


FIGURE 4.2-4. NO DUAL ANNUNCIATION

It is also easier, in general, to relate a specific failure mode of an IC within the processor to its effect on the processor operation than to start with the effect and then work in the other direction to the IC failure modes which would produce the effect. In other words, it is easier to do an FMEA than a fault tree analysis at this level. FMEA will be discussed in section 5.

Another reason for preferring FMEA to fault trees at this level is that in the course of performing the fault tree analysis, the analyst must account for all of the ways the processor can fail; that is, all of the ways in which the processor output can be wrong.

These ways are the failure modes of the processor. Each of these modes must then be traced to all possible combinations of IC pin failures which could produce the processor failure mode. Because processors have many different possible outputs, there are a high number of ways that the output could be wrong. There is no practical way of assuring that all of these possibilities have actually been covered in the fault tree. The FMEA requires that all pin-level IC failure modes be considered. These modes are much better understood, and there are fewer of them, so that it is much easier to be certain that they have all been covered. This is not meant to imply that a complete pin-level FMEA is easy or inexpensive; it is neither.

Due to the above considerations, the fault tree analysis of the processor was not continued below the level developed in figures 4.2-7 and 4.2-8. Instead, the FMEA approach was used as described in section 5.

To continue with the development of other branches of the fault tree, figure 4.2-9 develops the event of figure 4.2-6 in which the pilot is not warned that FCC No. 1 A channel is not generating a correct rudder command. This portion of the fault tree includes several software functions. In a production program, the test requirements of each of these functions should be reviewed to confirm that they satisfy the criteria of RTCA Document DO-178. In this tutorial, conducted for illustrative purposes, this review was not made.

Tree stubs similar to that developed in figures 4.2-6 through 4.2-9 were developed for the other required outputs from Channel A of FCC No. 1 and the other three channels (figure 4.2-5). They are not included here because they repeat the analysis shown.

The last of the Level-3 events of figure 4.2-1 is that the crucial phase is initiated without fail-operational servo capability, and a debilitating failure occurs. This is expanded in figure 4.2-10 into the three aircraft control axes: roll, pitch, and yaw. Figure 4.2-11 shows the fault tree for failure of the No. 1 yaw autopilot servo, with the servo failure not annunciated to the crew.

Fault tree stubs for the other five servos of figure 4.2-10 were developed to complete the analysis of the Level-3 events of figure 4.2-1. These are similar to the stub shown for the rudder servo and are not included in this discussion. This completes the discussion of the second of the Level-2 events of figure 4.2-1.

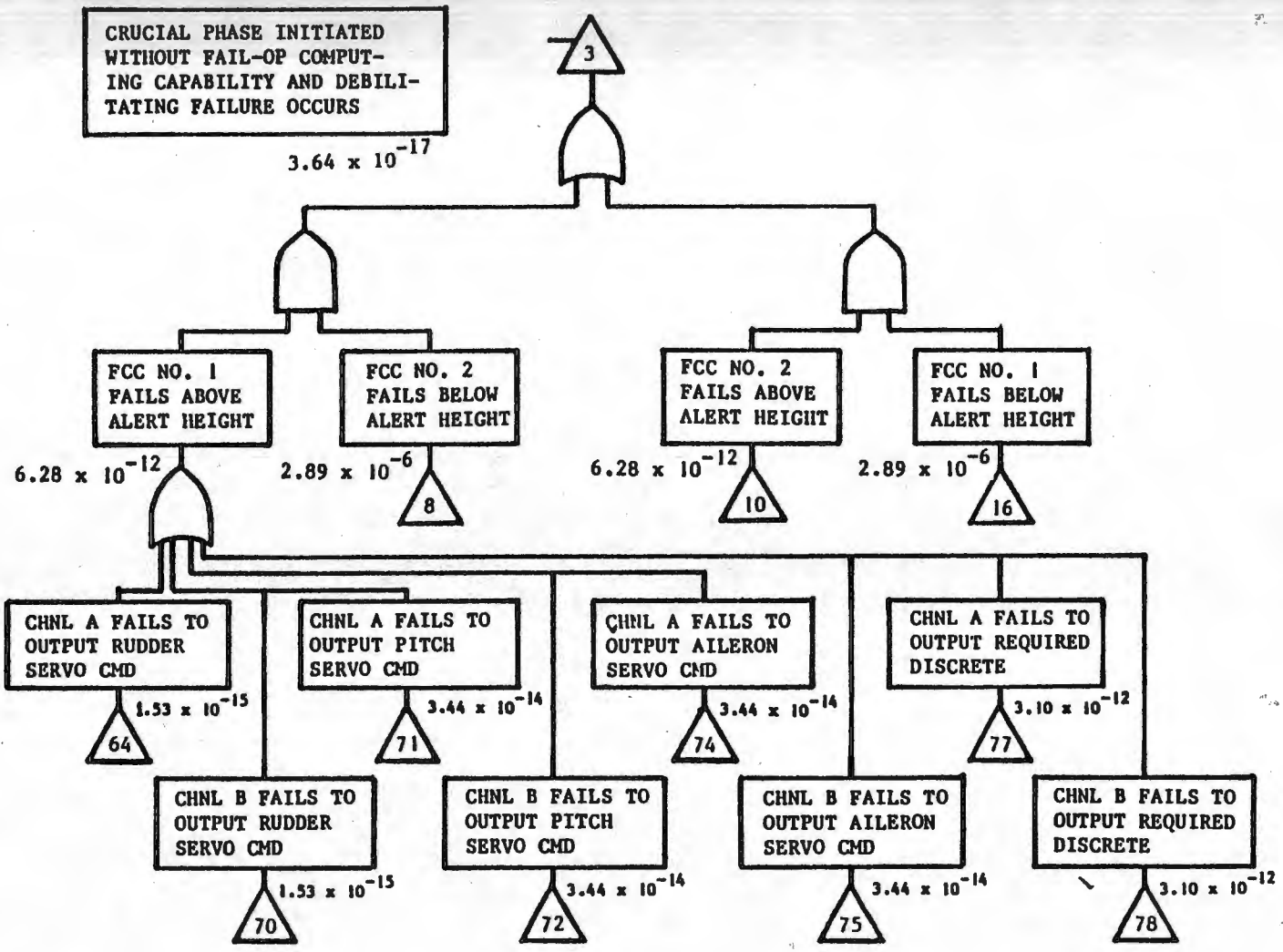


FIGURE 4.2-5. COMPUTING FUNCTION

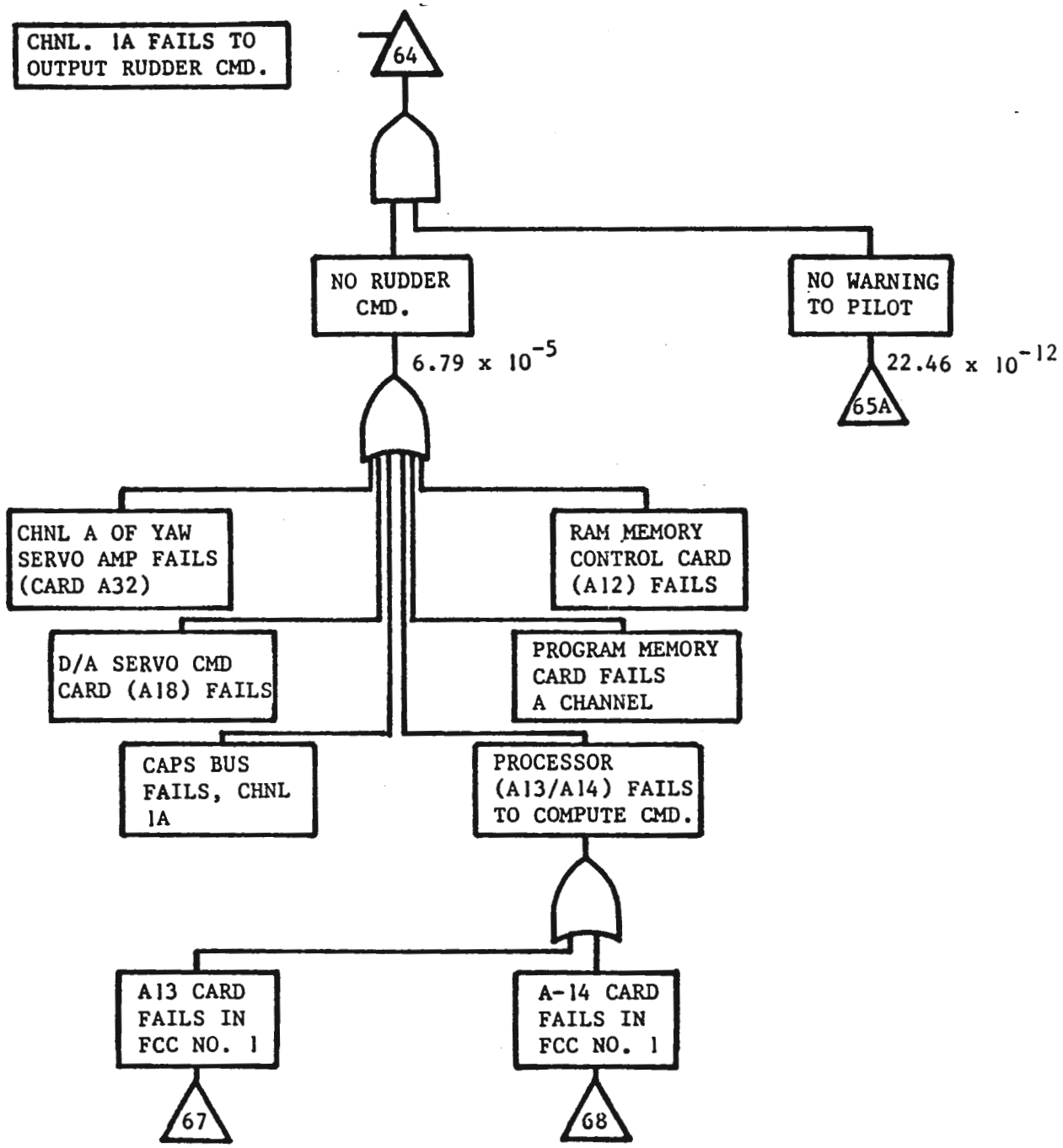


FIGURE 4.2-6. CHANNEL 1A RUDDER COMMAND

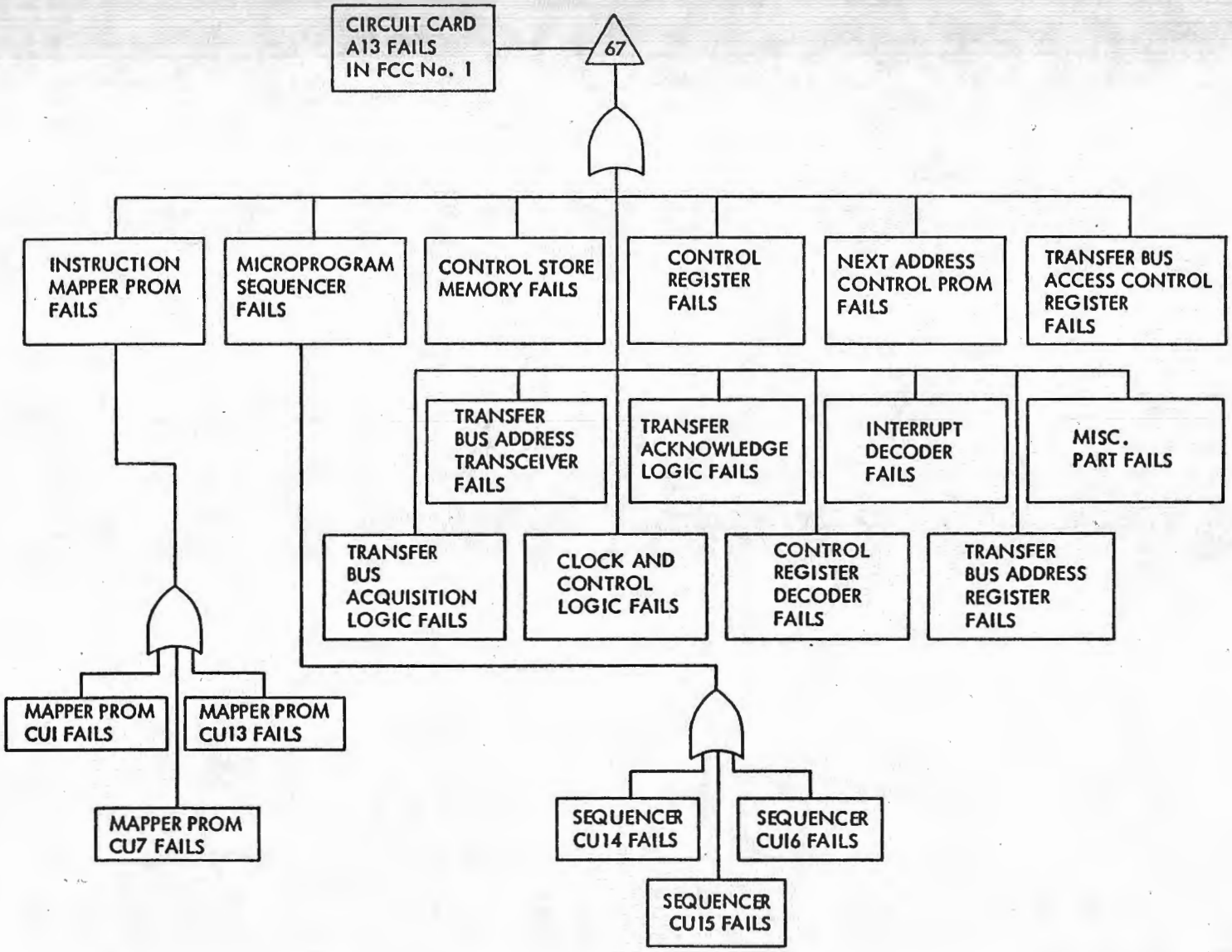


FIGURE 4.2-7. FCC PROCESSOR CONTROL CARD

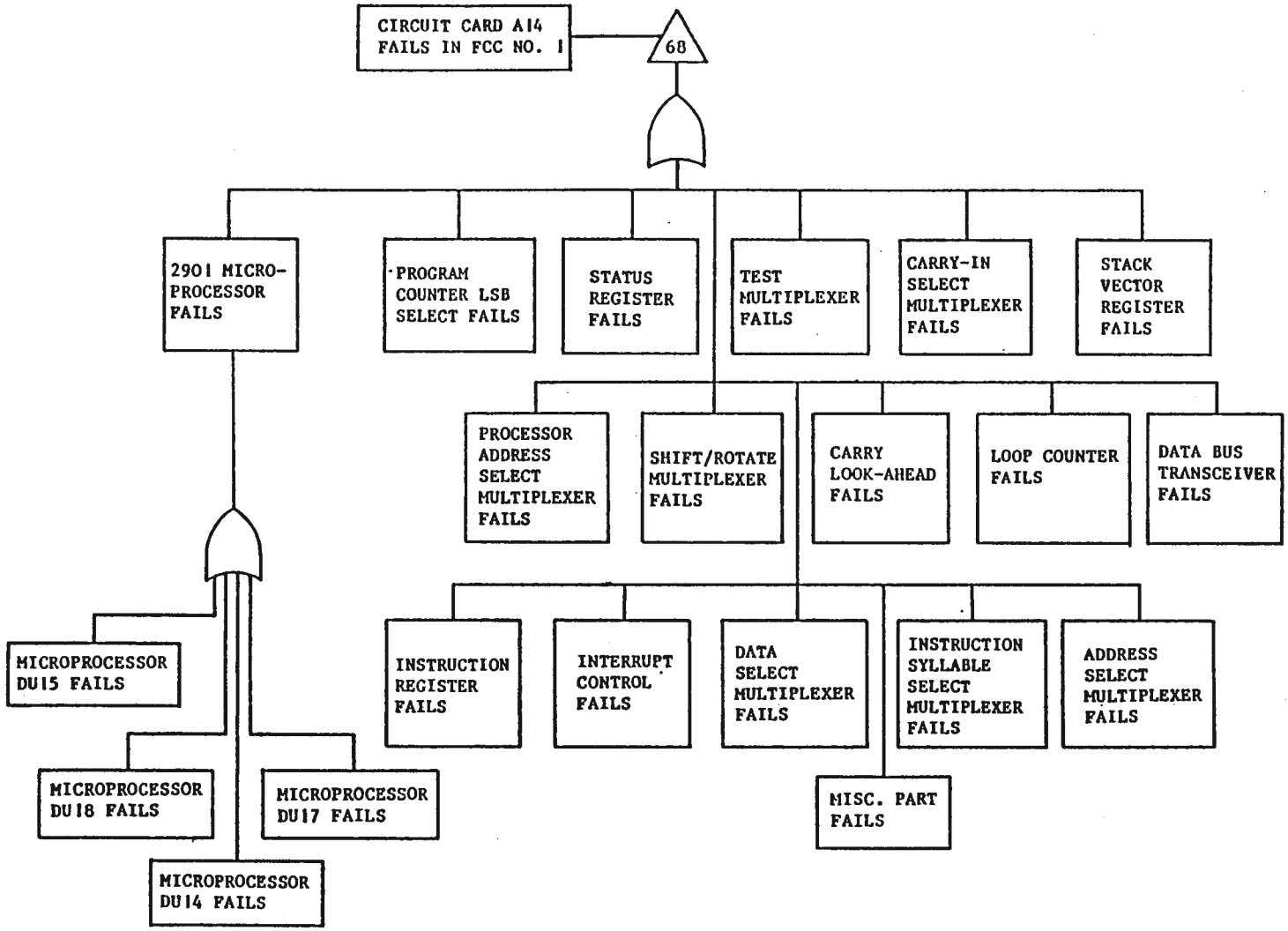


FIGURE 4.2-8. FCC PROCESSOR DATA PATH CARD

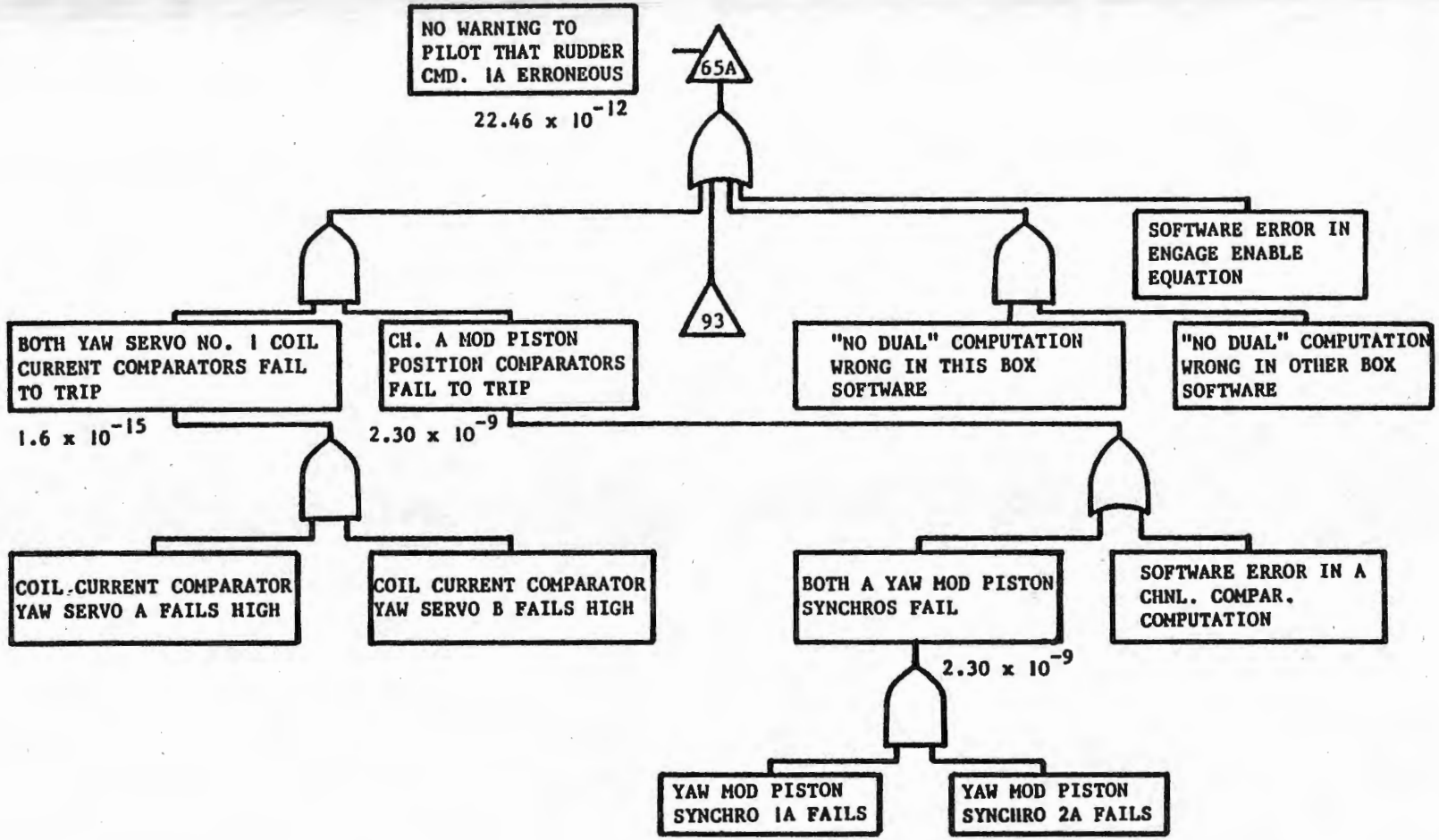


FIGURE 4.2-9. YAW AUTOPILOT SERVO COMMAND WARNING (1 of 2)

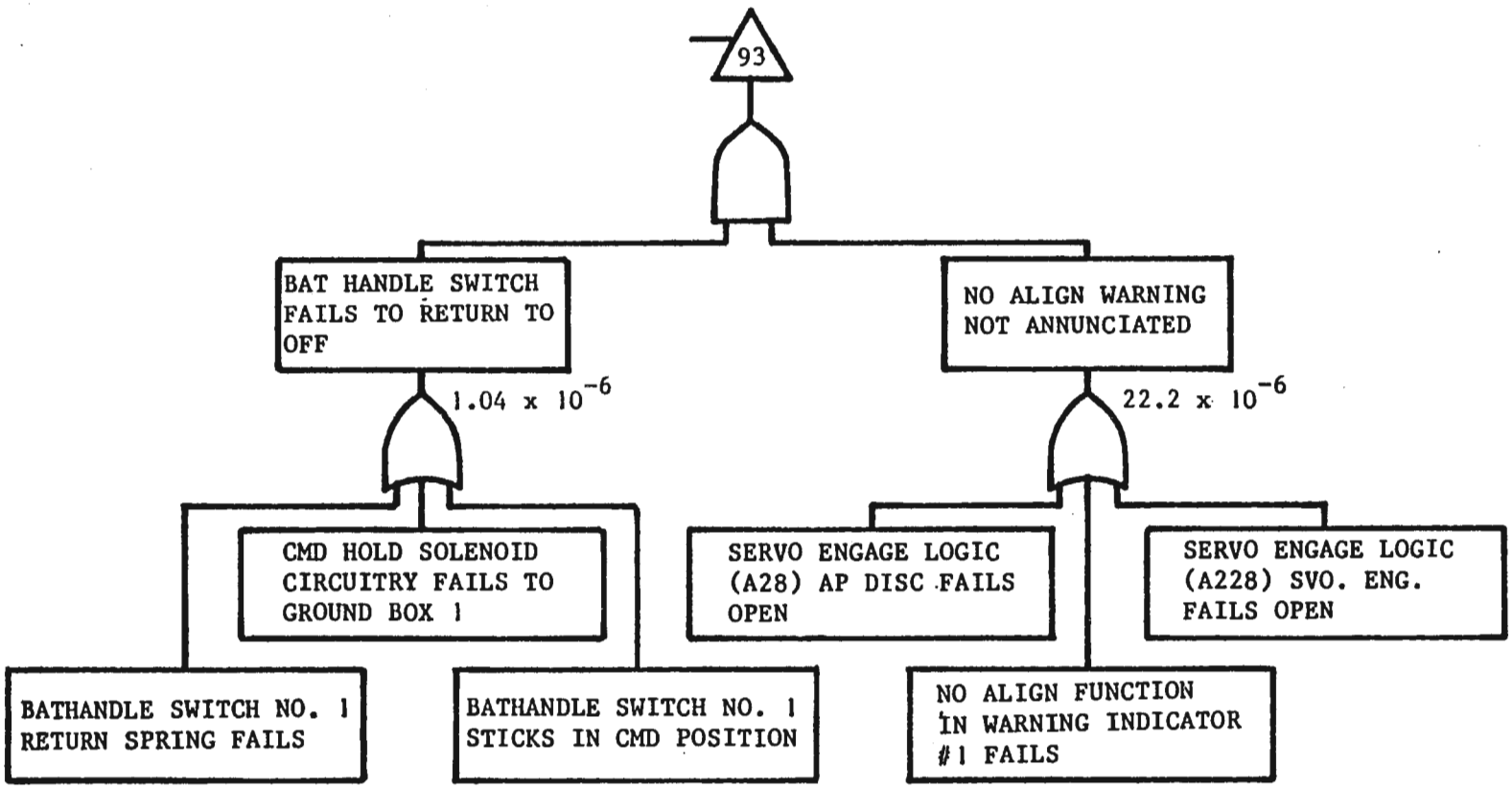


FIGURE 4.2-9. YAW AUTOPILOT SERVO COMMAND WARNING (2 of 2)

The third of the Level-2 events of figure 4.2-1 is that multiple failures occur during the crucial flight phase, and these occur in a combination which causes system failure. Figure 4.2-12 shows the initial development of this event to lower levels. Continuing this development produces a major branch of the fault tree, similar to but simpler than that for the second of the Level-2 events. It differs primarily in that the NO DUAL annunciation does not appear, because that particular warning is suppressed during the crucial phase. Since that major branch is similar to that already discussed, it is not described further here.

4.3. Quantitative Fault Tree Analysis

System failure probability was computed from the fault tree using the hardware failure rates presented in section 7. A failure rate of zero was used for each software function, since there is currently no acceptable way of predicting DFCS software failure rates (RTCA Document DO-178, section 2.2.1).

Considering hardware failure modes only, the probability of initiating the crucial phase with less than fail-operational capability and a second failure debilitating the system was calculated to be 2.46×10^{-14}. This is based on a flight time of 4.0 hours prior to the crucial phase and a crucial phase duration of 0.02 hours. The probability of the system failing because of multiple failures during the crucial phase was calculated to be 0.638×10^{-9}. This is based on a crucial phase duration of 0.02 hours. Table 4.3-1 compares the system failure probabilities obtained from the fault tree analysis with those obtained using the CARSRA reliability prediction program. The latter will be discussed in section 8.

The system failure probabilities computed are actually upper bounds on the actual failure probabilities. This is because the fault trees are based on the assumption, for many items, that all failure modes of the item render the item incapable of performing any of its functions. For example, certain buffers on the A26 Data Acquisition Card are used for sensor data which is not required for automatic landing. So at least some of the failures of these buffers would not prevent the card from correctly handling required data. However, the failure rates used in the analysis are for the entire card, including these buffers, so that the failure probability calculated for the card includes card failure modes which would not affect automatic landing.

The computation of the probability of system failure is accomplished by a process of "rolling up" from the lowest event levels. In computing the probability of an event's failure based on its subevents' failure probabilities, consideration must be given to several factors. Among these factors are the following:

- logical structure of the event: A OR B OR C, A AND B AND C, A OR B AND C, etc. (where A, B, and C are subevents)
- dependence or independence of subevents

TABLE 4.3-1. QUANTITATIVE RESULTS

Probability Of	Fault Tree Result	CARSRA Result
Unannunciated Failure in Cruise and Second Failure in Landing	2.46×10^{-14}	3.36×10^{-14}
Multiple Failures in Landing	0.64×10^{-9}	0.66×10^{-9}

The "rolling up" process continues from the lowest-level events through the top-level event. The actual number of computations can be large. Fault-tree algorithms use minimal cut sets (see section 4.2) to reduce the computational effort.

When dealing with these complexities, the approach to probability computation should always be conservative.

Poor event and subevent definition may result in excessive common failure modes within subevents. This results in having common failure modes simultaneously affect the operation of two or more separate systems, although these systems may otherwise be independent. Some common failure modes may be unavoidable. Examples of these are losses of electrical power, hydraulic power, or cooling. FMEA is useful in identifying common failure modes.

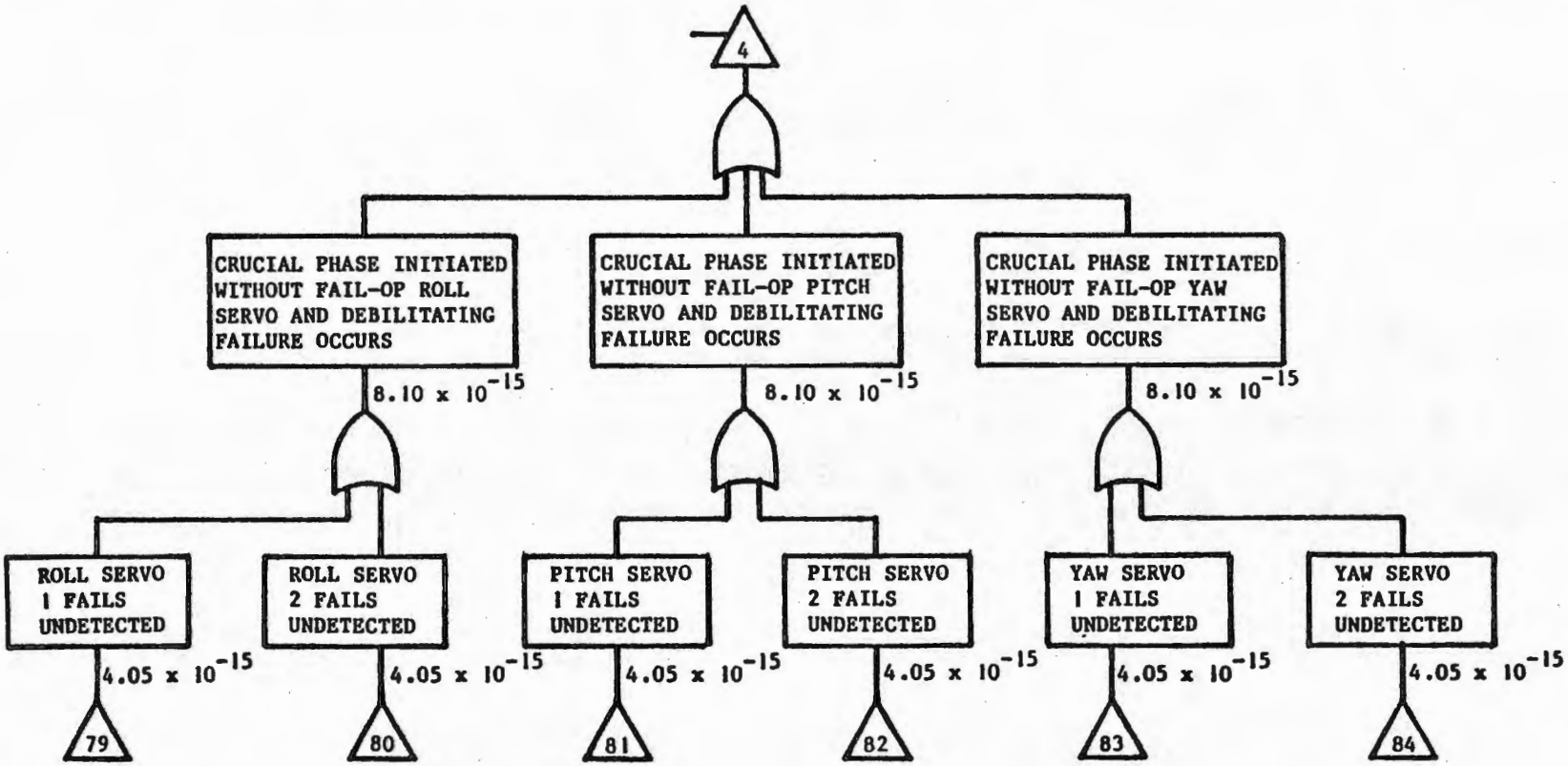


FIGURE 4.2-10. SERVO FUNCTIONS

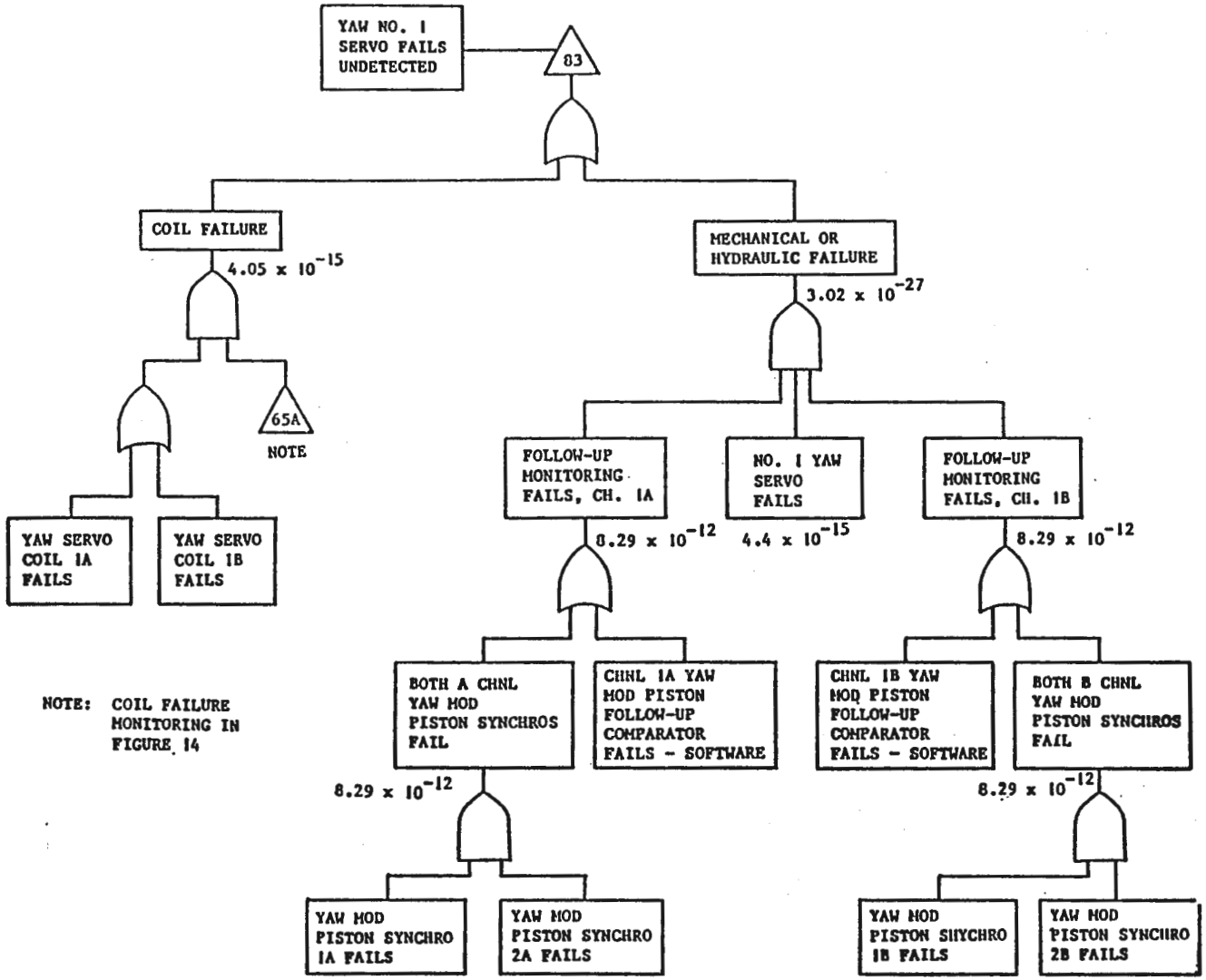


FIGURE 4.2-11. NO. 1 YAW AUTOPILOT SERVO

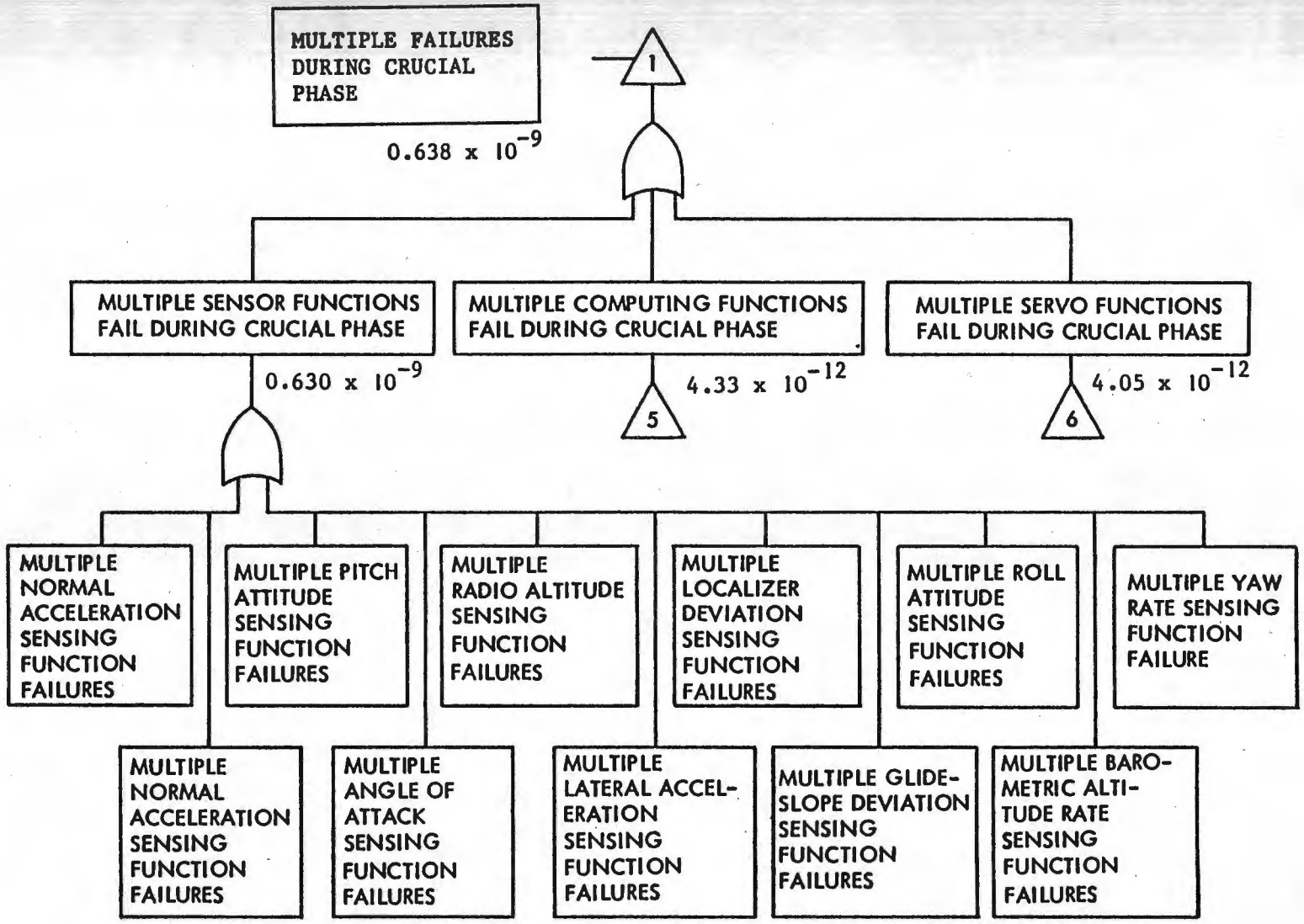


FIGURE 4.2-12. MULTIPLE FAILURES DURING CRUCIAL PHASE

5. FAILURE MODE AND EFFECT ANALYSIS

5.1. General Comments

The role of FMEA in IAA was specified in section 3.4. FMEA is a qualitative approach to be used in analyzing system failure. It is referenced in AC 25.1309-1 as a viable tool for both independent failure analysis and as an adjunct to a top-down tool, such as fault tree analysis.

AC 25.1309-1 defines FMEA as "an inductive bottom-up analysis which determines what happens to a system upon single failures of its individual parts." If the item being analyzed is a piece-part of a larger system, these identified failure modes will become the bottom events of a top-down analysis, such as fault tree analysis.

When using FMEA, all potential failure modes of a piece-part are identified. Each mode is then studied, and its impact on system performance is determined. This may be accomplished either empirically or by studying logic diagrams. The empirical approach may involve a technique such as fault insertion. Probabilities must be assigned for each of the failure modes. The development of these failure probabilities may involve some of the techniques discussed in section 7. MIL-STD-1629A is the best guidance document for FMEA.

The preceding paragraph defines the context in which FMEA was applied to RDFCS. Figures 4.2-7 and 4.2-8 depicted the subevents for two events related to the failure of Circuit Cards A13 and A14 in FCC #1. A number of these subevents dealt with microprocessor failure. Since the processor is a general purpose device, continuing the fault tree analysis, which is function oriented, was not desirable. An FMEA was performed instead.

In conducting the pin-level FMEA analysis of a processor, three factors were found to greatly reduce the effort. The first factor is that propagation of most faults under all conditions does not have to be considered. A single effect can usually be found which will totally debilitate the processor. For example, a faulted processor output pin will result in the processor trying to read about half of the data and machine level instructions from the wrong memory addresses. This will result in the coil current comparators tripping, sensor comparisons failing, and in the case of the RDFCS, the iteration monitor failing. In a system using checksums to monitor program memory integrity, these tests will fail.

The second factor which reduces the effort is that many pairs of faults will have the same effect. There are numerous instances of an output pin on one IC being connected only to one other pin. If either pin fails to open, the effect will be the same. Similarly, a ground fault in either pin will produce the same effect.

The third factor which reduces the effort is that there are many instances in which three pins are connected so that one output pin drives two input pins on different circuits. An open fault at each of the input pins can be evaluated first. An open fault at the output pin is then equivalent to both input pins failing open simultaneously. In most cases the effect is the "sum" of the effects of the input pins failing to open; that is, both effects occur. If both input pins are on the same chip, the effect of both being open is more likely to differ from the sum of the individual effects. See figure 5.1-1.

The effect of any of the three pins failing shorted-to-ground is the same in either of the two cases of figure 5.1-1.

Another frequently encountered condition involving three pins is two outputs connected to a single input (figure 5.1-2). In such a case, chips A and B will have three-state outputs, and one or both outputs should be in the high-impedance state at all times. An open fault on the output pin of chip A will then only affect chip C when A has its output enabled. Similarly, an open fault on the output pin of chip B will only affect chip C when B has its output enabled. An open fault on the chip C input pin will usually produce the sum of the effects of open faults on the two output pins. A ground fault on any of the three pins will have the same effect.

Still referring to figure 5.1-2, if a fault should occur, which results in both enable pins being in the enable state, there is a possibility of damage to the A or B chip. If one output is high and the other low, there could be a low impedence path to ground, through the output pins, which could burn out the A or B chip. This depends on the technology used in the individual chips. Frequently, the effect of the original ground fault can be judged to be a total processor failure whether or not the secondary damage occurs.

5.2. Application of FMEA to RDFCS

In the RDFCS analysis, three modules of the processor (figure 5.2-1) were considered at pin level.

- The instruction mapper Programmable Read-Only Memory (PROM), which consists of three PROM chips in parallel.
- The microprogram sequencer, which consists of three 2911 sequencer chips in parallel.
- The microprocessor module, which consists of four 2901A chips in parallel.

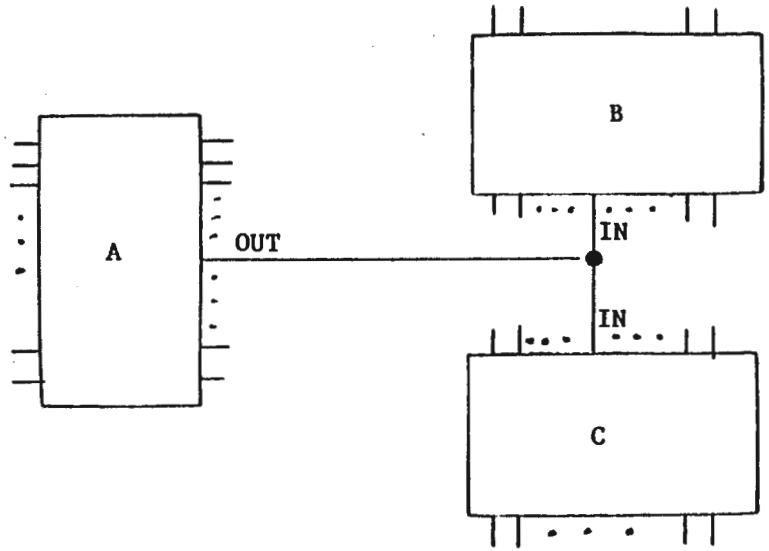
The instruction mapper PROM chips are read-only memory chips. There are two inputs to the chip: machine-level operation codes, and the depth of the stack maintained in the 2901 microprocessors. These are connected to the address pins of the mapper. The data stored in the PROM is the control store PROM address of the first microcode instruction required to execute the machine level instruction with the processor stack at a particular depth. The mapper output pins are only active at the beginning of a microcode sequence, at which time a chip enable signal is sent to the mapper from the next address control PROM.

The microcode address from the mapper PROM is routed to the microprogram sequencer module. This module generates a sequence of microcode addresses, beginning with the starting address from the mapper PROM. Some microcode routines involve jumps to a new address rather than sequential progression only. In such cases, the microprogram sequencer receives the jump address from the control store PROMs and resumes sequential generation of addresses.

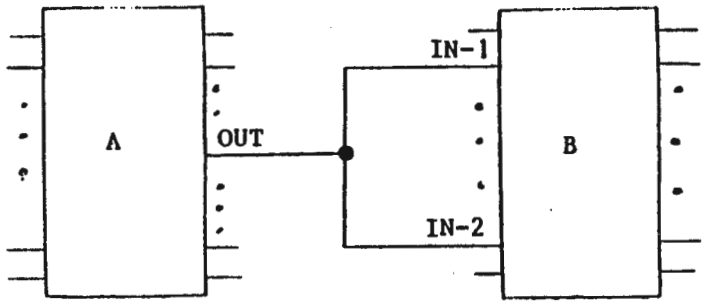
The microprocessor module is composed of four 2901A microprocessor chips. Each chip has a word size of four bits, so that the four chips in parallel are used to provide the processor 16-bit word size. This requires that carry signals be passed between 2901As during arithmetic operations. Other interconnections between 2901As are used for data shift operations. The 2901As are controlled primarily by control signals from the control store PROMs in conjunction with the outputs from various registers.

The FMEA, summarized in table C-1 (appendix C), considered three types of pin-level faults: open, grounded, and shorted-to-supply voltage. In most cases, the effect of a fault can be assessed by using the chip logic diagrams, a description of chip/module functions, and the schematic diagrams. The schematic diagrams are reproduced in appendix D.

The effect of certain pin faults cannot be determined by analysis using just the above information. In particular, the contents of specific PROM addresses was required or the machine-level code was needed along with the microcode sequences and addresses. Alternatively, the faults could be inserted and the effects observed. The Fault Insertion Approach was taken in this analysis. The results are presented in section 6. For example, it was known that failure of one of the processor pins, used in data shifts (R0, R3, Q0, or Q3 stuck high or low), would result in an immediate disconnect if certain integer words made up of packed Boolean variables were shifted. It was determined from the available information that such shifts might occur, but it was not definite. Similarly, if certain fixed-point numbers were shifted during computation, the commands to the servos would be in error, and the coil current comparators would trip. While both left and right shifts are normally used in multiplication algorithms, it could not be determined that a stuck shift bit would definitely cause such a trip. When the faults were actually inserted, the processor stopped immediately. ("Immediately," as viewed by the human observers.) In this way, fault insertion confirmed the overall effect, a massive processor failure and disengagement of the servos. The exact mechanism by which it occurred was not determined.

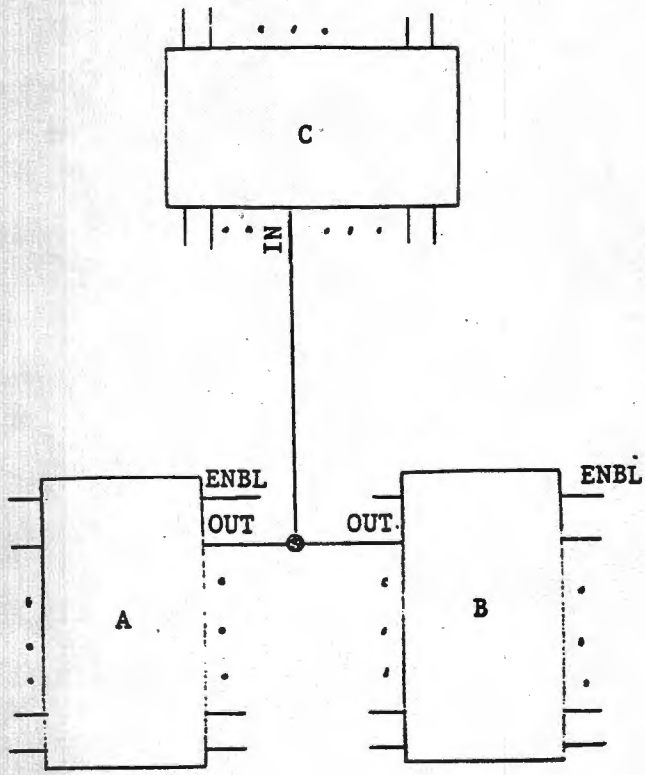


EFFECT OF OUTPUT OPEN ON "A" USUALLY SAME AS EFFECT OF INPUT ON "B" OPEN PLUS EFFECT OF INPUT ON "C" OPEN.



EFFECT OF OUTPUT OPEN ON "A" OFTEN NOT SAME AS EFFECT OF IN-1 PLUS EFFECT OF IN-2.

FIGURE 5.1-1. ONE OUTPUT, TWO INPUT CONDITIONS



BOTH "A" AND "B" ENABLED SIMULTANEOUSLY MAY DAMAGE CHIP.

FIGURE 5.1-2. TWO OUTPUT, ONE INPUT CONDITION

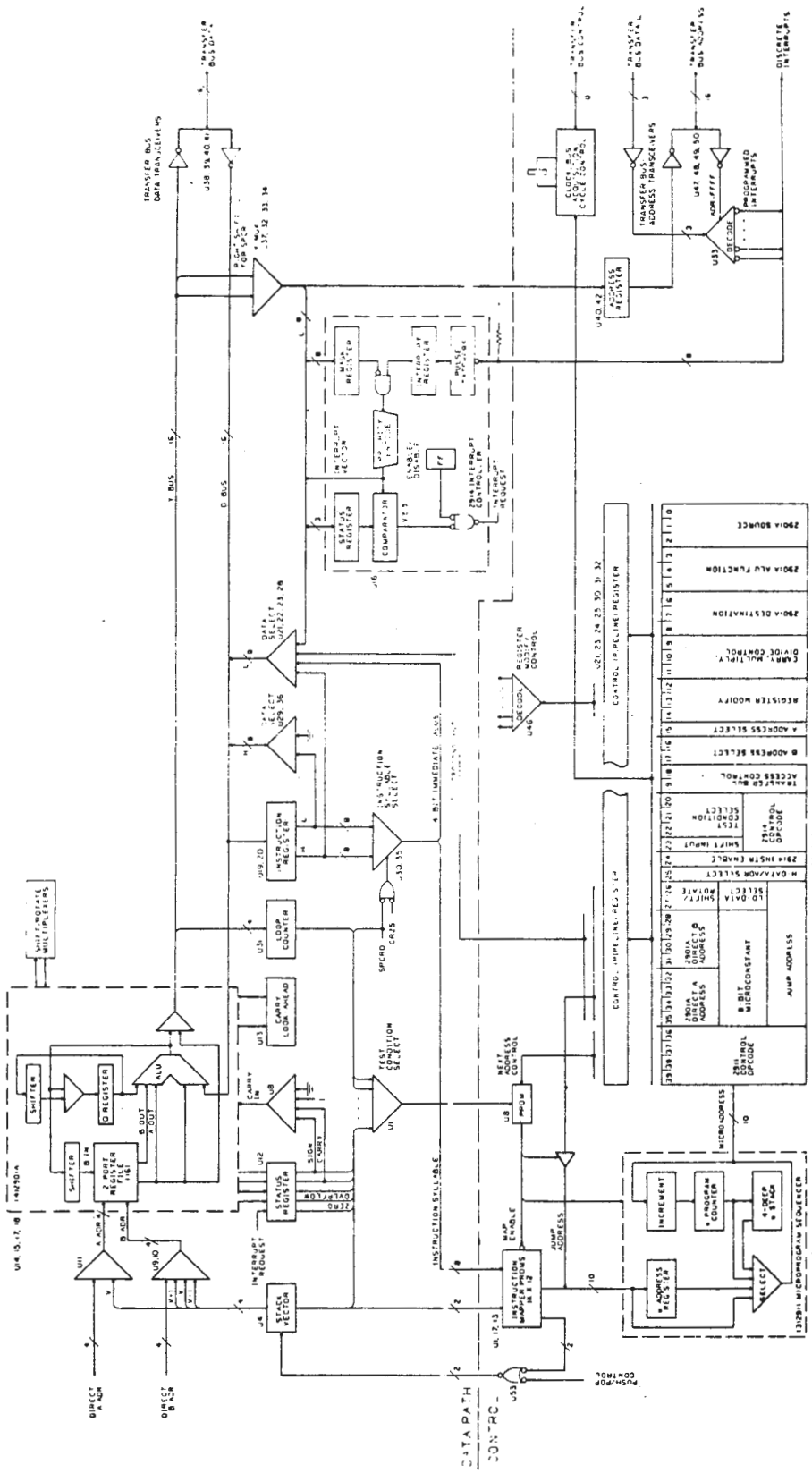


FIGURE 5.2-1. PROCESSOR BLOCK DIAGRAM

6. FAULT INSERTION

6.1. General Comments

The role of fault insertion in IAA was specified in section 3.5. This technique is required whenever the effect of certain pin faults cannot be determined analytically. The effect of the fault is determined by observing it. Fault insertion may also be used to confirm the effect of faults diagnosed analytically. Another use of fault insertion is to verify fault detection and annunciation systems. In the RDFCS analysis the technique was used for all three purposes.

6.2. Application of Fault Insertion to RDFCS

The RDFCS simulator at NASA-Ames was used to insert the faults shown in table E-1 (appendix E). The faults were of two general types: component level faults and IC pin faults. The component level faults were inserted using the FCC breakout panels (figure 6.2-1), the SSP (figure 6.2-2), and the MDICU. Single-sensor faults are those numbered 1 through 19 in table E-1 (appendix E).

Faults representing a dead sensor or a broken wire from the sensor to the FCC were inserted by pulling the appropriate jumper plug at the break-out panel. Faults representing missing sensor validity discretes were also inserted in this way, although they can also be inserted via the DSP (figure 6.2-3). Sensor hardovers and ramps were inserted using the MDICU. Servo faults were inserted using the SSP.

For monitoring the processor detection of sensor faults, the CTAs were used. One of the CTA address windows was set to the address of the Executive Failure (Status) Word (EFW) in each computer channel. The EFW is a 16-bit word with each bit representing a discrete piece of information. There is one EFW for each sensor type in each computer channel. The four low-order bits (0-3) represent, respectively, Executive Failure of My A (EFMA), My B (EFMB), Other A (EFOA), and Other B (EFOB) sensor signals. The data window of the CTA shows the status of the EFW as four hexadecimal characters, with the right-most character representing the bits of interest, 0-3.

The effect of a sensor signal being detected as bad by the software sensor monitor is that certain bits are changed from 0 to 1. With no failures detected, EFMA, EFMB, EFOA, and EFOB are all 0, which is represented in hexadecimal notation as 0. (0000 binary = 0 hexadecimal). When the number 1 sensor of a triple sensor complement is detected to have failed, bit 0 (EFMA) is set to 1 in both channels of FCC No. 1. Bit 1 is also set to 1 so that the comparison monitoring will work properly on the two remaining sensors. The EFW low-order bits will then be 0011, which is 3 in hexa- decimal. The net effect, then, of the number 1 sensor of a triple-sensor set failing is that the value displayed in the CTA window changes from 0000 to 0003. The left three

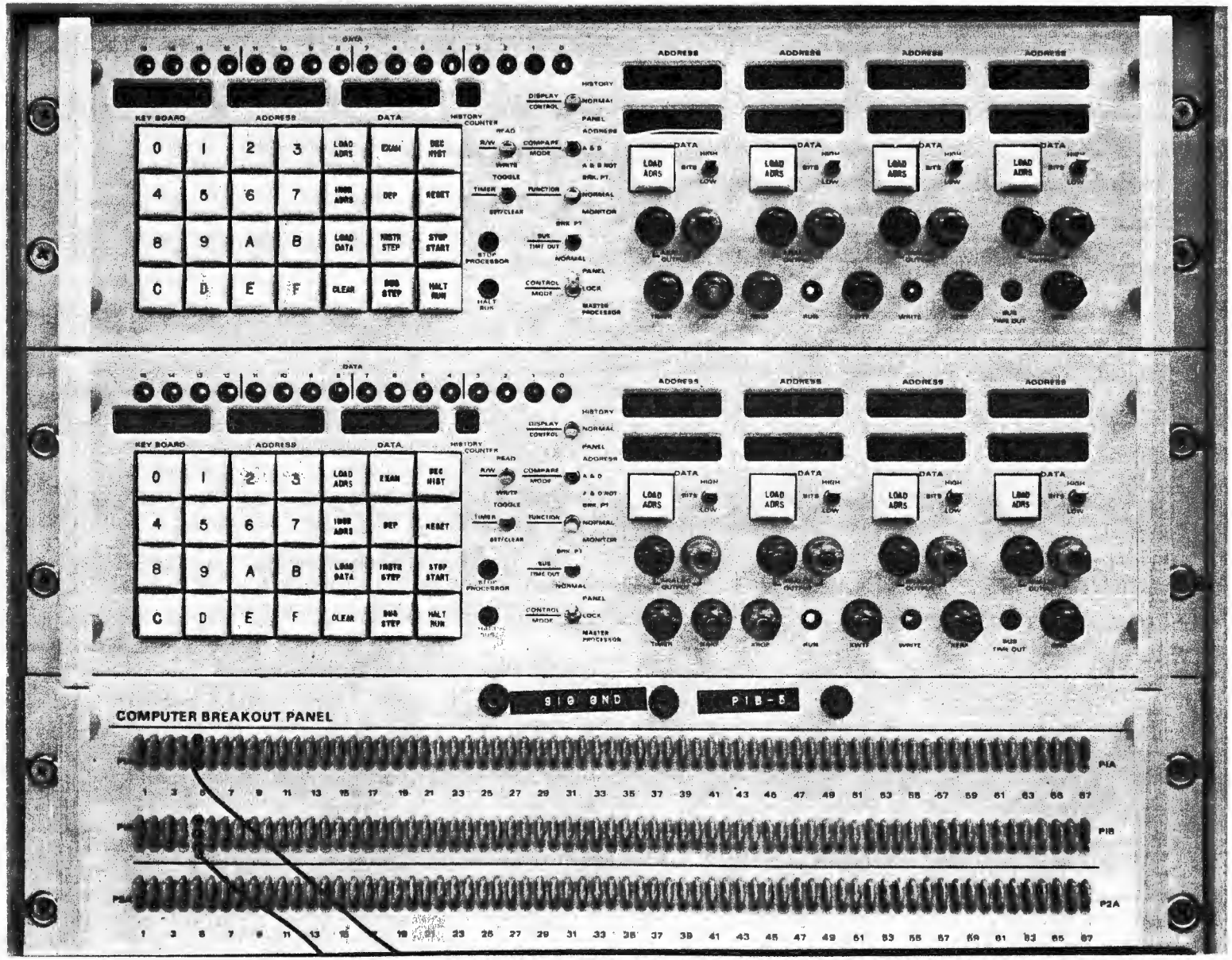


FIGURE 6.2-1. CTA AND COMPUTER BREAKOUT PANEL

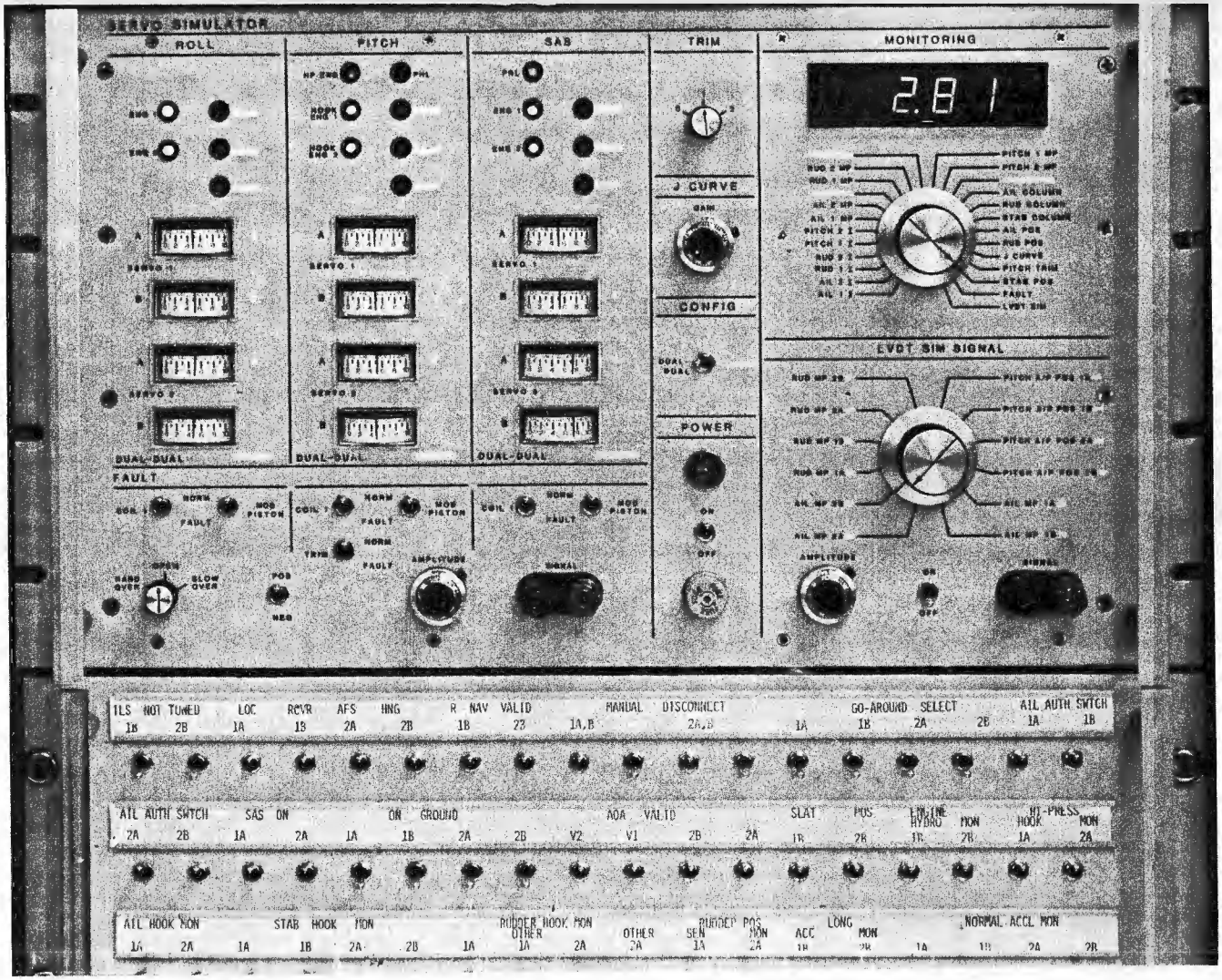


FIGURE 6.2-2. SERVO SIMULATOR PANEL

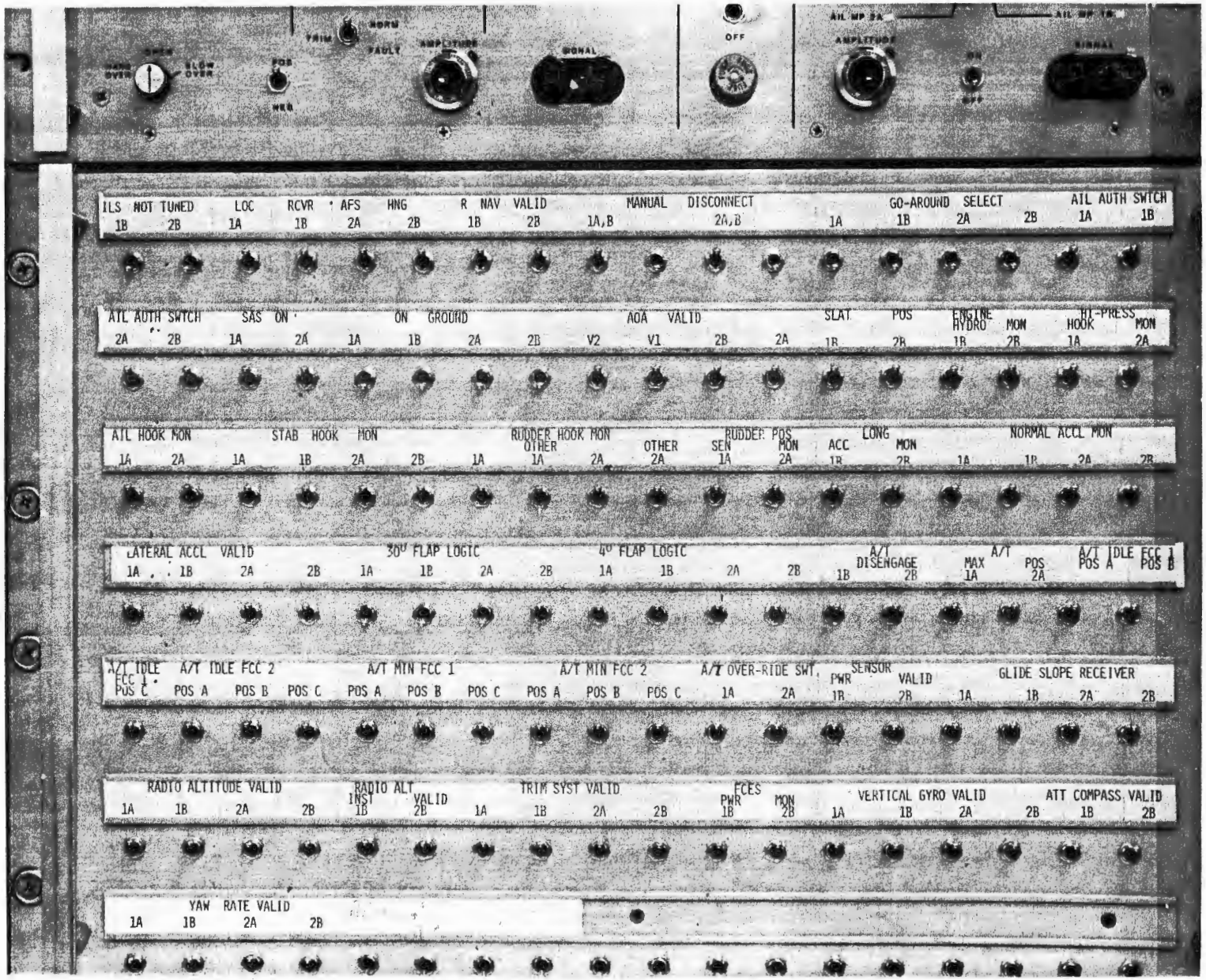


FIGURE 6.2-3. DISCRETE SWITCH PANEL

hexadecimal digits each remains at 0, since each of the corresponding binary bits (4-15) of the EFW remains at 0.

Fault cases 1 through 8 were used to show that the software sensor monitor subroutine is implemented correctly in the RDFCS by subjecting it to a number of different faults in the same sensor type. These cases were also used to show that the results of the sensor monitoring are accounted for in the implementation of the NO DUAL equation, which is also in software. Cases 9 through 16 were then used to show that the voter is involved for various sensor types. Rigorous validation of the system by testing would require that faults be inserted for all sensor types used in automatic landing. In this analysis, performed for illustrative purposes, the full complement of sensor types was not faulted.

In case 2E, NO DUAL did not annunciate even though the fault was inserted with the airplane inbound to the ILS beam intercept point. It is believed to be the result of the inbound leg being flown at an unrealistically low altitude, so that the airplane did not track the glideslope beam for 25 seconds before passing through a 150 ft altitude. A review of the NO DUAL annunciation logic shows that this is the most likely cause, since AP. ONEFAIL was set to true. Low approaches (1500 ft) were being simulated in the interest of time. Approach altitude was subsequently raised to 2000 ft.

Faults 17 through 19 were used to confirm the servo monitoring and the tie-in of the servo monitor outputs to the NO DUAL and disconnect logic. The servo monitors, in particular the coil current comparators, are quite important in ensuring that the airplane does not enter the crucial phase with a faulty computer or servo.

Fault cases 43 through 45 were used to confirm that the FCCs will both disengage upon loss of the second sensor, with the AP.DISC warning displayed, in accordance with the system description.

At the IC pin level, a number of open and ground faults were inserted to confirm the FMEA results of section 6. For this activity, one of the FCCs was removed from the pallet, and the card containing the chip to be faulted was extended for access as shown in figure 6.2-4.

Open pin faults, Cases 20 through 23, were inserted by using multiple sockets between the chip and the circuit card, with a jumper wire replacing the normal pin-to-socket connection. Each fault was inserted by physically pulling the jumper to open the connection. This is a slow procedure, since the chip must be removed and the jumper wire rigged on the desired pin.

The chip and sockets must then be installed and the processors brought back up. This method of inserting open pin faults is only marginally satisfactory. It would be much easier to do if a stack of five or six sockets could be used between the chip and the circuit card. However, the processor will not come up with more than three sockets stacked. The longer electrical paths resulting from the use of the extender card apparently come close to exhausting the avail-

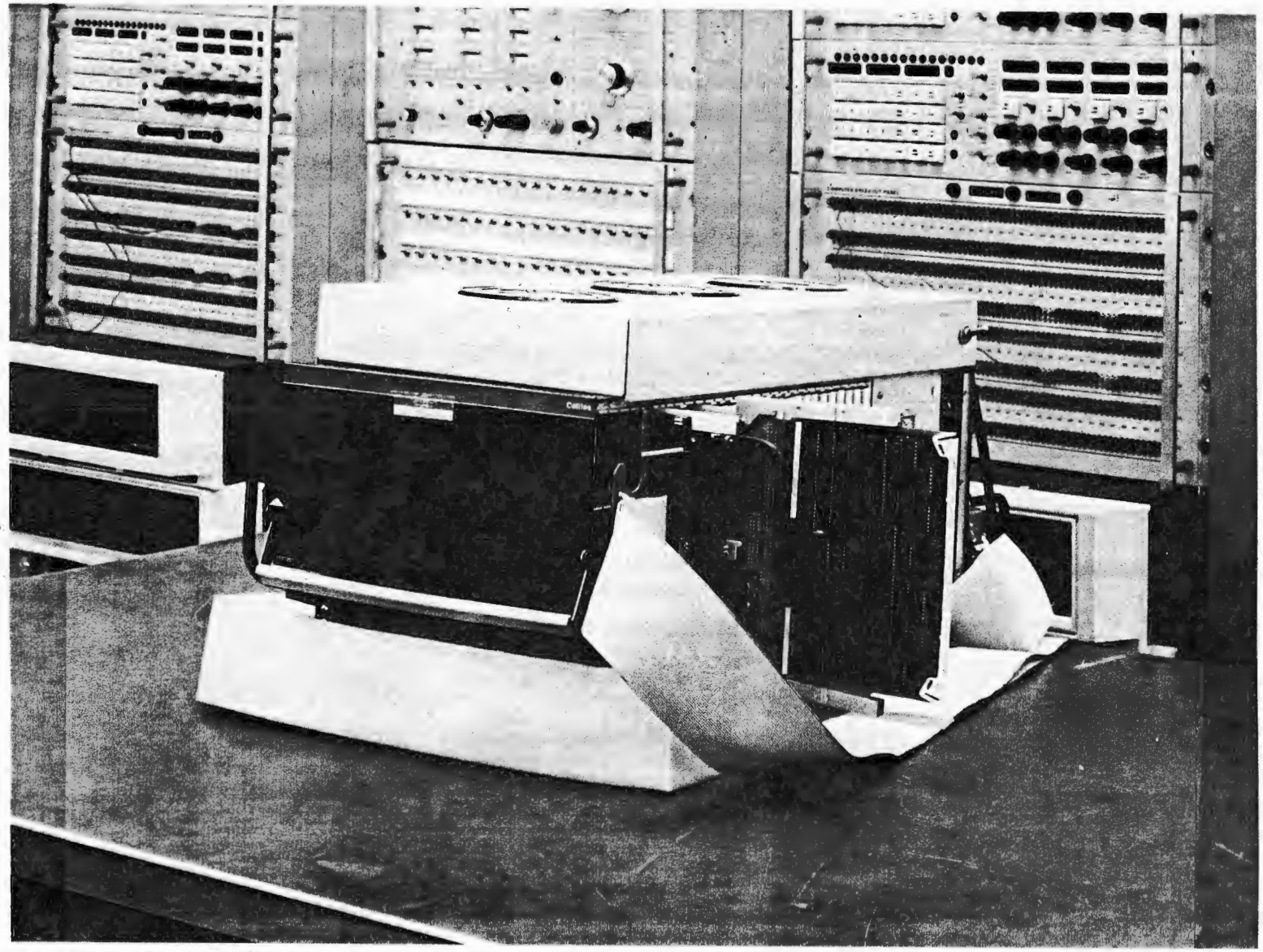


FIGURE 6.2-4. FCC WITH FLIGHT CARD EXTENDED

able tolerance in the timing of the individual microsteps, and the extra path length and capacitance caused by more than three sockets disables the processor.

Grounded pin faults are much easier to insert, since the chip does not have to be removed to set up each case. The processor does have to be brought back up each time, but this is a fairly rapid step. Before each fault was inserted, the data sheets from the chip manufacturer were reviewed (along with the card schematics) to determine that the fault would not damage any chips. No chips were damaged by the ground faults. The ground pin faults are cases 24 through 42 in table E-1.

The chip pin faults all disabled the processor, with the exception of open pin fault 21. This fault involves a pin of a quad 2-input NOR gate. The fault had no effect on the processor operation.

6.3. Fault Insertion Results

The faults inserted in the RDFCS simulator achieved the desired results in the assurance assessment of this analysis. More importantly, this confirmed that fault insertion is capable of providing the results required in the IAA. Specifically, the faults inserted confirmed (1) that the NO DUAL warning appears when it should, (2) that all sensor types faulted and required for automatic landing are monitored, (3) that the servo monitoring functions correctly, (4) that the effect of pin-level faults in the processor is in agreement with the FMEA, and (5) that fault insertion is a reasonable way of resolving uncertainty of the effect of open and grounded pin faults in digital hardware. While these results were obtained on a particular system, the approach is judged to be viable for validating other digital systems.

7. FAILURE RATE DEVELOPMENT

7.1. General Comments

While the development of failure rates is not a safety analysis technique, it is a task that is fundamental to the generation of system probability of failure. In order that fault tree analysis or a statistical prediction tool be able to produce a measure of reliability, individual failure rates must be known. This section discusses the different sources that were used for obtaining failure rates for the RDFCS analysis. The event trees used in the fault tree analysis in section 4 reference these failure rates; they were used in the failure probability computation. Likewise, the rates will be referenced in the reliability analysis found in section 8.

7.2. Rationale for Developing Failure Rates for RDFCS

The failure rates for servos, sensors, and indicators were taken from the database maintained by the Lockheed-Georgia Company Reliability Engineering Department. They are composite values for representative components of comparable complexity and construction.

The failure rates for the ICs of the Data Path and Control Cards were estimated using the formulas and tables of Military Standardization Handbook 217C. The formulas provide a means of accounting for a significant number of factors:

- Device technology
- Device complexity
- Junction temperature
- Package technology
- Application environment (voltage)
- Usage environment
- Quality level

For example, the equation for the failure rate of a monolithic bipolar device is:

$$f = K_0 \left[C_1 K_T K_V + (C_2 + C_3) K_E\right] K_L$$

Where:

f is the device failure rate

K_Q is the quality factor

K_T is the temperature adjustment factor for junctions

K_V is the voltage derating stress factor

K_E is the application environment factor

C_1 and C_2 are complexity factors based on transistor count

C_3 is a complexity factor based on package technology and number of pins

K_L is a learning factor

The quality factor, K_Q, has a value of 1 for devices procured in full accordance with MIL-M-38510, Class B requirements. This value was used for all circuits in this project. It should be noted that the quality factor is a direct multiplier, so that the predicted rate is proportional to it. More or less stringent quality factors can therefore greatly influence the prediction for any individual circuit, circuit board, or entire component.

Junction temperatures are used in determining the adjustment factors K_T. The junction temperature is ambient temperature plus the differential resulting from power dissipation through the case. An ambient of 60° was used, with the power dissipation taken from the circuit specification.

The voltage derating stress factor is 1 for the bipolar circuits used in the CAPS processor. The application environment factor is 3.5 for the airborne, inhabited, transport environment of the aircraft underdeck avionic rack. Failure rates for the circuit cards of the FCCs were obtained by summing the failure rates for the card and its components. Table 7.2-1 summarizes the failure rate prediction for the A13 control card. Failure rates for the other cards are shown in table 7.2-2.

Table 7.2-3 presents failure rates for the system components other than the FCCs.

In using these rates in the fault tree and CARSRA analyses, an adjustment was frequently required to include only a portion of the rate, since only certain failure modes are of interest. For example, each dual current comparator has a predicted failure rate of 0.03. Each half of the comparator is given a rate of .01 for the failure mode of failing to trip when the threshold difference is exceeded. This is a very conservative rate for this mode.

TABLE 7.2-1. FCC CONTROL CARD FAILURE RATE

ITEM	FAILURE RATE*
Integrated circuits	1.788
Resistors	.0018
Capacitors	.224
Oscillator	.25
Coil	.0007
Circuit Board	.023
Edge Connector	.16
Control Card Total	2.45

* All failure rates in failures per million hours.

TABLE 7.2-2. PREDICTED FCC CARD FAILURE RATES

CARD NO.	FAILURE RATE*
A1 Power Supply Monitor	0.555
A2-A5 PROM Card	.809 each
A6 Power Supply Monitor	.55
A7-A10 PROM Card	.809 each
A11 Terminator/Test Access	.555
A12 RAM Memory Control	1.18
A13 CAPS Control	2.45
A14 CAPS Data Path	1.98
A16 Cross-Channel Receiver	.70
A17 DITS Transmitter	1.75
A18 D/A Servo Command	1.75
A19 Terminator/Time Synch	1.40
A20 Discrete Output	2.79
A21 Data Transmitter/Receiver	.70
A22 Serial Digital Input No. 1	1.65
A23 Serial Digital Input No. 2	1.80
A24 Autoland Sensor Input	1.80
A25 Cruise Sensor Input	1.12
A26 Data Acquisition	1.20
A27 Discrete Input	1.30
A28 Servo Engage Logic	2.61
A29 Cross Channel XMTR	1.20
A30-A32 Servo Amplifier	3.00
A33 Speed Servo Amp	1.70
A300 Speed Command XMTR	1.70
A400 Power Supply	21.0
A500 Power Supply	21.0

* All failure rates in failures per million hours

TABLE 7.2-3. FAILURE RATES FOR MAJOR RDFCS COMPONENTS

COMPONENT	UNIT FAILURE RATE*
Pitch Angle Gyro	303
Roll Angle Gyro	303
Yaw Rate Gyro	200
Accelerometer	74
Radio Altimeter	756
ILS Receiver	252
Air Data System	167
Roll Autopilot Servo	14
Pitch Autopilot Servo	15
Yaw Autopilot Servo	14
EH Valve Drive Coil	1.0
LVDT	.72
Dual Current Comparator (Hardware)	.03
Warning Annunciator (per function)	8.3

* These are not actual failure rates for any particular airplane or for any single component produced by a particular manufacturer. They are representative rates determined by a review of generic component types on a number of airplane models in a variety of commercial and military applications. All failure rates in failures per million hours.

8. RELIABILITY PREDICTION USING CARSRA

8.1. General Comments

AC 25.1309-1 emphasizes the need for probability analysis in determining system reliability. This circular defines reliability to be "the probability that a system, subsystem, unit or part will perform its intended function for a specified interval under stated operational and environmental conditions." The following three probability classifications are specified:

- Probable: Probable events may be expected to occur several times during the operational life of each airplane. A probable event has an occurance probability on the order of 1.0×10^{-5} or greater.

- Improbable: Improbable events are not expected to occur during the total operational life of a random single airplane of a particular type, but may occur during the total operational life of all airplanes of a particular type. An improbable event has a probability on the order of 1.0×10^{-5} or less.

- Extremely Improbable: Extremely improbable events are so unlikely that they need not be considered to ever occur, unless engineering judgment would require their consideration. An extremely improbable event has a probability on the order of 1.0×10^{-9} or less.

Aircraft functions fall into three categories: Non-essential, Essential, and Critical. Section 3.1 discusses these three categories; AC 25.1309-1 provides a better discussion. Errors in a non-essential function may be probable. Errors in an essential function must be improbable. Errors in a critical function must be extremely improbable.

An IAA is performed on a particular aircraft system performing a function, e.g. the RDFCS. The RDFCS becomes a critical system when the airplane is executing an automatic landing, during the last 150 ft of descent.

8.2. RDFCS Reliability - Fault Tree Analysis

As demonstrated in section 4.3, the probability of system failure may be computed from the fault tree analysis. The two scenarios that could result in the aircraft's improperly accomplishing or losing the critical function performed by the RDFCS had a combined probability of 0.638×10^{-9}. A third scenario, that conditions encountered are outside the system design requirements or control laws are deficient, was not considered (see section 4.2). The probability of failure of the RDFCS is "extremely improbable." The concern is that an error was made producing a low system failure probability and that the error remains undetected. An independent confirmation of this reliability is needed.

8.3. RDFCS Reliability - CARSRA

CARSRA is an analytical reliability prediction program used in this IAA to obtain the probability of system failure (Bjurman, et al., 1976). In this analysis, the probability of failure is only considered during the crucial flight phase, which has a duration of 0.02 hours. Although CARSRA was used in this assessment, some other method (except fault tree analysis) could be used for the verification. These alternatives were discussed in section 3.5.

8.4. CARSRA Application

CARSRA is an application of stochastic processes. Stochastic processes concern sequences of events governed by probabilistic laws. Applications exist in many physical and social sciences, as well as engineering. The explosion of system failure into its elemental causes with probabilistic relationships (fault tree analysis) suggests that viewing the error tree as a stochastic process may be an aid in analysis. The reader need not have a strong statistical background in order to understand this discussion. There are a large number of possible references on stochastic processes. Karlin (1966) was used in the preparation of this tutorial.

8.4.1. Configuration Description

The CARSRA computer program requires that the user provide an input description of the system under analysis. Three levels of organization are implicit in the CARSRA inputs; the user must adhere to these levels. At the top level is the system, in this case the RDFCS. System failure probabilities constitute the primary output provided by CARSRA. The intermediate level is comprised of stages. Each stage consists of one or more identical modules, which are at the lowest level. In the RDFCS, each sensor is a module, and like sensors form stages. For example, each of the three Normal Accelerometers (NA) is a module, and the three NA together comprise a stage. CARSRA determines system reliability as a function of time; the success of the system is based on dependencies between stages, functional redundancy, and module redundancy.

8.4.2. Markov Models

A Markov process is a type of stochastic process. Chapter 1.3 of Karlin (1966) contains a mathematical definition of both stochastic and Markov processes. A Markov model is a Markov process used to describe a particular system or subsystem. Markov models were selected by the CARSRA developers as a major part of the program's analytical framework. The following discussion of these models includes some material on applying CARSRA to systems other than the RDFCS. This material is intended to benefit readers not familiar with the rationale of developing the input parameters for Markov models as used in CARSRA.

A Markov model is used to describe the number of failed and operating modules within each stage. The transition rates from state to state are used by CARSRA in computing state occupancy probabilities. A separate Markov model is used for each stage. State 1 is the no-failure state in each model, and the two states with the highest numbers correspond to stage failure. The Model always starts

in State 1. For example, a dual stage (one of two identical modules required for the stage to function) might have four states, as shown in figure 8.4-1. State 1 represents both modules working. State 2 represents one module failed and one working. States 3 and 4 represent both modules failed. The highest numbered state (state 4 in this case) represents undetected stage failure, while State 3 represents detected failure. Note that State 2 does not distinguish which module has failed.

State transition rates must be supplied to CARSRA by the user. These are generally functions of the module failure rates, and possibly other parameters. Returning to the example of the dual stage used previously, the Markov state diagram would be as in figure 8.4-1. Transition rate f_{12} is the rate at which transitions occur from State 1 to State 2. That is, if the system is in State 1, the probability that it will transition to State 2 during a short increment of time, dt is $f_{12}dt$. The other transition rates are similarly defined.

If there is no monitoring or switching required when the first module fails, and if there is no possibility of the stage failing undetected, the transition from State 1 will always be to State 2, and the transition from State 2 will always be to State 3. Transition rate f_{12} will be simply 2f, and f_{23} will be f, where f is the failure rate of a single module. The other transition rates will be 0. Note that this means that State 4 will never be occupied, consistent with undetected stage failure being impossible.

In many instances encountered in real systems, digital or otherwise, a reconfiguration must occur before the redundancy can be determined. In the example dual case, an output monitor could be used on each module. If the monitor can detect 97 percent of module failures, e.g. no output or unreasonable output, the monitor provides "coverage", c, of 97 percent. The transition rate f_{12} is then 2fc, so that 97 percent of the transitions from State 1 go to State 2.

Of the remaining three percent of the transitions from State 1, some fraction, e.g., 2/3, could go to State 3 and the rest to State 4. This would result in f_{13} being 2f(1-c)(2/3) or 2f(.02), and f_{14} being 2f(1-c)(1/3) or 2f(.01).

Note the distinctions between coverage (which relates to module failure detection) and undetected stage failure. Note also that the function of a particular stage could be such that it cannot fail undetected, even though individual modules within the stage may fail with coverage less than 1. In other cases, stage failure may be detected only by multiple module failures being detected.

The sum of transition rates out of State 1 is 2f. In general, if any state corresponds to N modules working, the sum of transition rates out of that state will be Nf.

Stages can fail for two reasons: spares exhaustion or coverage failure. In contemporary aircraft systems having critical functions to perform, coverage failures are of as much concern as spares exhaustion.

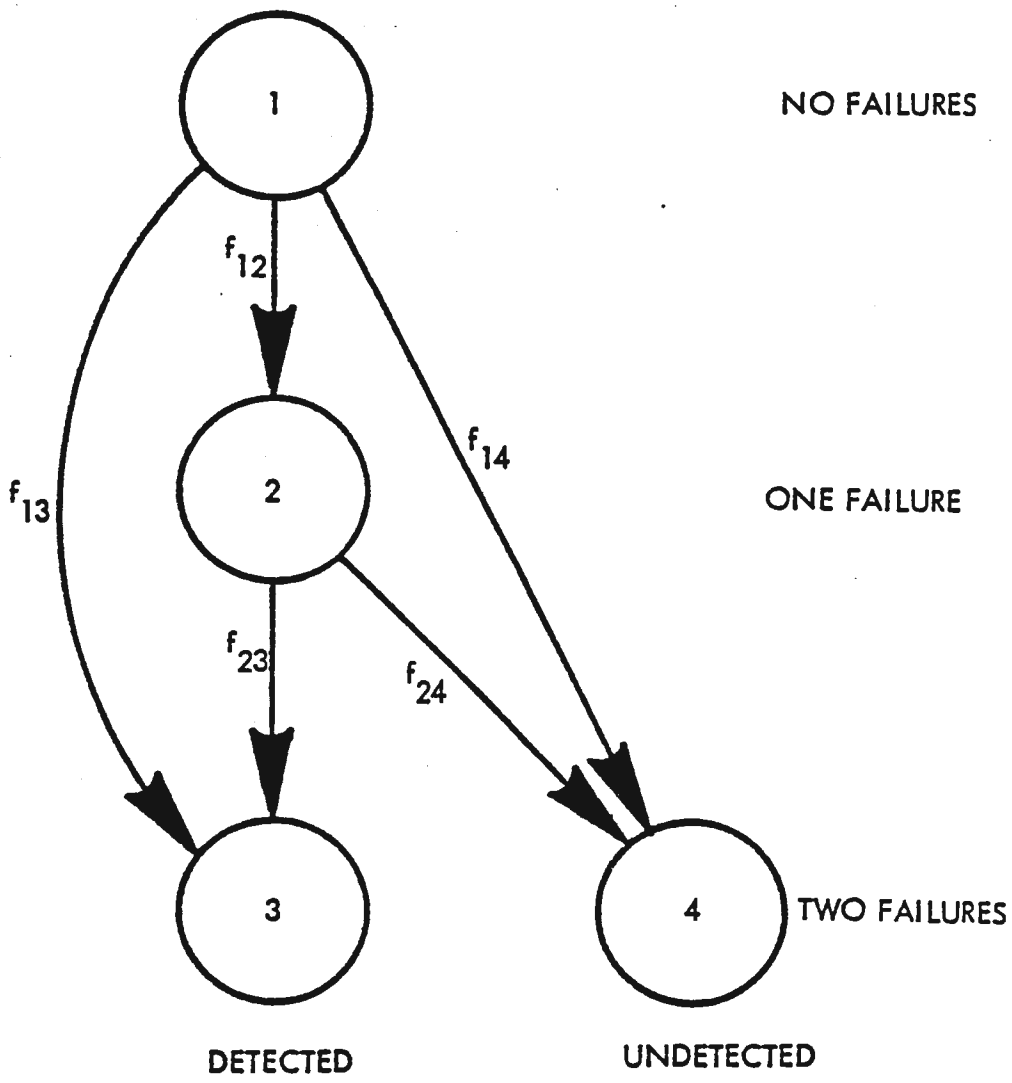


FIGURE 8.4-1. MARKOV MODEL OF DUAL STAGE

In the previous dual stage example with 97 percent coverage of the first module failure, no consideration of the failure rate of the monitor itself was included. The coverage factor of 97 percent means that 97 percent of the module faults are of such a nature that they can be detected by an unfailed monitor. The rest are outside of the monitor's capability. In cases where dedicated hardware monitors are used, it is appropriate to consider their failure rates and failure modes. A two-state monitor is the type most frequently encountered. It provides only a GOOD/BAD signal. Such a monitor has only two failure states: false indication of BAD when the module is good, and false indication of GOOD when the module is bad.

The simplest way of treating such monitors in CARSRA is to combine the monitors with the modules as a single stage. The transition rate from State 1 to State 2 is then

$$2fcr_m + 2f_m a$$

where:

- f = the failure rate of a single module
- c = the coverage
- r_m = the reliability of the monitor over the entire flight time
- f_m = the monitor failure rate
- a = the fraction of monitor failures resulting in a good module being declared bad.

The other transition rates would be similarly defined, recognizing the relation between detection of stage failure and component monitors. Each instance of such a stage must be evaluated individually in determining the applicable rate formulas.

Frequently, certain terms in a rate equation can be ignored because they are numerically negligible. For example, if $f = 120 \times 10^{-6}$ and $f_m = 0.1 \times 10^{-6}$, the term $2f_m a$ can be ignored in the formula

$$f_{12} = 2fcr_m + 2f_m a$$

provided c is not absurdly small. If c is 90 percent, a is 50 percent, and the flight time is ten hours,

$$f_{12} = 2(120 \times 10^{-6})(.90)\ \exp(-.1 \times 10^{-6} \times 10) + 2\ (.1 \times 10^{-6})(.50)$$

$$= 216 \times 10^{-6} + .1 \times 10^{-6}$$

Inclusion of the term yields a rate of 216.1×10^{-6}; ignoring it yields 216.0×10^{-6}. The difference is much less than that caused by uncertainty in the module failure rate, 120×10^{-6}.

8.4.3. Dependencies

CARSRA permits the user to describe instances in which failures of a module in one stage will prevent a module in another stage from being used. An example of

this in the RDFCS is the portion of each FCC channel which receives sensor data and makes it available to the other channels. Data Acquisition Card A26 in FCC No. 1 receives data from the No. 1 unit of each triple sensor type and relays it to another card for transmission to the other three channels and for use by its own channel. There are five triple-sensor types involved in the autoland mode: pitch, roll, and yaw-rate gyros, and lateral and NA. (The A26 card also handles data from other sensors, but only these five will be used for discussion here.) If the A26 card fails in FCC No. 1, the data will be lost from pitch gyro No. 1, roll gyro No. 1, yaw-rate gyro No. 1, lateral accelerometer No. 1, and normal accelerometer No. 1, just as if all five of these sensors had failed. The A26 card is called a dependency module and its stage a dependency stage. Each of the affected sensors is called a non-dependency module, and the corresponding stage a non-dependency stage.

Coverage for sensor failures is provided by comparison monitoring and reconfiguration. Each channel independently performs the sensor monitoring functions on the data it will use in control law computations. When an A channel detects a failed sensor, it does not transmit the identity of the individual sensor to the other channels. When a B channel detects a failure, it does transmit a discrete variable, AP.ONEFAIL, to the A channel in the same FCC. The A channel will turn on the NO DUAL annunciation based on its receipt of AP.ONEFAIL from B or its own detection of a sensor failure. The NO DUAL indication is provided to inform the crew that the RDFCS is not fail-operational. The No. 1 FCC drives the No. 1 Warning Annunciation Indicator (WAI) and the No. 2 FCC drives the No. 2 WAI, so that warning will be provided if either channel of either FCC detects the failure.

The sensor monitoring is part of the foreground flight software. Consequently, for a channel to detect a fault, the CAPS processor must function, as must the CAPS bus and portions of the program and data memory. These are the same hardware elements which perform other functions, such as control law computations and mode logic computations. Most faults in these circuits will result in a totally debilitated processor, so that the inability to the monitor sensors is inconsequential. Note also that even if one channel does lose the ability to monitor sensors, any one of the other three channels can force the NO DUAL warning.

Therefore, the only appreciable probability that the loss of fail-operational sensor capability will not be annunciated results from loss of both WAIS. The multiple-function WAI has a unit failure rate prediction of 33 per million hours. The failure rate of any one of the eight warning messages is conservatively taken to be one-fourth the unit rate, or 8.3 per million. The FCC activates the NO DUAL message by providing a ground to the WAIs, so that a broken wire or bad connector contact would prevent annunciation. A rate of 1.3 per million hours is included for such failures. Also, the Discrete Output (A20) and Servo Engage Logic (A28) cards are involved, with failure rates of 2.79 and 2.61 per million hours, respectively. Even though only a portion of the failures of these cards will affect NO DUAL, the full rate is used. Further analysis could reduce this rate substantially. The failure rate for NO DUAL is then

WAI	8.30×10^{-6}
Wiring	1.30
A20 Card	2.79
A28 Card	2.61
	15.00×10^{-6}

The probability of failure in a 4-hour time period is then 60×10^{-6}. The probability of both NO DUAL warnings being lost is the square of this number, 3.6×10^{-9}. The test button on the WAI results in the FCC circuitry and the wiring being tested as well as the WAI itself. Thus latent failures are not a problem, provided the indicators are tested prior to autoland.

The factor 3.6×10^{-9} is used as the probability that the first failure of a sensor type will not be covered. This does not constitute stage failure, either detected or undetected. Undetected stage failure is assumed to occur on second failure, provided the first failure was undetected. This is somewhat a misuse of the term "undetected"; the stage failure itself is not necessarily undetected, but the increased likelihood of its occurrence, following first failure, is not annunciated.

This treatment of sensor failures allows the availability feature of CARSRA to be used in computing the probability of loss of one sensor prior to 150 ft, failure of the NO DUAL annunciation, and another failure below 150 ft.

8.4.4. Availability

CARSRA permits system reliability to be computed for a mission phase which follows a period of operation with less stringent failure criteria. An obvious example of this is the RDFCS, which is fail-passive in cruise, but must be fail-operational in autoland below 150 ft. The availability feature allows the user to specify which modules may be failed at the beginning of autoland without forcing diversion to an alternate landing site. Each such availability configuration must provide adequate reliability for the landing, although not as much as if everything is working. The RDFCS requires all of the modules used in autoland to be operational, so that the availability feature might seem not needed in this assessment. It is needed, though, to compensate for a capability which CARSRA lacks.

The reliability of the RDFCS for automatic landing is predicated on the system being fail-operational as the alert height is passed. Therefore, the probability of the system having a latent failure at 150 ft and a second failure below that point must be quite small.

By setting up the CARSRA input to allow one sensor of each type to fail during cruise, with the transition rate from State 2 to the undetected failure state including the coverage factor of 3.6×10^{-9}, the undetected system failure probability computed by CARSRA will give the probability of an undetected latent failure at 150 ft and a second failure before touch-down. (See figure 8.4-2.)

What CARSRA will actually compute is:

$P_1 \times P_2 + P_3 \times P_4$

where,

P_1: probability of 0 failures at 4 hours

P_2: probability of undetected failure and detected failure between 4 and 4.02 hours

P_3: probability of 1 undetected failure at 4 hours

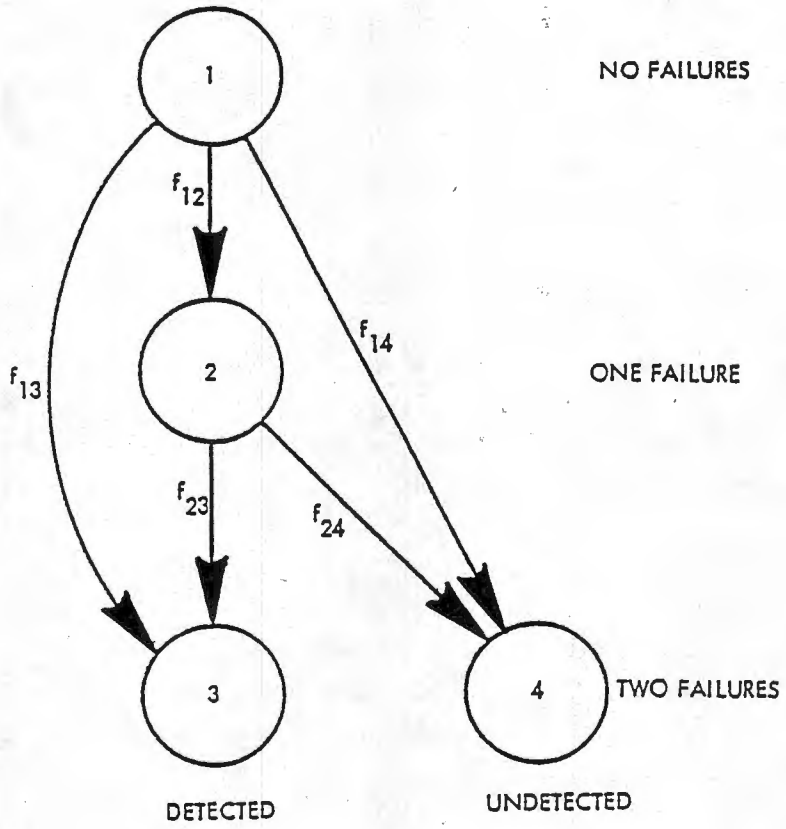
P_4: probability of second failure between 4 and 4.02 hours

Since the probability of both an undetected and a detected failure between 4 and 4.02 hours is very small, the first term is negligible and the output will be equal to the second term, which is the probability desired. This approach is used for the undetected (unannunciated) failures throughout the system. The definition of stages and the transition rates are shown in figure 8.4-3.

The CARSRA program is sensitive to the precision of the host computer. During this analysis, some negative probabilities for unannunciated failures were computed. It is suspected that this may have been caused by the 36-bit word length of the Univac 1100-series computer. The transition rates to the unannunciated failure states are quite small, in some cases 1.0×10^{-13}, and addition/subtraction of numbers of this magnitude with numbers close to 1.0 could produce some numerical accuracy problems on a 36-bit machine. If the program is run on a computer with a 64-bit word length, the problem is thought to be unlikely.

Because of the numerical problem encountered with the CARSRA output, the system failure probabilities reported here were actually manually calculated. This was done by manually computing the stage occupancy probabili-ties, and then combining these probabilities to account for dependencies between stages, using the same logic that the CARSRA program uses.

The probability of an undetected failure prior to the crucial phase, followed by a second failure in the crucial phase, is 3.36×10^{-14}, compared to 2.46×10^{-14} from the fault trees. The probability of multiple failures in the crucial phase, if everything is working just prior to the phase, is 0.658×10^{-9}, compared with 0.638×10^{-9} from the fault trees. These results are shown in table 4.3-1.



	DUAL SENSOR	TRIPLE SENSOR
f_{12}	2f	3f
f_{13}	0	0
f_{14}	0	0
f_{23}	f	2f
f_{24}	fa	2fa

f = MODULE FAILURE RATE

a = ANNUNCIATION FACTOR 3.6×10^{-9}

FIGURE 8.4-2. MARKOV MODEL CODING FOR SENSOR STAGES

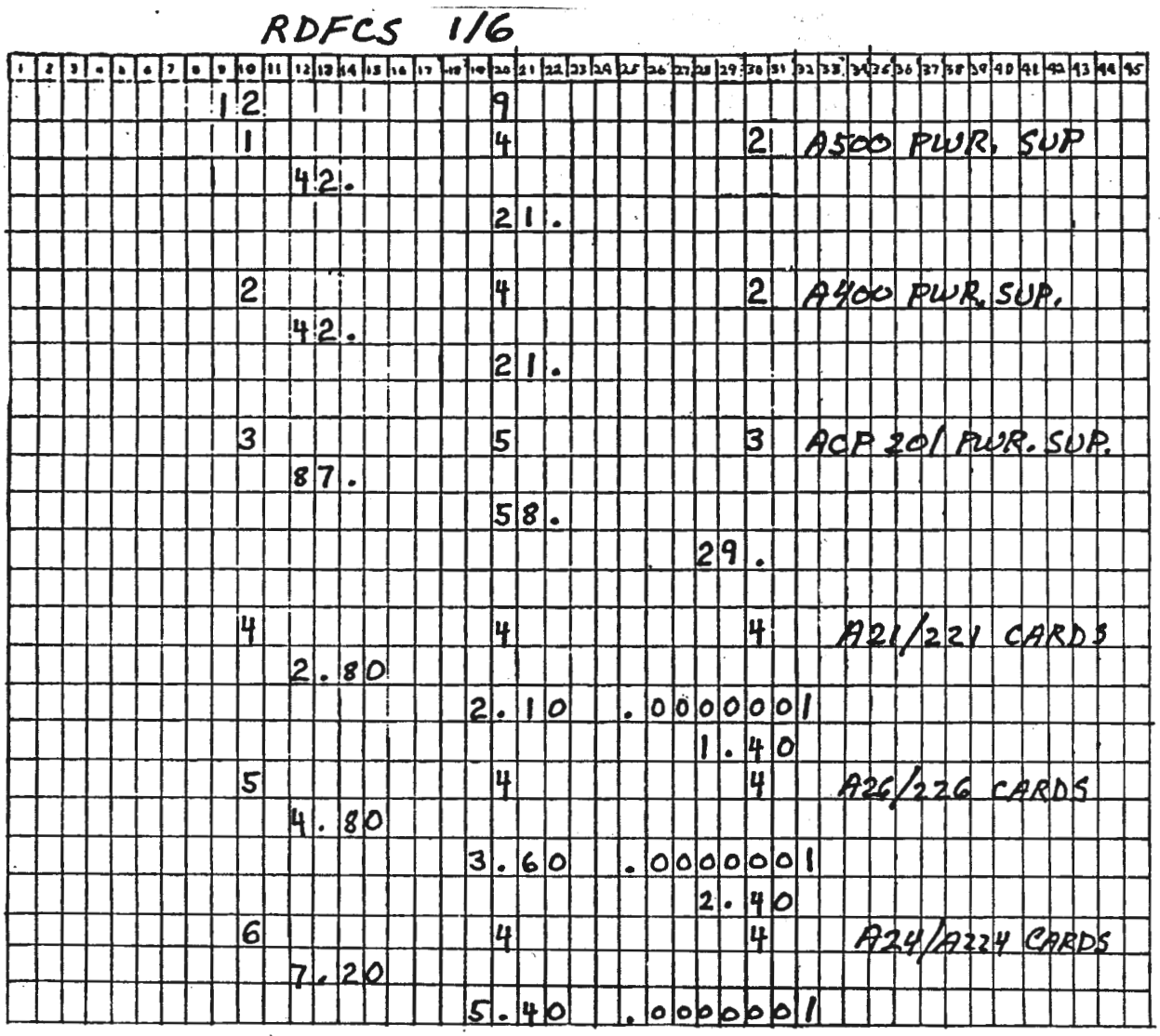


FIGURE 8.4-3. CARSRA INPUT (1 of 6)

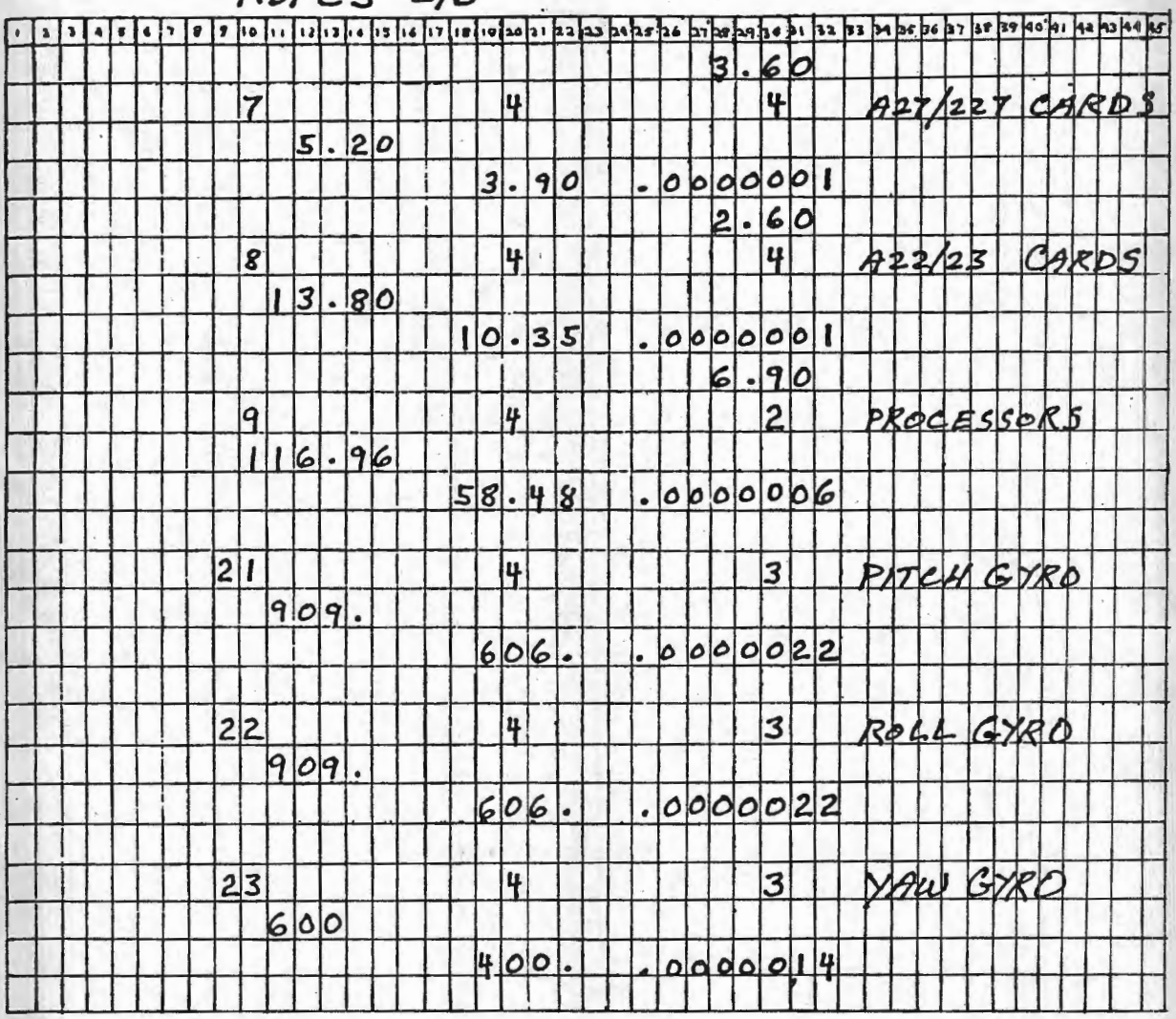


FIGURE 8.4-3. CARSRA INPUT (2 of 6)

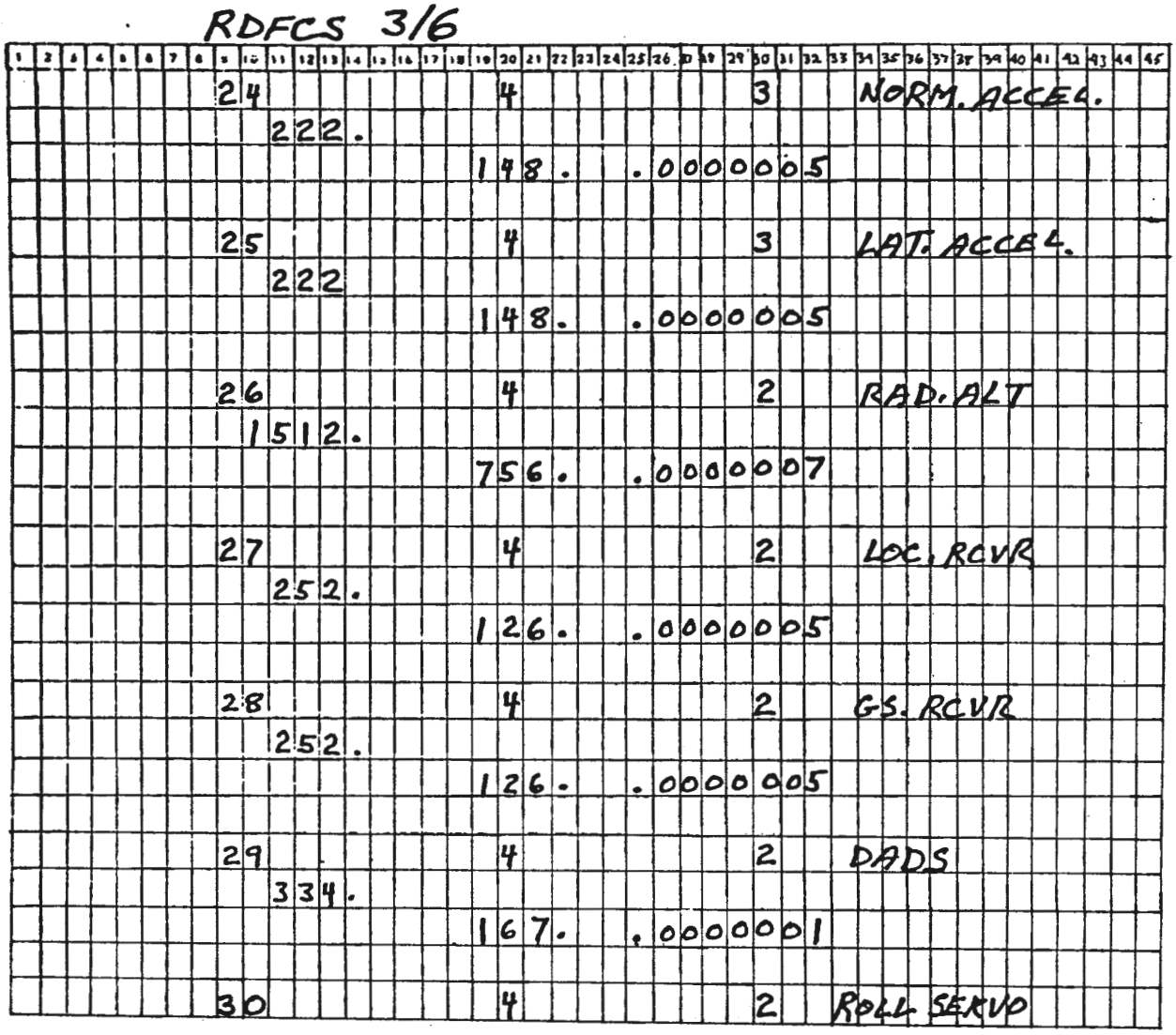


FIGURE 8.4-3. CARSRA INPUT (3 of 6)

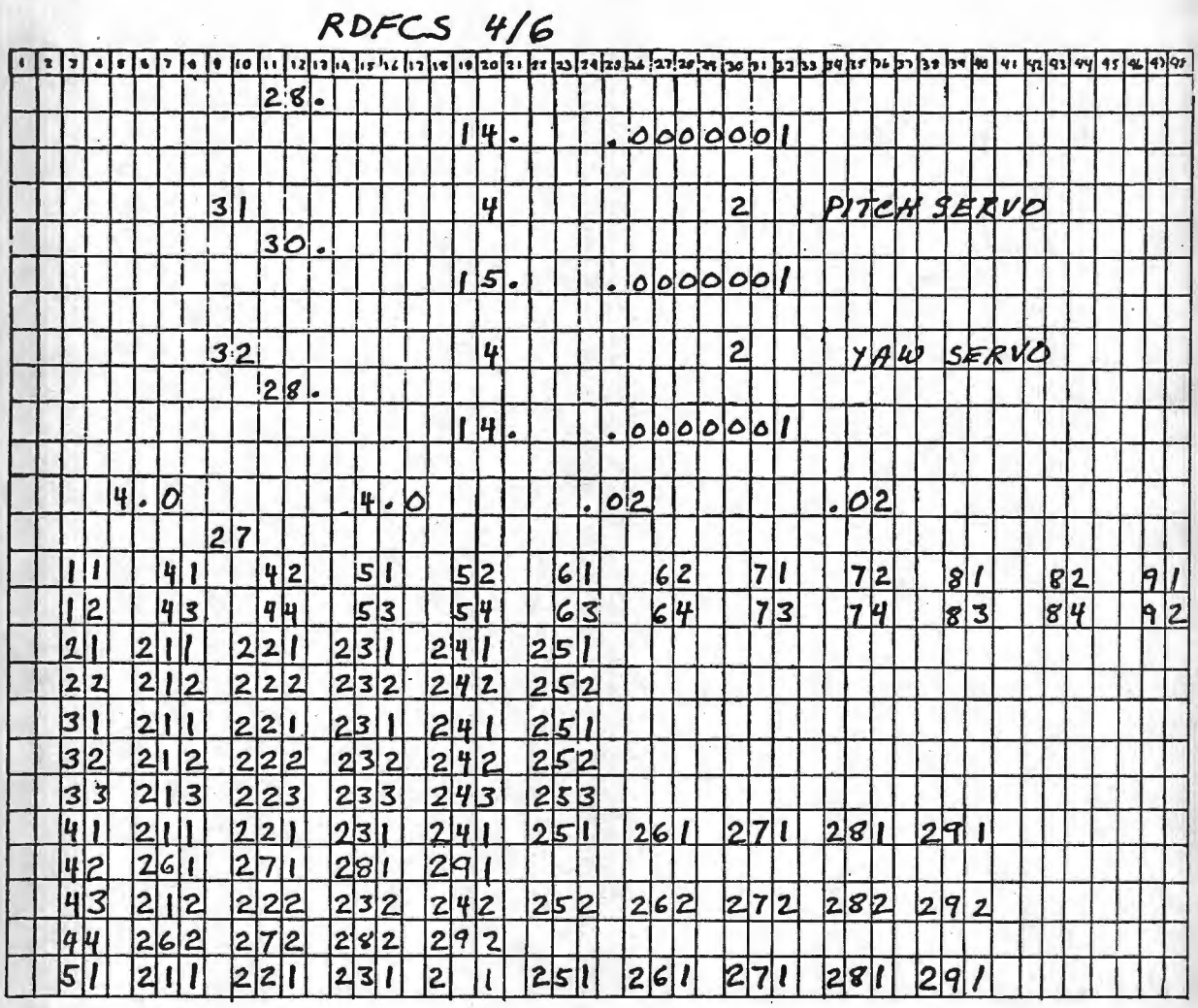


FIGURE 8.4-3. CARSRA INPUT (4 of 6)

RDFCS 5/6

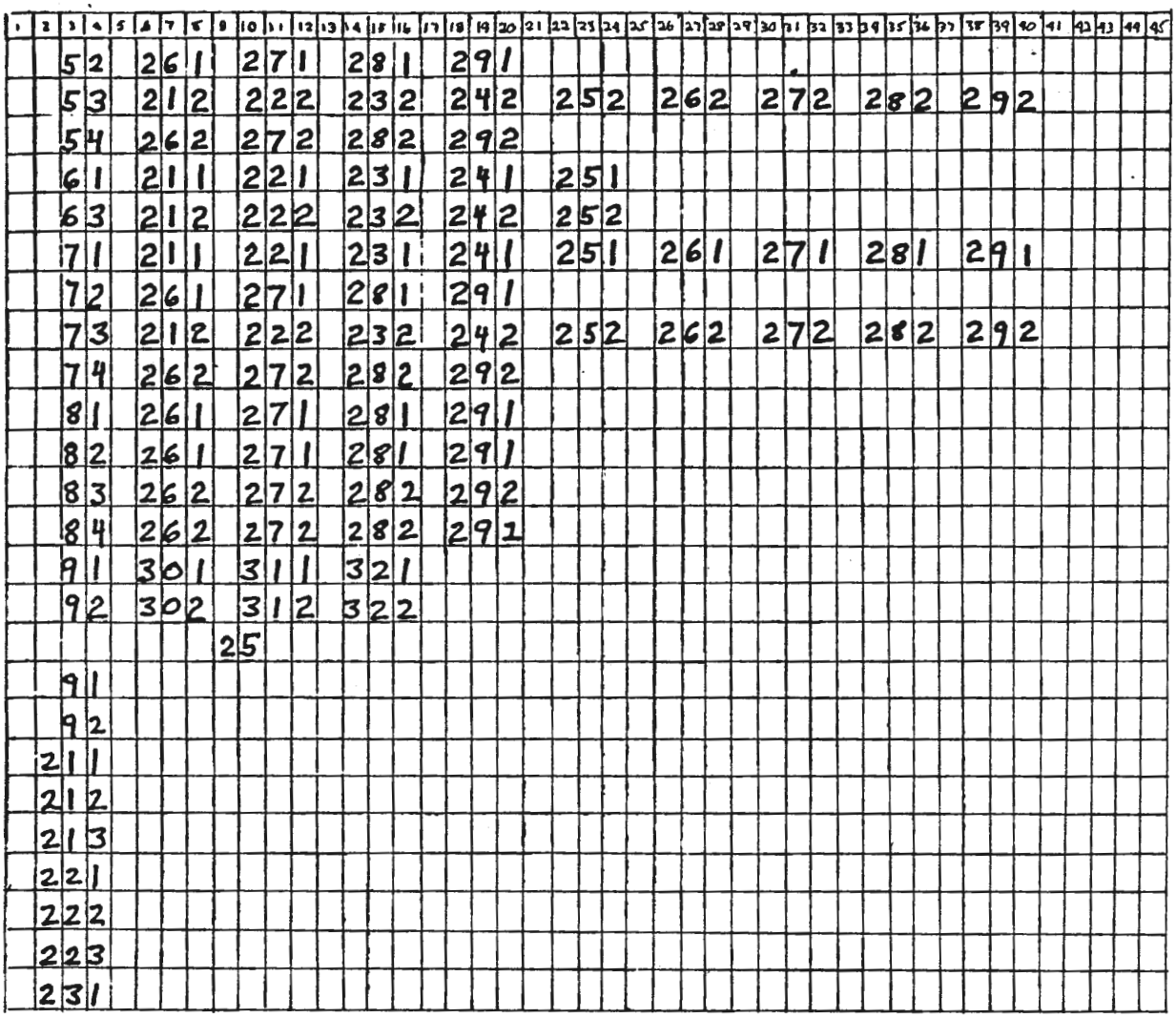


FIGURE 8.4-3. CARSRA INPUT (5 of 6)

RDFCS 6/6

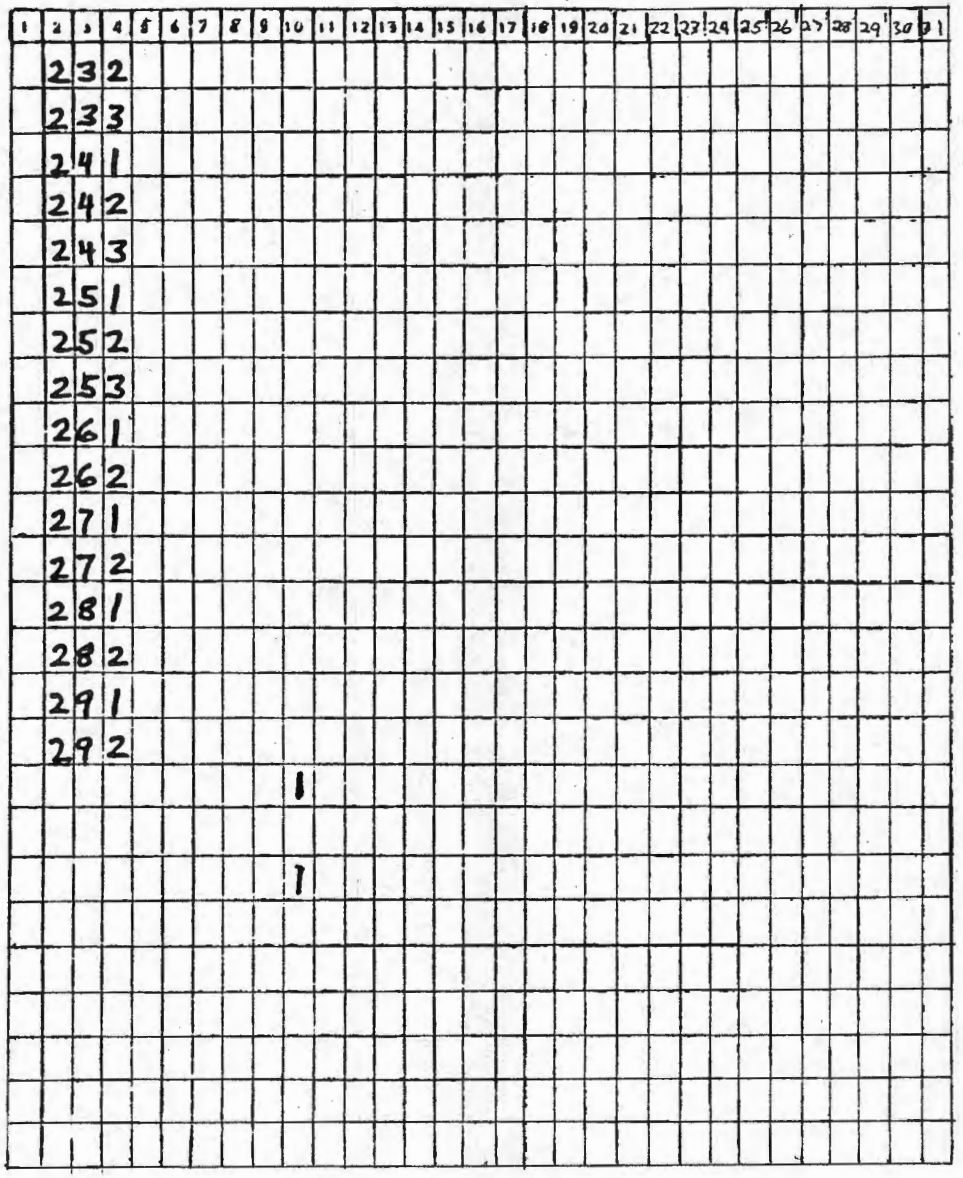


FIGURE 8.4-3. CARSRA INPUT (6 of 6)

9. SUMMARY

9.1. Background

The purpose of this tutorial is to provide certification engineers with a background in IAA. The vehicle for accomplishing this is to provide a walk-through of a system failure analysis using a critical system, the RDFCS. The complexity of airplane systems (RDFCS being a prime example) has necessitated the evolution of safety analysis techniques that permit the diagnosis of failure effects of multiple faults, both intra- and intersystem. A diagnosis of this type is important in demonstrating compliance with Part 25 of the FARs for "flight-essential" and "flight-critical" avionic systems. AC 25.1309-1 provides guidance material for complying with the above regulation. The techniques suggested in the AC were the basis for the RDFCS analysis. The analytical techniques were selected to mutually reinforce each other on the following bases:

- Fault tree analysis identifies the conditions and functional failures contributing to a defined failure condition. It is also used to compute a quantitative assessment of reliability (system failure probability).
- Analytical reliability prediction analysis confirms the system failure probability computed with fault tree analysis.
- FMEA was used to augment fault tree analysis in diagnosis below the circuit card level.
- Fault insertion was used to confirm the fault detection and tolerance capabilities of the RDFCS. The existence of the RDFCS Simulator made fault insertion possible.

The integration of the above techniques was exemplary. The selection of safety analysis techniques for integration is the responsibility of the test/certification engineer and must be customized to the system. The use of any IAA model must supplement, but not replace, the judgment of the FAA certification personnel.

9.2. Concluding Remarks

The concluding remarks for this tutorial relate to the benefits and limitations of the IAA model used in the RDFCS analysis. During the course of this analysis, certain enhancements were visualized that would have improved the study. These enhancements will be discussed in section 9.3. They are included in this tutorial, because of their potential application to other integrated analyses.

IAA is workable for a system, such as the RDFCS, which employs monitoring totally separate from the hardware/software being monitored. In the RDFCS, this monitoring includes the servo coil current comparators and the modulator piston followup monitoring. It also includes the warning annunciations which one FCC can generate following a failure in the other FCC. A single-string, self-monitored system might be much less amenable to this approach, depending on the monitoring approach used.

Fault tree analysis is a feasible analytical method for identifying system level faults. One benefit is that specific software failures are identified as the analysis progresses. These can be, and should be, used as a check on the validation test case selection to assure that the software function is rigorously tested. Fault trees can be extended to the circuit card level in a well organized computer such as that used in the RDFCS. In general, the analysis is facilitated by a design with clearly partitioned and identifiable functions, and an interface structure which is consistent for all card inputs and outputs.

FMEA is more easily accomplished than fault trees within the processor itself. This is because the processor is involved in a diverse set of functions defined by the flight software. Most individual pin-level faults have many effects. Usually, each fault can be traced to an effect which totally debilitates the processor. Other effects, which cause massive processor failure or erroneous results only under certain conditions, do not have to be analyzed in detail when their effects do not propagate across channels. In contrast, a fault tree analysis based on loss of required system functions would result in identification of the same hardware faults time after time. When a system simulator exists, fault insertion is an important adjunct to FMEA.

Fault tree analysis and reliability analysis tools, such as, CARSRA, provide comparable results for relatively straightforward redundancy conditions, e.g., the probability of multiple failures during the crucial phase when all components are working at the beginning of the phase. For more complicated situations, the two methods do not agree as closely. This is a result of different simplifications and assumptions being made to structure the problem to the two methods. For example, the third sensor of a triple sensor set (figure 2.1-1) has redundant input paths to the computers (the data input sections of the two computer B channels), but the other sensors have only a single data path (the A channel input sections). This is treated correctly in the fault trees, but the redundancy cannot be accounted for in CARSRA. The conservative assumption is therefore made that loss of either B channel sensor input capability will cause loss of the third sensor in all triple sensor sets. In validation work, any assumptions required can be made conservatively so that the computed failure probability is actually an upper bound on the true probability.

9.3. Analysis Enhancements

The RDFCS simulator was invaluable as a tool in the failure analysis function. Given sufficient access to the simulator, the FMEA and fault insertion test sessions should be scheduled on an iterative basis. After performing certain FMEAs, a fault insertion session should be used to confirm the analysis to that

point. The results should then be incorporated in the FMEA, and the entire FMEA reviewed in light of those results. This review may lead to identification of additional fault cases which should be simulated to resolve uncertainty which may have arisen. This iterative approach was not used in the subject study because of limitations on the availability of the simulator.

The RDFCS simulator has substantial capability for research investigations of DFCS validation issues. This capability would be significantly improved by an automated fault insertion and data recording capability. Such a capability should be preprogrammable with a list of faults to be inserted. It should include a means of recording the impact of each fault (e.g., changes in the values of discrete variables) for many more variables than the four accessible through the CTAs. It should allow variables in channels other than the faulted one to be accessed and recorded.

Care should be taken in selecting a computer tool for analytic reliability prediction. The nature of the analytical program requires a high degree of numerical precision when solving certain problems. In the subject study (see section 1.2) the system failure probabilities used to confirm the fault tree analysis computations were manually calculated. This was done by manually computing the stage occupancy probabilities and then combining these probabilities to account for dependencies between stages. The logic used was identical to that used by the CARSRA program. The problem was thought to be due to the small failure rates and the accompanying precision difficulties encountered on a computer with a 36-bit word length. If so, the use of CARSRA on a computer with a 64-bit word length would have remedied the problem.

APPENDIX A - SUMMARY DESCRIPTION FOR REDUNDANT DIGITAL FLIGHT CONTROL SYSTEM ARCHITECTURE

Redundant Digital Flight Control System

In most operational modes, the RDFCS is fail-passive, with a dual channel configuration. For automatic landings under Category IIIa conditions (see AC 120-28C), the system can be brought into a dual-dual fail-operational, fail-passive configuration. The classification dual-dual relates primarily to the four computer channels in the system. Each of the two FCCs has two channels which run frame-synchronously, with each channel driving one coil of a dual-coil servo in each axis. Any indication of disagreement between the two channels in an FCC causes the servo connected to that FCC to be disengaged by removing hydraulic pressure. Figure 2.1-1 of this report summarizes the dual-dual configuration.

Monitoring Configuration and Implementations

Extensive monitoring is employed in the RDFCS for fault detection. Coil current comparators for each servo provide coverage of faults resulting in erroneous commands to the servo coils. They also provide coverage for broken wire faults between the FCC and the servo or failures of the coils themselves. These monitors are made more effective by the insertion of opposing 5 ma bias currents. The bias currents permit circuit integrity to be monitored even when the FCC is not commanding the servo to a new position, such as when the aircraft is flying through very calm air at a stable altitude. It may be noted that this type of monitoring is equally applicable to analog and digital systems.

Response of the autopilot servos to commands from the servo amplifiers is monitored by modulator position signals fed back to the FCC. The feedback signals are averaged and passed through a high-pass filter to get a modulator rate that is compared with cold current. This comparison is used to detect jamming of the modulator piston, runaway conditions, or loss of hydraulic power. This type of monitoring can also be applied to either analog or digital systems.

In the pitch-axis servos, modulator piston position monitoring is implemented in hardware. In the other two axes, it is implemented in software. Together, the coil current monitoring and modulator piston monitoring detect any servo fault which prevents the servo from responding to commands. They also detect any fault in a computer channel which prevents that channel from generating a reasonable command for the servos in each of the three axes. All monitors and feedback sensors are dual to increase reliability.

Each computer channel has an iteration monitor implemented in hardware. This monitor observes the state of a discrete software variable which is changed at the end of each iteration of the foreground software. Since this software executes at a 10 Hertz (Hz) rate, the result is a 10 Hz square wave. Should the processor short-loop or hang up, the 10 Hz wave will not be presented. The

iteration monitor will withdraw its input to the engage logic, and the FCC will disengage.

Sensor monitoring is primarily accomplished by comparison and by validity discretes generated by the sensors. There is no one location that sensor monitoring takes place, since all four computer channels incorporate the monitoring function. This process ensures that the circuitry involved in getting the sensor signals to each channel is included in the monitoring.

The accelerometers are tested each time the system is powered up with the airplane on the ground. The ILS receivers are checked using a square wave test. This test checks for failure of the localizer and glideslope beam deviation inputs. During landing, the outputs of both receivers are compared, with reliance on the self-monitoring to identify which receiver is bad if the signals disagree. The comparison monitoring is used to check wire integrity between the receiver and the computer channels. The other dual sensors are comparison monitored in the same way.

Even though each channel monitors sensors individually, any channel can initiate the NO DUAL annunciation, which is the primary indication that the system is not fail-operational. If any channel detects a second failure of a sensor type, it will cause its FCC to disengage, but the other FCC will remain engaged.

Although NO DUAL is the primary warning of loss of one sensor, NO ALIGN will be annunciated if the course signals from the two compass systems do not agree.

Other monitoring within the FCC involves comparison of active operating modes. If the two channels within an FCC disagree on which modes are engaged, and the disagreement lasts for more than 0.1 sec, the FCC will disengage. If the two FCCs disagree, SPLIT will be displayed on the WAIs. This monitoring, together with the sensor data transfers, will detect most faults of the cross-channel data transfer circuitry.

APPENDIX B - DESCRIPTION OF RDFCS SIMULATOR

The RDFCS simulator is comprised primarily of a PDP 11/60 computer and the RDFCS pallet. The RDFCS pallet includes the FCCs, core memory, MDICU, SSP, DSP, CTA, and Computer Breakout Panels. The functions of these items are described in the remainder of this section. Figures 2.1-2 through 2.1-5 are pictures of these items.

PDP 11/60 Computer/Airplane Model

The PDP 11/60 computer hosts a discrete-state model of the airplane in which the RDFCS is installed. This airplane is a representative wide-body transport, and the model coefficients are changed according to the flight case being simulated. Each flight case, then, is a point simulation of the airplane in a particular configuration and operating in a specific portion of the flight envelope. The airplane model executes at a 50 Hz rate.

During the subject study (see section 1.2) the flight cases described and discussed in Lockheed-Georgia LG81E0126 (1981) were used in the simulation. As part of this study, a go-around case was added to the library of cases available. The go-around case is characterized as follows:

Airplane Weight	314,500 lb
Altitude	35 ft
Angle of Attack	10.91 °
Indicated Air Speed	168 kts
Flap Deployment	22 °

Transition capability was added to go from approach conditions to landing conditions, and from landing to the new go-around case. The transitions involve changing the model coefficients and establishing new trim values. The transition capability has been installed and checked out successfully.

Modular Digital Interface Control Unit

The MDICU receives the output of the airplane discrete-state model through a communication link with the PDP 11/60 computer. The MDICU converts the various pieces of information into the form needed by the FCCs. For example, roll angle and pitch angle are converted to three-wire AC signals, properly scaled, while localizer deviation is coded in Aeronautical Radio Incorporated (ARINC) serial digital format. The MDICU is described more fully in the NASA RDFCS System Interface Document.

The MDICU incorporates provisions for the signal for the No. 1 sensor of each type to be ramped up or down. This facility is accessed by means of the HP 2645A terminal physically located in the pallet.

Computer Breakout Panels

Each sensor signal going from the MDICU to the FCCs can be interrupted at the Computer Breakout Panels by removing the appropriate jumper plug. Every FCC back connector pin is routed through one of these plugs. The lower portion of figure 2.1-3 shows the rows of plugs for connector P1 and the "A" half of connector P2. Each FCC has its own breakout panel.

CAPS Test Adapters

Figure 2.1-3 also shows the CTA for one of the FCCs. The upper half of the CTA includes, on the right-hand side, four address and four data windows. An address can be loaded in each address window. The corresponding data window can be used to display the data on the FCC A-side processor bus data lines every time the address appears on the address lines. The CTA also has other capabilities, such as providing a history of the last 16 bus transfers and changing the contents of a specific memory location within the FCC, but during the subject study only the address monitoring was used. Discrete variables representing sensor voter status were monitored visually via the data windows. Continuous variables, such as inputs to the servo amplifiers, were monitored by using the analog output posts below the appropriate data window to drive a strip-chart recorder.

The lower half of the CTA performs the same functions as the upper half, but for the B side of the FCC.

Servo Simulator Panel

The servo amplifier outputs from the FCCs are routed to the SSP. The SSP simulates the dynamics of the autopilot and power servos, and generates the required feedback signals such as modulator piston position. The SSP has circuits which can simulate a hardover or slowover command to a servo coil. It can also simulate a hardover or slowover of a modulator piston, including the modulator piston position feedback signal and the command to the power servo. All of these apply to the No. 1 servo of each type.

Discrete Switch Panel

The DSP is located just below the SSP. This panel provides a centralized location for switches such as hydraulic pressure switches and autopilot disconnect switches. The panel also includes switches that can be used to insert sensor validity faults. These faults can also be inserted by pulling the appropriate jumper plug on the FCC Breakout Panel.

Core Memory

The pallet also contains core memory for the FCCs. This is used for both data and program memory to provide flexibility and convenience in using the pallet to simulate other airplanes or DFCS architectures. As used in an airplane, the FCCs have the flight software stored in PROM and use Random Access Memory (RAM) chips for data memory.

Glare-Shield Panel

The pallet also has a glare-shield panel, which is the control panel for the system as installed in an airplane. It includes the engage (bat handle) switches, mode select switches, altitude select knob, and other controls. The pallet also has a single Automatic Direction Indicator (ADI), Horizontal Situation Indicator (HSI), radio altitude display, Mode Indicator, and WAI.

Circuit	Function	Pin	Fault	Effect
Instruction Mapper Prom CU1	Produce direct input bits A0-A3 for control store memory micro-program start address	A0-A9	Open	Address bit low, wrong address read. Wrong output passed to control store proms as starting address bits A0-A3. Massive processor failure.
			Low	Address bit sticks, wrong address read. Wrong output passed to control store proms as starting address. Massive processor failure.
			High	Same as above
		CS1, CS2	Open	Output pins remain in high-impedence state. Input pins to microprogram sequencer CU16 low. Wrong starting address bits A0-A3 to control store proms. Massive processor failure.
			Gnd	Don't care.
			High	Same as open.
		O1-O4	Open	Prom output bit not fed to microprogram sequencer input bit. Input bit low, resulting in wrong microprogram starting address. Massive processor failure.
			Low	Corresponding bit (A0-A3) of microprogram start or jump address is always low. Massive processor failure.
			High	Corresponding bit (A0-A3) of microprogram start or jump address is always high. Massive processor failure.
Instruction Mapper Prom CU7	Produce direct input bits A4-A7 for control atore memory microprogram start address.			The fault of any pin of CU7 has the same effect as the same fault occurring in CU1, except that the affected address bits are A4-A7.

Circuit	Function	Pin	Fault	Effect
Instruction Mapper Prom CU13	Produce direct input bits A8-A9 for control store memory micro-program start address; produce push/pop signals to stack vector register DU4.	A0-A9	Open	Address bit low, wrong address read. Wrong output passed to control store proms as starting address bits A8-A9. Wrong output may also include wrong push or pop signal to stack vector.
			Low	Address bit stuck low, wrong address read. Bits A8-A9 of microprogram start address wrong. Push or pop signal to stack vector register DU4 may be wrong. Massive processor failure.
			High	Address bit stuck high, wrong address read. Bits A8-A9 of microprograms start address wrong. Push or pop signal to stack vector register DU4 may be wrong. Massive processor failure.
		CS1, SS2	Open	Control register cannot pull down enable, so that output pins are at high impedance. Start address bits A8-A9 always low. Massive processor failure.
			Low	Chip CU13 can pull down data input to microprogram sequencer when control register is trying to set it high as part of a jump address.
			High	Same as open.
		O1-O2	Open	Start address bit A8, A9 to control store always low. Massive processor failure.
			Low	Same as open.
			High	Start address bit A8, A9 high; address bad when bit should be low. Massive processor failure.

Circuit	Function	Pin	Fault	Effect
CU13 con't.		03-04		
			Low	POP commanded (pin 03 faulted) or PUSH commanded (pin 04 faulted) on each clock pulse, so that both commands will go to stack vector register when only non-faulted should be. Stack vector register will do nothing. Massive processor failure.
			High, Open	Fault in pin 03 causes stack vector register to broadside load instead of left shift when mapper prom tries to pop stack. Fault in pin 04 causes broadside load instead of right shift when mapper prom tries to push stack. Stack pointer not pointing to top of stack. Massive processor failure.
Microprogram Sequencer CU14	Generate sequence of microcode Code addresses for control store proms using starting address from mapper or control register. U14 generates microcode address bits A8,A9	D0, D1	Open	Address bit A8, A9 to control store proms always low when starting microcode sequence or on microcode jump. Massive processor failure.
			Gnd	Same as above
			High	Same as above, except that affected bit is always high.
		CN	Open	Carry-in from microprogram sequencer CU15 is always low. Wrong address will be sent to control store when address increment causes overflow in CU15. Effect depends on allocation of control store store addresses to microcode sequences.
			Gnd	Same as above

Circuit	Function	Pin	Fault	Effect
CU14 con't.			High	In incrementing address register, bit A8 will be toggled on each clock pulse. Wrong microcode address will be generated during most microcode sequences. Massive processor failure.
		OE	Open	Address bits A8, A9 always low.
			Gnd	No effect during operation. Maintenance troubleshooting affected.
		RE	Open	Initial microsequence address from mapper prom cannot be loaded. Massive processor failure.
			Gnd	Same as above.
			High	No effect.
		SO	Open	Sequencer will not jump to proper address when SO should be high. Massive processor failure.
			Gnd	Same as above.
			High	Sequencer will execute erroneous jump when SO should be low. Massive processor failure.
		SI		Same effect as pin SO faulted.
		FE	Open	Microprogram counter will always be pushed onto stack or stack will be popped, depending on PUP. Massive processor failure.
			Gnd	Same as above.
			High	Microprogram counter cannot be pushed on stack and stack cannot be popped. Massive processor failure.

Circuit	Function	Pin	Fault	Effect
Microprogram Sequencer CU14 con't.		PUP	Open	Stack will be popped when microprogram counter should be pushed on stack. Massive processor failure.
			Gnd	Same as above.
			High	Microprogram counter will be pushed onto stack when stack should be popped. Massive processor failure.
		CP	Any	Chip disabled. CP is clock pulse input, and all state changes occur on low-to-high transition of CP. Massive processor failure.
		ZERO	Open	Address bits A8, A9 to control store proms always low. Massive processor failure.
			Gnd	Same as above.
			High	Address bits A8, A9 not forced low when commanded by control register CU21. Effect depends on implementation of microcode.
		Y0, Y1	Open	Corresponding address bit to control store is always low. Massive processor failure.
			Gnd	Same as above.
			High	Corresponding address bit to control store is always high. Massive processor failure.
		Y2 CN+4, D2,D3	Any	No effect. Pins not connected.
		VCC	Open	Chip dead. Massive processor failure.
		Gnd	Open	Chip dead. Massive processor failure.

Circuit	Function	Pin	Fault	Effect
Microprogram Sequencer CU15	Generate address bits A4-A7 to control store proms. Effects of most pin faults are the same as for CU14, except the affected address bits are A4-A7. Only pins with different fault are discussed.	D2,D3	Open	Corresponding address bit A6 or A7 is always when address is from mapper prom or micro-coded jump address. Massive processor failure.
			Gnd	Same as above.
			High	Corresponding address bit A6 or A7 is always high when address is from mapper prom or microcoded jump address. Massive processor failure.
		CN	Open	Carry-in from microprogram sequencer CU14 is always low. Wrong address will be sent to control store when address increment causes overflow in CU14. Massive processor failure.
			Gnd	Same as above.
			High	Carry-in from microprogram sequencer CU14 is always high. Microcode address incremented on bit A4 each clock cycle. Massive processor failure.
		CN+4	Open	Same as CN open on CU14.
			Gnd	Same as above.
			High	Same as CN high on CU14.
		Y2,Y3	Open	Corresponding bit A6 or A7 always low in address to control store. Massive processor failure.
			Gnd	Same as above.
			High	Corresponding bit A6 or A7 always high in address to control store. Massive processor failure.

Circuit	Function	Pin	Fault	Effect
Microprogram Sequencer CU16	Generate address bits A0-A3 to control store proms. Effects of most pin faults are the same as for CU15, except that affected address bits are A0-A3. Only pins with different effects are discussed.	CN	Open	Microprogram address is not incremented during execution of microcode sequence. Massive processor failure.
			Gnd	Same as above.
Micro-processor DU17	Processes the four low-order bits of the 16-bit CAPS word in response to instructions from control registers.	A0-A3	Open	Wrong A pointer address when failed bit should be high. Massive processor failure.
			Gnd.	Same as above.
			High	Wrong A pointer address when failed bit should be low. Massive processor failure.
		B0-B3	Open	Wrong B pointer address when failed bit should be high. Massive processor failure.
			Gnd	Same as above.
			High	Wrong B pointer address when failed bit should be low. Massive processor failure.
		I0-I2	Open	Wrong data source selected when failed bit should be high. Massive processor failure.
			Gnd	Same as above.
			High	Wrong data source selected when failed bit should be low. Massive processor failure.

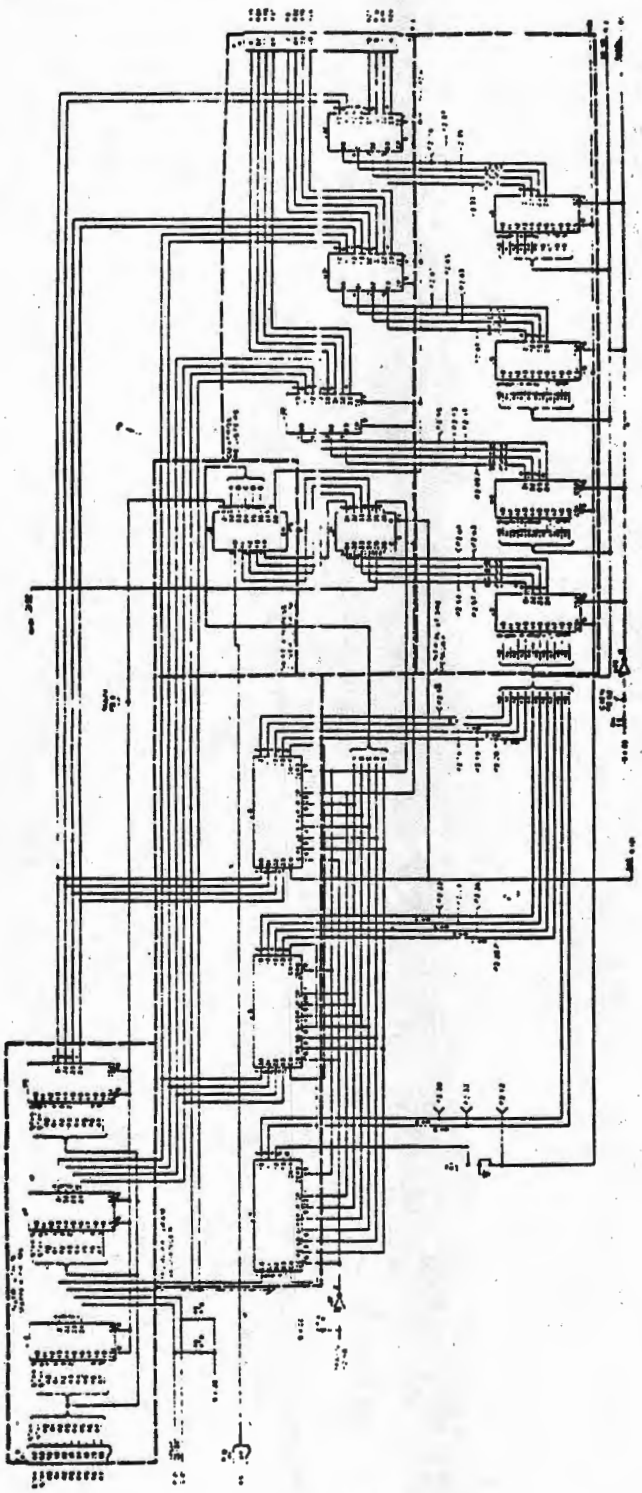
Circuit	Function	Pin	Fault	Effect
Microprocessor DU17		I3-I5	Open	Wrong operation performed when failed bit should be high. Massive processor failure.
			Gnd	Same as above.
			High	Wrong operation performed when failed bit should be low. Massive processor failure.
		I6-I8	Open	Wrong destination code when failed bit should be high. In most cases, the immediate effect will be internal to the chip involving load or shift of data in registers. Massive processor failure.
			Gnd.	Same as above.
			High	Wrong destination code when failed bit should be low. Massive processor failure.
		CP	Any	Chip dead. Massive processor failure.
		D0-D3	Open	Input to processor is wrong when failed bit should be high. Major effect caused by incorrect bit in packed Boolean data. Massive processor failure.
			Gnd.	Same as above.
		C	Open	Carry-in always low. Program counter not incremented on instruction fetch. Massive processor failure.
			Gnd.	Same as above.
			High	Carry-in always high. Foreground loop of flight software cannot execute paths 2 and 4; iteration monitor test bit not toggled; iteration monitor trips. FCC disconnects.

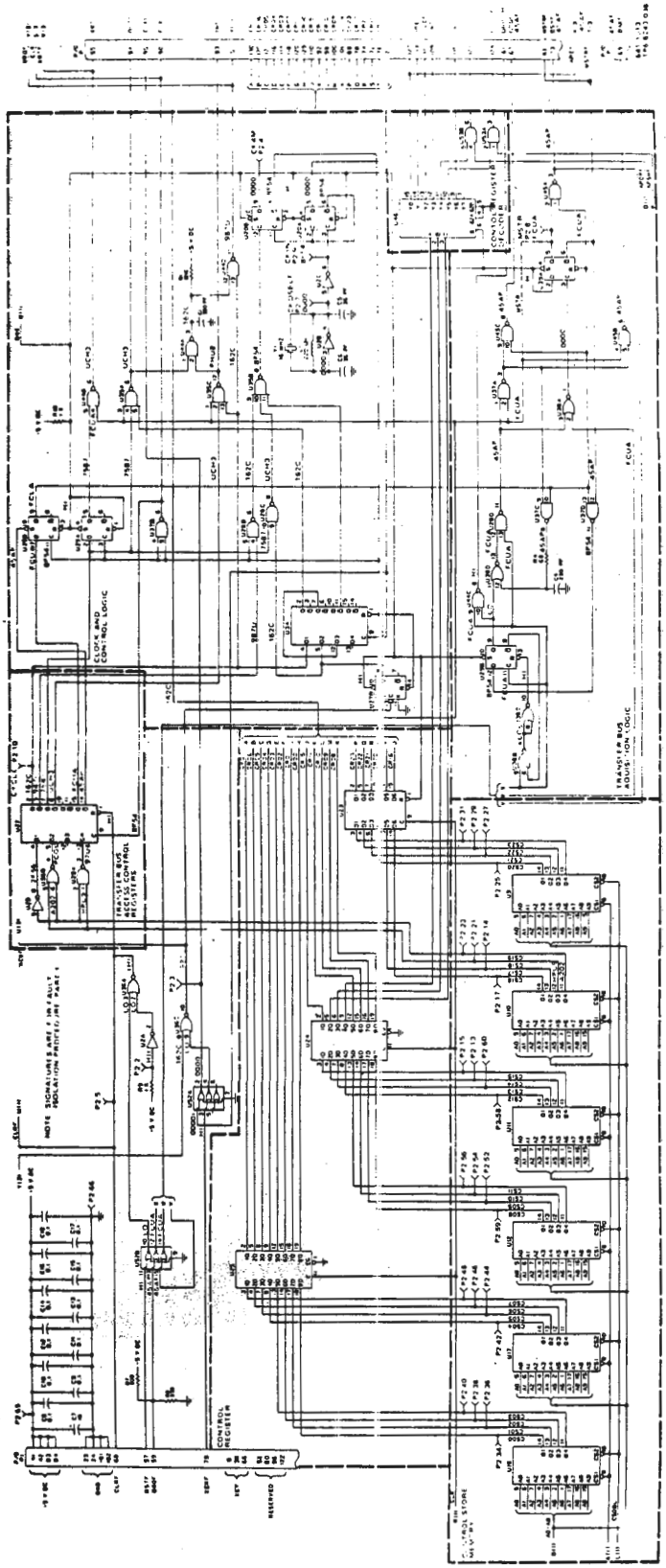
Circuit	Function	Pin	Fault	Effect
Microprocessor DU17 con't.		Y0-Y3	Open	Wrong address gated on CAPS address lines. Massive processor failure.
		P	Open	Carry propagate always sent to carry look-ahead logic. Massive processor failure.
			Gnd	Same as above.
			High	Carry propagate never sent to carry look-ahead logic. Double-precision integrators drift.
		G	Open	Carry generate always sent to carry look-ahead logic. Massive processor failure.
			Gnd	Same as above.
			High	Carry generate signal never sent to carry look-ahead logic.
		F=0	Open	DU17 cannot pull down F=0 line to status register, yielding false results for some logic tests. Massive processor failure.
			Gnd	DU17 always pulls down F=0 line to status register yielding false results for some logic tests. Massive processor failure.
			High	Same as open.
		Vcc	Open	Chip dead. Massive processor failure.
		OE	Open	Chip dead. Massive processor failure.
		Gnd	Open	Chip dead. Massive processor failure.
		R3	Open	Bit left-shifted into DU14 or right shifted into DU17 always low. Multiplication results erroneous. FCC disconnect.
			Gnd	Same as above.

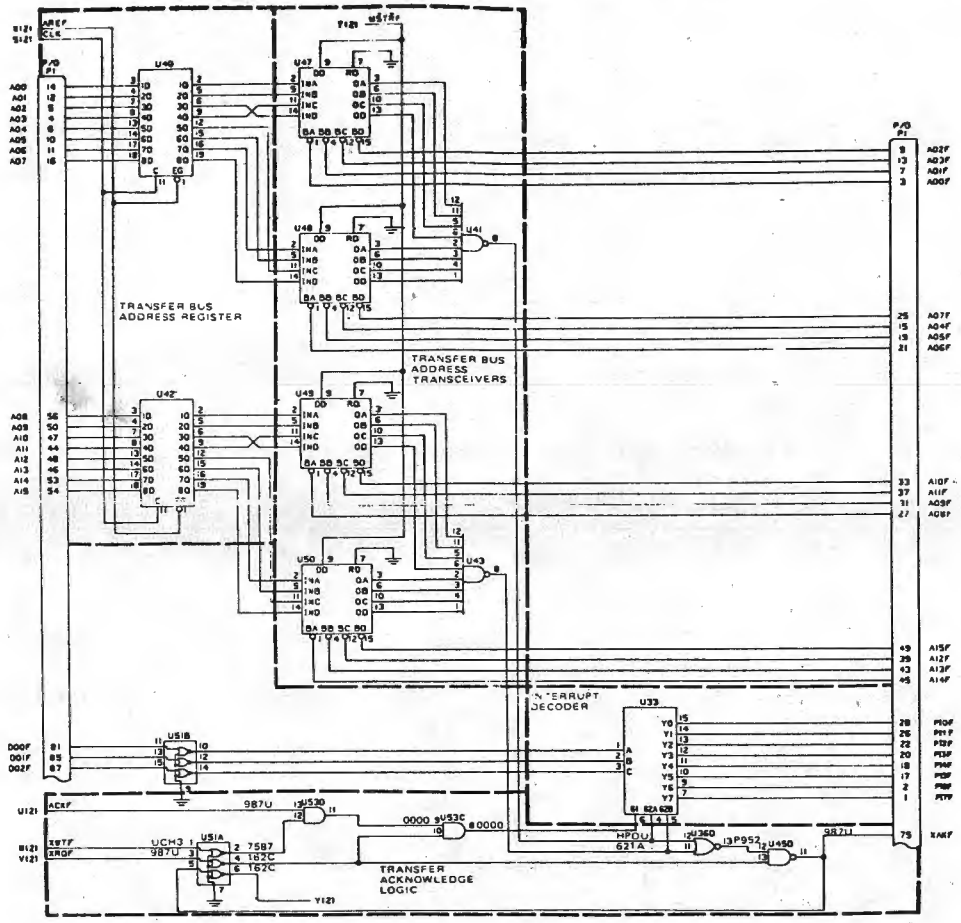
Circuit	Function	Pin	Fault	Effect
Microprocessor DU17 cont't.			High	Bit left-shifted into DU14 or right-shifted into DU17 always high. Multiplication results erroneous. FCC disconnect.
		Ro	Open	Bit right-shifted to shift/rotate multiplexer or input from shift/rotate multiplexer always low. Multiplication results erroneous. FCC disconnects.
			Gnd	Same as above.
		RO	High	Bit right-shifted to shift/rotate multiplexer or input from shift/rotate multiplexer always high.
		Q3	Open	Bit right-shifted into DU17 or left-shifted to DU14 always low. Multiplication results erroneous. FCC disconnects.
			Gnd	Same as above.
			High	Bit right-shifted into DU17 or left-shifted to DU14 always high. Multiplication results erroneous. FCC disconnects.
		Q0	Open	Bit right-shifted to shift/rotate multiplexer or left shifted from shift/rotate multiplexer always low. Multiplication results erroneous. FCC disconnects.
			Gnd	Bit right-shifted to shift/rotate multiplexer. or left-shifted from shift/rotate multiplexer always high. Multiplication results erroneous. FCC disconnects.
		F3	Any	No effect. Pin not connected.
		CN+4	Any	No effect. Pin not connected.
		OVR	Any	No effect. Pin not connected.

Circuit	Function	Pin	Fault	Effect
Micro-processor DU14	Microprocessor DU14 handles bits 4-7 of the 16-bit APS word in response to instructions from the control registers. Effect of fault as if fault had occurred on DU17, except that different bit positions in the CAPS word are affected.			
Micro-processor DU18	Microprocessor DU18 handles bits 8-11 of the 16-bit CAPS word in response to instructions from the control registers. Effect of pin faults is the same as if the fault had occurred on DU17, except that different bit positions in the CAPS word are affected.			
Micro-processor DU15	Microprocessor DU15 handles bits 12 - 15 of the 16-bit CAPS word in response to instructions from the control registers. Effect of pin faults is the same as if the fault had occurred in DU14, except that different bits are affected. Some differences result from the use of bit 15 as the sign bit in numerical computation.			

APPENDIX D - PROCESSOR SCHEMATIC DIAGRAMS

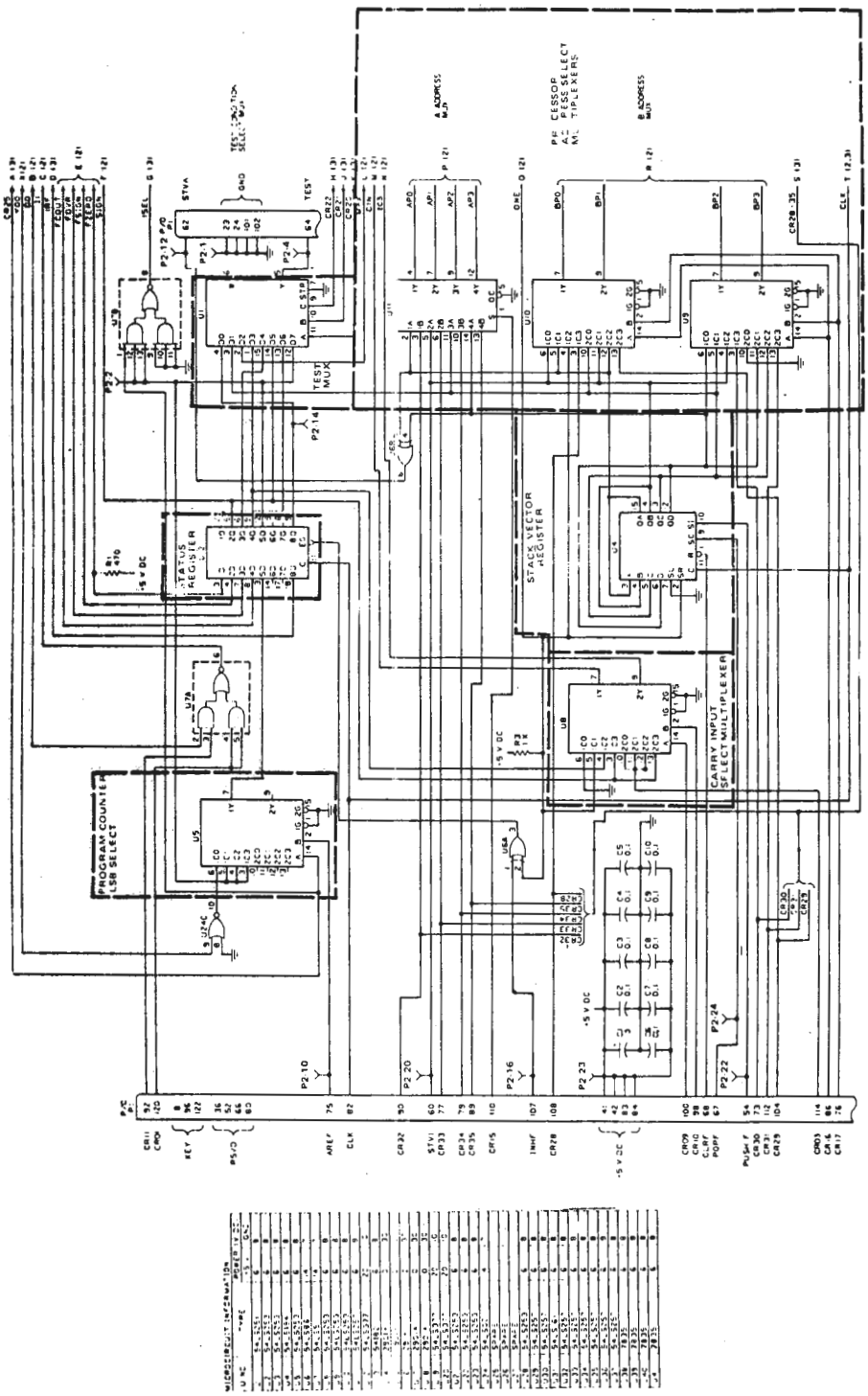


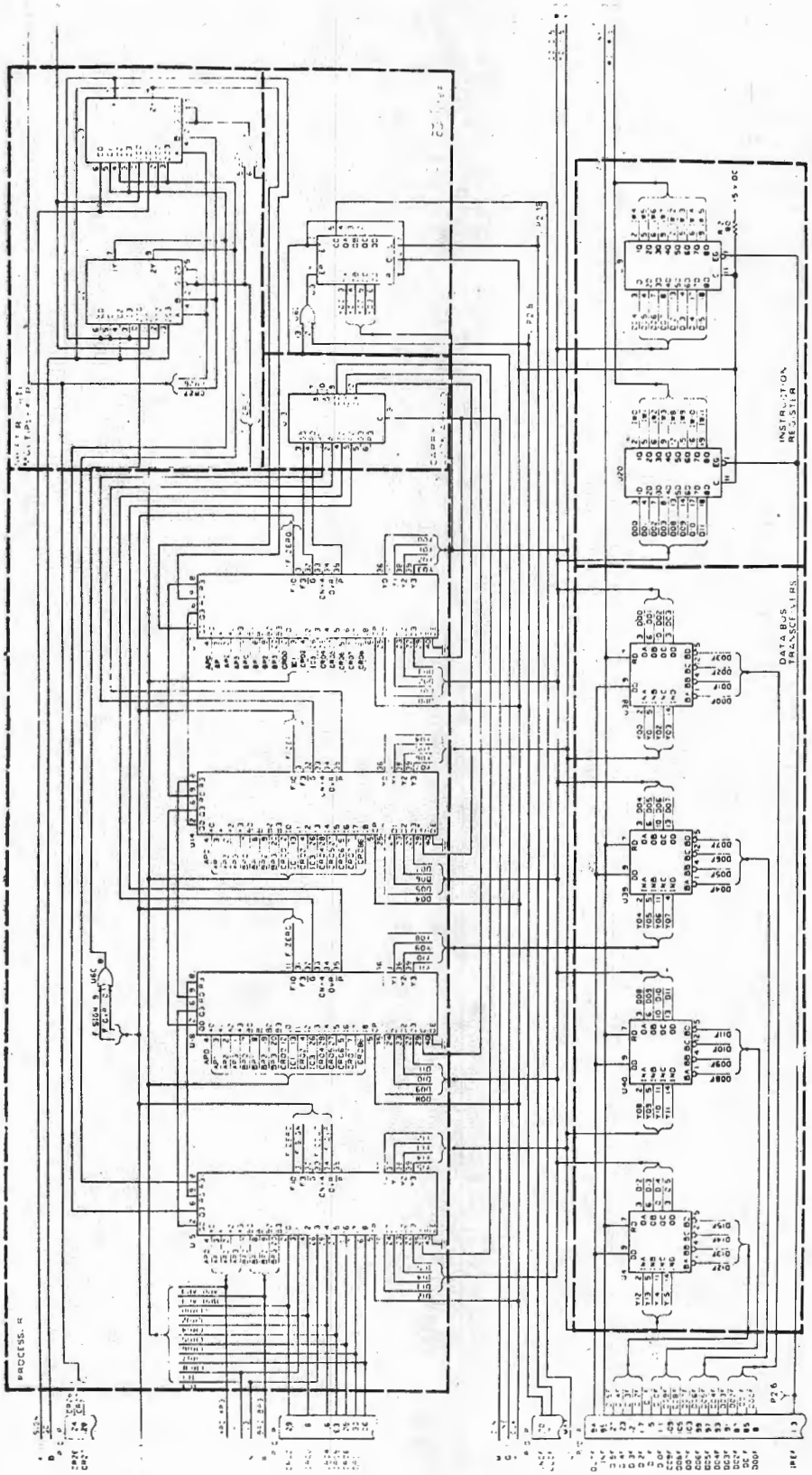


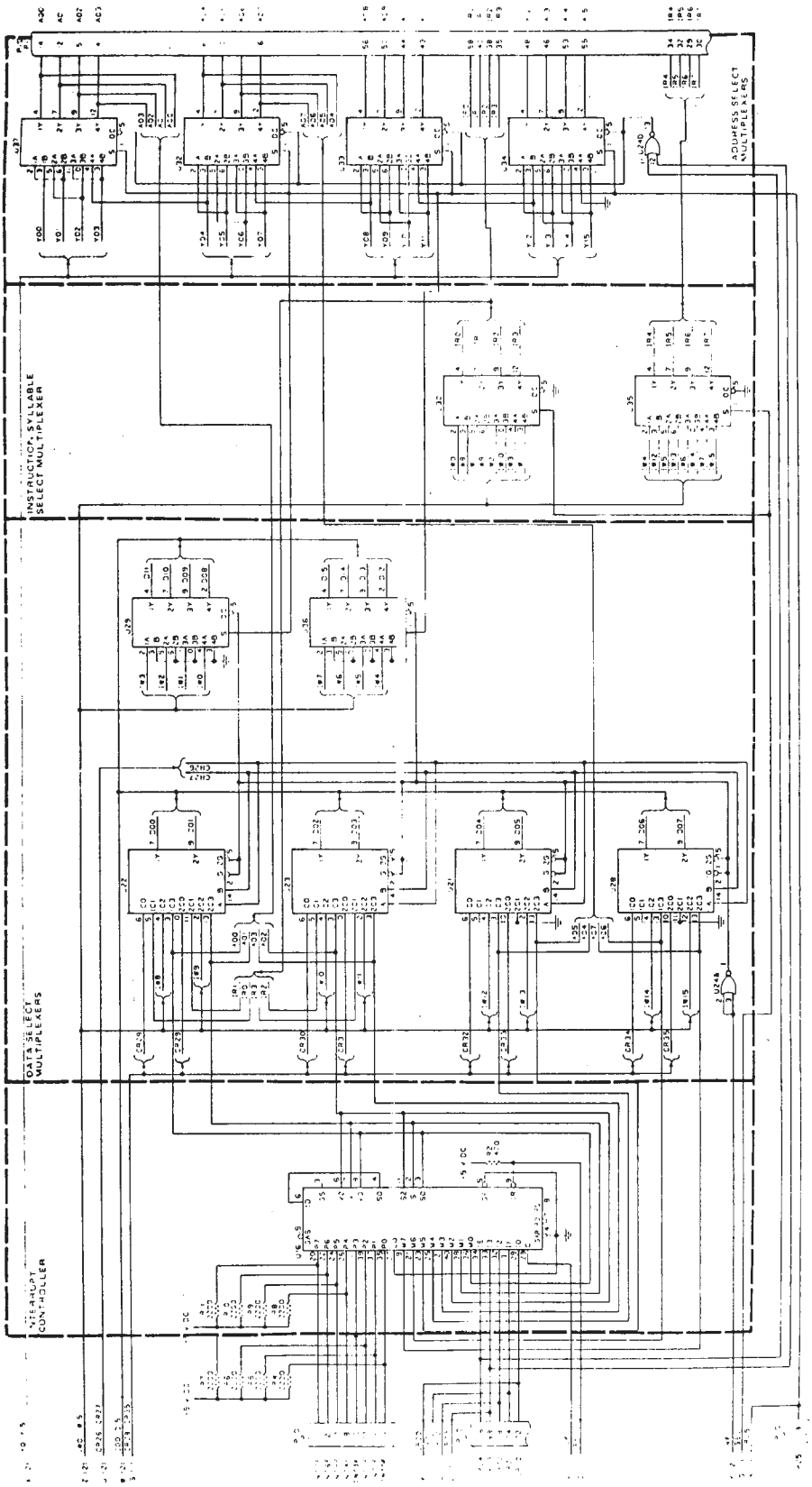


U NO.	TYPE	POWER DCI	
		+5	GND
U1	AM7643	18	9
U2	5404	14	7
U3	AM7643	18	9
U4	AM7643	18	9
U5	AM7643	18	9
U6	AM7643	18	9
U7	AM7643	18	9
U8	AM7643	18	9
U9	AM7643	18	9
U10	AM7643	18	9
U11	AM7643	18	9
U12	AM7643	18	9
U13	AM7643	18	9
U14	2911	2	8
U15	2911	2	8
U16	2911	2	8
U17	AM7643	18	9
U18	AM7643	18	9
U19	NOT USED		
U20	5474	14	7
U21	54LS174	16	8
U22	54LS175	16	8
U23	54LS174	16	8
U24	54LS377	20	10
U25	54LS377	20	10
U26	NOT USED		
U27	54LS112	16	8
U28	54LS00	14	7
U29	54LS74	14	7
U30	29LS18	16	8
U31	[illegible]	[illegible]	[illegible]
U32	29LS18	16	8
U33	54LS138	16	8
U34	54LS175	16	8
U35	54LC	14	7
U36	54LS02	14	7
U37	5453	14	7
U38	54LS22	14	7
U39	5474	14	7
U40	54LS377	20	10
U41	54LS32	14	7
U42	54LS377	20	10
U43	54LS30	14	7
U44	5438	14	7
U45	5438	14	7
U46	54LS38	16	8
U47	7835	16	8
U48	7835	16	8
U49	7835	16	8
U50	7835	16	8
U51	7832	16	8
U52	7832	16	8
U53	54LS08	14	7

NOTE SIGNATURES ARE FOR FAULT ISOLATION PROCEDURE PART 1


PROGRAM COUNTER LSB SELECT
STACK VECTOR REGISTER
CARRY INPUT SELECT MULTIPLEXER
STATUS REGISTER
TEST MUX
A ADDRESS MUX
B ADDRESS MUX
P2.10
P2.20
P2.16
P2.23
P2.24
P2.22
P2.14
-5 V DC


INSTRUCTION REGISTER
DATA BUS TRANSCEIVERS
PROCESS. II


INSTRUCTION SYLLABLE SELECT MULTIPLEXER
ADDRESS SELECT MULTIPLEXERS
DATA SELECT MULTIPLEXERS
INTERRUPT CONTROLLER

COMPONENT LEVEL FAULTS

CASE	ADDRESS VARIABLE	FB03 VG #1	FB01 $\dot{\theta}$ #1	3635 AP. ONE FAIL	3318 EXEC FAIL θ	FB03 VG #3	FB01 $\dot{\theta}$ #3	32E6 EXEC FAIL θ	331A EXEC FAIL $\dot{\theta}$	
	Strip Chnl	1	2			3				
1A	Vert. Gyro #1 X-Leg open inbound			0000 0001	0000 0003			0000 0003	0000 0003	Pin P1A-31, X-Leg V.G. #1 Open On Pin removal; ATS Disconnect warning - NO DUAL, NO ALIGN at 1500 ft.
1B	Vert. Gyro #1 Y-Leg open inbound			0000 0000 0000	0000 0003 0000			0000 0003 0000	0000 0000 0000	On pin removal ATS disconnect; prior to A/L track AP.ONEFAIL was cleared.
2A	Vert. Gyro #1 hard shift to fixed value inbound			0000 0001	0000 0003			0000 0003	0000 0000	First fault was not of sufficient magnitude to fault sys., fault level was increased resulting in the CTA values shown, NO DUAL was annunciated upon engagement of A/L TRK.
2B	Same as 2A but in Al. Arm			0000 0001 0000	0000 0003 0003			0000 0003 0003	0000 0000 0000	ATS Disc. WRN. on insertion NO DUAL did not light. AP.ONEFAIL reset.
2C	Same as 2A but in AL. TRK above 150 ft.			0000 0000	0000 0003			0000 0003	0000 0000	ATS. dropped out on insertion; no ATS warning; NO DUAL did not annunciate
2D	Same as 2A but in AL. TRK below 150 ft.			0000 0000	0000 0003			0000 0003	0000 0000	NO DUAL did not annunciate
2E	Vert. Gyro #1 Validity false inbound			0000 0001	0000 0000			0000 0000	0000 0000	NO DUAL. did not annunciate

COMPONENT LEVEL FAULTS

CASE	ADDRESS / VARIABLE	FB03 VG #1	FB01 $\dot{\theta}$ #1	3635 AP. ONE FAIL	3318 EXEC FAIL θ	FB03 VG #3	FB01 $\dot{\theta}$ #3	32E6 EXEC FAIL θ	331A EXEC FAIL $\dot{\theta}$	
	Strip Chat									
3A	Vert. Gyro #1 Open X-Leg In AL.ARM			0000 1	0000 0000			0000 0000	0000 0000	Disconnect of ATS on insertion; No dual annunciated.
3B	Vert. Gyro #1 Open Y-Leg In AL.ARM			0000 1 0	0000 3 3			0000 3 3	0000 3 0	ATS Disc. on Insertion. No dual annunciated. Exec. Fail Resets
4A	Vert. Gyro #1 Open X-Leg In AL.TRK above 150 Ft.			0000 1	0000 3			0000 3	0000 3	No dual annunciation, ATS Disc. Annun. ATS Disc.
4B	Vert. Gyro #1 Open Y-Leg AL.TRK above 150 Ft.			0000 1	0000 3			0000 3	0000 3	No dual & ATS Disc.annunciated, ATS Disc.
5A	Same as 4A but below 150 Ft.			0000 1	0000 3			0000 3	0000 3	ATS Disconn. without annunciation. No dual not indicated.
5B	Same as 4B but below 150 Ft.			0000 0	0000 3			0000 3	0000 0	No dual did not indicate; ATS Disc. & Ind.

COMPONENT LEVEL FAULTS

CASE	ADDRESS / VARIABLE	FB03 VG #1	FB01 $\dot{\theta}$ #1	3635 AP. ONE FAIL	3318 EXEC FAIL θ	FB03 VG #3	FB01 $\dot{\theta}$ #3	32E6 EXEC FAIL θ	331A EXEC FAIL $\dot{\theta}$	
	Strip Chnl									
6	Vert. Gyro #1			0000	0000			0000	0000	No dual flashed but did not
	Ramp Up			1	3			3	0	latch.
	Inbound			0	3			3	0	3635 reset when no dual flashed.
7	Vert. Gyro #1			0000	0000			0000	0000	No dual indicated.
	Ramp Down			1	3			3	0	
	Inbound									
8A	Vert. Gyro #3			0000	0000			0000	0000	
	Open X-Leg			1	2			2	0	
	Inbound			0	2			2	0	At 1500 Ft. No dual lit
8B	Vert. Gyro #3			0000	0000			0000	0000	No dual flashed, Reset 3635,
	Open Y-Leg			1	2			2	0	ATS Disc. Warning but ATS stayed.
	Inbound			0	2			2	0	

COMPONENT LEVEL FAULTS

CASE	ADDRESS VARIABLE	FB1F N.A.#1	3380 N.A.EXEC. FAIL	3635 ARONE FAIL	FB1F N.A.#3	334E EXEC. FAIL	
	Strip Chnl	1			3		
9A	Norm. Accel. #1 Open signal Inbound		0000 0	0000 0		0000 0	ATS held in; ATS Disc. warning at A/L TRK.
			0000 3	0000 1		0000 3	Case 9A repeated with turbulence; Fault detected & NO DUAL
9B	Norm. Accel. #1 Open Gnd. Inbound		0000	0000		0000	Case 9B involved signal gnd. Pulling pin has no effect.

COMPONENT LEVEL FAULTS

FCC # 1 CTA WINDOW

CASE	ADDRESS VARIABLE STRIP CHNL.	A1 FB07 ROLL RATE 1	A2 FB05 ROLL RATE 1	A3 3635 A.P.ONE FAIL	A4 33E8 EXEC. FAIL ROLL	B1 FB07 ROLL RATE 3	B2 FB05 ROLL RATE 3	B3 3386 EXEC. FAIL ROLL	B4 33EA EXEC. FAIL ROLL	
10A	Roll Gyro #1 Open X-Leg in AL..ARM			0000 1	0000 3			0000 3	0000 3	No dual at A/L TRK
10B	Roll Gyro #1 Open Y-Leg in AL.ARM			0000 1	0000 3			0000 3	0000 0	No dual at A/L TRK
11A	Roll Gyro #1 Ramp Up Inbound			0000	0000			0000	0000	No dual at A/L TRK
11B	Same as 11A but in AL..ARM			0000 1	0000 3			0000 3	0000 0	No dual at 1100 FT. (When Comparators Tripped).
11C	Same as 11A			0000 1	0000 3			0000 3	0000 0	No dual at 1100 FT. (When Comparators Tripped).
11D	Same as 11A but in AL..TRK below 150 Ft.			0000 1	0000 3			0000 3	0000 0	Landing Completed Without No Dual Indication.

COMPONENT LEVEL FAULTS

FCC # 1 CTA WINDOW

CASE	ADDRESS VARIABLE STRIP CHNL	A1 FB1B LOC.DEV. # 1	A2 339A EXEC.FAIL LOC.DEV.	A3	A4 33E8	B1 FB1B LOC.DEV. # 1	B2 3368	B3	B4 33EA	
12A	No Localizer Output In Inbound		0000 6003 4003		0000 0		0000 6003 4003		0000 0	No dual at AL.TRK
12B	Same as 12A but In AL.ARM		0000 4003		0000 0000		0000 4003		0000 0000	No dual indicated when fault detected (at 1000 Ft.)
12C	Same as 12A but In AL.TRK above 150 Ft.		0000 4003		0000 0000		0000 4003		0000 0000	No dual indicated when fault detected (at 1000 Ft.)
12D	Same as 12A but in AL.TRK below 150 Ft.		0000 6003		0000 0000		0000 6003		0000 0000	Fault detected; Land completed; No dual did not illuminate.
14A	Localizer No. 1, Validity 1 in AL. ARM		0000 4013		0000		0000 4013		0000	No dual at AL.TRK
14B	Localizer No. 1, Validity 2 in AL. ARM		0000 4023		0000		0000 4023		0000	No dual at AL.TRK
16	Lateral Accel. No. 1 Ramp Up in AL. ARM	FB1D	33B4	33B4 EXEC.FAIL 0000 400A	3635 AP. ONE FAIL 0000	FB1D		3382 EXEC.FAIL 0000 400A	3635 AP.ONE FAIL 0000	No dual at 1100 Ft. when comparator tripped.

COMPONENT LEVEL FAULTS

FCC #1 CTA WINDOW

CASE	ADDRESS VARIABLE STRIP CHNL.	A1	A2	A3 3364 AP.TWO FAIL	A4 3365 AP.ONE FAIL	B1	B2	B3 3364 AP.TWO FAIL	B4 3365 AP.ONE FAIL	
1/A	Pitch Servo Coil Discrete Fault Inbound			0000	0000			0000	0000	Instant disengage; No dual at A/L. TRK. Servo simulator panel pitch coil switch to fault. Box 2 engaged first.
1/B	Same as 1/A but In AL.ARM			0000 0000	0000 0000			0000 0000	0000 0000	Both bathandles dropped.
1/B	Rerun									On 3rd retry, only affected channel disengaged, no dual at A/L. TRK.
1/C	Same as 1/A, but In AL.TRK above 150 Ft.									Affected FCC disengaged, No dual
1/D	Same as 1/A, but In AL.TRK below 150 Ft.									Affected FCC disengaged; No dual not indicated.
18A	Pitch Servo Coil Current Ramp Up Inbound									Affected channel disengaged; No dual at A/L.TRK
19A	Roll Servo Discrete Open Inbound									
19B	Same as 19A but In AL.ARM									Affected bathandle dropped; No dual indicated.
19C	Same as 19A but In AL.TRK above 150 Ft.									Affected bathandle dropped; No dual indicated.
19D	Same as 19A but In AL.TRK below 150 Ft.									Affected bathandle dropped; No dual not indicated.

Case	Circuit	Pin No.	Pin Function	Open Pin Faults
20a	DU18 2901 No.3 (data bits 8-11)	24	Data Input Bit 1	Upon opening, both FCC's disengaged. NORMAL-STANDBY switch was in STANDBY.
20		11	F = 0	Faulted FCC disengaged. SPLIT and NO DUAL annunciated. Faulted pin transmits signal to the status register.
20b		4	Address bit A_0	Faulted FCC disengaged. SPLIT, NO DUAL, and NO ALIGN annunciated at initiation of AL TRK.
20c		17	Address bit B_0	Both FCC's disengaged. AP DISC and SPLIT annunciated.
20d		37	Data Output bit 1	Faulted FCC disengaged. AP DISC and SPLIT annunciated on disengagement. NO DUAL annunciated at AL TRK.
21	CU16	1		Fault had no effect on computer operation. Pin used only in reset of stack vector and transfer bus access control registers.
22a	CU30 Control Register 29LS18	1	Data Input bit 0	Faulted FCC disengaged. NO DUAL, NO ALIGN, and SPLIT annunciated. Pin 1 is coupled to Pin 2 (CR35, see fault following) and, when the next address control prom sets OE low, to direct input bit DI of microprogram sequencer CU14.
22b		2	Data Output bit Q_0	Faulted FCC disengaged. SPLIT, NO DUAL annunciated. Faulted bit drives control line CR35, which is address bit A3 of the 2901's when the processor address multiplexer couples CR35 to address line A03.
22c		10	Data Output bit Y2	Faulted FCC disengaged. SPLIT, NO DUAL, and NO ALIGN annunciated. Faulted pin is direct input bit D3 to microprogram sequencer CU15 when bit D3 is not being controlled by instruction mapper prom CU7. In turn, CU15 outputs this bit as address bit A7 to the control store proms when CU1E is in direct address mode.

Case	Circuit	Pin No.	Pin Function	Open Pin Faults
22a	CU10	4	Data Input bit D1	Faulted FCC disengaged; other FCC went to CWS. CMD DISC annunciated. Control line CR34 is latched to bit D1 on rising clock pulse, and when next address control prom sets OE low, to direct input bit D0 of microprogram sequencer CU14. When selected by data select multiplexer DU28, CR34 is used as data bit D06.
22c		4		In a repeat of previous case, faulted FCC disengaged. Other FCC stayed in CMD. SPLIT and NO DUAL annunciated.
22f		12	Data Input bit D2	Faulted FCC disengaged, other FCC stayed in CMD. NO DUAL and SPLIT annunciated. Control line CR33 is latched to Pin 12 on rising clock pulse. CR33 is used as processor A address bit AP1 when connected by the A address multiplexer. Also, CR33 can be coupled to processor input data bit D05 by data select multiplexer DU21.
23a	CU15 Micro-program Sequencer 2911	9	Not Zero	Faulted FCC disengaged. SPLIT and NO DUAL annunciated at ALIGN point in landing. Pin 9 forces all outputs of CU15 to zero when it is low. Open pin prevented HI signal from control register CU21 from reaching CU15, zeroing all outputs and causing erroneous address to control store memory.
23b		19	Not FE	Faulted FCC disengaged. Processor halted. System failed to capture glide slope. The FE signal is one of four used to control the operation of the 2911 micro-program sequencer. In most combinations of the signals, the absence of the FE signal causes a push or pop of a counter stack in addition to a jump.
23c		19	Not FE	Repeat of previous fault. Faulted FCC disengaged. Other FCC stayed in CMD. SPLIT and NO DUAL annunciated.
23d		20	PUP	Faulted FCC disengaged. NO DUAL and SPLIT annunciated. PUP is Push/Pop control signal. Open pin prevents pushing the microprogram counter contents onto the internal stack.

Case	Circuit	Pin No.	Pin Function	Open Pin Faults
23e	CU15 Microprogram Sequencer 2911	10	S0 Address Source Selection Control	Faulted FCC disengaged. NO DUAL and SPLIT annunciated. S0 is one of the four signals used in selecting the source of the next address. S0 open generally results in wrong source producing a jump to the wrong address.
23f		5	D2 Direct Input bit D2	Faulted FCC disengaged. NO DUAL and SPLIT annunciated. D2 is one of four bits which can be selected as the output of the 2911. Fault would cause the wrong control store memory address on selection of direct input when the bit should be high.
23g		15	Y3 Output bit	Faulted FCC disengaged. NO DUAL and SPLIT annunciated. Y3 is one of four output bits of the 2911. This bit being open causes the wrong microinstruction to be selected whenever this bit should be high.

Case	Circuit	Pin No.	Pin Function	Grounded Pin Faults
24	CU2 Hex Inverter	8	Inverter Output	Faulted FCC disengaged. Processor stopped. This signal fans out to several points, including NAND gate CU35C, which outputs the Read Enable signal to the data bus transceivers. NAND output is stuck high so that processor cannot read the CAPS data bus.
25	CU36 Quad. NOR Gate	4	Gate Output	Faulted FCC disengaged. NO ALIGN, SPLIT, and NO DUAL annunciated. Faulted pin being stuck low results in NAND gate U35A being stuck high, disabling the data bus transceivers from writing on the CAPS data bus.
26	CU28 Quad. NAND Gate	3	Gate Output	Faulted FCC disengaged. NO ALIGN, SPLIT, and NO DUAL annunciated. Processor stopped. Fault results in BROF being stuck high, so that processor cannot access CAPS bus. Also, all interrupt inputs to the interrupt controller are set high.
27	CU36 Quad. NOR Gate	11	Gate Output	Faulted FCC disengaged. NO ALIGN, SPLIT, and NO DUAL annunciated. Processor stopped. Fault results in the processor being unable to transmit XAKF (transfer acknowledge) on the CAPS control bus. XAKF is stuck high.
28	DU2 Shift/ Rotate Multi-plexer	2	Control Input	Faulted FCC disengaged. SPLIT, NO ALIGN, NO DUAL annunciated. Faulted processor stopped. Fault causes wrong data to be inserted into microprocessor in some shift operations.
		14	Control Input	Faulted FCC disengaged. Fault causes wrong data to be inserted into microprocessors during some shift operations.

Grounded Pin Faults

Case	Circuit	Pin No.	Function
29	DU17 Micro-processor	36	Y00
		37	Y01
		38	Y02
		39	Y03
30	DU14 Micro-processor	36	Y04
		37	Y05
		38	Y06
		39	Y07
31	DU18 Micro-processor	36	Y08
		37	Y09
		38	Y10
		39	Y11
32	DU15 Micro processor	36	Y12
		37	Y13
		38	Y14
		39	Y15

In each case, the faulted FCC disengaged. The faulted processor halted immediately. The y pins are the processor output pins for computed data. Under certain conditions, processor output is a memory address which is connected to the CAPS address bus, rather than data. Corruption of addresses is apparently the cause of the immediate processor halts.

Case	Circuit	Pin No.	Pin Function	Grounded Pin Faults
33	D017 Micro-processor	32	$\bar{G}$	
		35	$\bar{P}$	
34	D014 Micro-processor	32	$\bar{G}$	Faulted FCC disengaged. Faulted processor halted immediately.
		35	$\bar{P}$	
35	D018 Micro-processor	35	$\bar{P}$	
36	D016 Interrupt Controller 2914	16	V2	Faulted FCC disengaged. Faulted processor stopped immediately. V2 is the most significant bit of the interrupt vector output of the 2914. This bit is also address bit A02 of the CAPS address bus when the vector output is enabled, and is hard-wired to address line A02.
37		17	V1	Faulted FCC disengaged. Faulted processor stopped immediately. V1 is the middle bit of the three-bit interrupt vector of the 2914. This pin is hard-wired to CAPS address bus line A01.
38		18	V0	Faulted FCC disengage. Faulted processor stopped immediately. V0 is the least significant bit of the interrupt vector output of the 2914. This pin is hard-wired to CAPS address bus line A00.
39		28	I0	Faulted FCC disengaged. Faulted processor stopped immediately. I0 is a micro-instruction bit to the 2914.
40		31	I1	Faulted FCC disengaged. Faulted processor stopped immediately. I1 is a micro-instruction bit to 2914.
41		32	I2	Faulted FCC disengaged. Faulted processor stopped immediately. I2 is a micro-instruction bit to the 2914.

GROUNDED PIN FAULTS

Case	Circuit	Pin No.	Function	
42a	DU16 Interrupt Controller 2914	34	Instruction Enable	Faulted FCC disengaged. Faulted processor halted. Pin 34 is a logic-low instruction enable which should only go low when the instruction lines I0-I3 have been set. The pin stuck low causes the 2914 to read erroneous instructions.
42b		26	P4 Interrupt Request	Faulted FCC disengaged. Faulted processor halted. Pin 34 is a logic-low interrupt request. With the fault inserted, an interrupt request at priority 4 is generated whenever the corresponding mask bit is not set and a higher priority unmasked interrupt is not present.
42c		39	P2 Interrupt Request	Faulted FCC disengaged. Faulted processor halted. This is the same situation as in the previous case, except at a lower priority level.
42d		20	P7 Interrupt Request	Faulted FCC disengaged. Faulted processor halted. This is the same situation as in the previous two cases, except at the highest priority level.
42e		25	M4 Mash Bit	Faulted FCC disengaged. Faulted processor halted. This fault prevents priority Level 4 interrupts from being masked.
42f		19	M7 Mask Bit	Faulted FCC disengaged. Faulted processor halted. This fault prevents highest priority interrupts from being masked.

FCC #1 CTA WINDOW

CASE	ADDRESS VARIABLE	A1	A2	A3	A4	B1	B2	B3	B4	
				AP. ONE FAIL	AP.TWO FAIL					
	STRIP CHNL			3635	3634					
43	Open Roll Gyros 1 & 2 in AL.ARM			0000 0001	0000 0001 0000					Disconnected on Second Fault. 3634 flashed 0001 before reverting to 0000.
44	Open roll Gyros 1 & 3 to FCC #1 in AL.ARM									Sensors 2 & 3 still valid into box 2. No disconnect; No dual at AL.TRK.
45	Ramp Up Vert. Gyro #1 in APP; Open Vert. Gyro #2 in AL.ARM									Two sensors lost; both boxes disengaged.

BIBLIOGRAPHY

AC 25, 1309-1, System Design Analysis, Federal Aviation Administration, September 7, 1982.

AC 120-28C, Criteria for Approval of Category III Landing Weather Minima, Federal Aviation Administration, March 9, 1984.

Bjurman, B. E., et al., Airborne Advanced Reconfigurable Computer System (ARCS), NASA-CR-145024, Prepared for Langley Research Center, NASA by Boeing Commercial Airplane Company under contract NAS1-13654, August 1976.

Conn, R. B., et al, "CAST - A Complementary Analytic - Simulative Technique for Modeling Complex, Fault-Tolerant Computing Systems," AIAA Computers in Aerospace Conference, Los Angeles, California, November 1977.

DOT/FAA/CT-82/140, Digital Flight Control System Validation Technology Assessment, July 1982.

DOT/FAA/CT-82/154, Integrated Assurance Assessment of a Reconfigurable Digital Flight Control System, Volume I, April 1983.

Hitt, E. and D. Eldredge., "A Review and Application of Analytical Models in Fault Tolerant Avionics Validation," Proceedings of IEEE/AIAA 5th Digital Avionics Systems Conference, Fall 1983.

Karlin, S., A First Course in Stochastic Processes, Academic Press, 1966.

Lockheed-Georgia Company Engineering Report LG81E0126, Simulator Investigation Plan for Digital Flight Controls Validation Technology, as revised 10 April 1981.

MIL-M-38510, Microcircuits, General Specification for.

Military Standardization Handbook 217C, Reliability Prediction of Electronic Equipment, United States Air Force, Rome Air Development Center, 9 April 1979, with Notice 1, Supplement, 1 May 1980.

MIL-STD-1629A, Procedures for Performing a Failure Mode, Effects and Criticality Analysis.

Mulcare, D. B., et al., Industry Perspective on Simulation Methods for Validation and Failure Effects Analysis of Digital Flight Control Avionics, NASA CR-152234, Moffett Field, California, February 1979.

NASA RDFCS System Interface Document, April 8, 1981.

NASA Report prepared for Langley Research Center by Raytheon Company, Contract NAS1-12668, Reliability Model Derivation of a Fault-Tolerant, Dual, Spare-Switching Digital Computer System, March 1974.

Ng, Y. W., and A. Avizienis., ARIES 76 User's Guide, National Science Foundation Report No. NSF-MCS-7203633-78944, University of California, Los Angeles, Report No. UCLA-ENG-7894, December 1978.

Nuclear Regulatory Commission Report NUREG-0492, Fault Tree Handbook.

RTCA Document DO-178, Software Considerations in Airborne Systems and Equipment Certification, November 1981.

ACRONYMS AND ABBREVIATIONS

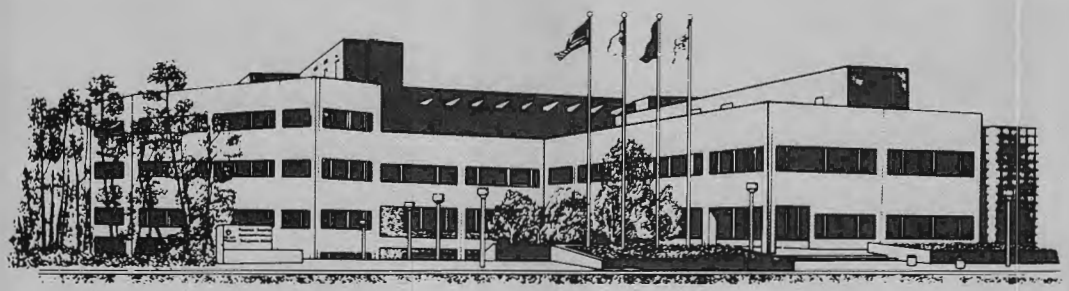
A/L	Approach/Land
AC	Alternating Current
AC	Advisory Circular
ADI	Automatic Direction Indicator
ALU	Arithmetic Logic Unit
ARIES	Automated Reliability Interactive Estimation System
ARINC	Aeronautical Radio Incorporated
CAPS	Computer Aided Production Simulator
CAST	Complementary Analytic Simulative Technique
CARE	Computer Aided Reliability Evaluator
CARSRA	Computer-Aided Redundant System Reliability Analysis
CTA	CAPS Test Adapter
DFCS	Digital Flight Control System
DOT	Department of Transportation
DSP	Discrete Switch Panel
EFMA	Executive Failure My A
EFMB	Executive Failure My B
EFOA	Executive Failure Other A
EFOB	Executive Failure Other B
EFW	Executive Failure Word
FAA	Federal Aviation Administration
FAR	Federal Acquisition Regulation
FCC	Flight Control Computer
FMEA	Failure Mode and Effect Analysis
HSI	Horizontal Situation Indicator
Hz	Hertz
IAA	Integrated Assurance Assessment
IC	Integrated Circuit
ILS	Instrument Landing System
LVDT	Linear Voltage Differential Transducer
mA	Milliampere
MDICU	Modular Digital Interface Control Unit
NA	Normal Accelerometers
NASA	National Aeronautics and Space Administration
PROM	Programmable Read-Only Memory
RAM	Random Access Memory
RBDCP	Reliability Block Diagram Computer Program
RDFCS	Redundant Digital Flight Control System
REL	Reliability
REL COMP	Reliability Computers
RTCA	Radio Technical Commission for Aeronautics
SSP	Servo Simulation Panel
TASRA	Tree Aided System Reliability Analysis
WAI	Warning Annunciation Indicator
XMTR	Transmitter

U.S. Department of Transportation
Federal Aviation Administration

DOT/FAA/CT-88/10

HANDBOOK-VOLUME II
DIGITAL SYSTEMS VALIDATION

CHAPTER 4
QUADRUPLEX DIGITAL FLIGHT CONTROL SYSTEMS



PREPARED BY:

COMPUTER RESOURCE MANAGEMENT, INC.
950 HERNDON PARKWAY, SUITE 360
HERNDON, VIRGINIA 22070

PREPARED FOR:

FEDERAL AVIATION ADMINISTRATION
TECHNICAL CENTER
ATLANTIC CITY INTERNATIONAL AIRPORT, NEW JERSEY 08405

NOTICE

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The United States Government assumes no liability for the contents or use thereof.

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the objective of this report.

TABLE OF CONTENTS

Section		Page
1.	INTRODUCTION	4-1
2.	BACKGROUND	4-3
3.	THE ASSURANCE ASSESSMENT PLAN	4-13
4.	THE USE OF SIMULATION AS AN ASSURANCE TOOL	4-19
	4.1. Assurance Assessment	4-19
	4.2. Digital Flight Control System Assurance Methodology	4-19
	4.3. Simulation Development	4-19
	4.4. Basic System Architecture	4-20
	4.5. Developing the Flight Control System	4-24
	4.6. Establishing System Reliability	4-29
	4.7. The Airplane Simulation	4-35
5.	VERIFYING THE SIMULATION	4-41
6.	SUMMARY	4-49
BIBLIOGRAPHY		4-53
GLOSSARY		4-55
ACRONYMS AND ABBREVIATIONS		4-58

LIST OF ILLUSTRATIONS

Figure		Page
2-1	BASELINE SYSTEM ARCHITECTURE	4-4
2-2	QUADRUPLEX DIGITAL FLIGHT CONTROL SYSTEM ARCHITECTURE	4-5
2-3	DIGITAL FLIGHT SYSTEM LIFE CYCLE	4-6
2-4	RECONFIGURABLE DIGITAL FLIGHT CONTROL SYSTEM FACILITY SETUP (FLIGHT TEST PALLET)	4-8
2-5	RECONFIGURABLE DIGITAL FLIGHT CONTROL SYSTEM FACILITY SETUP (SIMULATION SETUP)	4-9
2-6	INTERDEPENDENCE OF ASSURANCE METHODS	4-10
2-7	MULTILEVEL TESTING	4-11
3-1	ARCHITECTURAL DESIGN TASKS	4-14
3-2	MULTILEVEL-TESTING CLOSURE	4-16
3-3	RECONFIGURABLE DIGITAL FLIGHT CONTROL SYSTEM ALL-UP SIMULATION COMPONENTS	4-17
4.3-1	CONTROL LAW USED IN REDUNDANT FLIGHT CONTROL SYSTEM CHANNELS	4-23
4.5-1	TOP-LEVEL DIGITAL FLIGHT CONTROL SYSTEM CHANNEL LOGIC DESIGN	4-25
4.5-2	SYNCHRONIZATION PREDICATE/TRANSITION NETWORK	4-27
4.5-3	PARTIAL PREDICATE/TRANSITION NETWORK	4-28
4.5-4	TYPICAL DISCRETE-EVENT SIMULATION RESULTS	4-30
4.5-5	CONTROL GRAPH OF TOP-LEVEL SOFTWARE	4-31
4.6-1	RELIABILITY MODEL FOR IMPLEMENTED ARCHITECTURE	4-32
4.7-1	DIAGRAM OF STATE-VARIABLE AIRPLANE SIMULATION	4-36
5-1	AUTOMATED TEST SCENARIO	4-43
5-2	IMPLEMENTATION OF DIGITAL FLIGHT CONTROL SYSTEM SIMULATOR FACILITY	4-45
5-3	MECHANIZATION OF SIMULATOR TESTING	4-46
6-1	CROSS-CHANNEL SYNCHRONIZATION TIME-HISTORIES	4-50

LIST OF TABLES

Table		Page
3-1	DIGITAL FLIGHT CONTROL SYSTEM TESTING SCENARIO	4-15
3-2	RECONFIGURABLE DIGITAL FLIGHT CONTROL SYSTEM FACILITY CAPABILITIES	4-18
4.3-1	DEVELOPMENT PHASE ACTIVITIES AND PRODUCTS	4-21
4.3-2	STABILITY AUGMENTATION GAIN SCHEDULING	4-24
4.5-1	CYCLING STATE LOOP TIMING	4-26
4.6-1	DIGITAL FLIGHT-CONTROL SYSTEM RELIABILITY ASSESSMENTS	4-33
4.6-2	DEGRADATION OF FLYING-QUALITIES: 5-HOUR MISSION	4-34
4.6-3	DEGRADATION OF FLYING-QUALITIES: 10-HOUR MISSION	4-34
4.7-1	RELAXED STATIC STABILITY FLIGHT CASES	4-37
5-1	AUTOMATED TESTING TOOLS	4-44
5-2	MULTILEVEL TEST CASES AND MODES	4-47

1. INTRODUCTION

The purpose of this chapter is to provide background information pertinent to assurance assessments of quadruplex Digital Flight Control Systems (DFCS), to identify and discuss some of the current problems associated with assurance assessments, and to present related evaluation approaches and techniques.

Assurance assessment is a collective phrase that covers design correctness, system verification, and system validation. By ensuring that a proposed system functions according to design specifications, assurance assessments serve as tracking guidelines during system development.

The material contained in this chapter relies heavily on the development and validation of a double fail-operational DFCS architecture reported in Quadruplex Digital Flight Control Assessment, DOT/FAA/CT-86/30 (D. B. Mulcare, L. E. Downing, and M. K. Smith). This report serves as the basis for this presentation, and generous use has been made of its data, text, and illustrations.

2. BACKGROUND

The first DFCSs were introduced in the mid-1970s. They used fairly conventional architectures and were largely copies of prior analog systems. These systems used parallel replicated lanes with sensors fanning into the processors, which in turn fanned out to actuators or displays. Figure 2-1 shows an expansion of this architecture for the pitch-axis of an Augmented Fly-By-Wire (AFBW) function. Except for cross-channel communication, all signal paths are dedicated analog paths. Fan-in is minimized by having one set of sensors feed directly into each computational channel. In figure 2-1, each channel broadcasts its received sensor inputs to the other three channels.

These early Flight Control Systems (FCS) were flight-critical for a very limited time (e.g., autoland). By 1975, the Federal Aviation Administration (FAA) recognized the trend from analog systems to their digital implementations. The FAA also recognized the challenges that would accompany this transition to a new technology (Reed and Boothe, 1977). More recently, the FAA has identified a need to become familiar with the emerging full-time, full-authority, "flight-critical" fly-by-wire/light systems (Larsen and Carro, 1986). Besides the functional criticality of the systems, the concepts of envelope limiting, flying-qualities management, and system reconfiguration without loss of redundancy during the processing of hard and soft faults (transparent recovery) have brought new issues to the forefront. As a result, new areas of concern have emerged as fault-tolerant systems come online.

Advanced systems become interdisciplinary as their complexity increases. The overall aspects of certification become more system-integrity oriented. As a result, there is a need to formulate a uniform approach to assurance assessments. From a reliability point of view, analytical assessments of critical functions are demanding and tedious. They include the evaluation of the dissimilar versions of hardware and software used in the various lanes of redundancy, the need for backup systems, faulted recovery techniques, status instrumentation, and pilot workload.

Quadruplex Digital Flight Control Assessment (D. B. Mulcare, L. E. Downing, and M. K. Smith) defines a double fail-operational DFCS that was designed, analyzed, implemented, and validated relative to system fault-tolerance (figures 2-2 and 2-3). In this chapter, a subset of the DFCS system, the pitch-axis control functions, will be used to demonstrate important portions of an integrated approach to assurance assessment. The system-level, fault-tolerant design was developed around an engineering development model of the L-1011 FCS and will be verified using a predicate/transition network simulation (Barton, Mulcare, and LeBlanc, AIAA Paper, 1985).

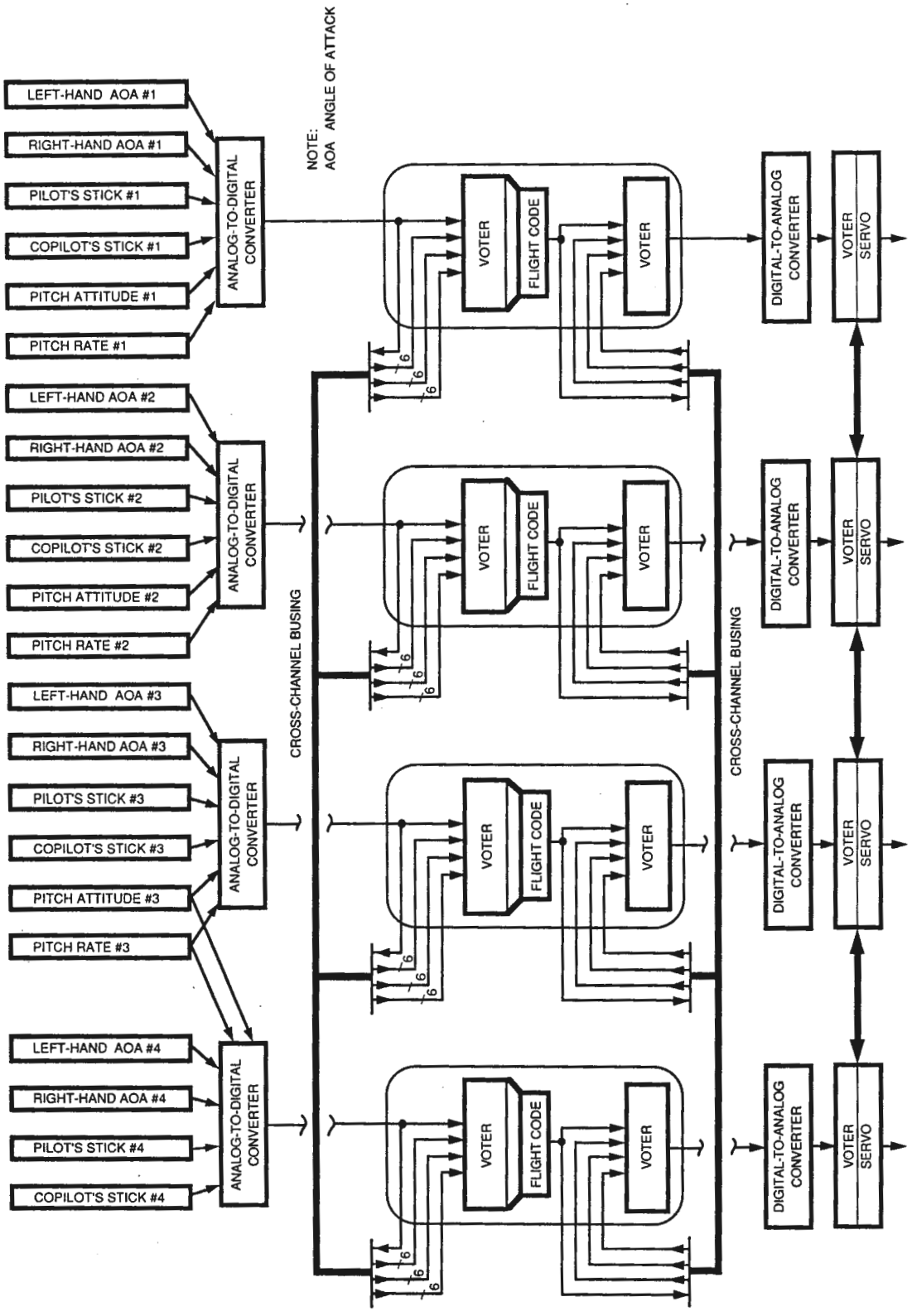


FIGURE 2-1. BASELINE SYSTEM ARCHITECTURE

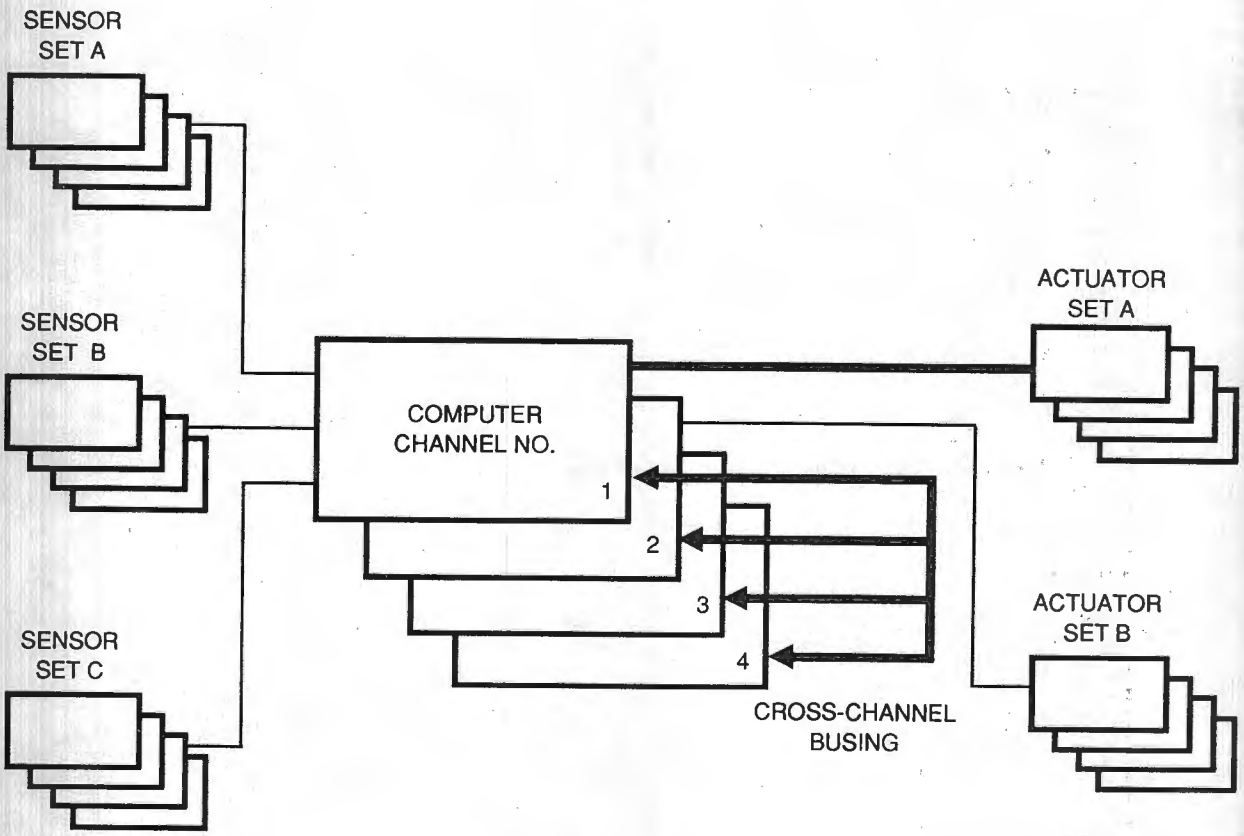


FIGURE 2-2. QUADRUPLEX DIGITAL FLIGHT CONTROL SYSTEM ARCHITECTURE

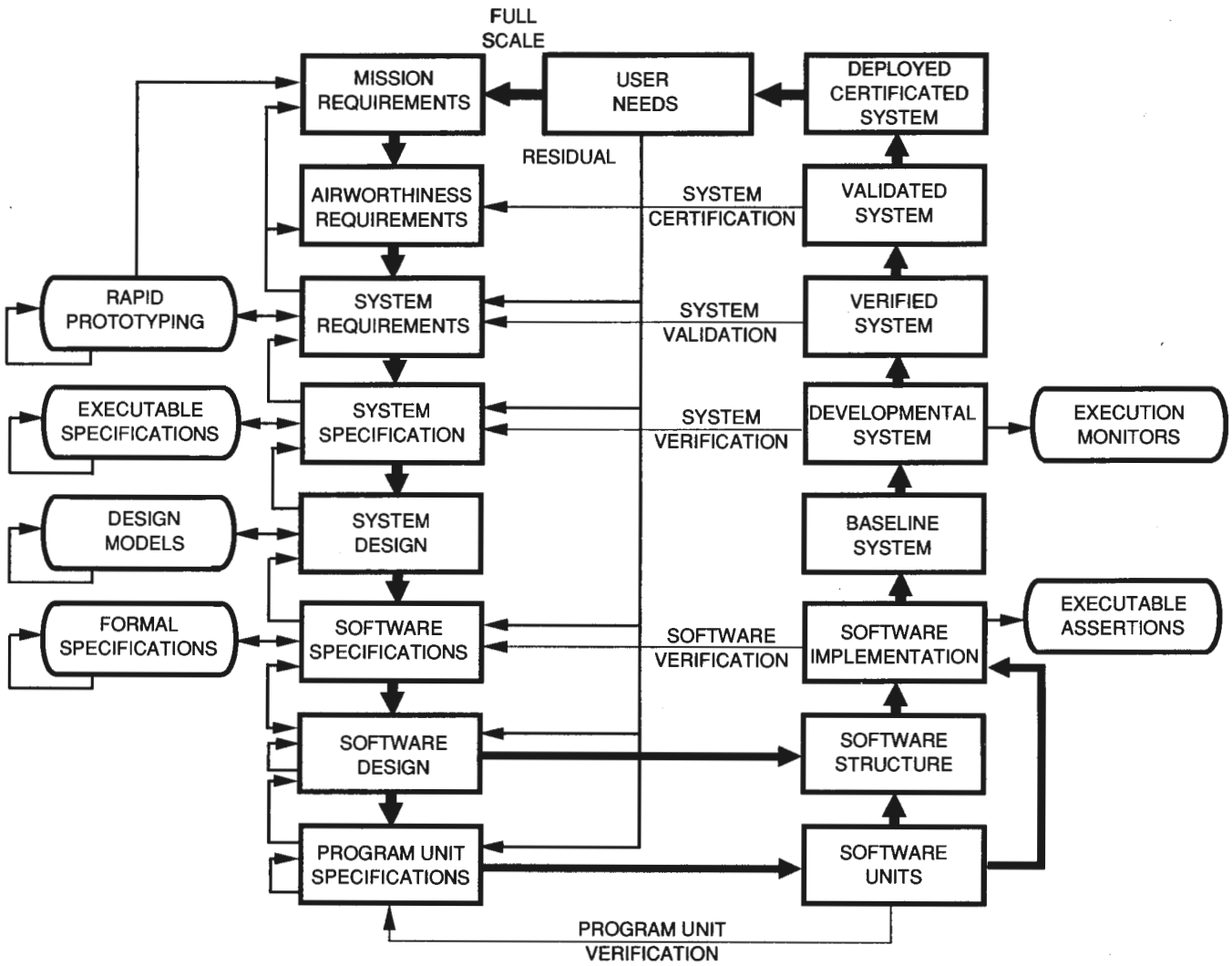


FIGURE 2-3. DIGITAL FLIGHT SYSTEM LIFE CYCLE

The data contained in this chapter have been developed from the DFCS and systems simulator shown in figures 2-4 and 2-5. (This system was located at Ames Research Center, Moffett Field, California.) The original dual-dual architecture was transformed into a quadruplex architecture by making software changes that extended the fault-tolerance capability to full-time flight-critical operation. The resulting implementation was not optimal for flight application, but it worked well for demonstrating a verified design and served to illustrate the integrated assurance methodology necessary throughout the development process for flight-critical applications.

With the introduction of full-time flight-critical control functions, substantially more effort must be directed toward guaranteeing airworthiness than was necessary for flight-phase critical functions. A tremendous increase in system complexity has also been incurred with (1) the advent of fault-tolerant (error-correcting) architectures, (2) the myriad assurance tools necessary to confirm system airworthiness, and (3) digital implementation. To attain requisite airworthiness of the flight-critical control system, lower levels of digital implementation must be examined. An integrated assurance methodology is essential for compliance with the provisions of AC 25.1309.1.

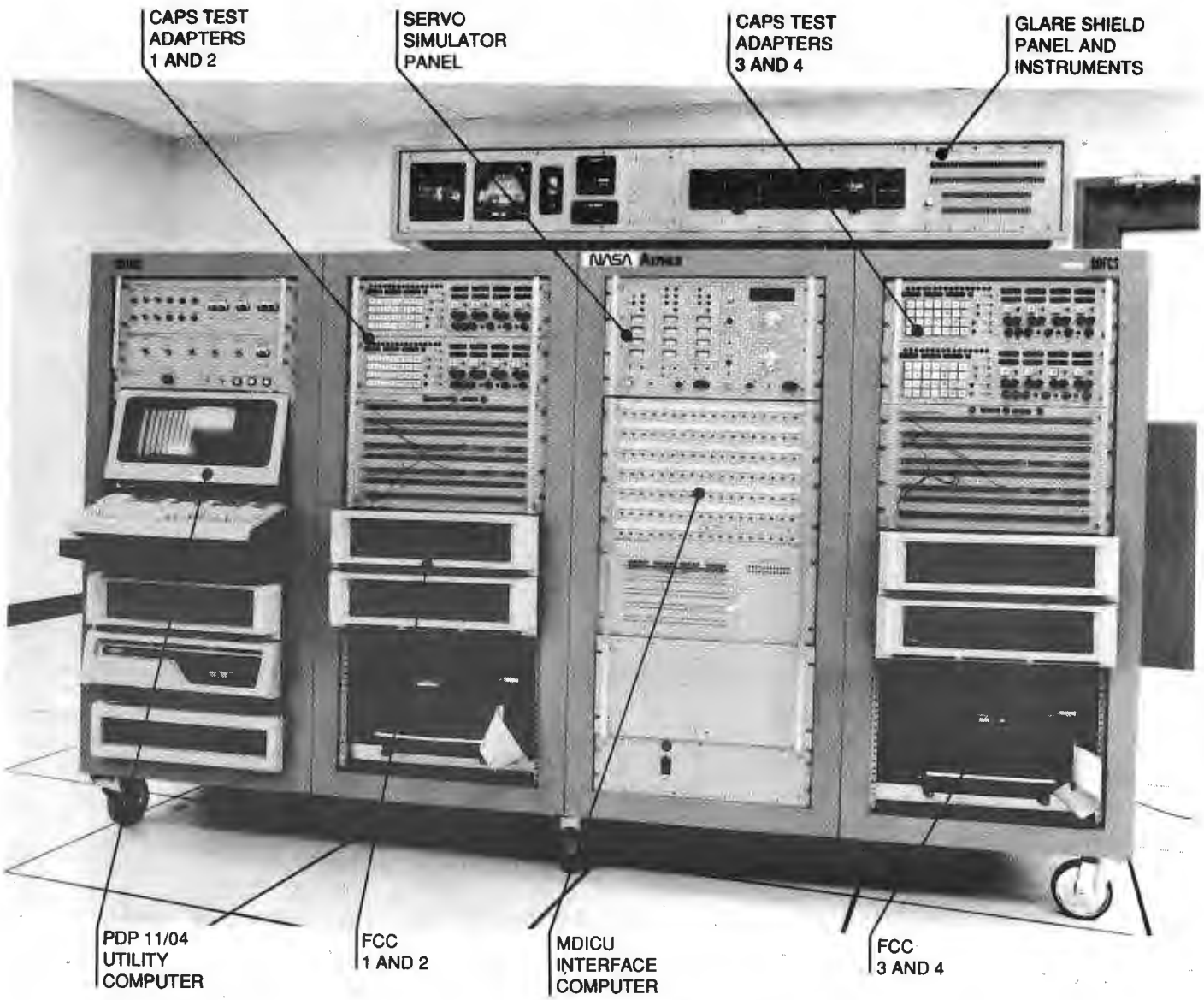
The assurance tools and methods required for use with these advanced systems have not been fully developed. It is not clear that their use ensures that the required level of reliability is achieved. In civil aircraft, current reliability levels are set at 10^{-9} or less.

In a comprehensive assessment process, several dimensions of an integrated assurance method should be reviewed and acknowledged. These dimensions include the performance, analysis, testing, and inspection of reliability analyses, failure effects analyses, simulation and functional performance assessment methods. The integration assurance methodology is shown in figure 2-6. Analysis is the dominant method for reaching high levels of assurance in the early development of an FCS, particularly when only models or abstractions are available. At this stage, a systems walk-through can be especially valuable.

Analysis forms the basis of conclusive testing. It must be accomplished at various stages of the development process to maximize the effectiveness of testing, as shown in figure 2-7. Testing based on proper analysis assures the system designer that the implementation is in accord with design specifications and requirements. The coincident application of various test cases derived through analysis provides an increased level of confidence in the correctness of the design and its implementation.

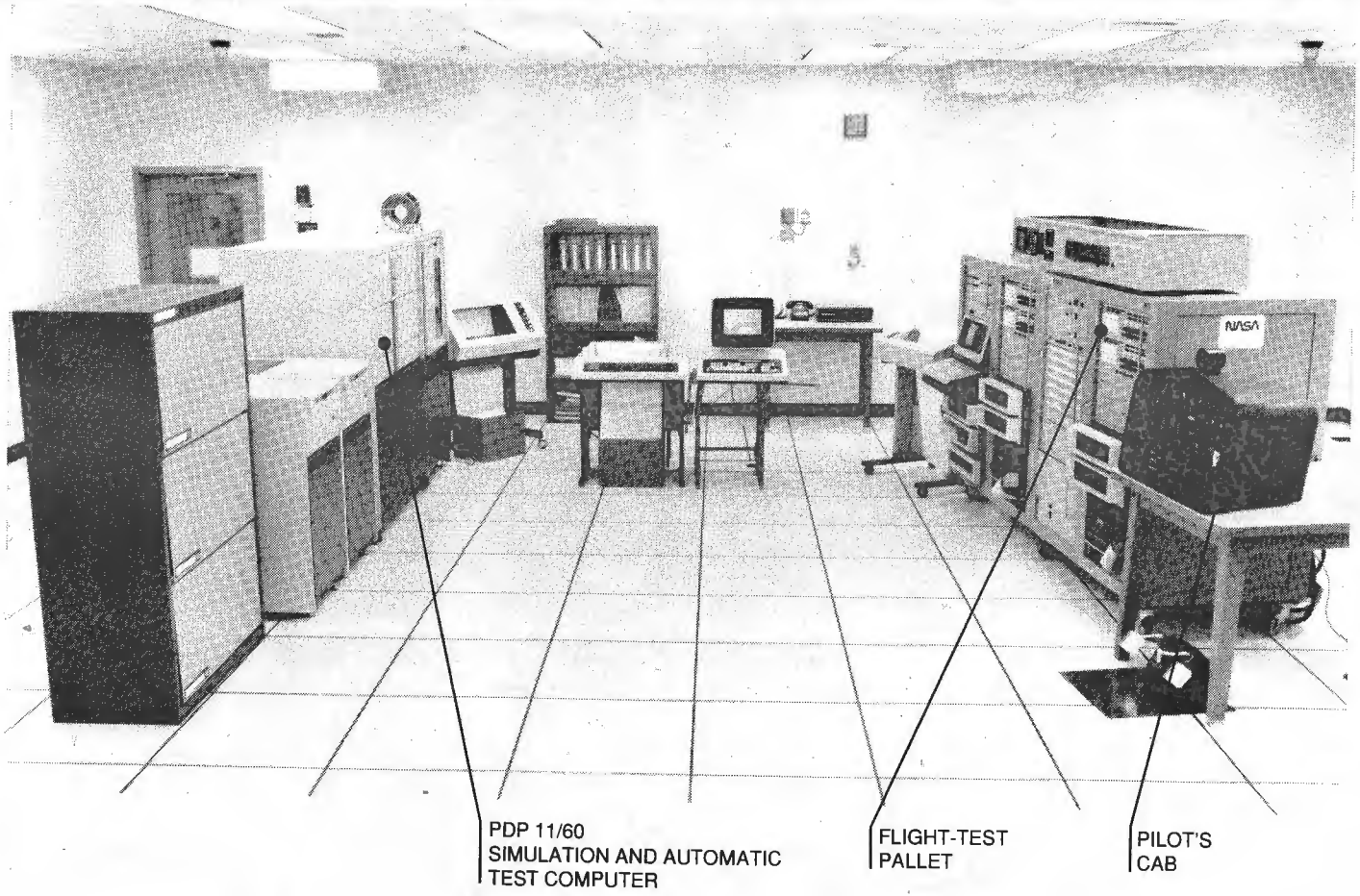
An integrated approach to proving implementation correctness is essential because testing can only be applied to the actual article. Testing primarily examines the validity of analytical results and the underlying assumptions.

The assessment methods presented are not necessarily the best or only techniques for every flight-critical DFCS use. The methods do, however, provide a representative example of the airworthiness assurance of the full-time, flight-critical DFCSs that will be making their appearances in the near future. Although the pitch-axis example discussed here is scaled, it is representative of the challenges in certifying double fail-operational quadruplex DFCSs.



(a) Flight-test Pallet

FIGURE 2-4. RECONFIGURABLE DIGITAL FLIGHT CONTROL SYSTEM FACILITY SETUP (FLIGHT TEST PALLET)



(b) Simulation Setup

FIGURE 2-5. RECONFIGURABLE DIGITAL FLIGHT CONTROL SYSTEM FACILITY SETUP (SIMULATION SETUP)

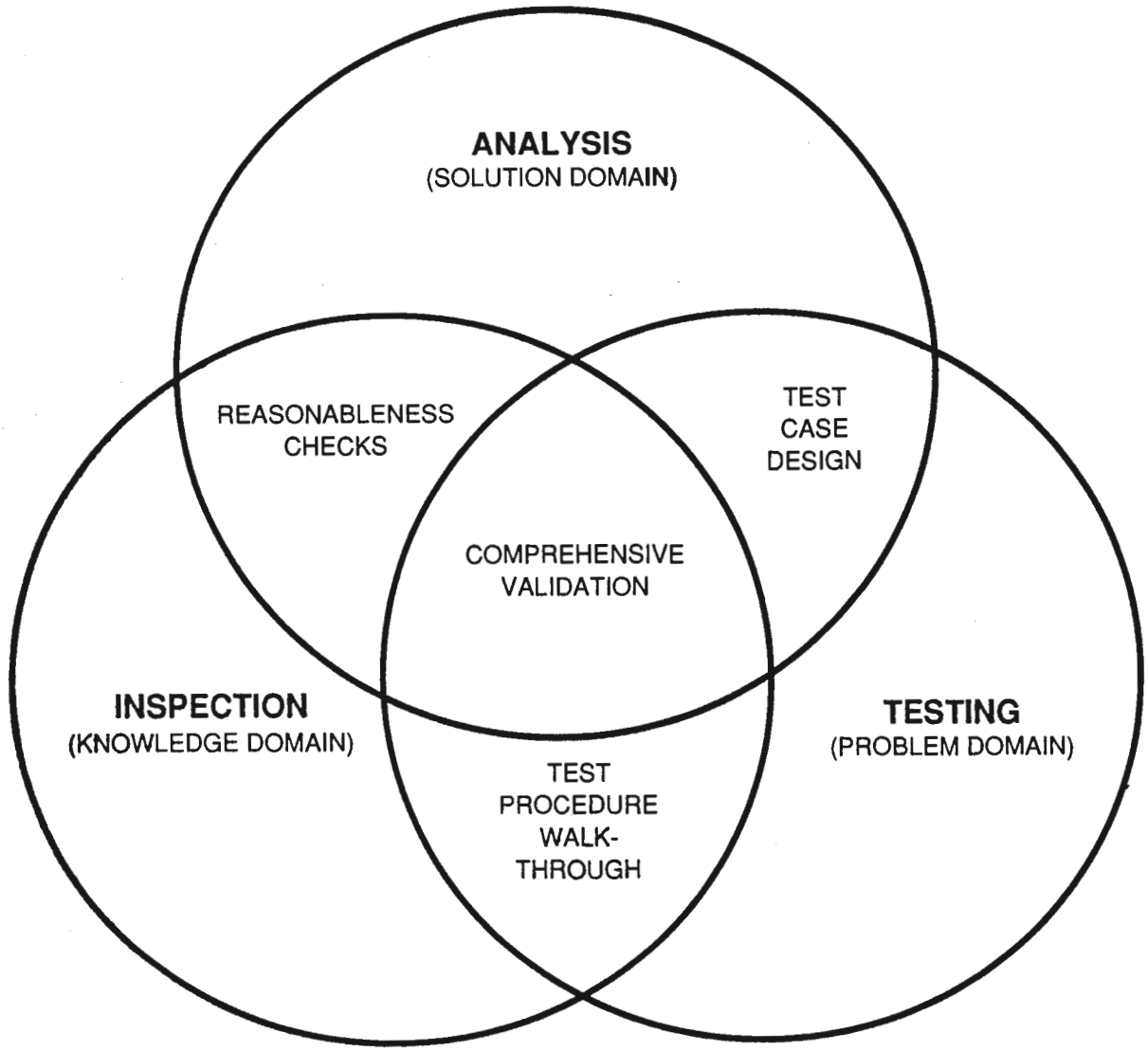


FIGURE 2-6. INTERDEPENDENCE OF ASSURANCE METHODS

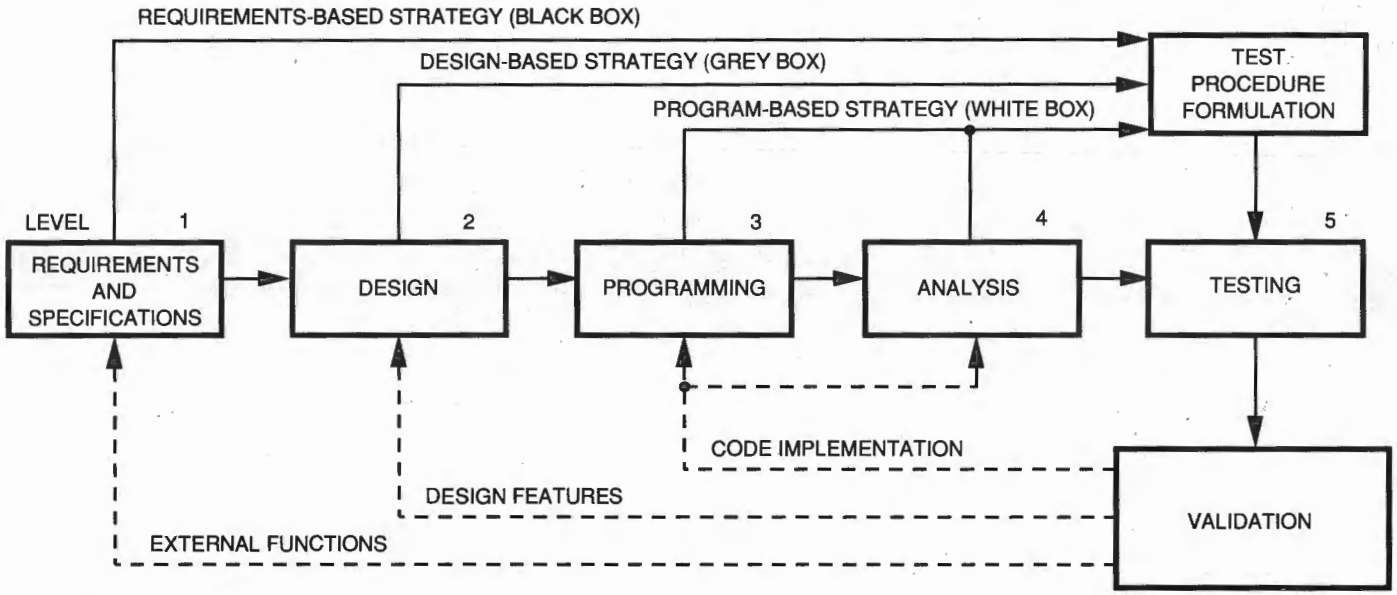


FIGURE 2-7. MULTILEVEL TESTING

There is a considerable difference in the airworthiness demands for a full-time flight-critical system or function and a flight-phase critical function. The increased demands for airworthiness assurances are primarily a function of system exposure time (the time from last system validation to completion of the present use). When a flight-critical function is implemented digitally, there is concern about the capability of ensuring its airworthiness. Therefore, these examples seek to demonstrate the requisite technology and its application in a representative DFCS development program for a pitch-axis augmented fly-by-wire/light system.

3. THE ASSURANCE ASSESSMENT PLAN

The FAA AC 25.1309-1 recommends procedures for demonstrating compliance with the requirements of Part 25 of the Federal Aviation Regulations (FAR) for flight-essential and flight-critical avionics systems. This publication outlines the use of a quantitative safety analysis that may include probability analysis, fault tree analysis, failure-modes and effects analysis, and other comparable techniques such as systems simulation. Such an integrated approach is discussed in detail in Ness, 1983 and chapter 3 of this handbook. This chapter describes a system-level failure-effects simulation that focuses on flying-qualities degradation and real-time multilevel assessment of software behavior stressing architectural fault tolerance.

The flying-qualities degradation testing is a conventional technique and incorporates an all-up simulation including sensors, servos, research cockpit pilot interface, aircraft model, and test electronics. The real-time multilevel testing is a considerably more sophisticated technique and a valuable new testing methodology for flight-critical systems. This approach to multilevel demonstration testing is a means of establishing high levels of confidence in the airworthiness of a given flight-critical DFCS.

Figures 2-3 and 3-1 present the developmental activities and expected results for a verification and validation process as it relates to a digital flight system design. Table 3-1 indicates five typical levels of testing that might be undertaken. The top level, most generally seen, is conventional system functional testing. Later on in this chapter the flying qualities for a Relaxed Static Stability (RSS) airplane test case are examined. Multilevel control structures that were also related to simulation developmental activities are examined coincidentally, as shown in figure 3-2. To conclusively examine sophisticated flight-critical systems requires that significant emphasis be placed on automated testing and wideband instrumentation. This level of testing is illustrated in table 3-1.

One major aspect of flight-critical DFCS verification and validation is the verification that logic for the top-level systems using fault-tolerant architecture has been implemented. Other testing might check the design and the redundancy of the design verification channel, because fault-tolerant architectures demand intricate hardware and software logic to maintain cross-channel coordination.

This kind of verification testing occurs in the latter stages of system development and is addressed using the fidelity of an all-up FCS simulator. If the test simulator lacks an automated test capability that would permit other than system-level checks, then it is necessary to develop real-time test programs running in parallel (commonly known as real-time test monitors). Real-time test monitors are extremely valuable in achieving the necessary level of assurance. Figure 3-3 presents the components of an all-up simulation facility (the

Reconfigurable Digital Flight Control System (RDFCS) Facility at Ames Research Center), and table 3-2 highlights some of the more significant capabilities of that facility.

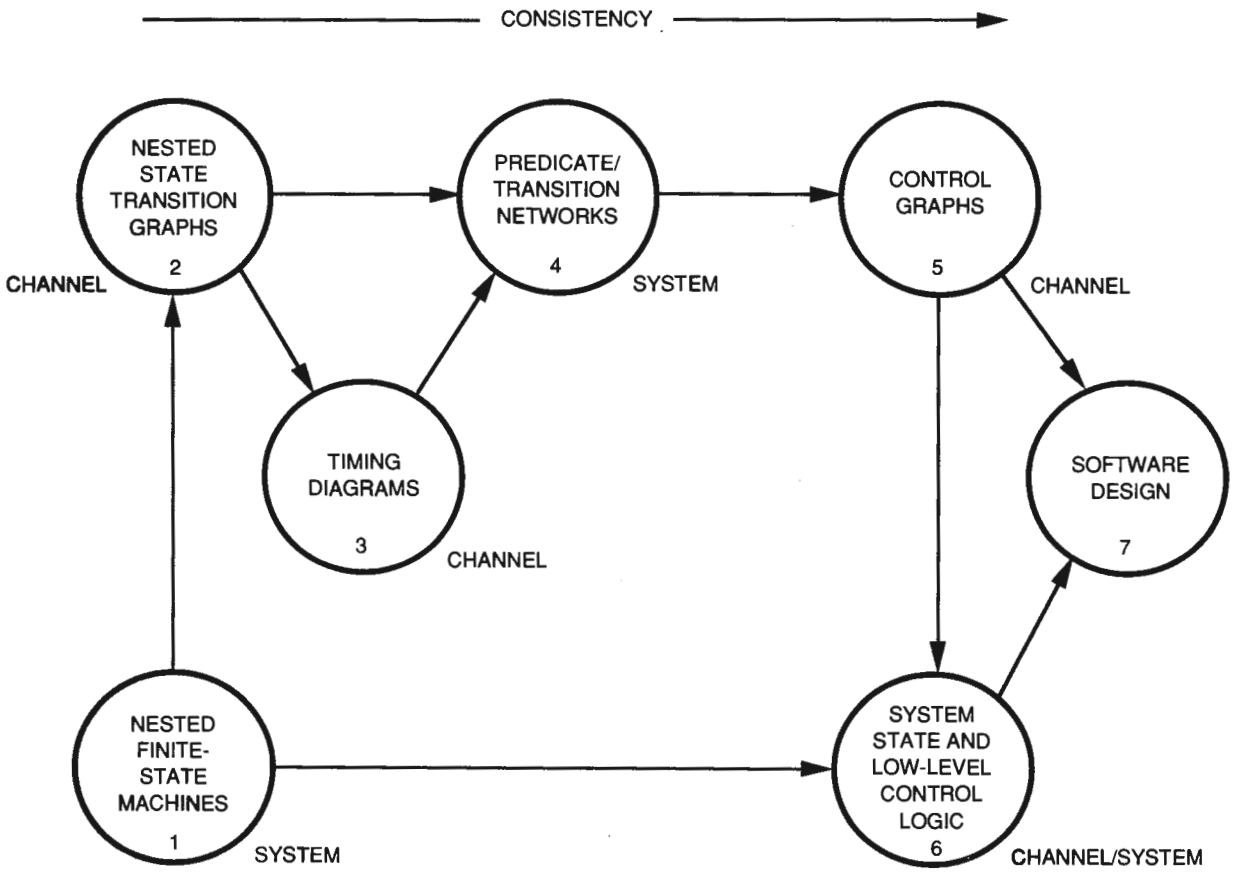


FIGURE 3-1. ARCHITECTURAL DESIGN TASKS

TABLE 3-1. DIGITAL FLIGHT CONTROL SYSTEM TESTING SCENARIO

Type of Testing	Focus	Execution Monitor	Monitor Location	Primary Concern
System validation	System performance	None	N/A	Fault Flying qualities
System verification	System state	Nested finite-state machines	Simulation/ test computer	Faulted states
	Channel synchronization	Predicate/ transition network	Single channel	Single-point failures
	Mode/fault logic	Boolean expressions	Single channel	Logic correctness
	Control flow	Control graphs with assertions	Single channel	Path traversals

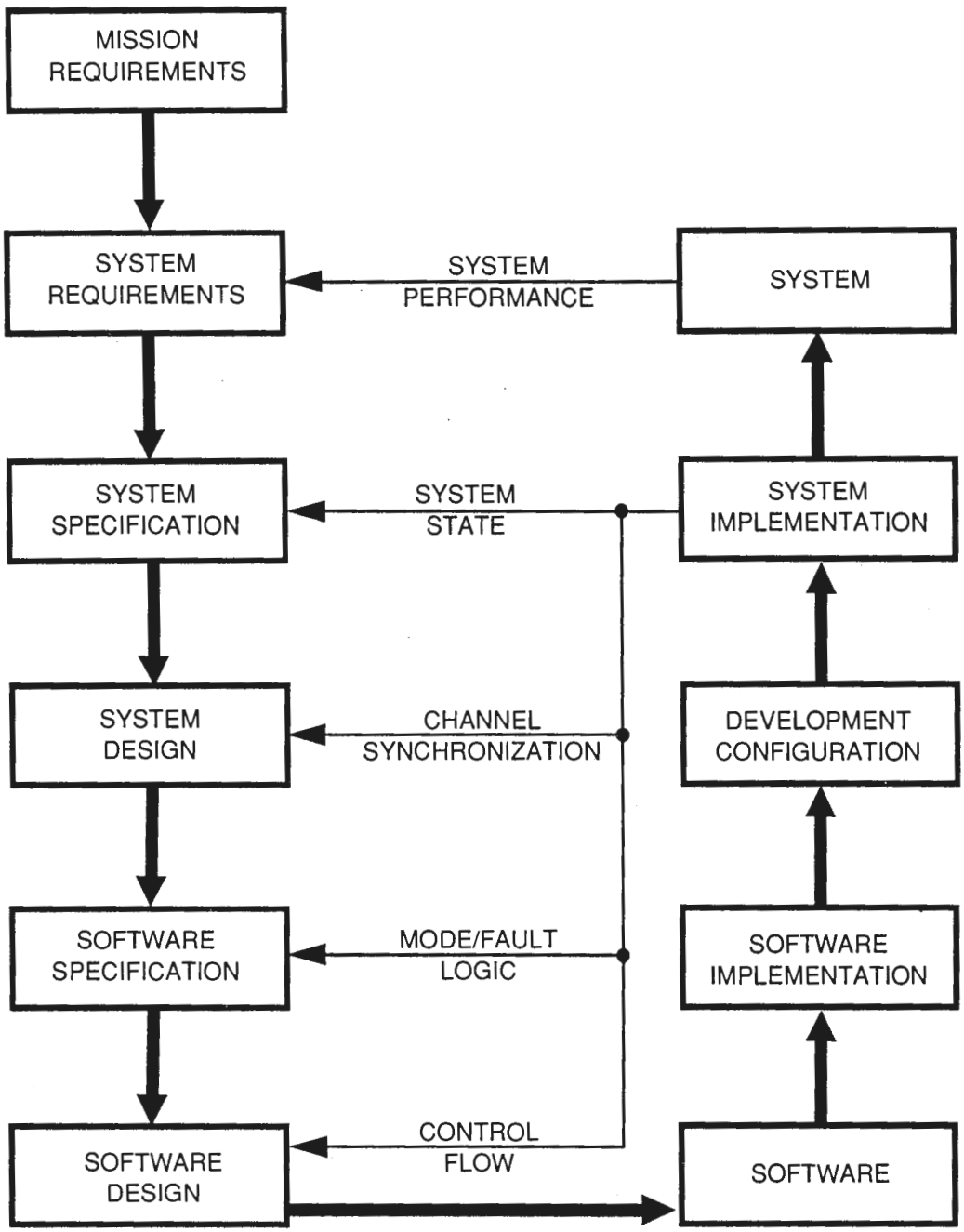


FIGURE 3-2. MULTILEVEL-TESTING CLOSURE

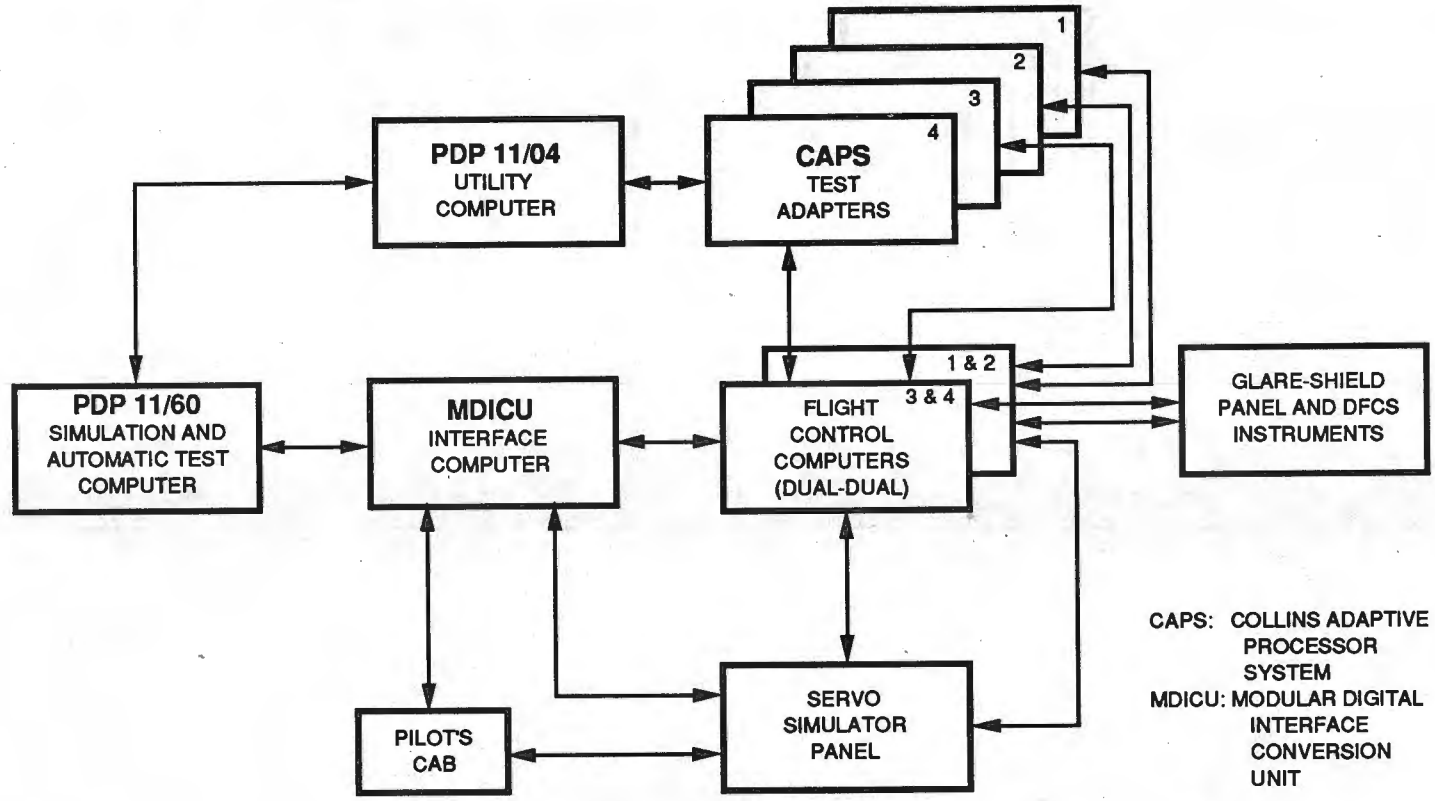


FIGURE 3-3. RECONFIGURABLE DIGITAL FLIGHT CONTROL SYSTEM ALL-UP SIMULATION COMPONENTS

TABLE 3-2. RECONFIGURABLE DIGITAL FLIGHT CONTROL SYSTEM FACILITY CAPABILITIES

Feature	Significance/Elaboration
Language for FCCs and MDICU Augmented higher order	Comprehensible, well composed software for DFCS or test purposes Instruction set augmented for machine-related use Mature support software
Efficient airframe simulation	75 percent quiescent frame time at 50 Hz Ample time for test software execution
Automated fault insertion	Simulated sensor faults in MDICU Simulated FCC memory faults via PDP-11/04 laboratories FCC hardware fault insertion via Draper
Software instrumentation	PDP-11/60 programmed data retrieval via the PDP-11/04 and Collins Adaptive Processor System (CAPS) test retrieval Background mode FCC data retrieval Data object access to all FCCs via link loader options

4. THE USE OF SIMULATION AS AN ASSURANCE TOOL

Engineering-level systems components and a piloted simulation constitute a central and critical part of the DFCS development process. They are used during the verification and validation process of systems development. The work described here is intended to further demonstrate a more effective use of system simulation during the assurance process. For both piloted flight , and system simulators the importance of the use of actual system elements in providing the required fidelity cannot be overemphasized, particularly for the m, re complex digital systems. The final assessment of the performance of the control-system handling qualities, degraded systems operation, electronic displays, and side-arm controllers can only be made with a piloted simulator. In digital systems, performance can be seriously compromised as a result of the inherent phase lags that are induced into the system.

4.1. Assurance Assessment

Throughout the development of a product, assurance features must be designed to accommodate the verification and validation processes (Goldberg, 1979 and Wensley, 1969). As the complexity of the system increases, particularly in fault-tolerant designs, early planning for accountability is of utmost importance to guarantee the conclusiveness of the assurance process.

4.2. Digital Flight Control System Assurance Methodology

The quadruplex DFCS described by Mulcare, Downing, and Smith, 1987, was developed primarily by making software changes to a dual-dual DFCS flight-test pallet. Although constraints were imposed in the changeover from a dual-dual to a quadruplex setup, the resulting configuration was adequately representative of a conventional quadruplex system. Since the modification was originally developed to demonstrate rigorous system and software design methodologies (Mulcare, Ness, and Davis, 1984) the facility is well-suited for use as an assurance assessment demonstrator.

4.3. Simulation Development

Within the constraints of the existing dual-dual FCS, a representative fail-operational-squared (fail-operational/fail-operational) architecture has been defined. The redundancy management coordination of this design can be verified through predicate/transition network simulation, and the high-level software design is then represented in nested control graphs. Cross-channel coordination is effected through respective channel control states broadcast over serial digital buses. The system is then tested under various simulated faulted and abnormal conditions using real-time software execution monitors to resolve low-level system management events.

In the development of the simulator, RSS flight cases are used and validated, using airplane time-histories. New sensor outputs are routed through the Modular Digital Interface Conversion Unit (MDICU) to the flight computers. New mode and fault logic signals are assigned on the logic discrete switch panel on the FCS simulator. Lastly, the pitch-axis inputs are introduced into the flight computer by means of a manual control stick. It is therefore possible to assess flight-qualities degradation through real-time closed-loop systems simulation with or without pilot input.

FCS definition evolved in accordance with the development products shown in figure 2-3, which highlights the stages of mechanization along the downward path on the left of the figure and the assurance steps along the upward path on the right of the figure. Further details of the development activities are presented in table 4.3-1. In general, the objective is to highlight key steps corresponding to progressive system definitions that should lead to a certifiable DFCS. Design definition usually addresses either system functions or architecture. System-function definition centers on control laws, whereas system-architecture definition centers on system and software structure. It is particularly vital that the organization of the flight software be explicitly described prior to digital implementation. These aspects of system definition, particularly system/software architecture, are central to reliable/fault-tolerant DFCSs and will be further discussed before the presentation of application examples (see paragraph 4.6). These design tasks, noted in figure 3-1 are illustrated through a sequence of examples.

To assist in the design definition of a flight-critical FCS, an RSS airplane with a Fly-By-Wire (FBW) FCS is defined. The control law shown in figure 4.3-1 has been used in each of the four redundant FCS channels. Pilot command is through a control stick. Short-period damping is provided from the simulated airplane, and angle of attack (AOA) is used to control the pitch divergence that is associated with the relaxed static stability. Control-surface scheduling is used, because control-surface effectiveness increases with airspeed. Six point-mass simulation cases were used, as shown in table 4.3-2, with their sensor feedback gains.

4.4. Basic System Architecture

Channel redundancy for fault tolerance in DFCSs demands intricate hardware/software logic to maintain cross-channel coordination. Such wide bandwidth processes must take place dependably despite component tolerances, inaccuracies in timing, occasional software errors, and hardware faults. At risk are the integrity and continued proper operation of the total system. It is vital to confirm the safety and acceptability of the cross-channel logic under a broad range of deviant or discrepant conditions.

In the latter stages of system development, these concerns can be addressed using the fidelity of an all-up, real-time FCS simulator. However, such simulators do not usually incorporate the automated test capability that is necessary to permit checks at other than the system level. Real-time test programs are desirable; they run in parallel with redundant DFCSs to ascertain the correct cross-channel logic operation, thereby guaranteeing a high degree of resolution and certainty.

TABLE 4.3-1. DEVELOPMENT PHASE ACTIVITIES AND PRODUCTS

1. Conceptual Phase	
Purpose	Provide for user needs
Input	Mission requirements
Mech.*	Formulate system requirements; explore design solutions
Assur.*	Validate requirements; analyze design solutions
Product	System requirements and concepts
Status	Functional architecture; feasible design solution
2. Definition Phase	
Purpose	Design for user needs
Input	Airworthiness and system requirements; system concepts
Mech.*	Formulate system specs; refine system concepts
Assur.*	Verify specs; validate system concepts
Product	System specs; conceptual system design
Status	Validated conceptual design; acceptable design solution
3. Analysis Phase	
Purpose	Design system to specs
Input	System specs
Mech.*	Design system structure; design control laws
Assur.*	Verify software specs, system structure, and control laws
Product	System design; software and hardware specs
Status	Verified system structure; superior design solution
4. Design Phase	
Purpose	Design software to specs
Input	Software specs
Mech.*	Design software structure; define software components
Assur.*	Verify software design and unit specs
Product	Software design; program unit specs
Status	Verified baseline design; comprehensive design definition
5. Coding and Checkout Phase	
Purpose	Implement software to specs
Input	Unit specs
Mech.*	Implement program units
Assur.*	Check and debug units; verify program units
Product	Software implementation
Status	Baseline system config.; comprehensive system definition

*NOTES: Mechanization process; Assurance process.

TABLE 4.3-1. DEVELOPMENT PHASE ACTIVITIES AND PRODUCTS (Continued)

6. Integration Phase	
Purpose	Construct system with hardware/software components
Input	Verified structure and components
Mech.*	Assemble/develop system
Assur.*	Identify and rectify inconsistencies
Product	System implementation
Status	Developmental system
7. Development Test Phase	
Purpose	Test to spec requirements
Input	System specs
Mech.*	Develop system; optimize performance
Assur.*	Identify and rectify deficiencies; verify performance
Product	Verified implementation
Status	Verified system
8. Validation Phase	
Purpose	Test for compliance with requirements
Input	System requirements
Mech.*	Modify system if necessary
Assur.*	Confirm acceptability
Product	Provisional configuration
Status	Validated system
9. Certification	
Purpose	Demonstrate airworthiness compliance
Input	Airworthiness requirements and certification plan
Mech.*	Modify system if necessary
Assur.*	Confirm airworthiness
Product	Production configuration
Status	Certificated system

*NOTES: Mechanization process; Assurance process.

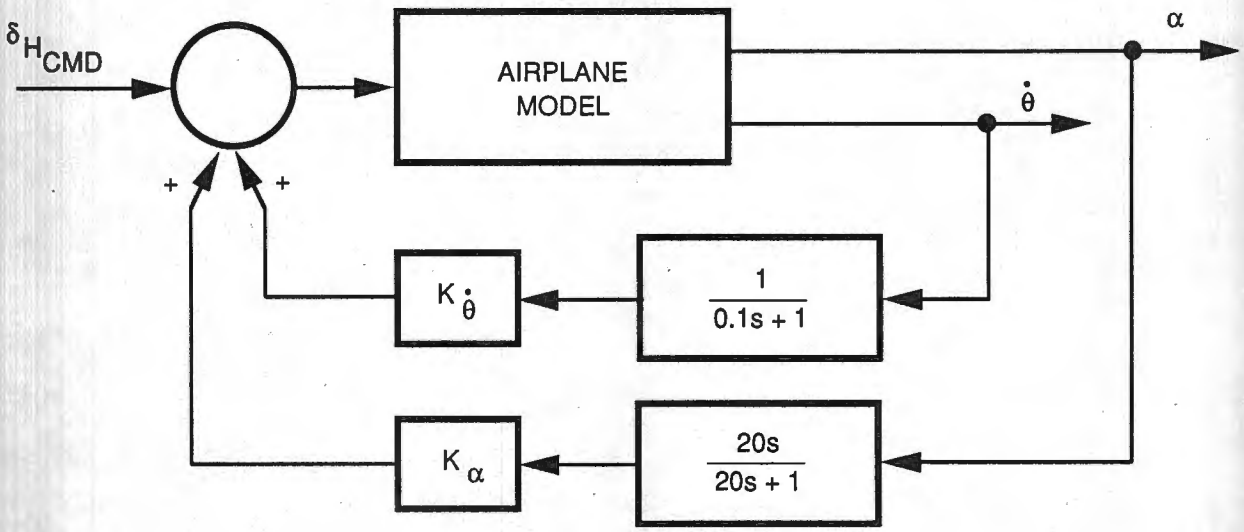


FIGURE 4.3-1. CONTROL LAW USED IN REDUNDANT FLIGHT CONTROL SYSTEM CHANNELS

TABLE 4.3-2. STABILITY AUGMENTATION GAIN SCHEDULING

Flight Case	True Airspeed	K_α, deg/deg	$K_{\dot{\theta}}$, deg/deg/sec
A1RSS	283.7	1.00	0.70
C13RSS	910.7	0.73	0.10
C15RSS	442.2	1.00	0.40
D2RSS	487.1	1.00	0.33
E3RSS	265.7	1.00	0.80
F6RSS	224.1	1.00	0.50

4.5. Developing the Flight Control System

The top-level primary FCS requirements are assumed to be MIL-F-9490D operational states (Specification MIL-F-9490D). These requirements encompass both airplane flying qualities and system safety in terms of redundancy margins. Operation State 1 denotes normal system status and Level 1 flying qualities (good); State 2 allows some deterioration in safety margins and Level 2 flying qualities (somewhat degraded); State 3 indicates marginal safety and Level 3 flying qualities (marginal); and State 4 designates unsafe components or flying qualities (worse). Knowledge of flying-qualities degradation under conditions of successive component failures makes it possible to view the operational state logic from an architectural point of view. The system design task then focuses on defining the channel-level logic to effectuate the system-level logic.

Figure 4.5-1 represents the top-level DFCS channel logic design; the labels in the circles correspond to the particular states that can be assumed by the logic variables. These logic variables denote states or substates of the channel. The names appearing on the arcs of the state transition graphs designate independent logic events such as pilot-mode selection, timer interrupt, or component failure. Such events in turn may effect changes in the channel states which must reflect the state of the system. Note that the transition graphs are nested to reduce complexity; the lower-level graphs capture substate information.

In the Gate substate of the Cycling state, for example, each channel develops its view of system status based on control state information for each of the four channels. This kind of consensus is one of the major determinants in calculating the operational state of the system. In all, the implementation of this design necessitates a substantial expansion of the logic in the flight code. First, however, it is necessary to examine the system-level coordination logic in terms of meeting the timing constraints given in table 4.5-1.

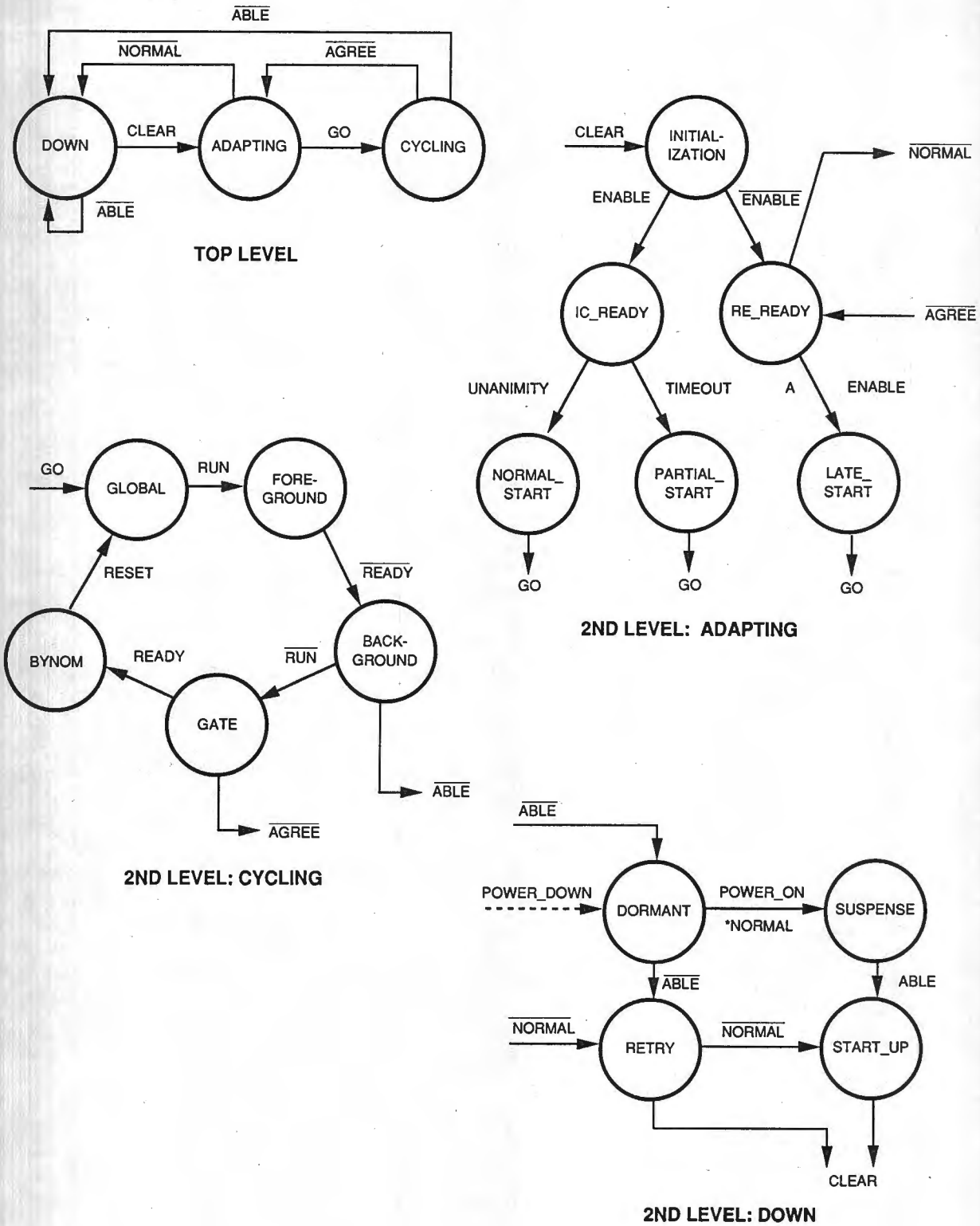


FIGURE 4.5-1. TOP-LEVEL DIGITAL FLIGHT CONTROL SYSTEM CHANNEL LOGIC DESIGN

TABLE 4.5-1. CYCLING STATE LOOP TIMING

EVENT	Substate and Duration				
	Global (0-1 msec)	Foreground (1-tfg)	Background (tfg-48)	Gate (48-49)	Synch (49-50)
Run	False	True	True	False	False
Ready	True	True	False	False	True
Reset	True	False	False	False	False

NOTES: Computational frame time = 50 msec; tfg <48 msec (where tfg is foreground time); timer interrupt at t = 48 msec ≥run; reset ≥t = 0.

System synchronization logic with a discrete time-base imposed is shown in figure 4.5-2, which represents a predicate/transition network (Goldberg, 1979). This view represents only one channel, but all four channels are the same for the subject architecture. Here the same logic nomenclature is retained where applicable, and some new logic variables are added. The logic names appear in the rounded boxes, which are called places in the network. At any given time, the collective values of all the places constitute the state of the system modeled. To examine and verify the correctness of all possible state sequences, the network's operation must be simulated using a computer program. Properly accomplished, such a simulation, under both faulted and fault-free conditions, verifies the system logic design.

The simulation is based on the firing of transitions (denoted by the rectangles in figure 4.5-2), which yield new values for the logic variables stored in the places. The top half of each transition box designates a predicate whose satisfaction by system logic values enables it to be fired. Whenever a transition fires, the new logic values described in the bottom half of the rectangle are assigned. These values are then reflected in the appropriate places, and a new network state produces a new set of transitions that can be fired.

This mechanism can be seen more clearly by noting the partial network in figure 4.5-3. Here the network captures a design where the hardware initialization within a channel sets its POWER ON to TRUE and its CLEAR to FALSE. This arms the top transition, whose firing corresponds to software initialization that sets CLEAR to TRUE among other logic-variable assignments shown. Referring to figure 4.5-3, note that the event of CLEAR being set to TRUE effects the top-level channel state transition from DOWN to ADAPTING. At this point, the channel tries to synchronize with the other channels, to enter the CYCLING state.

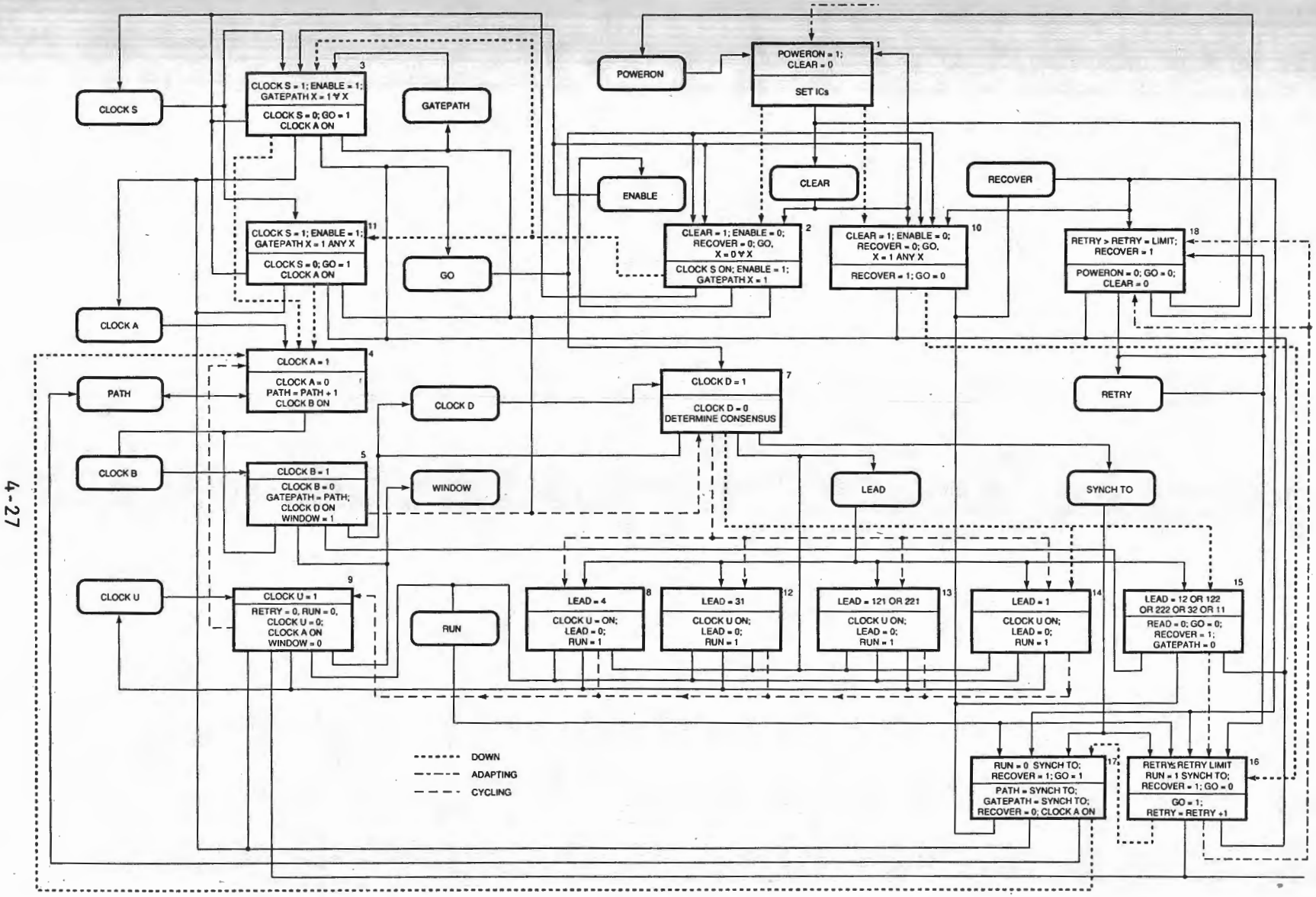


FIGURE 4.5-2. SYNCHRONIZATION PREDICATE/TRANSITION NETWORK

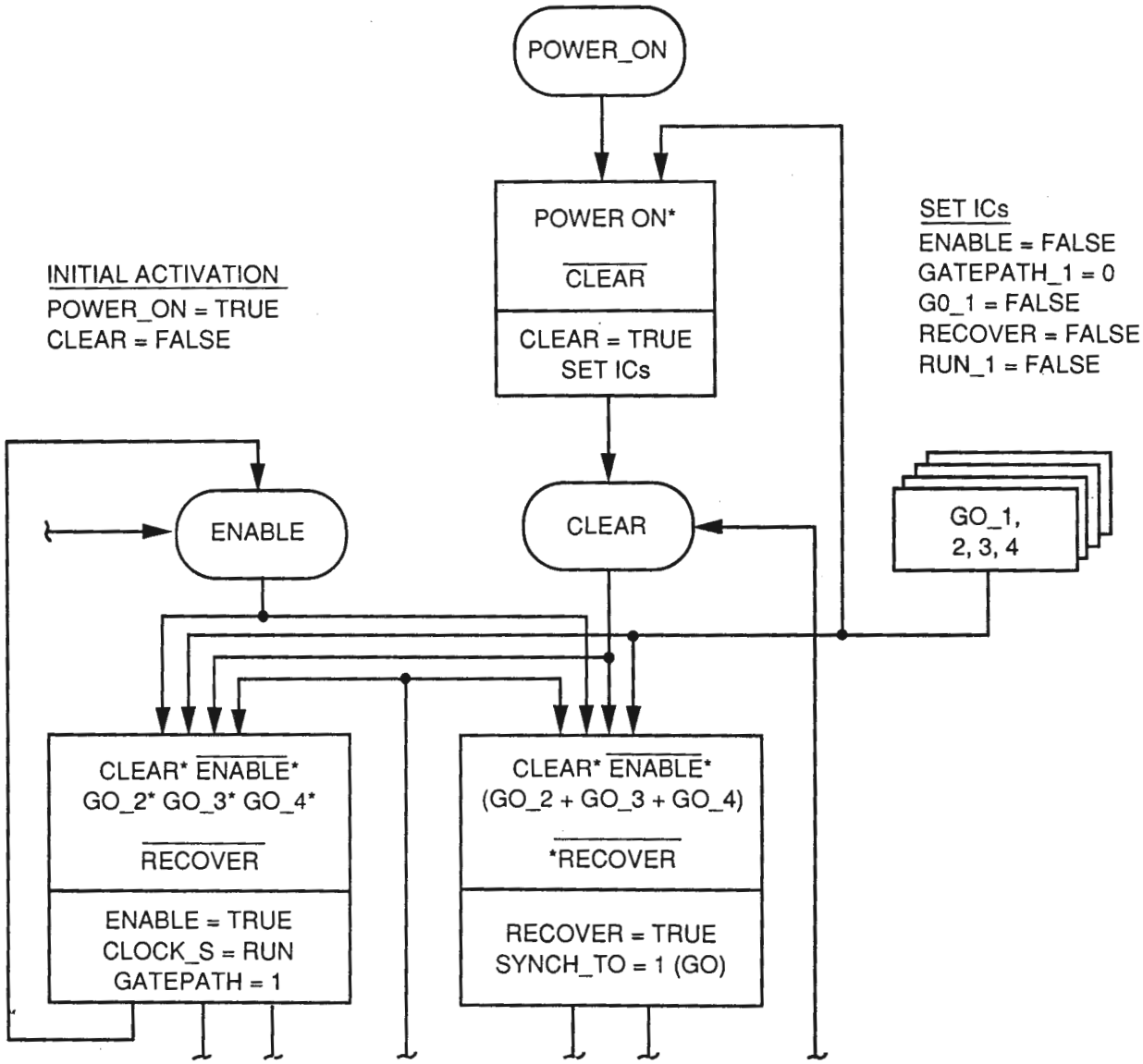


FIGURE 4.5-3. PARTIAL PREDICATE/TRANSITION NETWORK

Network simulation should always yield acceptable system states, and should never terminate unless all channels are failed. Determining that this is the case is a matter of defining and obtaining correct logic operation. In the demonstration, this was accomplished (with one exception noted later). Figure 4.5-4 illustrates some typical discrete-event simulation results for four-way channel synchronization. The pulses at the top indicate instances where individual channels were forced out of synchronization; that is, the corresponding logic-variable RECOVER went TRUE. The resynchronization logic in the network model then satisfactorily restored synchronization, and the RECOVER was set to FALSE, as indicated by the end of the pulse in figure 4.5-4.

Following synchronization design verification, which captured time-based hardware/software interaction, the design emphasis shifted to the top-level software design with the constraints imposed by the existing RDFCS hardware.

In the basic system, the flight software is hardware-interrupt-driven at the 60-Hz rate. Since the AFBW control laws were designed for 20-Hz operation, the executive software invokes the applications control functions every third interrupt. The executive program itself is somewhat austere, as appropriate for a dedicated system. Here the computational channels mutually coordinate themselves in a frame-synchronous, double fail-operational manner.

To achieve this, the channel design must be mapped into a software design with the objectives of maintaining consistency in the system-to-software design transition and minimizing the complexity of the software. Accordingly, the control graph shown in figure 4.5-5 represents a mapping of the top-level transition graph in figure 4.5-1 to a software control structure that preserves the design logic in a form exhibiting only moderate decision-logic complexity as measured by the cyclomatic complexity number. (This number is a measure of program complexity, thus a measure of program testability, maintainability, and, by implication, reliability. The number is a function of the number of control graph vertices, edges, and connected components. See McCabe, 1976 for details.) As byproducts, the control graph yielded test-case input vectors and logic assertions that were later used in verifying the implemented code.

4.6. Establishing System Reliability

To establish system reliability and the required architecture, a tradeoff study must be conducted. In establishing system reliability, the reliability assessments are directed toward contrasting levels and types of redundancy relative to their effect on system reliability for critical functions. Table 4.6-1 delineates 12 different system architectures for critical system function failure on both 5- and 10-hour missions. The system architecture shown in figure 2-1 actually corresponds to cases 1 and 2 in table 4.6-1, depending on whether inherent backup capability is invocable. Because of the arrangement of the mode-selection switch that was devised in the RDFCS laboratory, the backup mode was manually selectable. Figure 4.6-1 represents the reliability model for architecture case 1, the one actually implemented. Note that the dependencies and logic embedded in this reliability model do not, however, apply to case 2 in table 4.6-1.

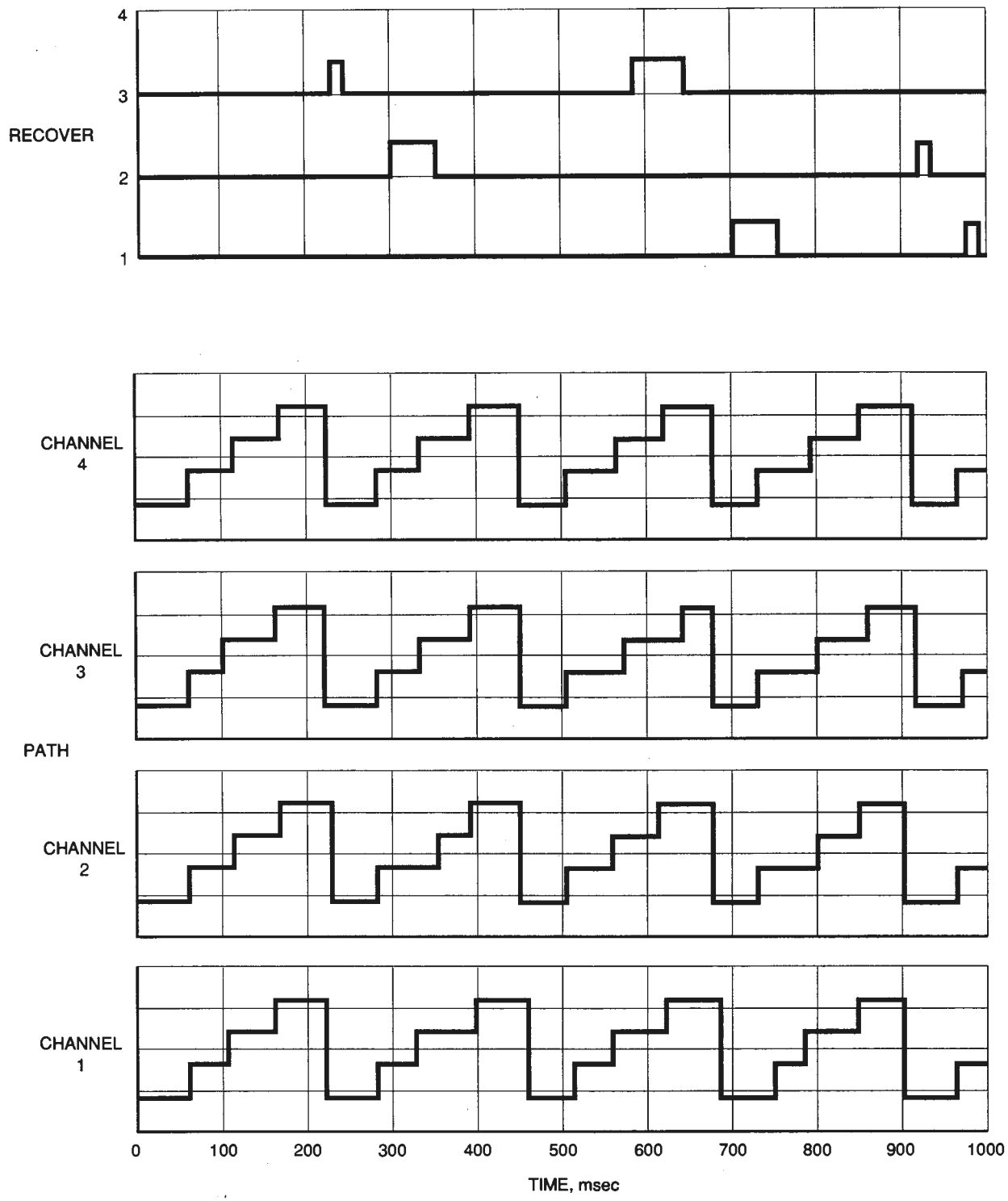


FIGURE 4.5-4. TYPICAL DISCRETE-EVENT SIMULATION RESULTS

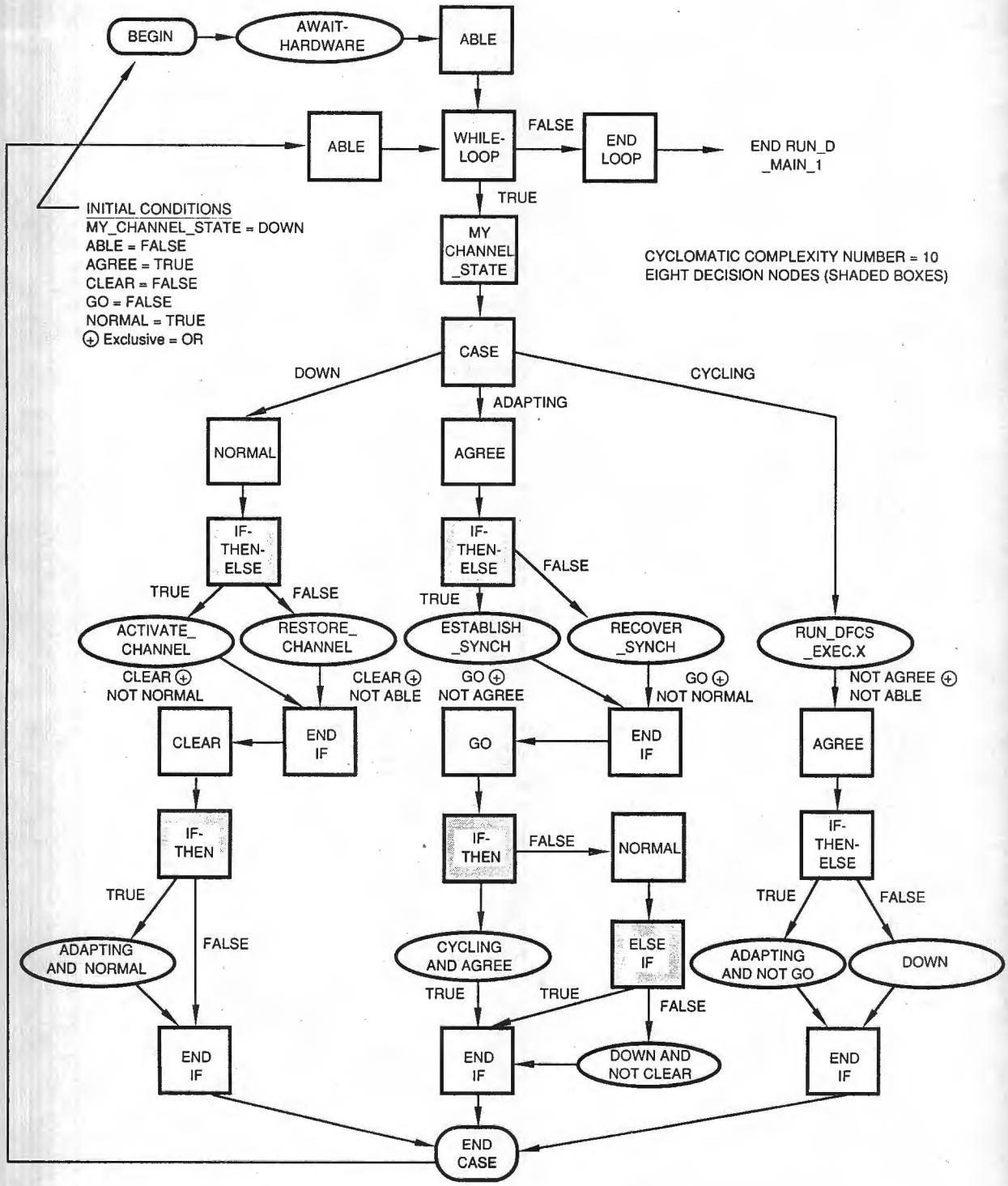
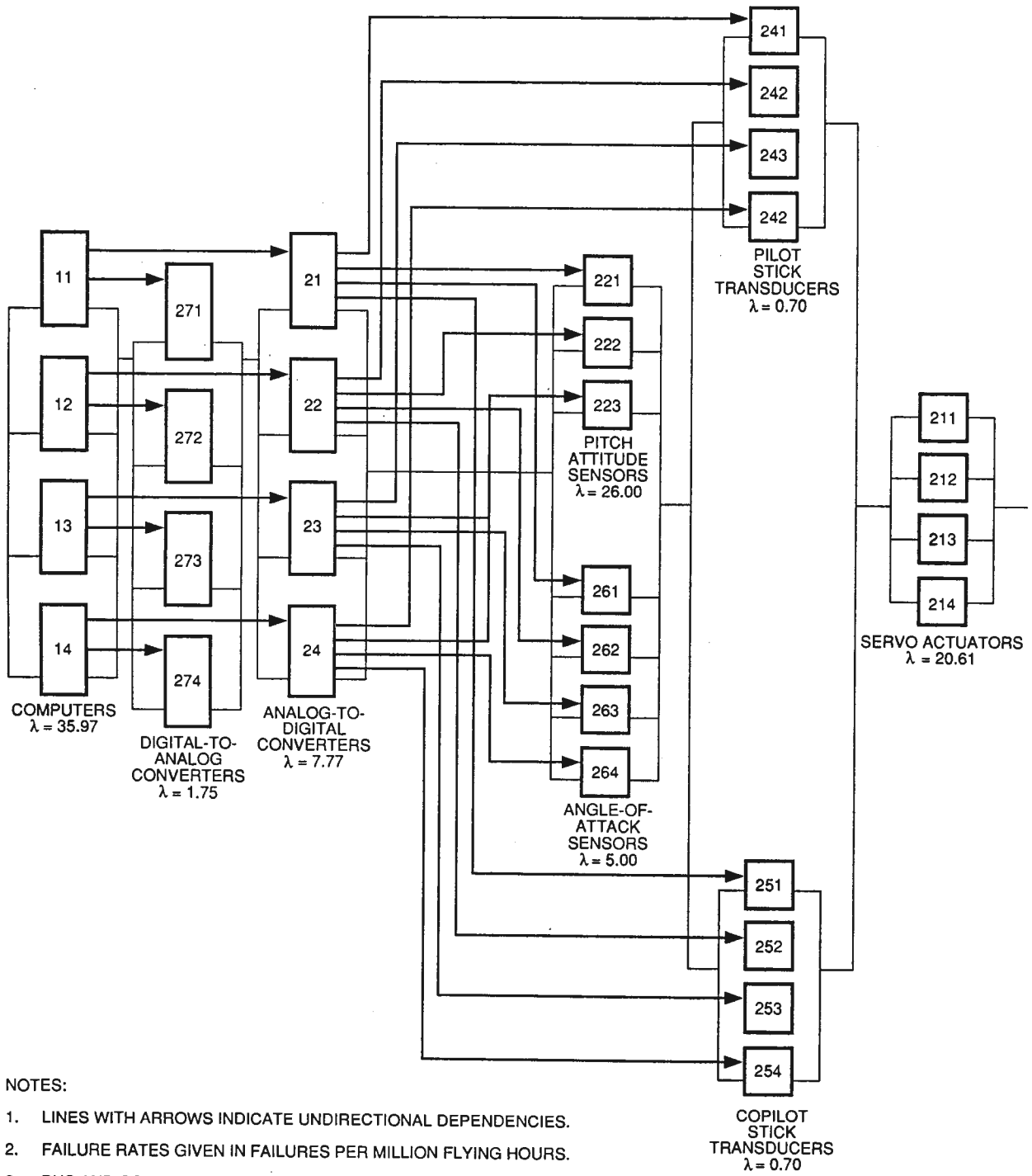


FIGURE 4.5-5. CONTROL GRAPH OF TOP-LEVEL SOFTWARE



NOTES:

1. LINES WITH ARROWS INDICATE UNDIRECTIONAL DEPENDENCIES.
2. FAILURE RATES GIVEN IN FAILURES PER MILLION FLYING HOURS.
3. BUS AND CONNECTOR FAILURES INCLUDED.

FIGURE 4.6-1. RELIABILITY MODEL FOR IMPLEMENTED ARCHITECTURE

TABLE 4.6-1. DIGITAL FLIGHT-CONTROL SYSTEM RELIABILITY ASSESSMENTS

Case	System	SAS Backup	Servo Setup	Failure Probability	
				5 hour	10 hour
1	Quad AFBW	Pitch hold	Quad	$0.478X10^{-10}$	$0.382X10^{-9}$
2	Quad AFBW	None	Quad	$0.615X10^{-10}$	$0.492X10^{-9}$
3	Quad AFBW	None	Triplex	$0.626X10^{-10}$	$0.500X10^{-9}$
4	Quad AFBW	None	Dual-dual	$0.107X10^{-7}$	$0.430X10^{-7}$
5	Dual-dual AFBW	None	Dual-dual	$0.200X10^{-6}$	$0.798X10^{-6}$
6	Quad FBW	N/A	Quad	$0.457X10^{-10}$	$0.365X10^{-9}$
7	Quad FBW	N/A	Triplex	$0.468X10^{-10}$	$0.374X10^{-9}$
8	Triplex FBW	N/A	Triplex	$0.153X10^{-6}$	$0.612X10^{-6}$
9	Quad SAS	Pitch hold	Quad	$0.478X10^{-10}$	$0.382X10^{-9}$
10	Quad SAS	None	Quad	$0.615X10^{-10}$	$0.491X10^{-9}$
11	Quad SAS	None	Triplex	$0.626X10^{-10}$	$0.500X10^{-9}$
12	Triplex SAS	None	Triplex	$0.188X10^{-6}$	$0.751X10^{-6}$

The cases of primary interest here (1 through 5 in table 4.6-1) are all-up AFBW architectures. Cases 1 through 3 meet the critical-function reliability requirements defined in Defeo, Doane, and Saito, 1982, but the analyses only take into account the hardware fault contributions to unreliability. Still, the data are instructive in several ways. Basically, the backup pitch-hold mode adds surprisingly little to survivability, partly because it would be needed so infrequently, and then only at a time when it too might well be unavailable owing to the loss of common components with the basic pitch Stability Augmentation System (SAS).

Cases 4 and 5 (table 4.6-1) are inadequate because of reduced redundancy levels. Cases 5 through 7 isolate the reliability properties of straight FBW architectures. A comparison of cases 2 and 6 shows, for example, that the critical pitch SAS function increases the probability of failure by about only a third for a straight FBW system. The straight SAS function is isolated in cases 9 through 12, and it is noteworthy that triplex AOA sensors and computers are inadequate for case 12. However, triplex servos with quadruplex sensors are satisfactory.

Tables 4.6-2 and 4.6-3 reveal flying-qualities degradation, as well as system failure, the extent and likelihood of the degradation being important measures of system acceptability. Cases 6 through 8 are omitted because the straight FBW function does not degrade in stages; instead, it usually fails completely when an appropriate combination of failures has occurred. Since stability-augmentation degradation is basically sensor-related, this set of servo-oriented architecture variations is not fully useful in delineating flying-qualities tradeoffs.

TABLE 4.6-2. DEGRADATION OF FLYING-QUALITIES: 5-HOUR MISSION

Case*	Flying-Qualities Level (probability)			
	1	2	3	<3
1	0.999+	$0.212X10^{-6}$	$0.017X10^{-11}$	$0.478X10^{-10}$
2	0.999+	$0.212X10^{-6}$	0	$0.615X10^{-10}$
3	0.999+	$0.212X10^{-6}$	0	$0.626X10^{-10}$
4	0.999+	$0.212X10^{-6}$	0	$0.607X10^{-10}$
5	0.999+	$0.120X10^{-3}$	0	$0.200X10^{-6}$
9	0.999+	$0.212X10^{-6}$	$0.137X10^{-10}$	$0.478X10^{-10}$
10	0.999+	$0.212X10^{-3}$	0	$0.615X10^{-10}$
11	0.999+	$0.212X10^{-6}$	0	$0.626X10^{-10}$
12	0.999+	$0.221X10^{-6}$	0	$0.188X10^{-6}$

*Cases 6 through 8 omitted because the FBW function does not degrade in stages (it fails completely given appropriate combinations of failures).

TABLE 4.6-3. DEGRADATION OF FLYING-QUALITIES: 10-HOUR MISSION

Case*	Flying-Qualities Level (probability)			
	1	2	3	<3
1	0.999+	$0.848X10^{-6}$	$0.250X10^{-14}$	$0.382X10^{-9}$
2	0.999+	$0.848X10^{-6}$	0	$0.492X10^{-9}$
3	0.999+	$0.848X10^{-6}$	0	$0.500X10^{-9}$
4	0.999+	$0.848X10^{-6}$	0	$0.430X10^{-7}$
5	0.999+	$0.240X10^{-3}$	0	$0.798X10^{-6}$
9	0.999+	$0.819X10^{-6}$	$0.108X10^{-9}$	$0.382X10^{-9}$
10	0.999+	$0.848X10^{-6}$	0	$0.492X10^{-9}$
11	0.999+	$0.848X10^{-6}$	0	$0.500X10^{-9}$
12	0.999+	$0.883X10^{-6}$	0	$0.751X10^{-6}$

*Cases 6 through 8 omitted because the FBW function does not degrade in stages (it fails completely given appropriate combinations of failures).

However, certain factors are of interest. Cases 11 and 12, although they show only a slight increase in flying-qualities degradation for the fully triplex architecture, indicate a notable disposition toward system failure. This suggests that the triplex computers are a weaker point in configuration 12 than are the three AOA pairs. Case 8 (table 4.6-1) tends to reinforce this inference.

The architectures used here are based on earlier-design DFCSs that have been used in retrofitting aircraft that had previously used analog systems. In retrofitted systems the interconnecting wiring is usually dedicated point-to-point signal paths, with little use of Multiplexer (MUX) buses. Newer systems commonly use parallel MUX buses, and considerably more complex bus topologies will be used when true digital computational capability comes into common use. Newer systems will use broader bandwidths, damage tolerance, and soft-fault processing. As a result, extended Mean Time Between Failures (MTBF) of up to 90,000 hours and system availabilities of 0.95 at a service life of 40,000 hours will be achieved. Considerably more system integration will also lead to additional diversity in MUX bus organization.

4.7. The Airplane Simulation

A negative static stability margin was postulated as a requirement for this demonstration, and six RSS flight cases were developed. These flight cases are then analyzed to determine the pitch-axis dynamic behavior, and a non-real-time simulation is developed and checked against the predicted pitch-axis dynamics for the free or, unaugmented, airplane. The same flight-case data are then used in the RDFCS facility simulation. Next, the non-real-time simulation time-history responses are used to check the RDFCS simulation.

Both the non-real-time and the RDFCS simulations discussed in this chapter are implemented using a state-variable approach (figure 4.7-1). Here the pitch-axis dynamics of the free RSS airplane, as captured in the matrix A (figure 4.7-1), are quite divergent or unstable, so certain states must be fed back to enable a stability augmentation function. The state variables here are pitch, pitch rate, vertical-axis velocity, and horizontal-axis velocity. The first two states are inertially oriented and are directly measured by airplane sensors. The second two are referenced to the airstream incident on the airplane, and are combined to form the directly measured signals: AOA and true airspeed. These two air-data signals are produced using matrix C.

The above sensor feedback signals then appear in vector u (figure 4.7-1). Pilot stick inputs are applied through vector d.

Table 4.7-1 summarizes conversions of six basic wide-body transport-type flight cases to six corresponding RSS flight cases with -5 percent static stability margins. For flight case A1RSS, for example, calculations revealed the neutral point to be 53 percent of the Mean Aerodynamic Chord (MAC), where neutrality denotes that there is no pitch-moment change with change in AOA. More specifically, the stability derivative describing the airframe behavior goes to zero. The -5 percent static margin then corresponds to shifting the center of gravity aft to 58 percent MAC.

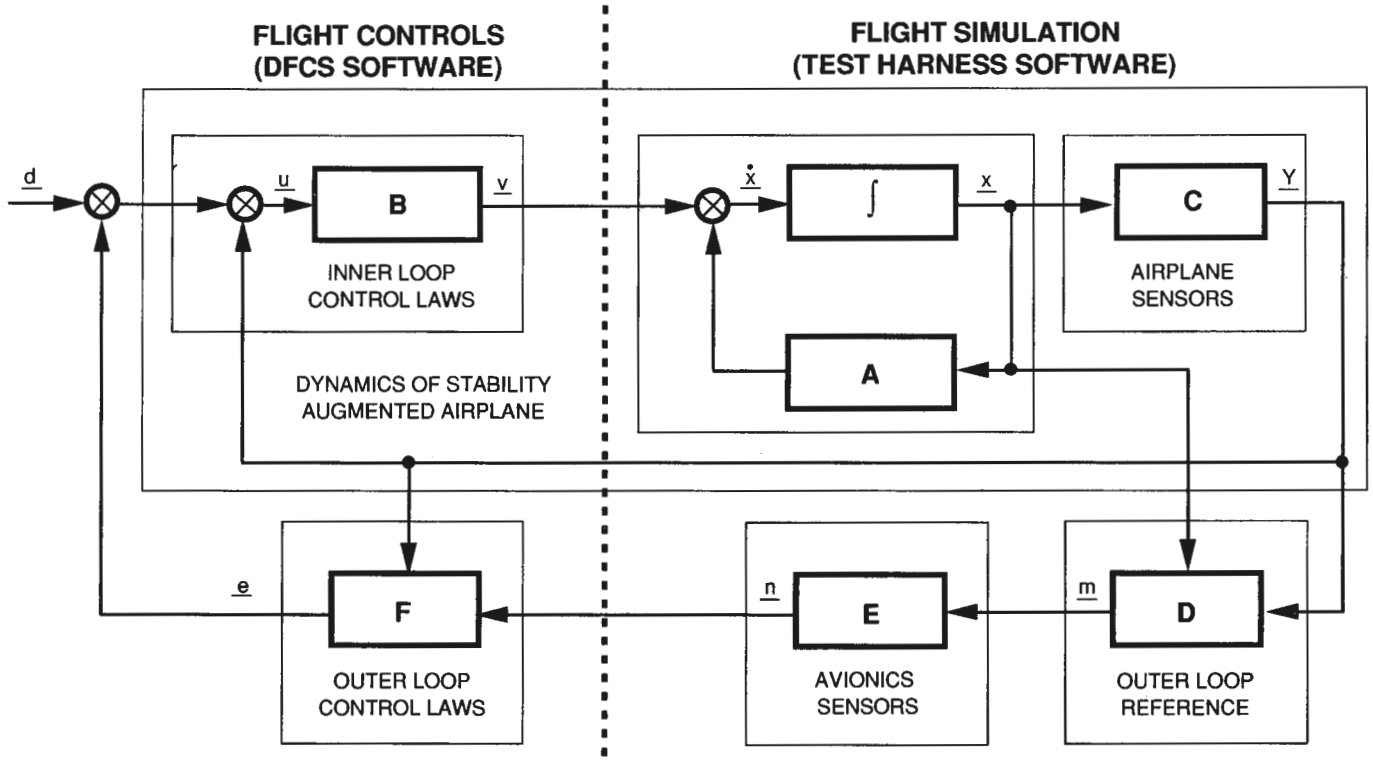


FIGURE 4.7-1. DIAGRAM OF STATE-VARIABLE AIRPLANE SIMULATION

TABLE 4.7-1. RELAXED STATIC STABILITY FLIGHT CASES

Parameter	A1RSS	C13RSS	C15RSS	D2RSS	F3RSS	F6RSS
c.g., $\%\bar{c}$	58.0	51.0	54.2	52.8	55.5	64.4
Neutral point, $\%\bar{c}$	53.0	46.0	49.2	47.8	47.8	49.4
V_T, ft/sec	283.7	910.7	442.2	487.1	265.7	224.1
τ_1, sec	1.099	0.160	0.122	0.660	0.938	0.920
τ_2, sec	-5.86	-2.40	-1.64	-7.55	-7.09	-3.17
ω_{PH}, rad/sec	0.169	0.008	0.134	0.125	0.167	0.209
ζ_{PH}	0.276	0.325	0.526	0.453	0.488	0.192
$\Delta C_M/\Delta\delta_H$, 1/rad	-3.247	-2.830	-2.415	-2.533	-3.138	-3.016
$\Delta C_M/\Delta_q$, 1/rad	-11.48	-14.99	-12.15	-12.75	-11.76	-11.23
$\Delta C_M/\Delta$, sec/rad	-3.14	-7.83	-3.44	-3.18	-3.14	-3.03
$\Delta C_M/\Delta C_L$	0.050	0.044	0.053	0.050	0.053	0.170
$\Delta C_M/\Delta$, 1/rad	0.295	0.353	0.288	0.292	0.306	0.960

This kind of alteration to the airframe dynamics yields a pair of real or non-oscillatory roots, τ_1 and τ_2 in table 4.7-1. Note that the negative time-constants for τ_2 correspond to positive roots, which plot on the positive axis of the s-plane and produce an absolute instability or exponential divergence that dominates the dynamic response of the airplane's pitch axis. With a -5 percent margin, moreover, the rate of divergence is quite rapid and clearly unacceptable. For flight case F6RSS, for example, the 3.17 second time-constant yields a doubling of pitch attitude in 2.2 seconds, which is so rapid as to be unflyable.

A non-real-time simulation was used to generate airplane time-history check cases before work was begun at the RDFCS simulator. The organization of the

previously been installed in the simulator test computer. For the RSS flight cases, no changes were made to the software organization itself, other than to add interfaces for the SAS control laws. The organization of the simulation provides for the conversion of flight-case data into a discrete-time model, trimming for specified initial conditions, and generation of dynamic time-history outputs. Ground effects, random gust options, or point simulation flight-case transitions may be selected.

Two stages of analysis were performed. First, the RSS airplane behavior was approximated by shifting the center of gravity aft to -5 percent MAC. It should be noted that six of the old flight cases have been converted into six derivatives sensitive to the reduced lever-arm of the empennage about the new center of gravity. As indicated in table 4.7-1, these all relate to the generation of a pitching moment. The analyses of the RSS flight cases then involved the examination of their respective dynamics. Specifically, the RSS stability derivatives were used to calculate the free-airplane response by finding the root solutions to the characteristic equation for each flight case. These results are also given in table 4.7-1.

Note that the sign on the pitching moment, owing to wing-lift, changes from negative for 25 percent to positive for 58 percent MAC. In the RSS case then, increasing lift produces more nose-up moment, which in turn further increases lift. This positive feedback causes an instability in the free, or unaugmented, airplane response. The rate of divergence is quite rapid. For flight case F6RSS, for example, the time-constant of -3.17 seconds yields a doubling of pitch attitude every 2.2 seconds. The given time-constants, radial frequencies, and damping ratios were obtained from the root solutions of the characteristic equations for the respective flight cases, as determined by the stability derivatives.

Simulation check cases were first run using the non-real-time dynamic simulation. Time-histories were generated for each case and checked against the root solutions. For the RSS cases, the divergent real root dominated the time-histories. The pitch-angle sequence exhibits an exponential time-constant roughly the same as the analytical calculated time-constant value of -3.17 seconds for flight case F6RSS once the initial transient has subsided. The initial upset of 3.0 degrees per second of pitch rate employed here is not really needed to exhibit the instability. It is used as a standard upset as needed for the corresponding augmented airplane.

Whereas these free-airplane responses are used as a cross-check on the real-time airplane simulation, the augmented airplane response is used to check both the simulation and the stability-augmentation control laws. Analytical-root-solution (eigenvalue) checks can also be made for the augmented airplane response, where the order of the characteristic equations is increased as a result of the presence of the sensor feedbacks. All these checks should be in general agreement. They then serve to corroborate the control laws used in different stages of development.

In general, the requirement for stability augmentation is to effect desirable flying qualities for the augmented airplane. Basically, the eigenvalues of the closed-loop airplane should exhibit suitable damping and frequency characteris-

closed-loop airplane should exhibit suitable damping and frequency characteristics. In effect, the augmentation control law should override the RSS airplane's positive pitching moment caused by increasing wing lift. Restoration of the negative pitching moment is possible through a nose-down stabilizer input for increasing AOA.

For the subject demonstration, the design criterion was simply to provide approximately the same, or better, pitch-axis handling and response for the stability-augmented RSS airplane than was available on the basic 25 percent MAC center-of-gravity free airplane.

5. VERIFYING THE SIMULATION

There are two major concerns with the verification of DFCSs: cross-channel synchronization and cross-channel consensus. Cross-channel synchronization poses the greater challenge because coincident, wideband access to the required logic signals, either hardware or software, usually demands some special instrumentation. In addition, means must be provided for rapidly appraising the observed logic states so that anomalies can be noted and captured before new events mask the attendant circumstances. Such appraisals must correlate events over the different channels, as well as reference external events such as fault insertions. For closure on assurance objectives, the online assessment must be based on precise representations of the verified design. These then constitute real-time test criteria when incorporated into execution monitors.

Observation and assessment of system availability and function/mode states are more tractable because of relaxed bandwidth requirements. Moreover, monitoring hardware test points is not generally necessary. The overall magnitude and complexity of the associated logic considerations means that a substantial amount of logic must be monitored. It is therefore desirable to minimize the number of logic variables that must be monitored. Perhaps the most formidable problem for the system monitor is that of judiciously defining test cases. The intent here is to maximize system-level coverage while acknowledging the unit-level test cases and coverage of previous programs. This is, of course, an issue largely independent of the construction and operation of the execution monitor itself. Note that in the case of the cross-channel synchronization monitor, previous unit-level testing is not very significant. This is because definitive testing is possible only after the hardware/software is available and operating.

As in most exploratory investigations, the scope of the DFCS test article was purposely limited, even though the quadruplex system architecture logic was fully implemented. Appropriate aspects of this logic are examined by two real-time execution monitors. The channel synchronization monitor for this implementation resides in one of the flight computers, and the system state monitor has been installed in the main simulation/test computer. The first monitor examined certain low-level events. On a larger scale it noted the readiness of each channel to operate with the flight-computer consensus.

Here the term consensus refers to the computer subsystem operational state as manifested by the majority of operable computers in terms of their synchronized, cooperative behavior. Most of the flight-computer faults that can occur compromise the affected computer's capacity to function and its ability to participate in this consensus. Such a loss would be noted by the system state monitor directly, as well as in terms of the DFCS system state consensus. Also, the synchronization monitor will capture some of the constituent logic sequences in the respective DFCS channels that culminate in the system state noted.

An automated testing scenario as generalized in figure 5-1 and elaborated on in table 5-1 has been implemented to simulate, observe, and record DFCS coordination phenomena. The test-case design has resulted from analysis of the DFCS specification and design, within the constraints of the capabilities of the test facility. Those constraints discouraged the use of automated test drivers, such as those used in Benson, Mulcare, and Larsen, 1987. However, this is not a significant limitation in this example, in which the emphasis has been on the real-time monitoring and results processing, instead of on testing productivity.

Multilevel testing is used to obtain conclusive evidence of the consistency of the DFCS implementation. Test-case design focuses on the architectural system/software logic indicated in figure 3-2. Here system validation is not at issue, so verification test scripts for the all-up system are devised only for the multiple levels indicated. Actually, these various level test requirements should be by-products of the corresponding stages of system development. The key point is that low-level design and implementation details should be examined simultaneously during system testing.

The DFCS simulator facility was implemented as shown in figure 5-2. The dual-dual architecture was replaced by a double fail-operational (quadruplex) architecture where each channel is relatively independent of all others. As a consequence, only one computer channel would fail at a time. The utility computer is used to retrieve selected software state data, through the test adapters, for the simulation/test computers.

Simulator testing was mechanized as shown in figure 5-3. The system consensus execution monitor, which was in the simulation/test computer, observed each channel's view of the consensus along with certain lower-level logic states. These parallel sets of data were used to check the consistency of the logic states of the operable channels. These states were referenced to the specified system states, according to component availability, and to lower-level logic design descriptions.

Not surprisingly, the bandwidth of the instrumentation computer was inadequate for monitoring and evaluating cross-channel synchronization logic. The fourth DFCS flight computer was used to monitor the synchronization logic of the other three, a function made possible because of the built-in cross-channel serial data transmission broadcast from each channel. This was also the medium used for synchronization control under the software-only modification rules for the test facility. Consequently, the monitor was able to view the synchronization process without having to fetch the wide bandwidth signals of interest.

Table 5-2 provides a breakdown of the multilevel monitoring functions and their focus. All but the fourth category, control flow, have been implemented and run. Still, there was ample quiescent time in the testing part of the system to conduct concurrent additional tests. The purpose of all of the monitors was to observe logic variable changes and their effects on related states. Figure 4.5-1 illustrates state-transition graphs that define the synchronization design within each channel. The monitor can then be interpreted as a data structure for each channel coupled with the specified outcome taken from the system level.

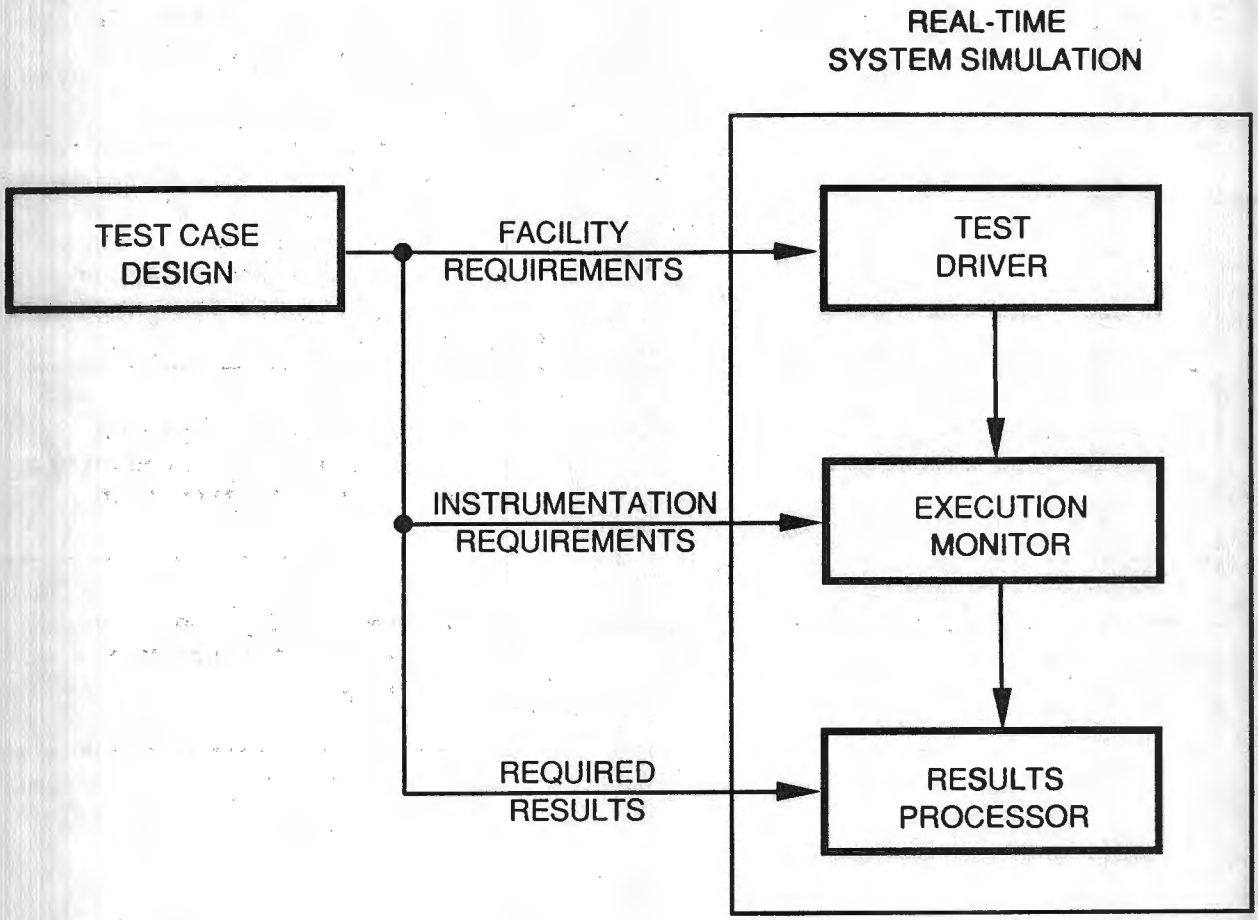


FIGURE 5-1. AUTOMATED TEST SCENARIO

TABLE 5-1. AUTOMATED TESTING TOOLS

Function	Type of Tool	Purpose
Test case design	Static analysis	Assessment of properties without actual program execution
Test driver	Automated testing	Automated control of test stimuli, environment, and data acquisition in a non-interfering manner
Execution monitor	Instrumentation	Implementation of probes for monitoring or recording during program execution
Results processor	Dynamic analysis	Assessment of properties during or after program execution

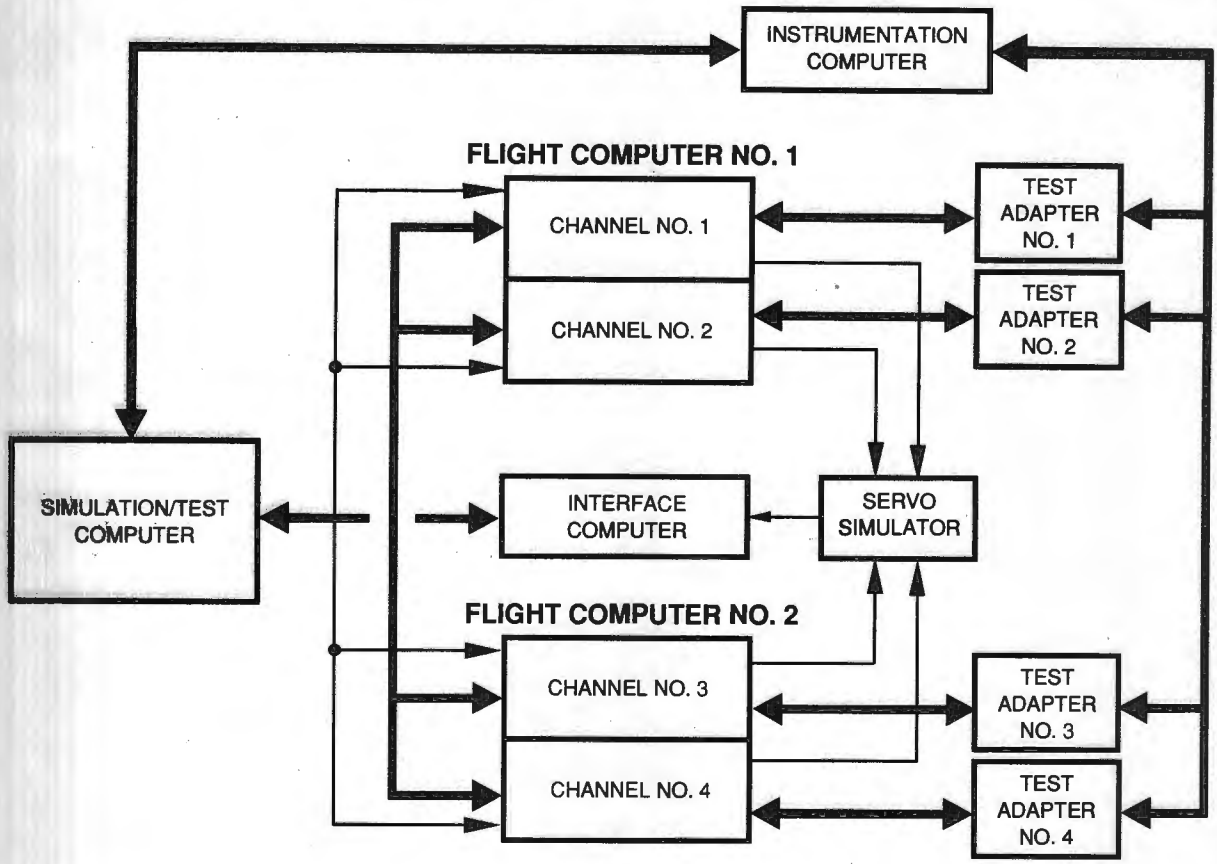


FIGURE 5-2. IMPLEMENTATION OF DIGITAL FLIGHT CONTROL SYSTEM SIMULATOR FACILITY

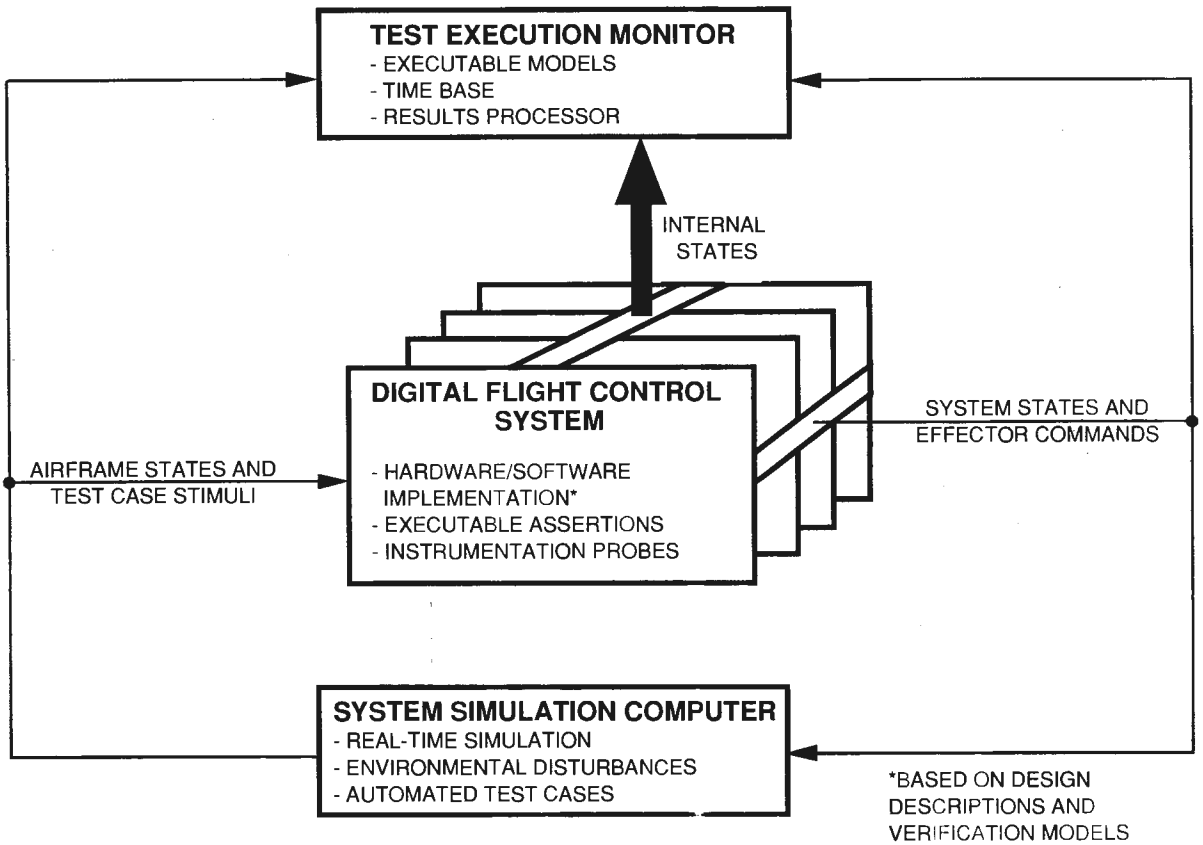


FIGURE 5-3. MECHANIZATION OF SIMULATOR TESTING

TABLE 5-2. MULTILEVEL TEST CASES AND MODES

Verification Focus	Execution Monitor	Monitor Location	Primary Concern
System State	Nested-state transition graphs	Simulation/test computers	System state consensus
Channel synchronization	Nested-state transition graphs	One flight computer channel	Coincidental faults
Mode/fault logic	Boolean expressions	Simulator/test computer	Logic correctness
Control flow	Control graphs with assertions	One flight computer	Software path traversal

6. SUMMARY

In addition to a variety of simulated hardware fault insertions, the quadruplex DFCS used for this example actually sustained an out-of-tolerance Flight Control Computer (FCC) clock problem during testing. This was at first thought to be a new DFCS design problem, but examination showed that the system was responding properly to discrepant hardware. It is not certain if the frequent, rapid resynchronizations would have otherwise been noticed since the defective channel remained online most of the time. This is one example of the ease with which a discrepancy, which might not be readily apparent at a higher level, can be noted at a low level. It is a case supportive of the subject execution monitor.

A distinction must be made between a physical error and a design error as the source of a discrepancy. As an instance of actually locating a design fault during simulator testing, one of the synchronization test cases that was applied disclosed a flaw in the verified design that was implemented in the DFCS. Specifically, the case involved two channel pairs, each synchronized for clock resetting and path traversal. But each pair continued to run without attempting to synchronize to the other pair. The resultant skew tended to yield some otherwise unwarranted voter/comparator logic trips. This occurred in flight-qualified hardware.

This problem was identified by iterating part of the design process, namely, that of reconstructing the discrepant operation in the predicate/transition network analysis simulation. Once this was done, the logic flaw and the required fix were identified. This illustrates not only the power of the diagnostic information provided by the synchronization execution monitor, but the need for better design verification test-case definition.

All of the other simulator test results merely affirmed (as intended) the correct operation of the DFCS logic as noted, for example, in figure 6-1. Affirmation was sufficient because verification is a process of confirming specified operation under the range of admissible conditions. Such confirmation was thought to have been achieved with a high degree of certainty through automated, multilevel testing, which depended crucially on the real-time execution monitors.

Despite certain inconveniences inherent in the RDFCS facility, the effectiveness of real-time execution monitors was considered excellent. There seems to be no substitute for the fidelity, precision, and conclusiveness provided in verifying fault-tolerance implementation.

As shown here, the execution-monitor testing approach should be supported by rigorous, early design verification as described in Barton, Mulcare, and LeBlanc, October 1985. Essentially, the same monitor criteria and logic model can be used both for design verification in pure simulation and for implementation verification in a system simulator. The interplay of these two assurance

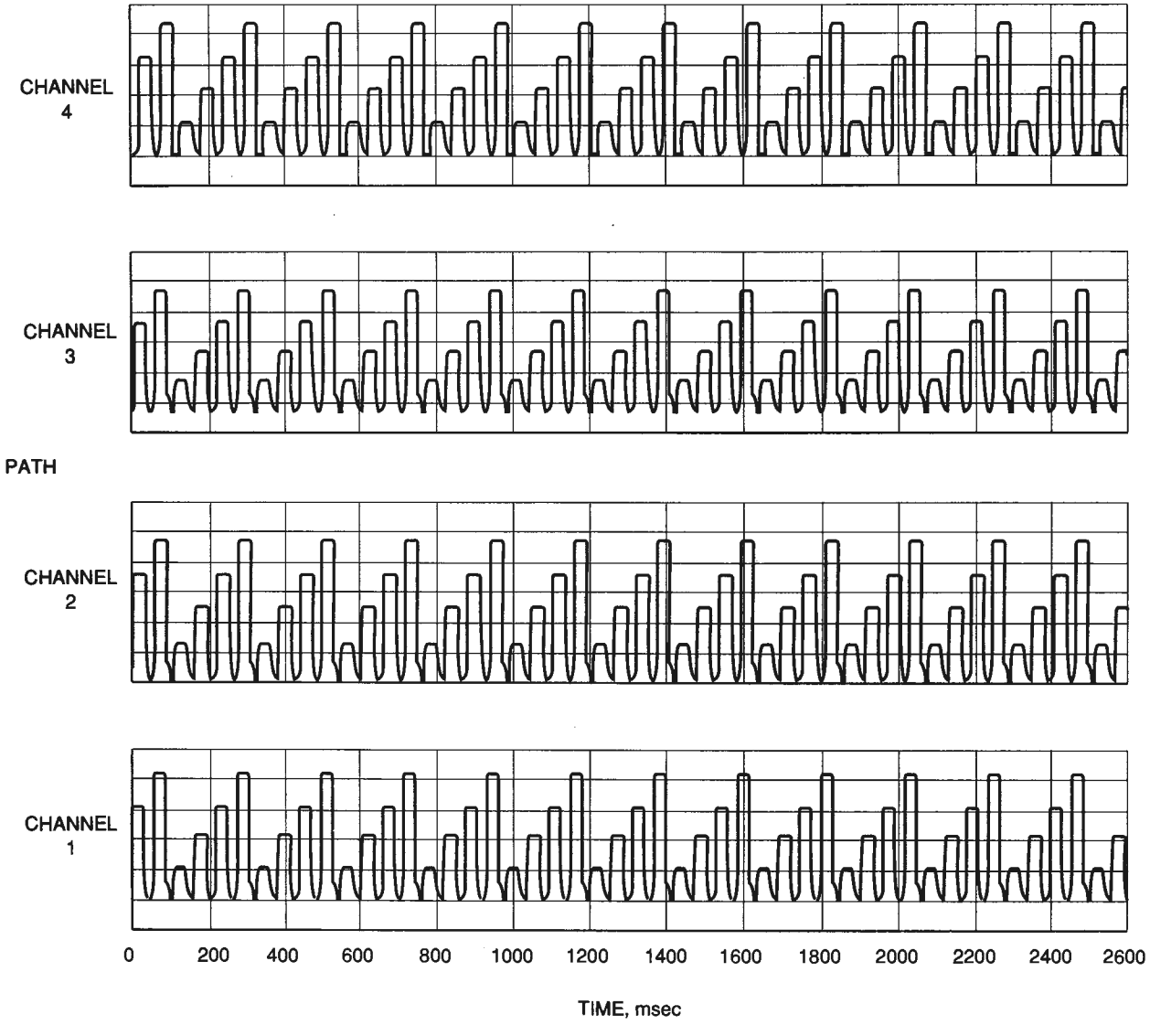


FIGURE 6-1. CROSS-CHANNEL SYNCHRONIZATION TIME-HISTORIES

activities, explicitly connected through the same precise criteria, tends to maximize confidence in the integrity and feasibility of system fault tolerance. This is fortunate, for generally the greatest source of fault-tolerant system unreliability is the fault-tolerance mechanisms themselves (Driscol, 1985).

The system state monitor aids in the interpretation of the synchronization monitor results. It can also be related to the status of DFCS components other than the flight computers. Moreover, the observed logic states can be readily correlated with flight software path traversals in an integrated testing strategy. In the interest of test productivity, comparable low-level monitoring of functions associated with different path traversals could be accomplished simultaneously. Multilevel testing would then be complemented by multifunction testing.

In summary, the demonstration of the execution monitors supports the addition of certain features to system simulators to facilitate the fault-tolerant architecture testing described. Moreover, such changes in the composition and use of system simulators appear to be a DFCS trend. As another example, similar or complementary low-level testing features have also been shown to be most valuable in the task of statistically calibrating low-level computer hardware fault-detection capability through intensive automated testing (Bensen, Mulcare, and Larsen, 1987).

BIBLIOGRAPHY

Barton, L. A., D. B. Mulcare, and R. J. LeBlanc, "Predicate/Transition Network Analysis of Redundant Channel Synchronization," Computers in Aerospace V Conference, October 1985.

_____, "Predicate/Transition Analysis of Redundant Channel Synchronization," AIAA Paper 85-6020P, 1985.

Bensen, J. W., Development and Implementation of the Real-Time Six Degree of Freedom Airplane Simulation for the Reconfigurable Digital Flight Control System, Report Contract NAS2-10270, December 1981

Bensen, J. W., D. B. Mulcare, and W. E. Larsen, Hardware Fault Insertion and Instrumentation System: Experimentation and Results, DOT/FAA/CT-86/34, March 1987.

Boehm, B. W., "Verifying and Validating Software Requirements and Specifications," IEEE Software, January 1984.

Defeo, P., D. Doane, and J. Saito, An Integrated User-Oriented Laboratory for Verification of Digital Flight Control Systems: Features and Capabilities, NASA TM-84276, 1982.

Driscol, K. R., "Reversion, A Response to Generic Faults," AIAA Paper 85-1981CP, 1985.

Goldberg, J., General Concepts of Validation. Methods for Fault-Tolerant Avionics and Control Systems, March 1979.

Larsen, W. E. and A. Carro, "Digital Avionics Systems: Overview of FAA/NASA/Industry-Wide Briefing," 7th AIAA/IEEE Digital Avionics Systems Conference, December 1986.

McCabe, T. J., "A Complexity Metric," IEEE Transactions on Software Engineering, December 1976.

MIL-F-9490D, General Specification for Design, Installation, and Test of Piloted Aircraft Flight Control Systems, U.S. Air Force, 6 June 1975.

Mulcare, D. B., L. E. Downing, and M. K. Smith, Quadruplex Digital Flight Control System Assessment, DOT/FAA/CT-86/30, November 1987.

Mulcare, D. B., W. G. Ness, and R. M. Davis, "Analytical Design and Assurance of Digital Flight Control System Structure," AIAA Journal of Guidance, Dynamics, and Control, May-June 1984.

Ness, W. G., et al., Integrated Assurance Assessment of a Reconfigurable Digital Flight Control System, DOT/-FAA/CT-82/154, April 1983.

Reed, J. E. and E. M. Boothe, "Digital Avionics, Active Controls, and the FAA: Advanced Integrated Flight Systems (AIFS)," 2nd AIAA/IEEE Digital Avionics Systems Conference, November 1977.

AC 25.1309-1, System Design Analysis, September 7, 1982.

Wensley, J. H., Design for Validation. Validation Methods for Fault-Tolerant Avionics and Control Systems, March 1969.

GLOSSARY

ANALYTICAL ROOT SOLUTION. Information obtained from the roots of the characteristic equations of the airplane model such as short-period frequency response.

ANGLE OF ATTACK. Angle between the longitudinal axis of an aircraft and the direction of movement.

ASSURANCE ASSESSMENT. Procedures whose purpose is to ensure that a proposed system functions according to design specifications.

BROADCAST. Transmission of messages to all terminals without reference to the identification of the receiving station or terminal.

CHORD. The straight line segment intersecting or touching an airfoil profile at two points.

DECOUPLED MANEUVERS. Changes in an aircraft's direction and attitude in one axis without affecting direction or attitude in other axes.

DESIGN ERROR. A functional flaw resulting from a misinterpretation of the specifications of the system.

DOUBLE FAIL-OPERATIONAL SYSTEM. A quadruplex (or higher) redundant flight control system which is designed to incur failures in two redundant lanes (or channels) before it fails.

DUAL-DUAL ARCHITECTURE. Two parallel dual computers with a voting plane at the output of each dual computing lane.

ENVELOPE LIMITING. General or additional limits imposed on the structural, "g" limits, speed, attitude, etc. of the aircraft. In some cases, envelope limiting imposes additional constraints on the envelope that cannot be exceeded regardless of pilot inputs.

ERROR. A mistake in specification, design, production, maintenance, or operation of a system causing undesirable performance.

EVENT, EXTREMELY IMPROBABLE. An event with a probability of occurrence on the order of 10^{-9} or less.

EVENT, IMPROBABLE. An event with a probability of occurrence on the order of 10^{-5} or less.

EVENT, PROBABLE. An event with a probability of occurrence greater than 10^{-5}.

FAILURE. The inability of a system, subsystem, unit, or part to perform within specified limits.

FAILURE, HIDDEN. A failure that is not manifested at the time of its occurrence.

FAULT. An error in the operation of a system.

FAULT, HARD. A defect in the hardware or software of a digital control system that permanently affects some functional performance of the system.

FAULT, SOFT. A transient defect in the software of a digital flight control system that can be overcome by error-correctable code or by recycling of power to the computer system.

FAULT INSERTION. A testing technique used to obtain information about data latency and built-in test coverage of a digital flight control system.

FAULT TOLERANT. Software which continues to operate satisfactorily in the presence of faults.

FAULT TREE ANALYSIS. A top-down deductive analysis that identifies the conditions and functional failures necessary to cause a defined failure condition. The fault tree can be used to establish the probability of the ultimate failure condition occurring as a function of the estimated probabilities of contributory events.

FLIGHT CODE. The application software of the digital flight control system.

FLIGHT-CRITICAL. A description of functions whose failure would contribute to or cause a failure condition preventing the continued safe flight and landing of the aircraft.

FLIGHT-ESSENTIAL. A description of functions whose failure would contribute to or cause a failure condition which would significantly affect the safety of the airplane or the ability of its crew to cope with adverse operating conditions.

FLIGHT-PHASE CRITICAL. A description of functions which are critical only during certain phases of flight.

FLY-BY-LIGHT. Flight control system where fiber optics carry the signal.

FLY-BY-WIRE. Flight control system with electric signaling.

GROUND EFFECT. Increase in aircraft lift when operating near the ground.

MEAN AERODYNAMIC CHORD (also mean chord). The chord of an airfoil whose length is equal to the area of the airfoil section divided by the span.

NEGATIVELY STABILIZED. Aircraft design in which the point of effective lift is aft of the center of gravity.

POINT-MASS SIMULATION. Same as state variables airplane simulation (q.v.)

POSITIVELY STABILIZED AIRCRAFT. Aircraft design in which the effective point of lift is forward of the center of gravity.

PREDICATE/TRANSITION NETWORK. A bipartite graph (a type of linear graph) to model concurrency between redundant concurrent events. Basically a modified generalized Petri net.

QUADRUPLEX ARCHITECTURE. The use of four separate lanes (or channels) of computer redundancy. Each lane can fail separately providing a fail-operational squared capability for the digital flight control system.

RELAXED STATIC STABILITY AIRCRAFT. An aircraft whose center of gravity is behind the wing's point of effective lift.

RELIABILITY ANALYSIS. A means of determining the probability of failure in a system. Military flight-critical systems typically are required to have reliability levels of 10^{-5} to 10^{-7}, whereas civil flight-critical systems have reliability levels of 10^{-9} or less.

STATE-VARIABLE AIRPLANE SIMULATION (also point-mass simulation). Fixed aerodynamic variables are used in the solution of the equations of motion of the model instead of using look-up tables in which each aerodynamic derivative varies with airspeed, altitude, etc. The model performance is only accurate at or near the point in the flight envelope for which the variables are chosen.

STATIC MARGIN. The degree of instability in a relaxed statically stable airplane.

SYSTEM EXPOSURE TIME. The period during which a system may fail. This period extends from the last verified proper functioning to the completion of the next required performance.

TIME CONSTANT. Time required to double amplitude of the divergent real root in the pitch axis of the aircraft model.

TRANSPARENT RECOVERY. Correcting a soft fault without interrupting the system's intended performance.

VALIDATION. Demonstration and authentication that a final product operates in all modes and performs consistently and successfully under all actual operational and environmental conditions founded upon conformance to the applicable specifications.

VERIFICATION. Demonstration by similarity, previous in-service experience, analysis, measurement, or operation that the performance, characteristics, or parameters of equipment and parts demonstrate accuracy, show the quality of being repeatable, and meet or are acceptable under applicable specifications.

ACRONYMS AND ABBREVIATIONS

AC	Advisory Circular
AFBW	Augmented Fly-By-Wire
AOA	Angle of Attack
CAPS	Collins Adaptive Processor System
DFCS	Digital Flight Control System
FAA	Federal Aviation Administration
FAR	Federal Aviation Regulation
FBW	Fly-By-Wire
FCC	Flight Control Computer
FCS	Flight Control System
MAC	Mean Aerodynamic Chord
MDICU	Modular Digital Interface Conversion Unit
MTBF	Mean Time Between Failures
MUX	Multiplexer
RDFCS	Reconfigurable Digital Flight Control System (facility)
RSS	Relaxed Static Stability
SAS	Stability Augmentation System

U.S. Department of Transportation
Federal Aviation Administration

DOT/FAA/CT-88/10

HANDBOOK-VOLUME II
DIGITAL SYSTEMS VALIDATION

CHAPTER 5
ADVANCED FAULT INSERTION AND SIMULATION METHODS



PREPARED BY:

COMPUTER RESOURCE MANAGEMENT, INC.
950 HERNDON PARKWAY, SUITE 360
HERNDON, VIRGINIA 22070

PREPARED FOR:

FEDERAL AVIATION ADMINISTRATION
TECHNICAL CENTER
ATLANTIC CITY INTERNATIONAL AIRPORT, NEW JERSEY 08405

NOTICE

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The United States Government assumes no liability for the contents or use thereof.

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the objective of this report.

TABLE OF CONTENTS

Section		Page
1.	FAULT TOLERANT SYSTEMS RELIABILITY PREDICTION BACKGROUND	5-1
1.1	Introduction and History	5-1
1.2	Reliability Requirements	5-2
1.3	Fault Tolerant Systems Performance Issues	5-2
1.3.1	Processor Redundancy	5-2
1.3.2	Byzantine Resilience	5-3
1.3.3	Synchronization	5-3
1.3.4	Scheduling	5-4
1.3.5	Deadlock Protection	5-4
1.4	Reliability Prediction Issues	5-4
1.5	Fault Modeling	5-7
1.5.1	Origin of Faults	5-7
1.5.2	Fault Models	5-9
1.6	Fault Tolerant Techniques	5-11
2.	FAULT INJECTION SIMULATION AND TEST METHODS	5-19
2.1	NASA FT System Research Summary	5-20
2.1.1	Architectural Design Methods	5-21
2.1.2	Detail Design Methods	5-21
2.1.3	Developmental Phase Methods	5-21
2.1.4	Systems Implementation and Evaluation Methods	5-22
2.2	Fault Insertion Methods	5-22
2.3	Software Simulation Fault Insertion Methodology	5-26
2.3.1	Method	5-26
2.3.2	Application	5-26
2.3.3	Example Results	5-26
2.3.4	Limitations	5-29

TABLE OF CONTENTS

Section		Page
2.4.1	Method	5-30
2.4.2	Application	5-31
2.4.3	Example Results	5-31
2.4.4	Limitations	5-32
2.5	Fault Emulation Fault Insertion Methodology	5-33
2.5.1	Method	5-33
2.5.2	Application	5-33
2.5.3	Example Results	5-34
2.5.4	Limitations	5-39
2.6	Physical Fault Insertion Methodology	5-39
2.6.1	Method	5-39
2.6.2	Application	5-39
2.6.3	Example Results	5-40
2.6.4	Limitations	5-45
3.	ADDITIONAL FAULT INSERTION AND TEST EXAMPLES	5-47
3.1	Draper Labs FTMP Tests Using Fault Emulation	5-47
3.1.1	Gate-Level Model For Redundancy Communicator Interstage	5-50
3.1.2	Fault Detection Effectiveness of Diagnostic Tests	5-52
3.1.3	Experimental Results on Hardware/Software	5-54
3.2	Electromagnetic Induced Transient Injection Test Examples	5-57
3.2.1	Electromagnetic Transients and Fault Injection	5-57
3.2.2	Upset and Failure Detection	5-60
3.2.3	Digital Engine Controller Test Examples	5-62
BIBLIOGRAPHY		5-71
GLOSSARY OF TERMS		5-76
ACRONYMS AND ABBREVIATIONS		5-78

LIST OF ILLUSTRATIONS

Figure		Page
1.4-1	FAULT TREE RELIABILITY MODEL	5-5
1.4-2	MARKOV MODEL QUADRUPLEX WITH LATENT FAULTS	5-6
1.5-1	SIMPLIFIED BREAKDOWN OF FAULT TYPES	5-8
1.6-1	FAULT CONTAINMENT REGION REQUIREMENTS	5-12
1.6-2	ARCHITECTURE FOR CONGRUENT TRANSFER FROM SIMPLEX SOURCE	5-14
1.6-3	FAULT TOLERANT PROCESSOR - FUNCTIONS	5-16
1.6-4	FAULT TOLERANT PROCESSOR - PHYSICAL VIEW	5-17
2.0-1	FAULTED PROCESSOR PERFORMANCE MEASUREMENT	5-20
2.2-1	FAULT LATENCY DATA	5-25
2.3-1	FAULT LATENCY DISTRIBUTION - GATE LEVEL FAULTS	5-27
2.3-2	FAULT LATENCY DISTRIBUTION - COMPONENT PIN LEVEL FAULTS	5-28
2.3-3	LOGIC DIAGRAM ILLUSTRATING DETECTABLE FAULTS	5-29
2.4-1	FAULT EMULATION COMPUTATION TECHNIQUE	5-30
2.5-1	FIAT FAULT INSERTION AND EXPERIMENT MONITOR	5-34
2.5-2	FAULT INSERTION MONITOR SOFTWARE	5-35
2.5-3	FIAT EXPERIMENT INTERFACE TREE	5-35
2.5-4	DETAILED VIEW OF FAULT INJECTION RECEPTOR SOFTWARE	5-36
2.5-5	REAL-TIME CHECKPOINTING SYSTEM	5-37
2.6-1	FAULT INSERTION AND INSTRUMENTATION SYSTEM (FIIS)	5-40
2.6-2	FTMP VAX DATA ACQUISITION ENVIRONMENT	5-41
2.6-3	DEVICE PIN ACCESS FOR FAULT INJECTION AND MONITORING	5-41
2.6-4	FAULT INJECTOR HARDWARE	5-43
3.1-1	EXPERIMENT DIAGRAM	5-47
3.1-2	C/I EXCHANGE NETWORK	5-49
3.1-3	ACTUAL CIRCUITRY FOR CROSS CHANNEL LINKS	5-51
3.1-4	SYSTEM FLOW DIAGRAM	5-53
3.1-5	C/I MODEL FOR DIAGNOSTIC EMULATION	5-55
3.1-6	DIAGNOSTIC TEST PERFORMANCE	5-56
3.1-7	COMBINED PERFORMANCE OF ALL NON-VOTER TEST	5-56
3.2-1	TIME-DOMAIN TRANSIENT WAVEFORM GENERATED BY LIGHTNING	5-58
3.2-2	BLOCK DIAGRAM OF THE HAMILTON STANDARD EEC	5-63
3.2-3	SIMULATION OF EEC INPUTS FOR EM TRANSIENT EXPERIMENT	5-64
3.2-4	BLOCK DIAGRAM OF EM TRANSIENT TEST INSTRUMENTATION	5-65
3.2-5	CLARIFICATION OF GATE DISTANCE DEFINITIONS	5-67

LIST OF TABLES

Table		Page
1.5-1	FAULT EFFECTS: MANUFACTURING VS. OPERATIONS	5-9
1.5-2	FAULT MODELS AT DIFFERENT LEVELS OF ABSTRACTION	5-10
2.1-1	VALIDATION AREAS FOR THE DESIGN PROCESS	5-21
2.2-1	FAULT INSERTION METHODS	5-22
2.2-2	FAULT INSERTION EXAMPLES	5-23
2.5-1	ERROR DETECTION STATISTICS - CHECKPOINTING ONLY	5-38
2.5-2	ERROR DETECTION STATISTICS - CHECKPOINTING AND CHECKSUMS	5-38
2.6-1	EXPERIMENTAL RESULTS SUMMARY FROM FIIS	5-44
3.1-1	TESTS USED FOR EXPERIMENT	5-54
3.1-2	SUMMARY OF UNDETECTED FAULTS	5-57
3.2-1	ERRORS IN SIMULATED EM TRANSIENT ON EEC	5-69
3.2-2	ERRORS IN SIMULATED EM TRANSIENT ON MICROPROCESSOR MODULE	5-69

1. FAULT TOLERANT SYSTEMS RELIABILITY PREDICTION BACKGROUND

1.1 Introduction and History

Application of microprocessor-based computers to flight critical systems is becoming more commonplace. The flight management systems on board the newest series of aircraft bring to aviation a new capability in management of the air traffic and additional capability for safe flight in marginal weather conditions. All-electronic engines have come into use on the latest generation aircraft as well. Stability augmentation puts the control of aircraft loads under computer control. The latest Airbus has brought military fly-by-wire concepts into commercial aviation for stability and control. This Airbus, operated by a new generation of on-board computers, offers greater fuel efficiency and smoother flight path control under varying aircraft configurations. Additional computer applications are under development to increase future aircraft system automation for stability and control, and for flight management. Considerable attention has been directed toward the technology and methods for validating the performance of these flight critical systems (National Aeronautics and Space Administration [NASA] Conference Publication 2114, Working Group Meetings I and II).

Failures of these advanced electronic systems would place a severe load on the pilot and crew in operating the aircraft without automation. The reliability of electronic components is not good enough to ensure a sufficiently low probability of failure in flight critical systems. Computer processors, having millions of components on a single chip of silicon, are available. Testing of these devices under all possible conditions is not feasible. Single logic circuit failures within these high density Very Large Scale Integration (VLSI) components must not cause loss of function. Such loss of function could arise from either a random failure or a defect present within the device that escaped detection in the original design and fabrication of the device. If a microprocessor is part of a system, the user must assume that the device may have some inherent fault that eluded detection during manufacturing.

The addition of duplicate, redundant, computers is a promising approach to ensure that at least one good computer will be available in the event of failure during flight. Redundancy has been useful in many prior aircraft systems; the low cost, weight, and power requirements of microelectronic devices make this attractive. Pioneers in reliability modeling demonstrated mathematically that system reliability could be enhanced by redundant hardware. These results are based upon the assumptions that the devices were initially fault free, that failures occur independently, and that errors can be detected by comparing the results from redundant components. More recently it has been shown that increased reliability can be attained, even if the components have inherent faults and not all failures can be detected (Arnold 1973).

The critical factor in achieving increased reliability through redundant components is the determination of how the system detects and processes faults. Therefore, the study of faulted behavior of digital circuits has expanded for two reasons: (1) to determine whether the output is correct for a given input and the device is sufficiently complete to manufacture and distribute, and (2) to determine the effectiveness of the device or system to detect, isolate, and recover from faults and errors. Because of the complexity of the computer software and the system (i.e., the number of component circuits), the entire study and testing is viewed as a stochastic process.

This chapter deals with system reliability issues in depth, through discussion of the concepts and methods for system testing by inserting faults and measuring the recovery process. An extensive bibliography is included for further reading on this subject (references that are given in the text by the author's last name and date refer to entries in the bibliography). A majority of the discussion material results from efforts by NASA Langley over the last ten years. When NASA began these investigations, reliability modeling was almost entirely based upon the failure modeling of single components. Data for these models were obtained for military programs from accelerated life cycle testing and 100 percent acceptance tests on receipt. These methods are too costly and too restrictive for commercial programs. Furthermore, the concept of 100 percent testing of microcomputers has proved impractical to implement, even for MIL-SPEC parts, due to the size and complexity of the test effort. Some of the work on fault insertion and testing of fault tolerant systems is still in progress and all the answers are not available for this challenging topic.

1.2 Reliability Requirements

The reliability of a system is the probability of performing a given function from some initial time, t=0, to time t. Typically, it is more convenient to deal with the probability of failure, which is one minus the reliability. For mission critical applications, a typical requirement is that the probability of failure be no more than 10^{-5} per hour. For flight critical safety related applications, the acceptable probability of failure is 10^{-9} per hour.

1.3 Fault Tolerant Systems Performance Issues

1.3.1 Processor Redundancy

Redundancy uses the execution of the same code on two processors to detect and correct the effects of faults before they propagate errors into other tasks in a system. Reliability predictions are based upon independence of the faults in the individual processors; i.e., a fault affecting one processor must not be able to cause an error in another processor. Fault independence can be enforced by separating the computations into fault containment regions (FCRs). An FCR is a collection of components that operates correctly regardless of any arbitrary logical or electrical fault outside of the region. If the entire system is partitioned into FCRs, an arbitrary fault inside such a region of containment cannot cause any component outside that region to fail in any manner. Therefore, fault tolerance includes containment of faults and correct handling of errors.

In order to use hardware redundancy effectively against random component failures, it is necessary that faults in different FCRs be independent and uncorrelated. To avoid common mode failures, each FCR must have a power supply and clock independent and separate from those of any other region. Furthermore, it is necessary to provide electrical isolation at all inter-FCR communication points.

Duplex redundancy is sufficient to detect errors, however, triplex redundancy is needed to mask errors. Redundancy is very effective if all the computations are performed in lock step with identical inputs and intermediate steps. Exact match voting can then be used to detect and mask errors. Thus any value not bit-for-bit identical to the other two is considered to be in error and is discarded. The input values must be interchanged between the FCRs to ensure that the same values are used by all three processors. If the processors are not in lock step synchronization, exact match voting cannot be used, and fault detection and masking is compromised.

1.3.2 Byzantine Resilience

A fault tolerant process is termed "Byzantine Resilient" if it is tolerant of intermittent faults that can send good information part of the time. The term alludes to the problem faced by Byzantine generals trying to authenticate messages sent by traitorous messengers. Unfortunately computer components do not often fail in benign ways. Components involved in communication between processors have been known to fail in ways that send conflicting information to different parts of the system (Johnson 1989). If the redundancy is not sufficient to isolate these failures, then they are very difficult to isolate and correct. The difficulty arises because a processor correctly processing incorrect input will appear faulty to the rest of the system. In order for a system of parallel processors to be Byzantine resilient the network topology must be divided into four separate FCRs. Each computation must be performed in two separate regions; whenever data values pass from one containment region to another the communications must be via separate paths. If a containment region no longer has three separate paths for communications, then the system will no longer be Byzantine Resilient. The reconfiguration of such a system is very complex.

1.3.3 Synchronization

Synchronizing individual processors in a redundant computation makes it possible to schedule completion of individual process steps. This simplifies the determination of whether a processor has failed or simply is slow to complete a task. Failures of synchronized processors must not propagate beyond a containment region. Global clocking methods are not recommended because they permit too high a chance of a single point of failure. Much research has been done on synchronization methods. Various methods range from loosely coupled methods to lock step synchronization. The computation overhead for correctly dealing with processor synchronization is usually very large.

1.3.4 Scheduling

Task scheduling in the redundant processors requires a real-time executive that can operate in the presence of faults. This executive must be responsible for reconfiguration and fault masking on data between processors. The executive must also maintain data integrity in the presence of faults. This means that multiple copies of the executive must exist in the various FCRs.

1.3.5 Deadlock Protection

It is necessary to guard against the deadlock condition where every processor is waiting for a task to be completed as a subtask on another processor. Parallel processors require specific software measures to prevent deadlock. If the operating system provides no deadlock control during execution, then it must be shown, for each process, that no deadlock can occur in the worst-case combination of events. Implementation of a program to ensure that deadlock cannot occur is extremely difficult. The information necessary to determine deadlock is usually not available until operational testing is performed. Deadlock recovery schemes often use rollback to a prior processing step where the state was recorded. A combination of avoidance and recovery is usually sufficient to handle deadlock. If avoidance is used, it must be shown that the system will avoid deadlock under all possible circumstances or that deadlock is extremely unlikely (i.e., the probability acceptably low). If roll-back recovery is used, the probability that the system will miss a real-time deadline due to roll-back must be shown to be acceptably low.

1.4 Reliability Prediction Issues

Modern reliability prediction of fault tolerant systems is a mathematically intensive problem usually requiring computer modeling and analysis tools such as those developed at or in conjunction with NASA Langley (Hybrid Automated Reliability Predictor [HARP], Fault Tree Compiler [FTC], Computer Aided Reliability Evaluator [CARE III], Pade Approximation With Scaling [PAWS], Scaled Taylor Expansion Matrix [STEM], Semi-Markov Unreliability Range Evaluator [SURE], etc.). These computer programs differ from one another in their modeling complexity and solution techniques, but have been shown to be largely in agreement for predicting reliabilities of systems within each of their stated limitations.

The reliability models utilized are limited by their techniques for mathematically describing hardware-type events and solution algorithms. They are also limited by the availability of accurate data for input to the model.

Typically, the failure and recovery of a system of components is described by some type of fault tree diagram, such as those illustrated in figures 1.4-1 and 1.4-2. These modeling examples provide just a small glimpse of the complexity of modern reliability prediction problems.

One of the most direct reliability models is a fault tree. The fault tree shown in figure 1.5-1 is used to calculate the failure probability of the system for raising and lowering aircraft landing gear. This type of model was developed in 1961 by Bell Telephone Laboratories. The technique starts with a single

event and delineates all possible "basic event" paths to that single event. Each basic event must be independent, the characteristic distinction of a random/stochastic process. Each fundamental event must also have a known probability (perhaps calculated from another fault tree). This fault tree, in itself, will not allow for the reconfiguration in a modern fault recovery process, but it is a valuable tool for ascertaining the probabilities and failure rates of other parts of a system.

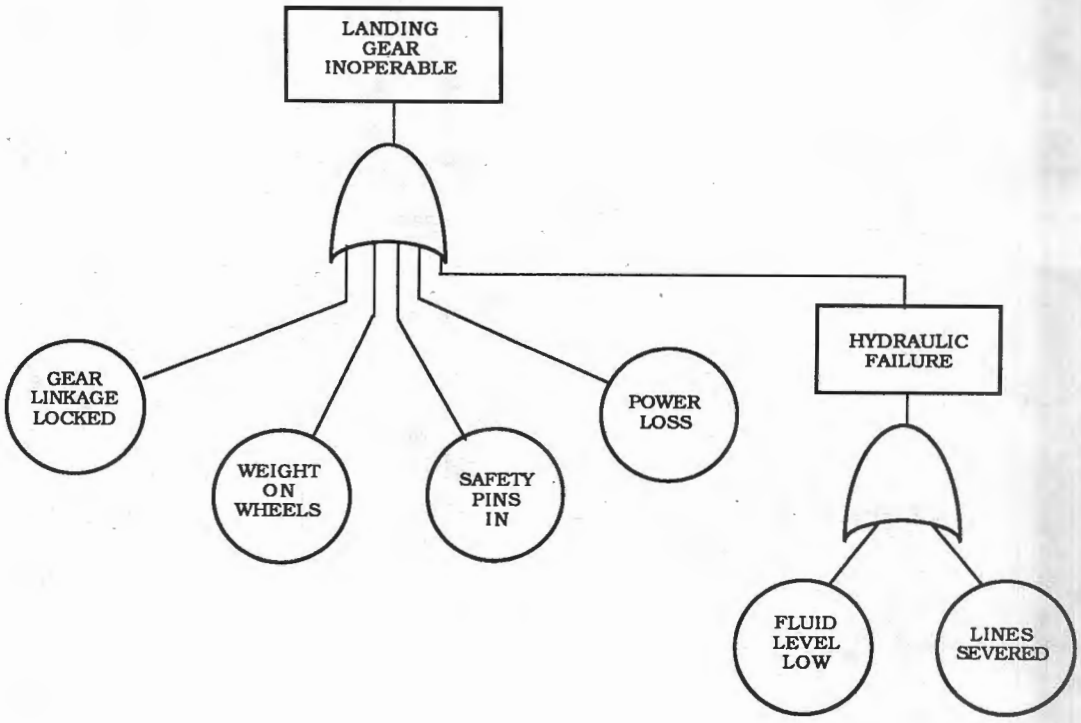
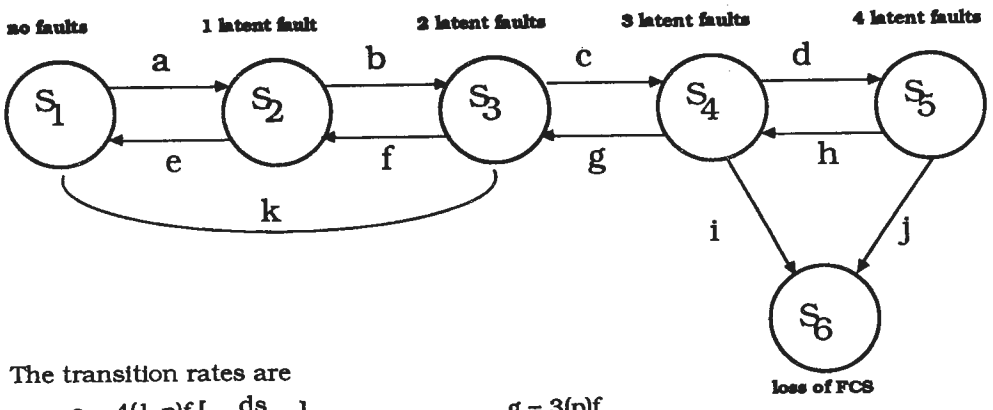


FIGURE 1.4-1. FAULT TREE RELIABILITY MODEL

A Markov model is required to incorporate the fault recovery reconfiguration process. Figure 1.4-2 illustrates a Markov model for a quadruplex Flight Control System (FCS) with latent faults (McGough 1988). The key difference from the fault tree of figure 1.4-1 is the incorporation of multiple end states, thus allowing for reconfiguration paths through activation of redundant components or degradation to simplex states (node 7 in figure 1.4-2). Transitions to each state are characterized by a time rate for failure and recovery. The Markov model becomes semi-Markov if the recovery rates, F(t), are not simply approximated by the common exponential distribution rates used for component failures. However, this adds significant complication to the mathematics required for solution of the model.

Computation of the reliability of a fault tolerant system is more complex than the examples shown in figures 1.4-1 and 1.4-2. The reliability of the overall system is a function of the reliability of the individual components of the system. It is also a function of the fault recovery process used in the fault tolerant system. Individual part reliabilities may be obtained effectively from life cycle testing and materials analysis. This data is fundamental to the failure prediction methods used by the most common component reliability prediction (MIL-HDBK-217E 1986). Including behavior such as fault recovery in the system during the presence of a fault is even more challenging. This data must be obtained from tests run on both hardware and software under controlled fault injection into real and simulated systems. Data obtained in this manner is ordinarily specific to the system under test.



The transition rates are

$a = 4(1-p)f\left[\frac{ds}{as+ds}\right]$

$b = 3(1-p)f$

$c = 2(1-p)f$

$d = (1-p)f$

$e = as + (p)f$

$f = 2(p)f$

$g = 3(p)f$

$h = 4(p)f$

$i = as$

$j = as$

$k = as$

Note that the backwards transitions (e, f, g, h, and k) are the result of repairs due to detection of faults.

FIGURE 1.4-2. MARKOV MODEL QUADRUPLEX WITH LATENT FAULTS

As has been indicated, the failure rate of components is an essential part of the data needed for reliability prediction. Studies have centered around the testing of components in manufacturer's plants and during incoming quality assurance inspections and testing at systems manufacturing facilities. The most comprehensive data is available for the large volume components and originates from the larger manufacturers. Testing is almost nonexistent at small and low volume commercial facilities. Device testing is much more stringent for MIL-SPEC parts.

Logic devices may be tested by generation of test pattern inputs to a device. Their outputs can then be compared to a known correct response. This approach

works well for combinatorial devices, but not so well for microprocessors. Testing of microprocessors has become a significant problem due to the incredible complexity of VLSI designs. Millions of logic gates, sequential and parallel data propagation, and very fast (to the tens of nanoseconds) timing relationships make it virtually impossible to expose the microprocessor to all possible combinations of inputs and outputs. For most applications the level of testing is a direct function of economics.

The inclusion of redundant fault tolerant hardware helps circumvent overall system failure due to the failure of a single component. Studies (Arnold 1973 and Johnson 1989) have shown that system reliability is enhanced through redundancy even with less than 100 percent fault detection; however, the fundamental operation of the fault tolerant system is fault detection and recovery. Therefore, fault recovery data is crucial to the accuracy of the reliability prediction model. One of the significant purposes of controlled fault injection into fault tolerant systems is the procurement of fault recovery data which may be used in the reliability prediction model. Complex and innovative system architectures make it difficult to predict fault recovery behavior based upon history and experiments on other types of systems. Experimental fault injection is usually required to ascertain a specific system's reliability.

1.5 Fault Modeling

Validation of fault toleration systems requires an understanding of faults. This section discusses the origin of various faults and models that have been defined for them.

1.5.1 Origin of Faults

The wide range of faults and where they occur in digital systems performance is illustrated in figure 1.5-1. These faults may be classified in three general areas: design errors, manufacturing errors leading to faults, and faults occurring in-service.

Design errors are the result of improperly translating an idea or concept into an operating system. Gathering information on design errors is difficult because each error occurs once per system development and the errors are usually complex. Design errors have been extensively studied in the software realm due to the increased emphasis on reliable software and the fact that software faults are entirely due to design and coding errors. (In general, coding errors will be identified and eliminated during testing.) Such errors often enter into the design of algorithms where partitioning into separate regions of validity is necessary; for example, with square roots, arc tangent, arc sine, and so on. If sufficient testing is not done between regions, the value at the region boundary can become undefined or very large or small. The effects of these errors are difficult to generalize due to their infrequency and diversity, hence few generalized models are available. These faults may produce little or no effect until just the exact combination of input conditions and time history present themselves.

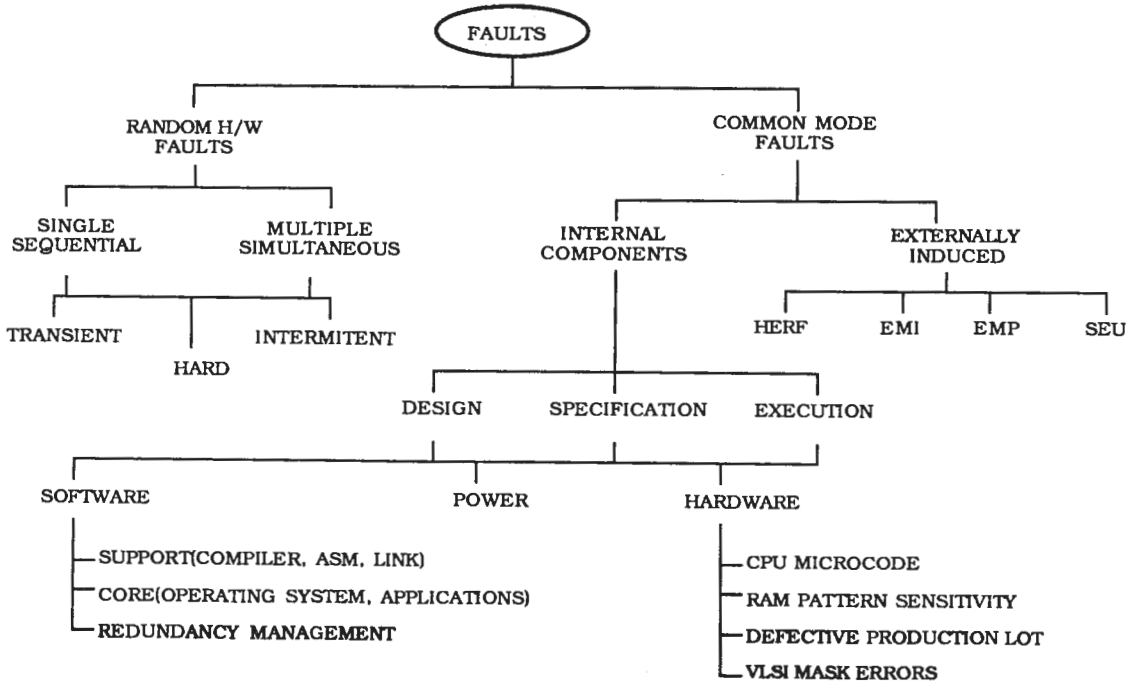


FIGURE 1.5-1. SIMPLIFIED BREAKDOWN OF FAULT TYPES

Manufacturing defects may be introduced by improper processing or by the use of defective materials in the fabrication of the system. Examples of manufacturing defects in semiconductor devices include cracked dies, over or under doping of the semiconductor material, and flaws in the mask. Effects of manufacturing defects on Complementary Metal-Oxide Semiconductor (CMOS) circuit behavior have been characterized. In a manufacturing research program, defective masks were utilized to fabricate several circuits and defective parts examined for faults (Ferguson 1987, Marchall 1985). Results of this study determined that 99 percent of all faults can be characterized by bridging or breaks, while only 50 percent of the faults can be modeled by stuck line faults.

In-service failures are caused by physical processes occurring through normal and abnormal use over the life of the system. Following are examples of CMOS device failures:

- Oxide breakdown due to high electric fields in the insulator, usually in the gate oxide.
- Electromigration, drifting of metal atoms toward the cathode, influenced by high current density in the semiconductor.
- "Hot electron" trapping in the gate oxide in the channel of a Metal-Oxide Semiconductor (MOS) transistor due to high temperature and high electric fields.

- Electron hole pairs or alterations to stored charge due to impacts by cosmic rays and alpha particles.

- Physical failures due to electrical overstress caused by overvoltages, electromagnetic (EM) environment (interference (EMI), radiation (EMR), lightning), and electrostatic discharge.

Other life cycle failures are caused by many other sources including design deficiencies, production techniques, mechanical stress, thermal stress, moisture, and corrosion.

Very little has been published on the distribution of in-service faults due to manufacturing processes and the operating environment. Attempts to map physical failures into logical models are not satisfactory. Several failures resulted in stuck-at behavior, while others resulted in parametric faults. An interesting contrast is apparent between in-service and manufacturing operations. The results of one study (Ferguson 1987, reported in Czeck, Siewiorek, and Segall 1989), are shown in table 1.5-1.

TABLE 1.5-1. FAULT EFFECTS: MANUFACTURING VS. OPERATIONS

FAULT EFFECT:	MANUFACTURING STAGE	FIELD
Oxide	Less than 10%	25% - 75%
Metal	30% to 40%	4% - 17%

During in-service operations, the highest failure rate is for gate oxide (25 to 75 percent) and the next highest is for opens and shorts in metal runs (four to 17 percent). Oxide failures result in transistors being stuck off. Metal failures result in opened and shorted lines, and in breaks and bridges. During manufacturing, the most common failure (30 to 40 percent) is due to metalization, with a minor component (less than 10 percent) of faults causing transistors to have stuck-off faults.

1.5.2 Fault Models

Fault models are abstractions of failure mechanisms. They range from the physical device level to the gate, functional, and architectural level. A wide range of fault models have been developed to generate test programs. Fault models at various levels of abstraction, illustrated in table 1.5-2, cover the levels of system detail from network to switch.

TABLE 1.5-2. FAULT MODELS AT DIFFERENT LEVELS OF ABSTRACTION

LEVEL	FAULT MODELS	BASIS	LIMITATIONS
Network	Communication lost, delayed or unordered. Lost nodes.	Abstraction of behavior.	Failure modes are unknown and complex.
Physical Message Switch	Data Change. Message or process lost. Data inconsistent. Time outs.	Abstraction of behavior.	Failure modes are unknown and complex.
Remote Terminal	Data change. Wrong assertion, source, or destination.	Abstraction of behavior.	Based on RT models not implementation.
Functional	Complement or dual function Truth table modification.	Observation? Ad hoc.	Many realizations and fault modes.
Gate	Gate output stuck at 0 or 1, Single stuck line.	TTL and PC board behavior.	Technology outdated.
Switch	New and missing devices, Shorts and breaks (opens), Transistors stuck on/off.	Processing defects.	Simulation overhead, difficult to observe and fault insert.

Switch-level fault models are used for MOS devices where unidirectional logic gate models do not provide the bidirectional behavior of bridging, stuck open, and stuck closed transistors. Switch level models contain nodes connected by bidirectional transistor switches. Faults are modeled as nodes stuck high or low, transistors stuck open or closed, and extra or missing transistors. Low level simulations and models are required because the failure modes of the devices, especially CMOS, cannot be modeled at the gate or higher levels. Switch level simulations are a great aid in the design and generation of test programs, and assessment of test coverage, but the simulation is difficult to use for large systems due to the excessive run time for the simulations.

Gate-level fault models assume that inputs and outputs of gates are stuck at high or low logic levels, but the gates function correctly. These faults are based upon printed circuit boards, Transistor-Transistor Logic (TTL), and pre-TTL logic of the 1960s. The gate-level model is not applicable to MOS implementations since failure modes in CMOS can transform a combinational circuit into a sequential circuit. Furthermore, complex MOS circuit implementations do not map gate lines to circuit nodes.

Studies were conducted to determine the effects of processing mask defects on faults by mapping the defects to circuit fault models. These studies showed that only 50 percent of the faults are representable by gate-level models having a single line stuck-at fault. Further studies showed that the development of

fault models should be independent of the technology and updated as technology advances.

There is continuing research underway on higher levels of abstraction in fault models. Czeck, Siewiorek, and Segall 1989 provides a guide to this literature. This level of abstraction is driven by the VLSI in the electronic circuitry, and the complexity of dealing with testing based upon implementation faults. Some work has been completed laying the ground work for functional testing based upon the possible scenarios of failures occurring within the control section, data section, or data storage section of a generalized microprocessor. These models were independent of the implementation, hence they did not contain a fault library. The models were useful, however, in presenting a methodology for developing a set of test procedures for microprocessors.

1.6 Fault Tolerant Techniques

The degree to which a system can recover from faults depends critically upon the mechanisms for detecting and isolating faults. A careful distinction must be drawn between those techniques which are approximate and those which achieve near perfect fault detection and recovery. The following discussion is based upon a reference document from Charles Stark Draper Laboratory (CSDL) (1986).

Consider, for example, a dual channel system that relies upon bit-for-bit comparisons of the two outputs for fault coverage. This approach provides nearly perfect fault detection coverage. If a fault in one of the channels affects the correct operation of the channel output, then the output of that channel must differ from that of the other correctly operating channel. The latency between the occurrence of the fault and its appearance as an output error depends upon how frequently the malfunctioning hardware is exercised by the operating software (control routines and diagnostics). A fault will be readily detected when miscompare occurs, but it will not be obvious which channel is correct.

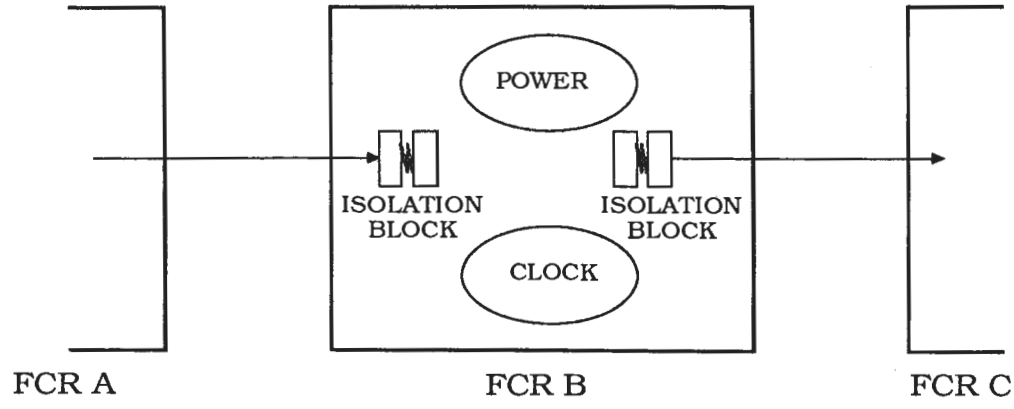
Diagnostic software routines may be run to isolate the channel which is faulty in a dual-channel system, but the coverage of the self-test software is often quite limited in its ability to detect faults. An example of a troublesome fault that is difficult to isolate is a pattern memory error in one bit position of the system memory. This fault may appear as a "dying one" (i.e., a one that is surrounded by all zeros occasionally turns into a zero). To catch such a fault requires a large number of runs with a great number of patterns on the memory. The likelihood that a "dying one" fault will be detected in a diagnostic test is not very large, and certainly not close to one hundred percent.

The addition of ad hoc solutions such as parity checks can help to isolate the memory pattern faults. Special circuitry to disassociate faults in the rest of a system adds considerable overhead in hardware and increased complexity. Furthermore, there is no technique for adding self-tests to logic circuits.

Other common techniques, which result in less than perfect coverage, include temporal checks, integral checks, and diagnostic checks. Watch dog timers have been used to catch certain hardware faults as well as programs stuck in infinite

loops. Integral checks are built into the basic number representation. Parity, Hamming codes, and Bose-Chaudadhary codes are all examples of programs written specifically to exercise the hardware. Examples of this type of check include Read-Only Memory (ROM) checksums, voter tests, and processor opcode tests. These methods add to the coverage of faults, but each one addresses only a limited number of possible faults, and unless all fault types are considered, the system will not achieve high fault coverage.

Current thinking on fault tolerant computer systems has identified four requirements that must be met to achieve a high degree of fault coverage. Any process which is critical to the operation of the system must be replicated. Critical processes must be correctly mapped into FCRs, such as defined in figure 1.6-1. There must be a sufficient number of FCRs and communication channels between them. And finally, there must be an "exact consensus" (bit-for-bit and synchronous, to be discussed later) among replicated processes. These requirements, described more fully below, are applicable for all processes, whether application program or operating system code.



FAULT CONTAINMENT REGIONS (FCR)
REQUIREMENTS:

1. INDEPENDENT POWER
2. INDEPENDENT CLOCKING
3. ELECTRICAL ISOLATION
4. PHYSICAL SEPARATION

FIGURE 1.6-1. FAULT CONTAINMENT REGION REQUIREMENTS

Most critical real-time control applications demand uninterrupted operation. Effects of faults must be masked by detection, isolation, and correction. These three processes must be mapped into three independent FCRs. In general, to mask the effects of f simultaneous faults, the nominal requirements call for (2f+1)

FCRs. However, this nominal requirement is not sufficient when the requirements for fault isolation are considered.

Fault isolation imposes additional requirements. The isolation of a faulty channel may be accurately determined for any output if the channels in the voting scheme are synchronized and have bit-for-bit agreement under normal, no fault, operation. Two processes which are initiated identically, receive identical inputs, have no failures, and operate on the inputs in the same way, are said to be congruent processes.

Congruence allows for a precise definition of the process of fault detection and isolation:

- Fault Detection - Two congruent processes not in bit-for-bit agreement indicate the presence of a fault.

- Fault Isolation - A process not in bit-for-bit agreement with the majority vote is the faulty process.

The concept of congruence has greatly simplified our thinking about fault detection and isolation. Since the processors are synchronized, the majority vote for a congruent process requires a simple truth table to determine the correct output and to isolate the faulty processor.

Incongruent processes need more complex fault detection and isolation algorithms and circuitry. These unsynchronized processes, having only approximately equal outputs, must rely on having "windows of agreement" or thresholds. In this case, fault detection coverage is a function of how precisely one can define these thresholds. For most dynamic processes these thresholds are a function of inputs and outputs. These parameters change with time and the operating mode of the systems. The use of application process driven thresholds for fault detection and isolation put a serious and uncalled for burden on the applications programs to ensure fault tolerance in the system.

Incongruent processes have been used successfully for tasks such as flight control, where computers were used as a one-for-one replacement of the analog counterparts. These implementations suffer from all the ills discussed earlier, including low fault coverage and excessive involvement of the applications programmer in redundancy management. Moreover, they also exhibit a high false alarm rate. These inexact consensus techniques will not work at all in more complex applications that require real-time fault isolation and reconfiguration of distributed system elements. Such systems communicate in abstract messages that have no semantic meaning (like actuator commands) if their redundant counterparts are only approximately equal.

Thus in the final analysis, fault detection, isolation, and masking need congruent processes. Achieving exact consensus in congruent processes having f faults, requires (3f+1) FCRs. This is more severe than stated earlier. In addition, processes must go through an algorithm for consensus that requires (f+1) communication rounds over links providing at least (2f+1) disjoint paths. For example, to tolerate a single fault there must be a least four FCRs, two communication rounds, and three disjoint paths.

This more precise mathematical basis provides a proof of fault tolerance from any arbitrary failure mode. This is driven from a need to account for more than Stuck-At-One (S-A-1) or Stuck-At-Zero (S-A-0) faults. A particularly difficult mode is one in which one receiver detects a one while the other receiver detects a zero. Such failures are termed Byzantine failures and can occur at sensor signal level, logic level, and even at program levels. For example, a redundancy management program in computer A can send the message to computer B that computer C is bad, while it tells computer C that computer B has failed. Although such modes may appear to be remote, they are in fact not unlikely. At least one in-flight failure of a triplex system has been traced to such a Byzantine fault and the lack of safeguards against such faults.

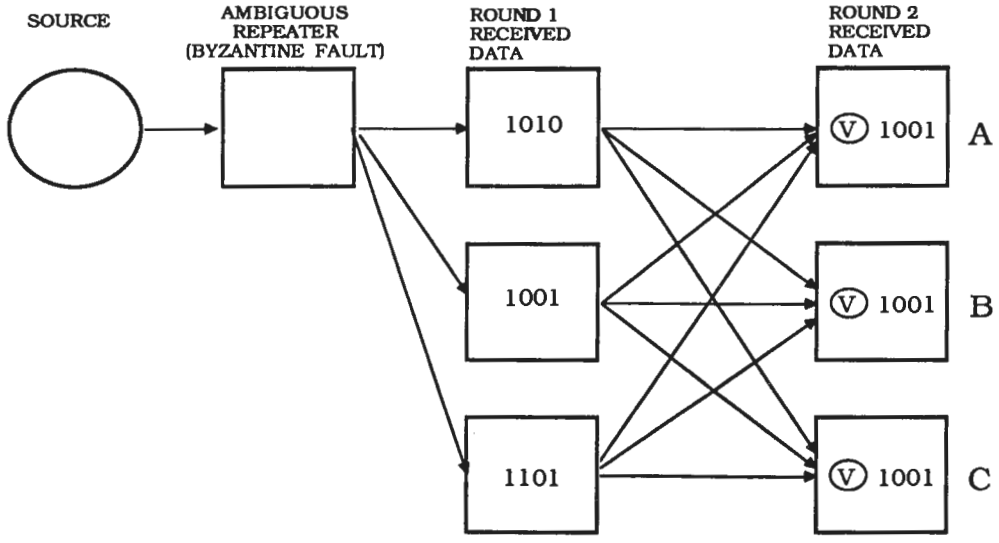


FIGURE 1.6-2. ARCHITECTURE FOR CONGRUENT TRANSFER FROM SIMPLEX SOURCE

An architecture that meets all the requirements for exact consensus in the presence of Byzantine faults is shown in figure 1.6-2. This diagram shows six FCRs that are interconnected by three disjoint paths and go through two-round communication protocol. Data is distributed, in each channel, to voters that produce majority voted outputs. The voters perform a two-out-of-three vote on a bit-for-bit basis. The least significant bits of the three values are voted to produce the least significant bit of the majority output. The next order bit in the three words is then voted to produce the next bit of the output. This process continues until the most significant bit is voted to produce an output word that is the bit-for-bit majority of the three inputs. In the diagram, a Byzantine fault in the form of an ambiguous repeater is postulated in the channel that is distributing the value of the sensor attached to that channel. After the two-round exchange and the bit-for-bit vote, it is evident that all correctly operating channels have identical or congruent values.

The CSDL Fault Tolerant Processor (FTP) architecture for implementing the requirements for fault containment and consensus voting illustrates these considerations. A picture of that fault tolerant implementation is shown in two views: a functional view shown in figure 1.6-3, and a physical layout shown in figure 1.6-4.

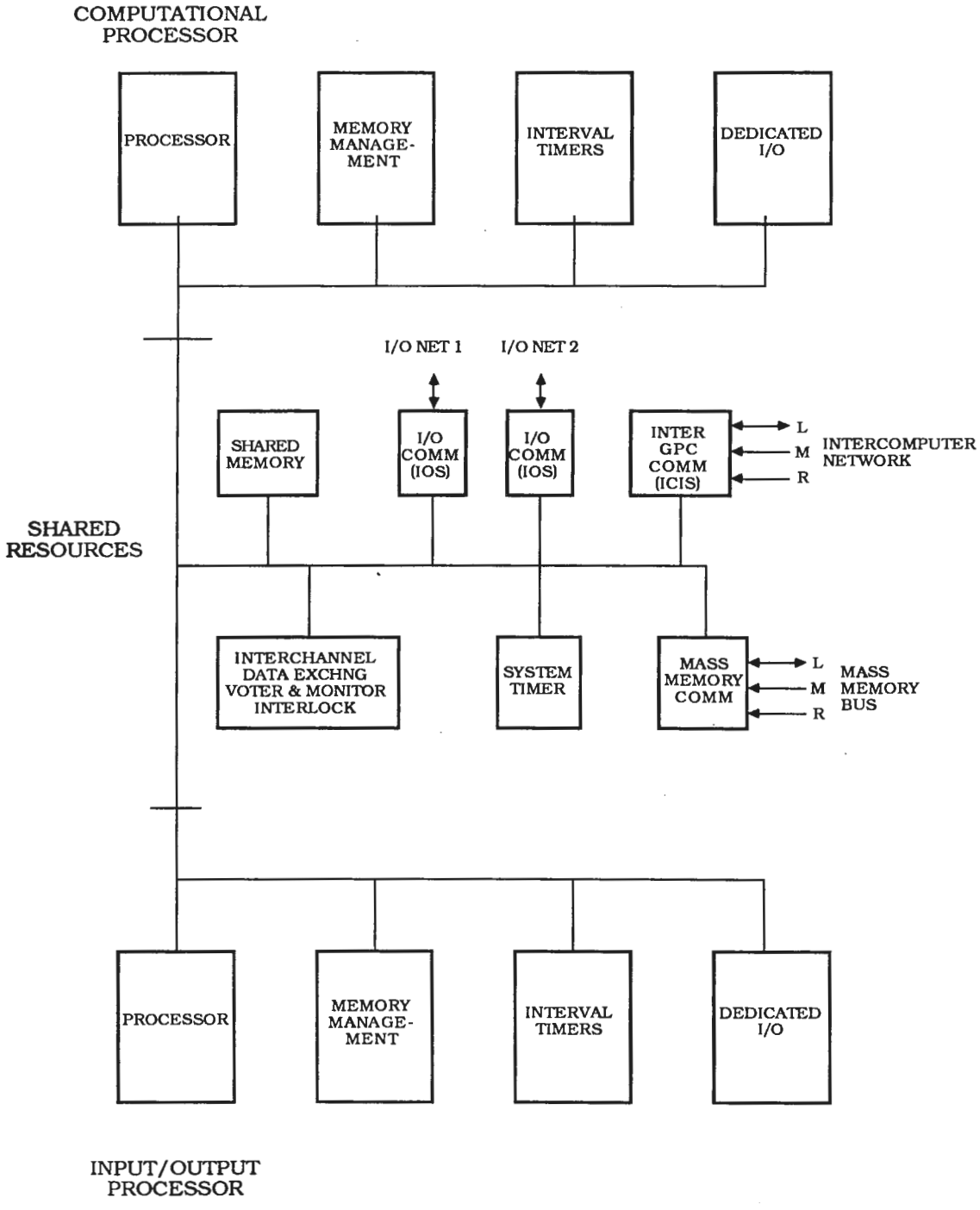


FIGURE 1.6-3. FAULT TOLERANT PROCESSOR - FUNCTIONS

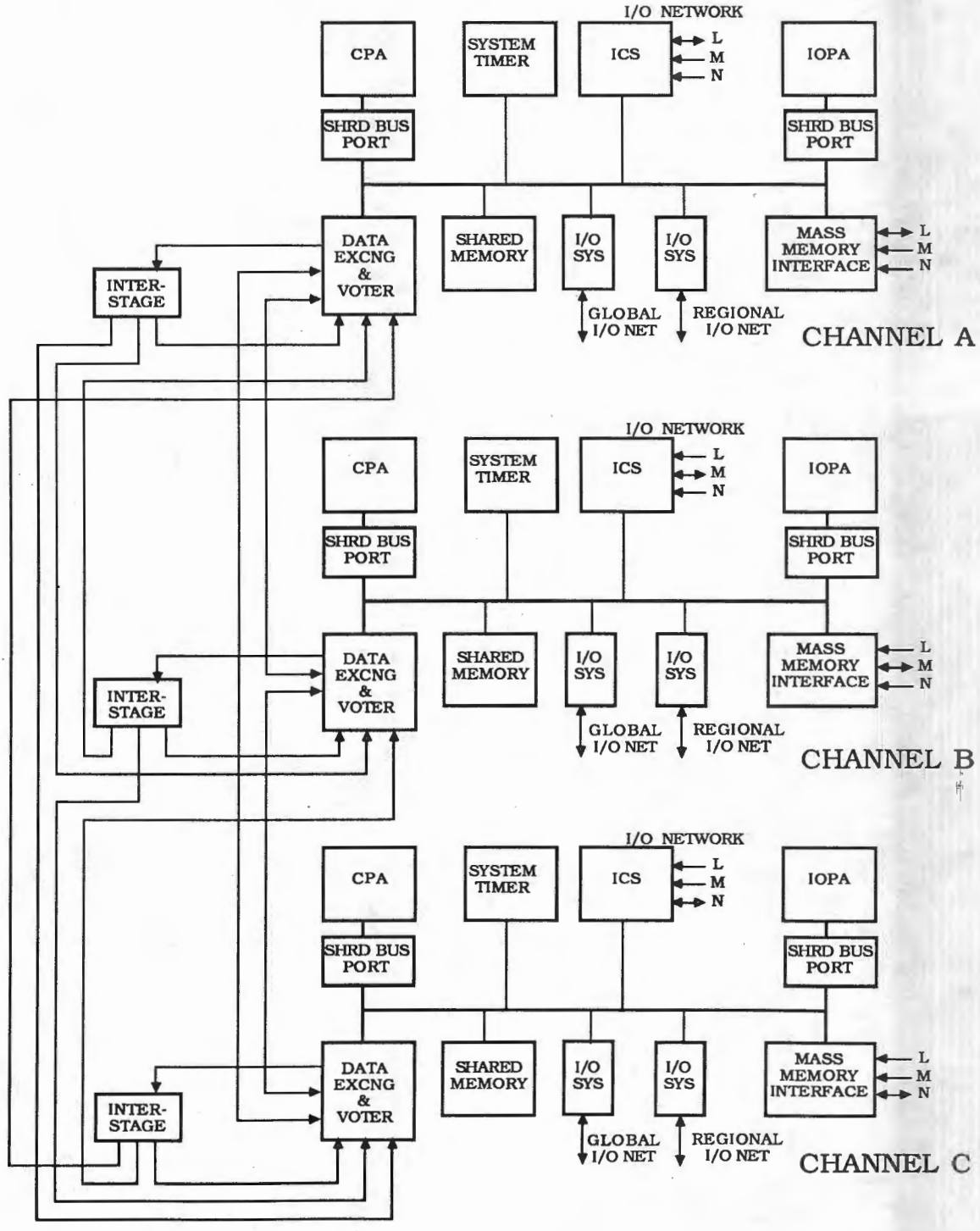


FIGURE 1.6-4. FAULT TOLERANT PROCESSOR - PHYSICAL VIEW

2. FAULT INJECTION SIMULATION AND TEST METHODS

Simulated faults are needed to demonstrate that an FT system response to faults is appropriate. Measurements are needed to demonstrate faulted behavior of the systems. These measurements also give added confidence that the predicted reliability, determined from coverage and component failure modeling, is correct. Furthermore, these tests validate that a system's performance under faulted conditions meets the need for safety and reliability of aircraft systems.

A typical reliability requirement for life critical operations is 10^{-9} per hour of operation. This translates into approximately 10 failures per million years per unit. This is about five orders of magnitude greater reliability than today's non-redundant systems. More importantly, for the purpose of demonstrating the reliability, life cycle testing will not be possible without simulation of the effects of faults.

Life cycle testing offers realism, but the rate of component failures is about one per 1,000 hours of operation per system. Fault insertion speeds up the rate at which synthetic faults occur. The use of inserted faults is needed to allow testing to cover all possible fault types and locations. For example, a small board consisting of 50 packages, each having 20 pins, has a possibility of 1,000 different pin-level (hardware) faults. This is without considering time dependence or sequence faults. Software faults must also be considered as the software becomes more complex. The number of potential software faults for a small operating system is also in the tens of thousands and time dependent as well.

Much work has been done on simulation and test methodology for insertion of faults in digital systems. (For details refer to McGough and Swern 1981; Benson, Mulcare, and Larsen 1987; Padilla 1988; Baker, Mangum, and Scheper 1988; and Migneault 1988.) A typical fault injection test setup involves the elements shown in figure 2.0-1. The faulted behavior is measured comparing faulted processor responses against those of an unfaulted processor. Measurement parameters include faults that are inserted and found, faults that are inserted and not found, and no-fault situations that are identified as faults.

A great deal of work is still needed in this area. The variation in test approaches ranges from entirely simulated software faults to injecting faults on real hardware operating under simulated flight conditions. The modeling accuracy, assumptions required, and interpretation of results also varies among the different methods. NASA, more than any agency, has developed the tools and techniques for fault insertion and testing to validate FT systems reliability.

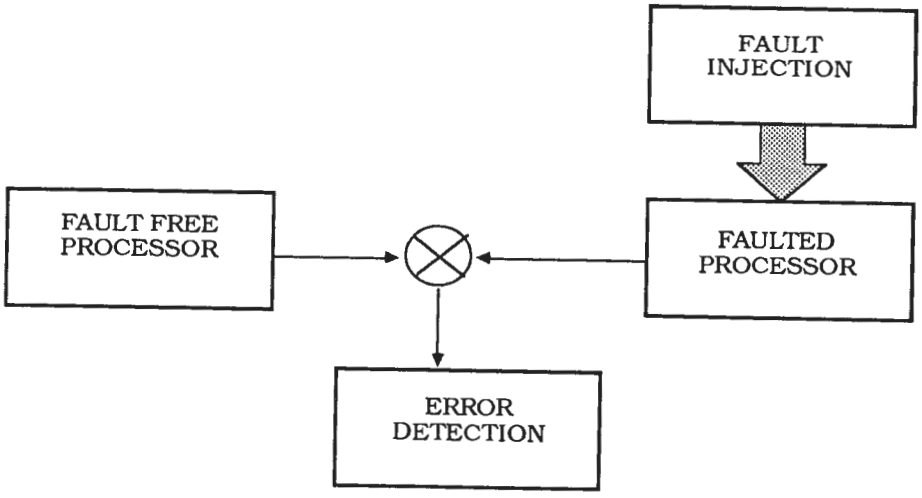


FIGURE 2.0-1. FAULTED PROCESSOR PERFORMANCE MEASUREMENT

2.1 NASA FT System Research Summary

Validation is the process of demonstrating that a system meets its specifications. For high-reliability systems, the specifications contain reliability requirements that necessitate operation under fault conditions. To demonstrate these extreme conditions, prediction methods must be used to determine nominal behavior of fault detection, isolation, and recovery schemes. Complementary to these methods are experimental methods, such as fault insertion, which are well suited to support modeling and analysis. Experimental methods are extremely useful when too many assumptions are needed in the analytical models or when analytical models have too much uncertainty.

Recent NASA research has emphasized fault injection methods for testing fault tolerant systems. The establishment of the Avionics Integration Research Laboratory (AIRLAB) at the NASA Langley Research Center provides a capability for government, industry, and research oriented institutions to investigate the faulted behavior of digital systems. NASA Ames is also involved in research in this area through efforts at the Reconfigurable Digital Flight Control System (RDFCS) Facility. Efforts recently completed at NASA Ames include development of the Fault Insertion and Instrumentation System (FIIS) to measure behavior of FCSs. (This work was sponsored by the Federal Aviation Administration (FAA) Technical Center.)

Several approaches have been developed for application throughout the system life cycle from design to deployment (Carter et al. 1989). Much work remains to be accomplished. Some of the methods are at different stages of development ranging from initial investigations to nearly final form (well understood and

ready for engineering application). A summary of methods and application areas is given in table 2.1-1.

TABLE 2.1-1. VALIDATION AREAS FOR THE DESIGN PROCESS

DEVELOPMENT LEVEL	ABSTRACT		CONCRETE
	DESIGN PROOFS	ANALYSES	TESTS
Architecture	Prove Architecture Against Requirements	Reliability and Error Rate Markov Models	Design Review High Level Simulations
Implementation	Prove Implementation Against Architecture	Fault Tree Analysis	Simulation and Emulation of Hardware
Realization	Prove Realization Against Implementation	Failure Modes and Effects Analysis	Support Assumptions from Analysis, Fault Insertion

2.1.1 Architectural Design Methods

In the requirements and specifications phase, the tools needed include simulation and modeling for the hardware and software. These tools provide data for studies and analysis including conceptual design, hardware/software trade studies, architectural design options, and "what if" parameter bounds.

2.1.2 Detail Design Methods

In the design phase, little hardware is available for testing. During this time the architectures of redundant processors, operating system software, and some applications are partially designed. Many portions are in process. The tools needed for this phase include emulation and simulation of the software and processor architecture.

2.1.3 Developmental Phase Methods

In the development phase, the software is better defined and available for testing. Hardware is becoming available in prototypes. Everything may change as this phase continues. Tools needed for this phase include in-depth emulation and simulation for software, and prototype evaluation tools for hardware items such as error detection and communications.

2.1.4 Systems Implementation and Evaluation Methods

In the implementation and evaluation phase the first items are completed. The tools needed include methods for inserting faults directly into the system hardware, and methods to exercise the special capabilities built into the hardware.

2.2 Fault Insertion Methods

Four methods for insertion of faults are in use. The advantages and disadvantages of each are shown in table 2.2-1. These four methods (Czeck, Siewiorek and Segall 1989) are based upon:

- Software simulation using fault insertion by code modifications or special functions of the simulation engine.
- Hardware emulation using hardware representative of the system under test, such as engineering prototypes, for study.
- Fault emulation imitating fault behavior through software control of the hardware or special capabilities built into the hardware.
- Physical fault insertion inducing faults through special hardware directly into the actual system under test.

TABLE 2.2-1. FAULT INSERTION METHODS

	FAULT INSERTION METHODS			
	SOFTWARE SIMULATION	HARDWARE EMULATION (BREADBOARD)	FAULT EMULATION	PHYSICAL FAULT INSERTION
Advantages	Access to system at any level of detail. Fault types and control are unlimited.	Representative hardware with favorable access and monitoring.	True hardware and software in use.	True hardware and software in use.
Disadvantages	Simulation time explosion. Lack of tools limit ability.	Implementation and other parameters will change with deployed system.	Fault types are limited.	VLSI limits access and monitoring points. Task is difficult.

These methods have been developed and used extensively for at least ten years. They are used for measuring fault coverage, error detection coverage, and other fault tolerance performance parameters. Measurement of these parameters requires special methods such as fault insertion.

Fault insertion has been used extensively for a number of purposes including test coverage, generation of fault dictionaries, study of error detection, and system evaluation. Table 2.2-2 summarizes these studies.

TABLE 2.2-2. FAULT INSERTION EXAMPLES

SOURCE	TARGET	METHOD/LEVEL	GOAL
Avizienis and Rennels '72	JPL Star breadboard	Permanent and Transient/Pin (Gate) Level	Estimate of Detection, Recovery Procedures, and Coverage
Goetz '72	ESS Microstore	Simulation/Gate Level	Detection and Coverage Measure
Courtois '79	MC 6800	Simulation/Opcode (RT) Level	Detection Time
Decouty Michel and Wagner '80	GORDINI: Fault Tolerant micro	Physical/Pin (Gate and RT) Level	Tool and Methodology Development
Kurlack and Chobot '81	GE MCP-701 CPU	Physical Faults with FMECA analysis	Evaluation of Watchdog Timer
McGough and Swern '81	Bendix Simplex BDX-930	Simulation/Gate and Pin Level	Coverage Measurement
Lala '83	FTMP Engineering Model	Physical/Pin (Gate) Level	System and Coverage Evaluation
Yang, York, & Siewiorek '85	iAPX 432	Fault Emulation/ Memory Words (RT)	Coverage Management
Schuette et al. '86	MC 68000	Transient, Physical/ Bus (RT) Level	Evaluation of Error Detection Techniques
Czeck et al. '87	FTMP Engineering Model	Fault Emulation/ RT Level	Methodology Study
Finelli '87	FTMP Engineering Model	Permanent Physical/ Gate Level	Fault Recovery Distributions

By far the two most common methods involve computer simulation of hardware faults and physical fault insertion. Fault simulation has been used at all levels for the fault models discussed in section 1.5. Physical fault insertion has been used to determine fault coverage of test programs, fault latency, and fault detection efficiency. Experimental agreement among the different techniques provides reasonable confirmation of the validity of these test methods. However, there is considerable difference in fault detection coverage between tests with models at the gate level and pin level of detail. The selection of injection points and the number of points tested also affects the measurements.

Since system fault detection is a critical parameter, NASA initiated a pilot study to investigate the methodology for measuring the fault latency of digital computers. The methodology consisted of simulating a 1,000 gate equivalent computer in a host Cyber 173 computer. The simulated computer was a paper design and was referred to as the hypothetical machine. This hypothetical machine was simulated at the gate level. Actually two copies of the hypothetical machine executed identical copies of code in synchronism although one machine received a fault at the onset of the simulation. Detection or non-detection was determined after the non-faulted processor completed its execution. At that time, the computational results of the two machines were compared bit-for-bit; any difference constituted detection. If no detection occurred, the code input variables were randomly altered, and the programs were repeated for the same fault. This scheme was repeated for up to eight executions for the same fault. If a detection occurred in eight repetitions or fewer, or no detection occurred after eight repetitions, then a new trial began. This over-all process was repeated for up to 211 faults randomly selected throughout the circuitry of the computer. The 211 faults were selected as a function of the piece-part failure rates and distributed across the nodes of the gates. The latency time (i.e., time to detect) is expressed in terms of code executions or repetitions.

Results of the pilot study were both surprising and disturbing. Using six different programs ranging from simple fetch and store to a very complex linear convergence scheme, the pilot study indicated that only 50 percent of the induced faults were detected after eight repetitions for each of the six programs. Figure 2.2-1 provides a typical picture of these results.

Studies conducted by McGough and Swern (1983) on the Bendix 930 FTP showed quite significant differences between gate-level and pin-level fault injection. Preliminary results obtained from gate-level fault simulation, using a hypothetical simple computer, showed that only 50 percent of the induced faults propagate to an output port where detection is possible. This pessimistic and surprising result prompted the design and implementation of several follow-on studies using realistic avionic computers.

The same set of experiments were applied to the minicomputer with the addition of realistic flight control code. The results were once again surprising: the percentage of undetected faults was about the same for all the programs simulated. Only about 50 percent of the injected faults were detected when different instruction sets, executing in two different machines (one hypothetical and one real), were used. More complex code tends to alter the shape of the

histogram, so that almost all the faults are detected in the first execution of the code. The latency time decreases somewhat with increased code complexity, but not the percentage of faults detected.

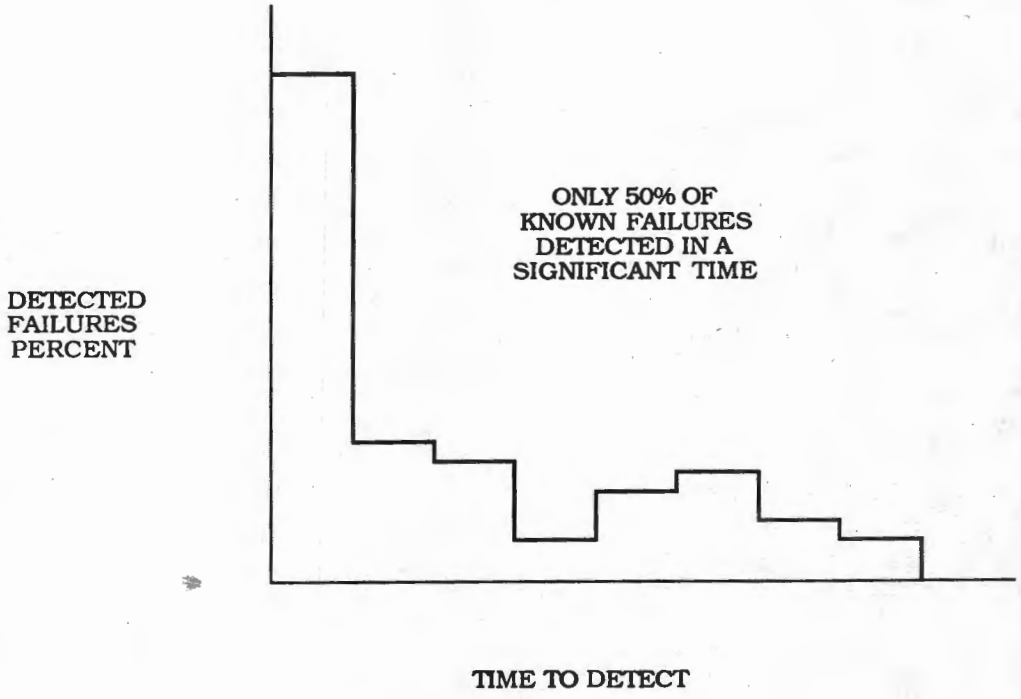


FIGURE 2.2-1. FAULT LATENCY DATA

More recently, experiments were conducted by the FAA Technical Center at NASA Ames (Benson, Mulcare, and Larsen 1987) on a Flight Control computer operating a simulated closed loop flight control. The computer configuration used was two CAP-6 processors in a dual configuration. This configuration is part of the dual-dual FCS used for flight control of the Lockheed L-1011 Tristar. An enhanced fault injection and data analysis system, based upon the CSDL Fault Tolerant Multiprocessor (FTMP) test setup, was used to control the experimental fault injection, and to collect and analyze the data. This computer was a much larger processor than that used for the Bendix study. The FCS program had 11,000 words, and the FCS computer had approximately 1,200 pins. Faults were injected on pins stuck at zero or one; approximately 2,500 faults were injected. The finding was that a much higher percentage of faults were detected; nearly 96.5 percent of all the injected faults were detected. These results were consistent with the earlier CSDL measurements on the FTMP. Additional discussion of these experiments is in section 2.6.3.

The following subsections discuss the four fault injection and test methods, and provide further examples and discussion of these issues.

2.3 Software Simulation Fault Insertion Methodology

2.3.1 Method

For this method of testing, the system digital logic circuits are simulated at the gate level-of-detail using high speed, specially designed, fault simulations. These simulations are like those for performance analysis, but stripped down to provide higher speed and specific outputs for fault detection and analysis. Logic levels are limited to zero or one. Little or no information is available for prediction of the system performance, such as speed and throughput.

Gate level-of-detail logic simulators were among the first tools for investigating the behavior of fault tolerant processors. Starting in 1977, NASA Langley initiated pilot studies on methodology for dealing with faults in digital systems. These efforts utilized a Cyber 173 to model a 1,000 gate equivalent processor. This was a paper design for a hypothetical machine. Following the pilot studies, a Gate Logic Software Simulator (GLOSS) was implemented to study the behavior of the Bendix 630 miniprocessor, a 5,000 gate machine. Subsequent to the Bendix study, it was realized that a more general capability could be useful for other processors. Several interim and improved versions of the GLOSS were developed: S-GLOSS by Stevens Institute of Technology; B-GLOSS by Bendix; and IGGLOSS, an improved version developed at Langley.

The current version being developed by Research Triangle Institute is called the Generalized Gate-Level Logic System Simulator (GGLOSS). In this version detailed gate-level simulations are run with faulted and unfaulted processors. The results are checked by comparison monitoring at critical outputs in the simulations.

Simulations may be run with faults injected at the gate level, pin level, or component level of detail. Since the entire processor is modeled at the gate level, faults may be injected at the opcode level. The gate faults are injected by holding the output node at zero or one. Pin-level faults are injected by holding the pin or terminal at zero or one.

2.3.2 Application

This method is most useful in early stages of development where no hardware is available, or for small enough computers so that a complete gate-level simulation may be processed.

2.3.3 Example Results

Researchers at NASA Langley began to study development of highly reliable prediction programs (B-GLOSS) for application to ultrareliable fault tolerant digital systems in the 1970s. One of the first efforts was modeling and analysis of the earlier Bendix 930 pilot study results. These new efforts were designed to test the validity of the pin-level injection result obtained from physical fault injection. It was desired to confirm that similar results could be obtained from a real processor executing practical software.

The pilot study results were tested in three phases using simulations of the Bendix 930 miniprocessor, a 5,000 gate equivalent computer. In the first phase, six application programs were coded using the primitive instruction set of a hypothetical machine: load, store, add, subtract, and branch. The next phase allowed the six programs to be coded using the rich instruction set of the Bendix 930. Finally, comparison monitoring detection was measured for an FCS code in lieu of the six programs.

The surprising outcome of these simulations is typified in figure 2.3-1. The results of all six programs follow the same trends. The percentage of undetected faults is about the same for all the programs, instruction sets, and two different machines: about 50 percent were undetected. For the more complex code, the shape of the histogram changes: most of the faults are detected in the first execution of the programs. The latency time decreases with increasing code complexity, but not the percentage detected.

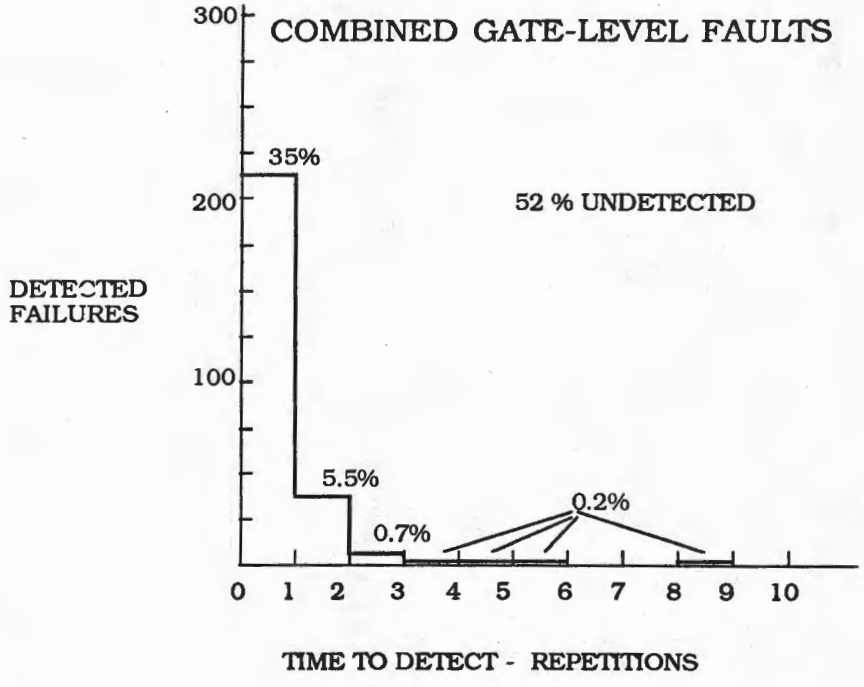


FIGURE 2.3-1. FAULT LATENCY DISTRIBUTION - GATE LEVEL FAULTS

The next set of experiments with pin-level fault injection produced more disturbing results. When the simulations were run with faults injected at pin level, but using the same programs and gate descriptions, the results shown in figure 2.3-2 were obtained. The notable difference is that the level of detection increases to about 62 percent of the injected faults.

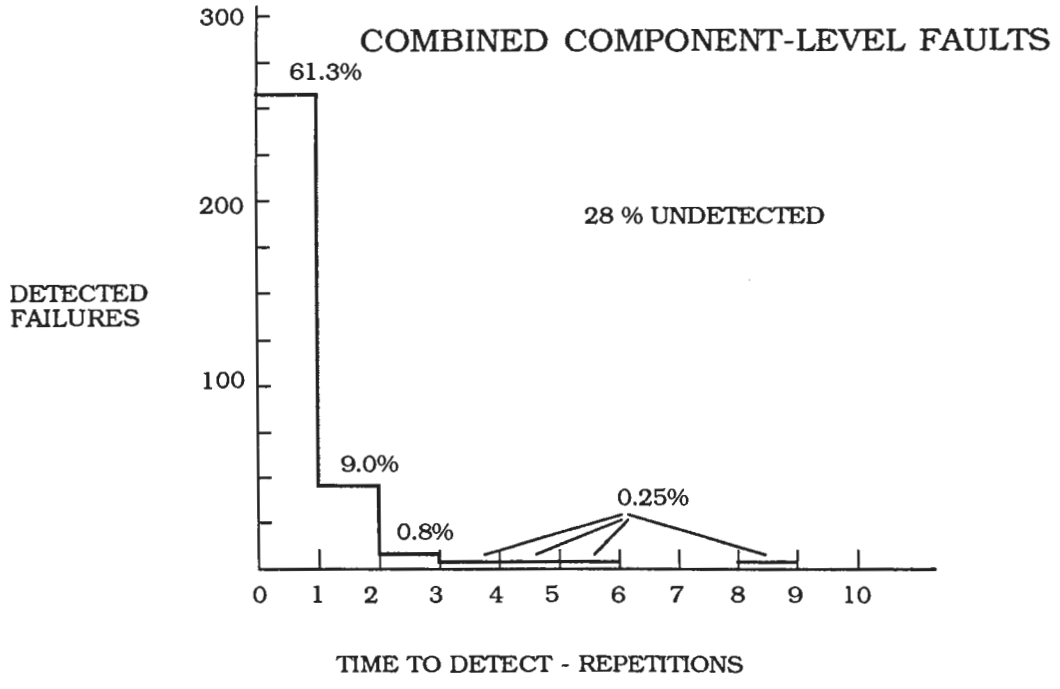
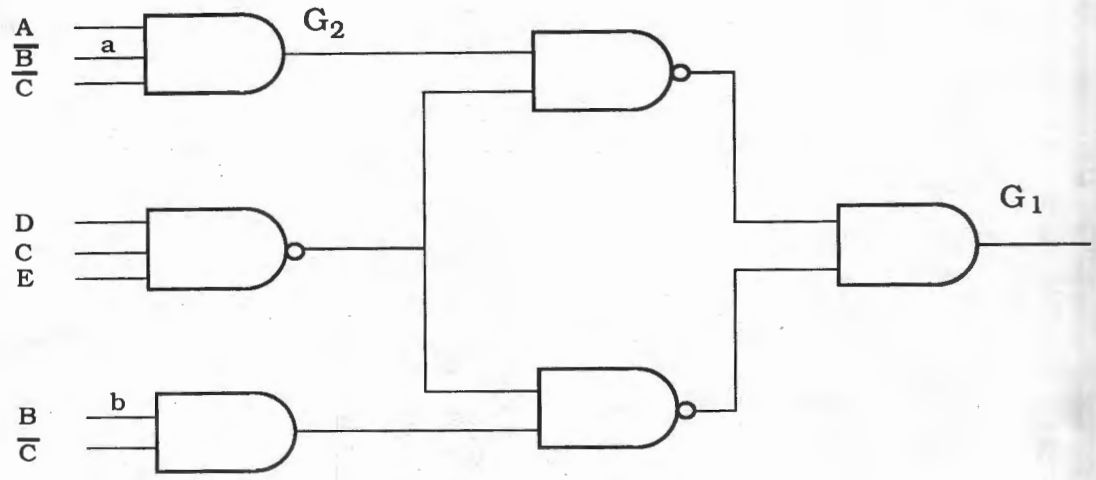


FIGURE 2.3-2. FAULT LATENCY DISTRIBUTION - COMPONENT PIN LEVEL FAULTS

Investigation of the difference between faults at the gate level and faults at the pin level determined that there is a class of faults that can never have an effect on the system and will never be detected. An example of these faults is a stuck-at fault located at a unused flip-flop output. The study found 16 percent of the faults were injected at such "don't care" locations.

An example of digital logic, shown in figure 2.3-3, may clarify the difference between pin and gate faults. The eight device terminal pins are A, $\overline{A}$, B, $\overline{B}$, C, $\overline{C}$, D, and E. Gate faults may be injected at any of the six gate output nodes, but only one is a pin. The "don't care" pin faults are line-a S-A-1. If line a is stuck at one, then line-b S-A-0 cannot be detected.

As an extension to the pilot project, the GLOSS simulation method was applied to a 2,000 instruction self-test program for the Bendix 930 processor. While the simulation exercised the program, faults were injected at the gate level and subsequently in another experiment at the pin level. The design goal of the self-test program was five percent coverage. For the experiments run with gate-level fault injection, the undetected faults were about eight percent of the detectable injected faults. With faults injected at the pin level, only two percent of the detectable injected faults were undetected. These data have been corrected for the 16 percent of "don't care" faults in this system.



THE SET OF TESTS $A\bar{B}CD$, $\overline{ABC}$, ABCDE, BCD, $AB\overline{CD}E$

CAN DETECT ANY DETECTABLE FAULT IN THE CIRCUIT G_1

NOTE THAT:
1. FAULTS ON PIN b ARE DETECTED BY $AB\bar{C}$
2. BECAUSE OF REDUNDANCY, THE FAULT PIN a: S-A-1 CANNOT BE DETECTED
3. IF THE FAULT PIN a: S-A-1 EXISTS, THEN FAULT PIN b: S-A-0 CANNOT BE DETECTED.

FIGURE 2.3-3. LOGIC DIAGRAM ILLUSTRATING DETECTABLE FAULTS

Work is still underway on the GGLOSS simulator and validation plans for it were recently developed (Bavuso and Miner 1989). GGLOSS has the capability to simulate S-A-1 or S-A-0 at any point in the circuit. Several innovative methods are used to eliminate the overhead in the usual event driven simulation. Sequential and combinatorial circuit elements can both be modeled. The combination of features gives GGLOSS the capability to simulate approximately 10^6 gate evaluations per MicroVAX II Central Processing Unit (CPU) second, while allowing users to monitor the effects of simulated faults.

2.3.4 Limitations

The types and numbers of faults are limited to logic levels in most implementations. Faults must be modeled as gate nodes or logic lines stuck at zero or stuck at one. Faults are inserted one at a time in an order determined by an experiment plan. Logic lines may be held for several cycles or inserted momentarily.

The selection of faulted lines or devices is often at random. This imposes a limitation on the measured coverage of possible faults. A great many "don't care" faults may be reported. Recent workers emphasize measurement of critical faults, such as detection and recovery times for reconfiguration processes (Padilla 1988, 1989).

2.4 Hardware Emulation Fault Insertion Methodology

2.4.1 Method

This approach uses special purpose digital hardware to emulate portions of the system under development. (Frequently the system under development is still on a breadboard.) The concept is similar to the use of emulators in the design and test of software using an emulation of a future microprocessor design. The instruction set of a new processor can be emulated in code using the instruction set of an existing microprocessor. In this way operating system software can be developed and tested while the new processor is still under development.

In the fault insertion emulation approach, algorithms for fault detection and recovery can be emulated by a special purpose computer consisting of hardware and software for emulating a wide variety of digital logical elements. The operation is based upon high-speed emulation of algorithms describing the digital logic at various levels of detail down to the gate level.

The technique and its concept, depicted in figure 2.4-1, are independent of any particular hardware or software implementation. Migneault (1988) contains a detailed discussion of this technique.

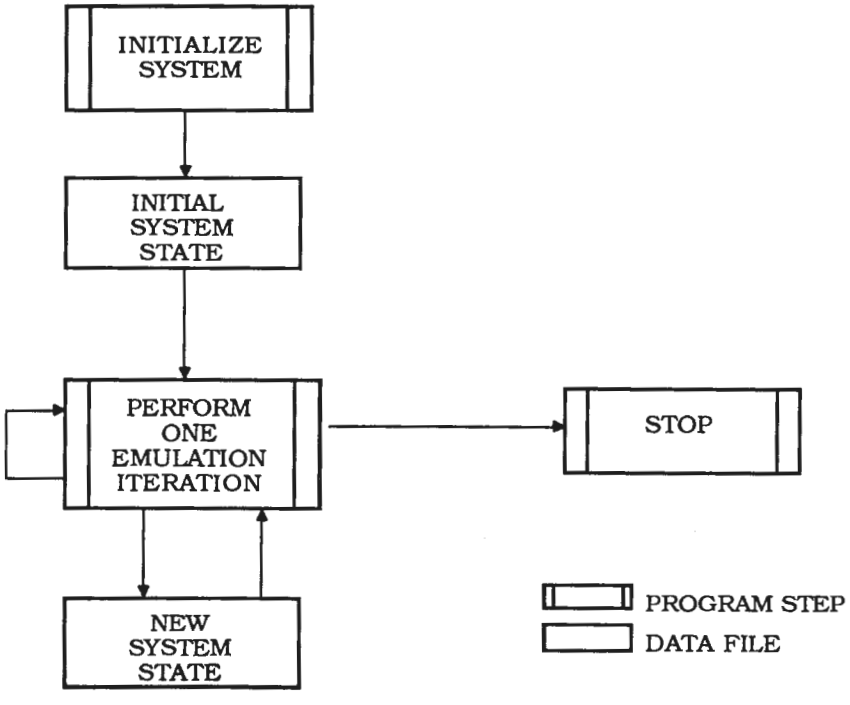


FIGURE 2.4-1. FAULT EMULATION COMPUTATION TECHNIQUE

2.4.2 Application

Fault tolerant digital systems are implemented by first setting reliability goals, and then designing the hardware and software systems as algorithms for fault detection and recovery. The tolerance to faults is usually in the form of redundancy components and algorithms which can detect faults and reconfigure the components in case of faults.

2.4.3 Example Results

This approach was developed at NASA Langley Research Center. In this approach (Bavuso and Miner 1989) an algorithm was developed to emulate any network of logic gates, flip-flops, and tristate devices. This algorithm is independent of any particular piece of hardware being emulated. A description of the target system is fed into a translator which converts the description to a form that the emulator can process. The processing of this representation of the target hardware by the software-implemented algorithm consists of the gate-level emulation of the target hardware. During emulation, faults can be injected and their effects studied.

Several unique features were incorporated in the Langley emulator system. Emulation speed is important because the target system must be run for many iterations of the fault insertion, detection, and recovery process. Memory space is necessary due to the large number of gates, flip-flops, and tristate devices in any modern digital system. The algorithm employs a general model for all types of gates (i.e., AND, OR, NAND, NOR, NOT, XOR), and a generalized model for all types of flip-flops. These general models allow for efficient use of computer memory. Time is conserved by processing, in a given cycle, those devices whose input has changed during the previous cycle.

This emulation system has a very general hybrid capability to model some parts of a system at the device level and other parts at the functional level. This capability allows the user to specify large portions of the system in a functional manner. At the same time, the user retains the capability to model other critical portions in detail, to the gate level, without the need to define every gate in the entire system.

In operation the emulator process is very straight forward. The state of the entire system at a given time consists of state descriptions of all logic devices in the network, state descriptions of all connections among devices, and a state description of the functional subsystem. The emulation must have the initial state of the entire system. Then for each time step T, the emulator calculates the output states for the next time T+1. The process continues to be driven by the series of events until the user-specified stop time is reached. The basic time step is the propagation time for input signals to propagate through a device to reach the output. For the initial emulator, the propagation time is taken as the same for all devices.

Fault insertion capabilities built into this implementation are as follows:

- The ability to insert or remove faults at user-specified times into the logic gates or ROMS.

- The ability to input to the digital logic at user-specified times from sources external to the emulation.

- The ability to output from emulated logic to external ports at times specified by the user or times driven by the system states.

As an initial experiment, a horizontally-microprogrammable computer, the Nanodata QM-1, was chosen as the host system. The emulation was coded at the microcode level to take advantage of the parallel capabilities of the host machine and to exploit the speed advantages of executing code at the most primitive level of the host computer. All processing of the hardware description and fault-injection data, as well as all post-processing of fault data, is performed on a Digital Equipment Corporation VAX 11 which is interfaced to the QM-1.

The emulation algorithm has been used to emulate a simplified model of the central processor of the Bendix BDX930, and the communications interface unit of the CSDL FTMP. Both of these system components have been the subject of other studies and comparative performance data is available to validate the emulation.

2.4.4 Limitations

The principal limitations of the hardware emulation technique are in speed and system modeling accuracy.

The speed of the emulation is dependent upon the size of the models and the speed of the emulation engine. The hybrid technique is important in reducing the size of the models. Speed is important both in shortening the run time and for emulation of real-time processes.

Modelling accuracy has some limitations due to the hybrid nature of the process. In this hybrid, system states are determined from the data structure and a subprogram module operating on that data structure and optionally on the gate-level network. This may not be serious, but care must be exercised in any applications to maintain consistency between the global and gate-level models.

The faults modeled are limited to pin faults, S-A-0 or S-A-1, or gate-level faults that can be defined by effects on the logic algorithms of the system under test. Conceptually this allows for a wide range of faults, but the effects of faults may be very difficult to model accurately and input to the emulation.

Propagation delay times are taken as the same for all devices. This is unrealistic and future versions of the emulator should address this parameter allowing the user to specify the delay time for each device in the gate model library.

2.5 Fault Emulation Fault Insertion Methodology

2.5.1 Method

The fault emulation technique inserts software faults, or the effects of faults, into the real system software. A special purpose processor is used to control fault insertion and monitor the system under test. Faults, or the errors manifested by faults, are inserted by triggering the error detection mechanisms or by seeding errors into memory.

Faults are injected into software controlling the system hardware or into the special capabilities built into the hardware such as communications registers, comparison-monitoring detector, configuration status registers, and sensor output registers. Possible software faults include lost or delayed messages, corrupted messages, delayed tasks, abnormal task termination, timer corruption, and system clock corruption. Injection of faults directly into system software is accomplished by transfers over the data bus of the system under test. The process is controlled by timing and monitoring of data and status information received from the data bus. Instrumentation of the effects of testing is also accomplished using the data and status information on the system data bus.

Fault emulation fault insertion offers the following features:

- Allows insertion directly into the substantial software portion of a system. These software errors can be used to examine the software and the interactions of the hardware and software.
- Is often simpler, in terms of time and effort, than hardware fault insertion.
- Is complementary in terms of functionality to hardware fault insertion and does not exclude hardware fault insertion.
- Provides a direct means for testing software implemented fault tolerant detection and recovery strategies.

Data collection and analysis is also implemented by using information on the system or processor bus. Historical data records of normal events and functions may also be collected. Then, error reports can be generated indicating exceptions and abnormal events, allowing measurement of faulted performance. Data analysis supports developing data on runs, experiments, and multiple experiments.

2.5.2 Application

Software fault insertion, as described in this subsection, is most useful during the system development and validation. Faults may be inserted into developmental software modules such as the communications processor. The equipment utilized is essentially a general purpose test and evaluation instrument. This concept should have application during the operation and maintenance phase as well.

2.5.3 Example Results

As an example we will discuss the implementation of the Fault Injection Automated Testing (FIAT) environment, illustrated in figure 2.5-1. FIAT was developed by Carnegie Mellon University under a NASA Langley grant. This system is described in Czeck, Siewiorek, and Segall 1989. The FIAT hardware consists of four International Business Machines (IBM) personal computer (PC) remote terminals (RTs) connected via a 10 Megabit token ring. This hardware provides the operating environment for FIAT, without limiting the generality of the system. The software structure consists of two modules: (1) the Fault Injection Manager (FIM) for experiment control and data analysis, and (2) the Fault Injection Receptor (FIRE) for data collection and processing.

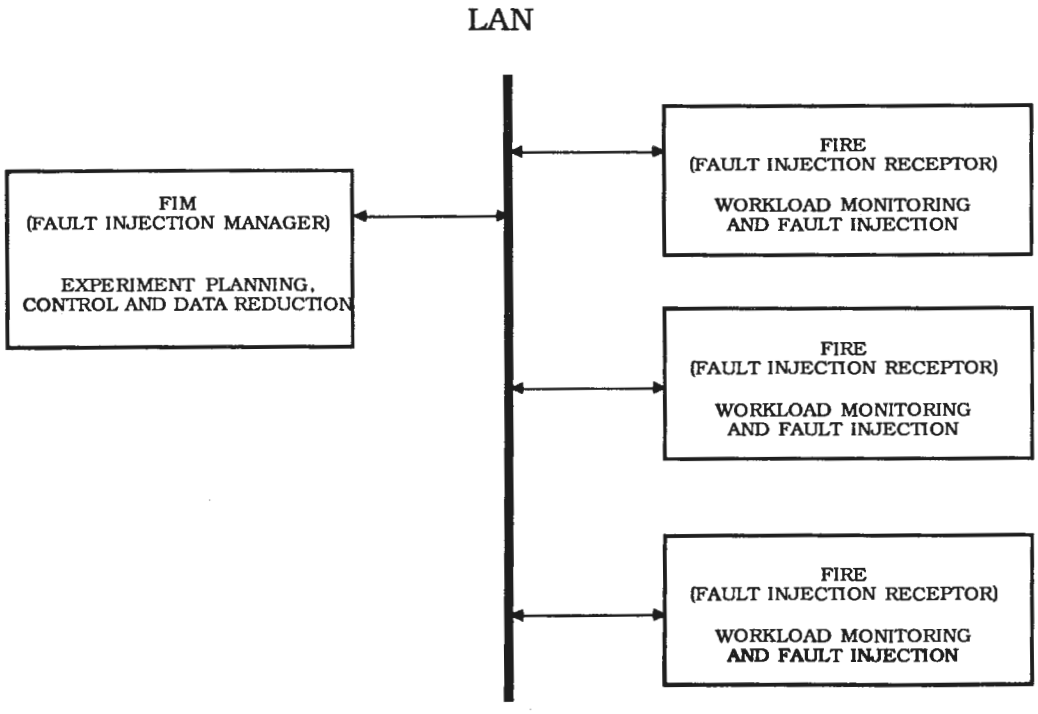


FIGURE 2.5-1. FIAT FAULT INSERTION AND EXPERIMENT MONITOR

The FIM software, shown in figure 2.5-2, processes users' programs and experiment definition to control the injection of faults and collection of data from the system under test. The experiment interface, illustrated in figure 2.5-3, has four modes of processing:

Library Preparation - Assists in preparation of task development, image generation, and attribute extraction. The fault class library groups domains and methods for selection of the fault insertion method during experiment definition.

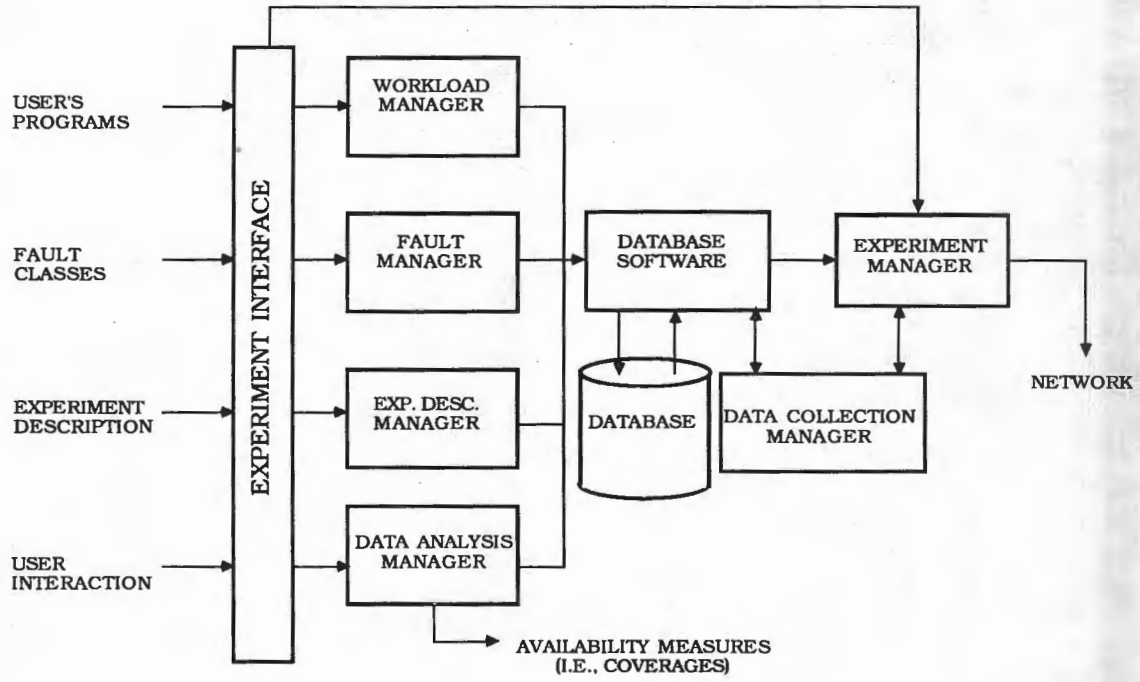


FIGURE 2.5-2. FAULT INSERTION MONITOR SOFTWARE

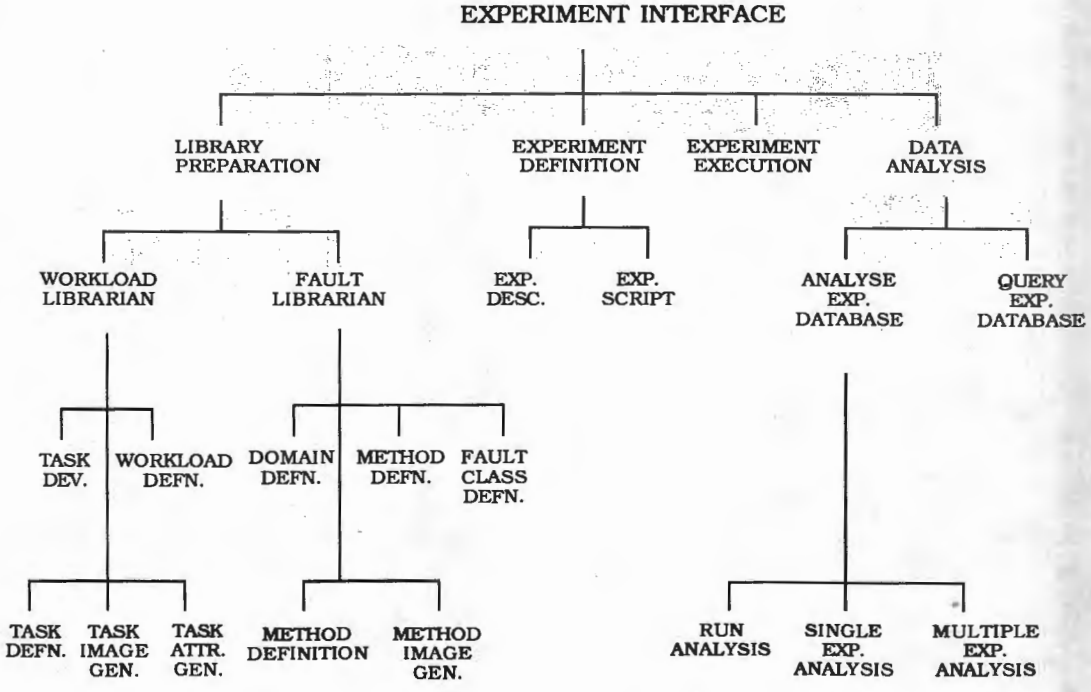


FIGURE 2.5-3. FIAT EXPERIMENT INTERFACE TREE

- Experiment Definition - Assists in developing an experiment description and in script generation.

- Experiment Execution - Utilizes the prepared scripts and executes the sequence of fault injections.

- Data Analysis - Utilizes the prepared experiment definition and provides two types of analysis: (1) user defined queries using Informex Software Query Language (SQL), and (2) generic FIAT data analysis routines.

The data collection and monitoring operate via the FIRE modules shown in figure 2.5-4. The function of each module is as follows:

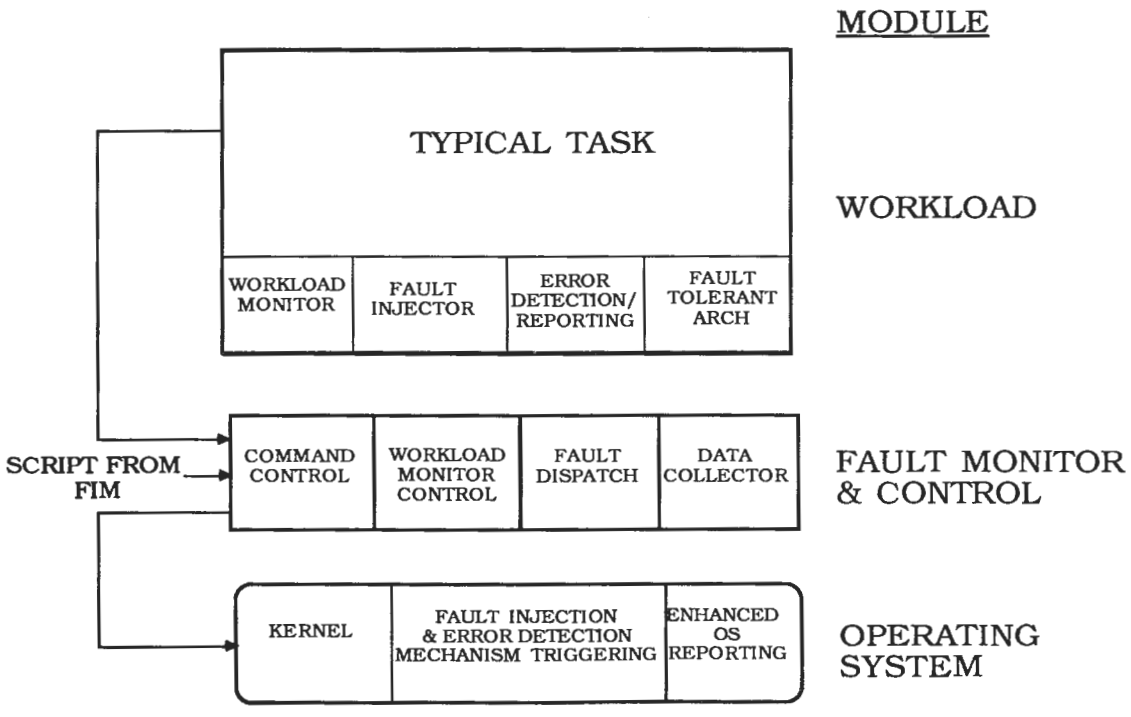


FIGURE 2.5-4. DETAILED VIEW OF FAULT INJECTION RECEPTOR SOFTWARE

- Command Controller - Stores and organizes commands from the experiment manager in a command queue and synchronizes the commands, such as task starting and fault insertion timing within the work load control flow.

- Workload Monitor Controller - Provides the functions of monitoring and control of the workload, task control, communications, and synchronization.

- Fault Dispatcher - Manage fault insertion of workload tasks, communications, and the operating system. Operating system faults include memory

and register faults and triggering of existing hardware or software error detection mechanisms.

- Data Collection - Transfers records generated by the monitor and error report scripted commands to a local experiment database storage.

As an example of the use of FIAT, a distributed checkpointing system was implemented using two FIRE machines. The block diagram, figure 2.5-5, consists of a two-computer system. The primary computer informs the secondary computer of task initiation and the time frame for the next task initiation. The primary machine executes the task and the secondary machine waits for the next interaction. If the time between interactions exceeds the time frame for completion (indicating primary failure), the secondary then initiates a recovery process and becomes the primary. If the primary detects that no secondary exists (indicating secondary failure), it creates a new secondary.

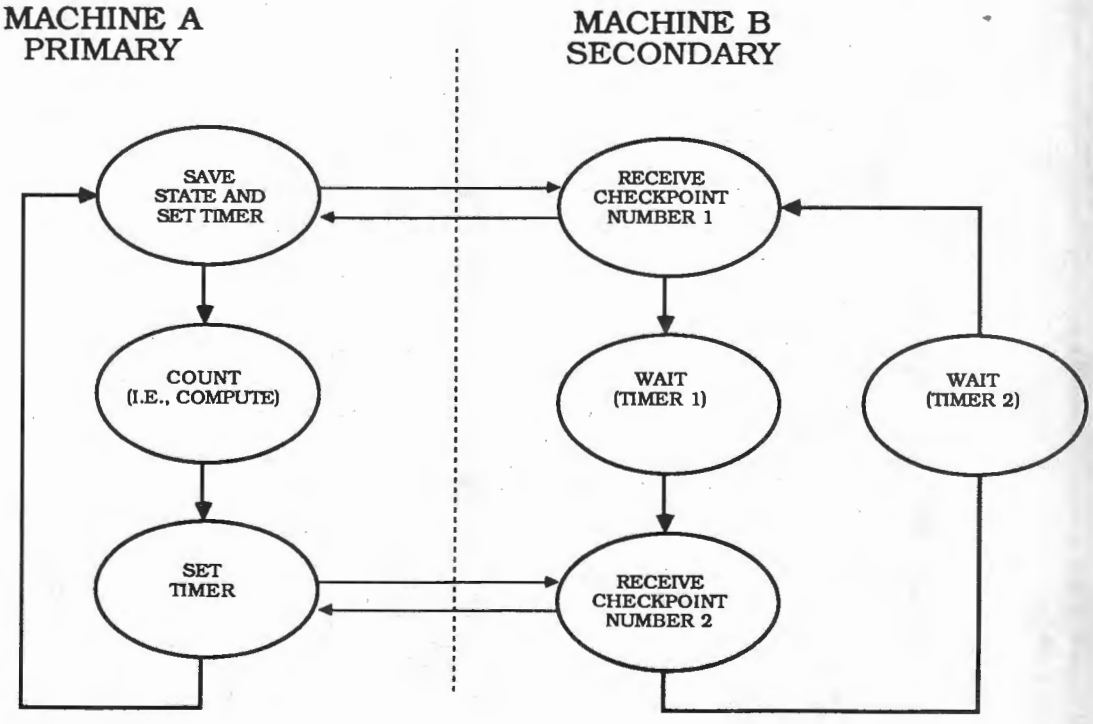


FIGURE 2.5-5. REAL-TIME CHECKPOINTING SYSTEM

Experimental results of inserting faults into the checkpointing system are shown in table 2.5-1. The faults considered were bus faults having double bit compensating errors. These double errors would not be detected by hardware parity and must be detected in software. Note the large latency time and that one-quarter of the inserted faults were not detected. To illustrate how fault tolerance strategies may be compared, the checkpoint system was improved by

TABLE 2.5-1. ERROR DETECTION STATISTICS - CHECKPOINTING ONLY

```
METHOD:  DETECTION STATISTICS

Experiment = 1a
Detection Statistics:

Number of Fault Injections ... 206
Number of Fault Detection  ... 153
Detection Coverage = 74.271845%

Average Detection Time = 4.219567 Seconds
Minimum Detection Time = 1.406250 Seconds
Maximum Detection Time = 6.218750 Seconds
```

TABLE 2.5-2. ERROR DETECTION STATISTICS - CHECKPOINTING AND CHECKSUMS

```
METHOD:  DETECTION STATISTICS

Experiment = 1b
Detection Statistics:

Number of Fault Injections ... 91
Number of Fault Detection  ... 86
Detection Coverage = 94.505495%

Average Detection Time = 3.102834 Seconds
Minimum Detection Time = 1.203125 Seconds
Maximum Detection Time = 4.203125 Seconds

END OF METHOD:  Detection Statistics
```

adding double checksum comparison to the primary and secondary machines. A checksum is appended to code and data blocks during compilation. This code is regenerated when reading blocks from memory during run-time and is compared with the code determined at compilation to check for errors. Hence the detection methods are timeouts for the first experiment and checksums for the second. Improvements in the results may be seen from table 2.5-2. Note the increased fault detection probability and the decreased detection time.

2.5.4 Limitations

Fault emulation fault insertion has limitations primarily in its inability to force low level errors, such as gate output S-A-0 or S-A-1. These limitations can often be worked around by injecting faults or errors at another location. Since designers are interested in the behavior of the entire system this may not be a serious limitation.

2.6 Physical Fault Insertion Methodology

2.6.1 Method

In this technique faults are actually inserted into real hardware by computer controlled switches holding the voltage levels high or low on individual wires at connector pins of selected devices. Because only the device terminals are accessible, these faults are usually pin-level S-A-0 or S-A-1 faults. Some testing has been performed using inverted levels (i.e., change a zero level to one and conversely). The existence of faults is detected by the real hardware comparator monitoring systems. Measurements include the fault insertion time and detection time. Statistics are collected on the number of faults inserted, detected, detected in error, and not detected.

These experiments can be run at the gate, pin, or resistor or other electronic component. Since real system hardware is used for testing, the common method of fault injection is at the pin level of logic circuits including gates; flip-flops; memory clips; processors; and Small Scale Integration (SSI), Large Scale Integration (LSI), and VLSI components.

2.6.2 Application

Physical fault insertion is used for validation of completed hardware/software systems. This method is a validation tool. Modeling and simulations will have been used prior to final implementation. Those tools will have provided predictions and reliability estimates. A final level of testing using faults injected on real hardware provides the realism, and minimizes the assumptions and modeling errors that may exist in the other techniques.

This technique is most useful during full scale development. Some applications of this technique have been made using development hardware of portions of the system, such as the communications interfaces between multiprocessors.

2.6.3 Example Results

Physical fault injection has been used for several systems including the Bendix 930, the CSDL FTMP, and more recently on a dual-dual FCS for the Lockheed L-1011 Tristar. This system used four Rockwell/Collins CAPS-6 digital computers. Figure 2.6-1 illustrates the experimental setup for data acquisition experiments (Benson, Mulcare, and Larsen 1987). The additions from prior experiments include an external clock and an associated DR11C interrupt so that fault detection is quickly reported to the PDP-11 through the DR11C interrupt. Considerable software modifications were made to analyze, reduce, and present the data for reports. Prior efforts injected faults on the pins of the CSDL FTMP (references 4, 6, and 7, from Benson, Mulcare, and Larsen 1987) system which also uses the CAP-6 processors. Investigations into this methodology are in progress. Figure 2.6-2 shows the setup for data acquisition on this current effort on the CSDL FTMP at NASA Langley.

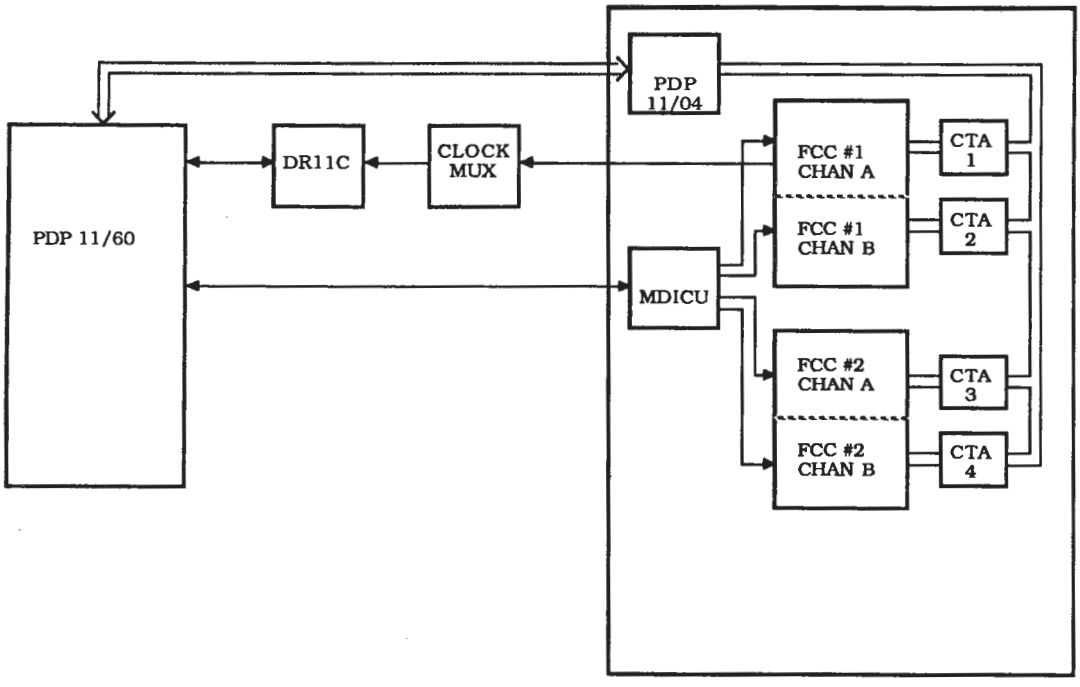


FIGURE 2.6-1. FAULT INSERTION AND INSTRUMENTATION SYSTEM (FIIS)

Both of these systems share common features in fault insertion circuitry (FI). The FI is connected to the UNIBUS in a manner similar to the Collins Test Adaptor (CTA) shown on figure 2.6-1.

The FI provides an interface between the UNIBUS and the system components under test. The fault injection is accomplished in the FI by signals sent to Field Effect Transistors (FETs) inserted between the socket and device, as illustrated in figure 2.6-3. These FETs also allow monitoring of the signals at device

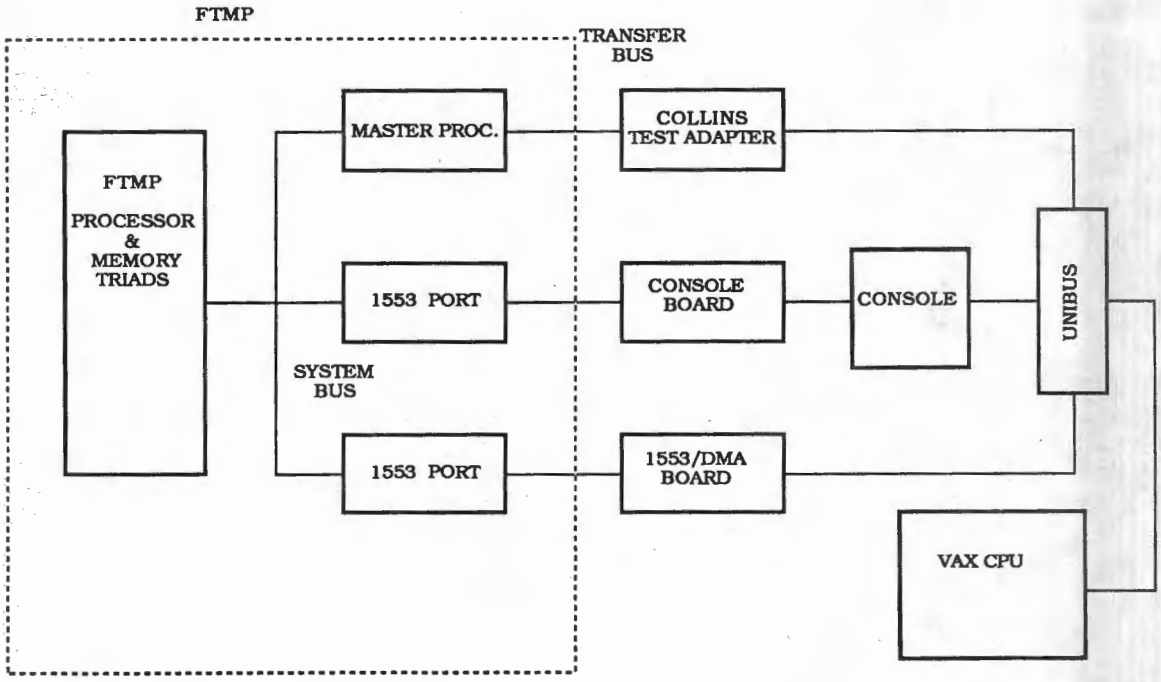


FIGURE 2.6-2. FTMP VAX DATA ACQUISITION ENVIRONMENT

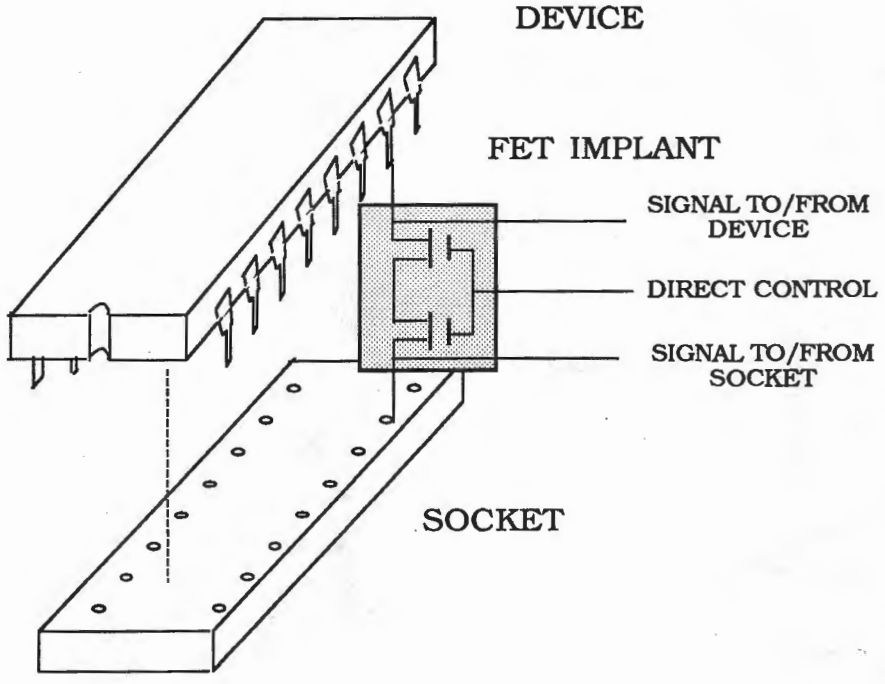


FIGURE 2.6-3. DEVICE PIN ACCESS FOR FAULT INJECTION AND MONITORING

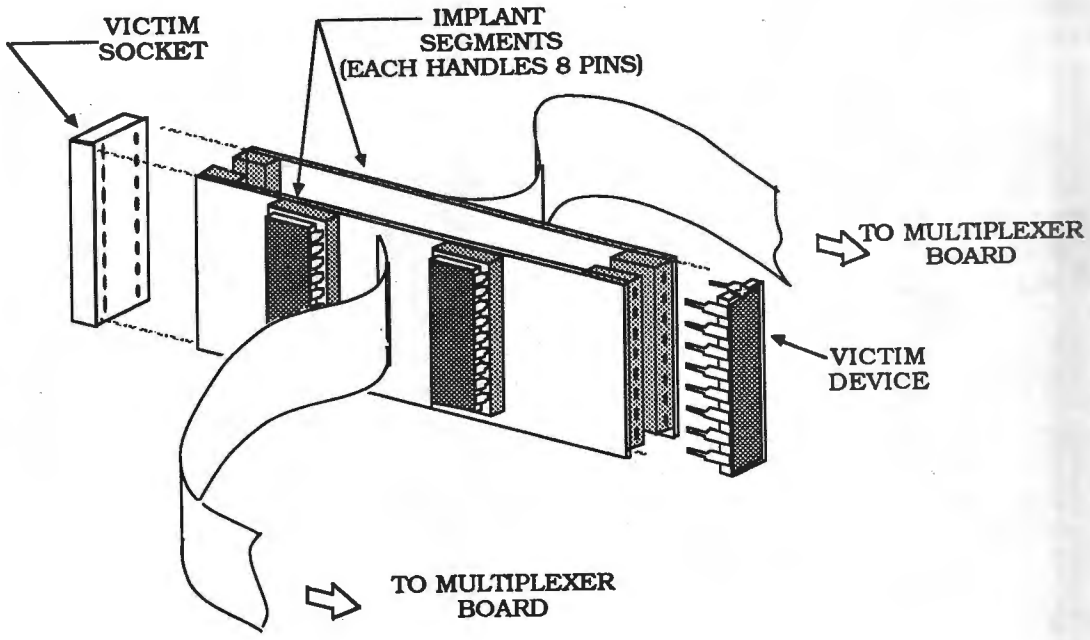
terminals. Turning on the FET establishes a direct connection between the device and the socket. This is the normal condition when no faults are injected. The connection can be severed by turning the FET off. In the off state, any desired signal can be applied to the device or the socket pin. Usually the pin selected is an input. It should be noted that an output pin is usually connected to an input pin of some other device. Added flexibility is provided by multiplexing signals to and from the package and socket under control by the VAX or PDP-11. A choice of eight possible injection signals is provided to the experiment via the multiplex system:

- Direct connection to the socket
- Multiplex A
- Multiplex B
- Multiplex C
- F(AB) Boolean combination of signals from Mux A and B
- F(A,B,C) Boolean function F(AB)and output of Mux C
- Ground, logic zero
- External signal

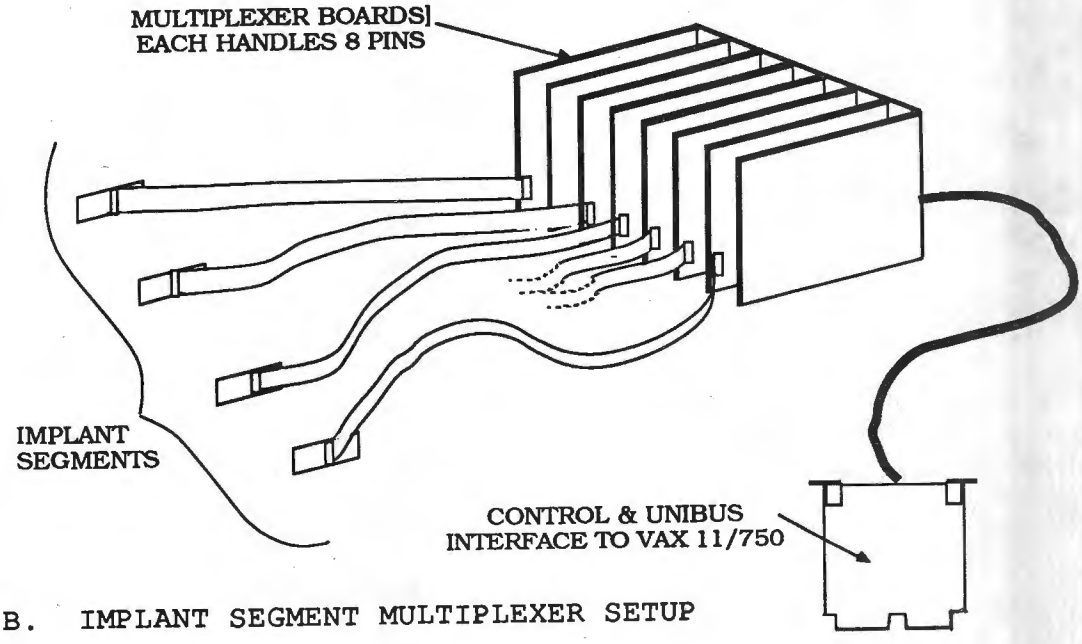
Each of these eight signals may be inverted before application to the victim pin, thus providing sixteen possible sources. The usual experimental run uses S-A-1, S-A-0, and "complemented signal" types of faults. This setup has powerful flexibility. For example, certain faults in integrated circuits can change a NAND gate into a NOR gate. It is possible with the fault injector to simulate such a fault by extending the pins of the target gate with the FETs and generating the Boolean functions. The main utility of this injector is to inject faults on tristate lines. The direction of the fault can be made a function of other signals on the device, signals that determine the state of the tristate control pin. Thus, it is possible to inject faults into the data pins of memory chips, bus drivers, and other tristate devices.

The injector hardware is packaged as shown in figure 2.6-4, with FET pairs on an extender segment. These extenders are used to provide space for the multiplex signals and interface to the sockets. Each multiplexer board accommodates eight pins. The six boards are connected to the instrumentation processor via a unibus card into the VAX or PDP-11.

The recent experiments using the FIIS at NASA Ames provided information on the behavior of a completed FCS operating in a simulated aircraft environment. The CAP-6 processors used in this system had approximately 1,200 pins. For this system a nearly complete coverage of single faults was injected on all possible pins. These results are an important addition to the information obtained from the earlier work on the Bendix 930 and on the FTMP. In these experiments over 2,700 simulated faults were applied to the chip pin level. Most testing was done open loop. The more persistent faults were subjected to closed loop



A. DEVICE TEST ADAPTER



B. IMPLANT SEGMENT MULTIPLEXER SETUP

FIGURE 2.6-4. FAULT INJECTOR HARDWARE

testing in order to involve the built-in fault detection mechanisms. The test results, summarized in table 2.6-1, are in reasonable accord with similar data, confirm well with the fault models, and indicate the value of this type of testing for practical system validation efforts.

The data shown in table 2.6-1 are the result of injecting 2549 "care" faults. While running the baseline FCS program under open loop control conditions, 2461 were detected and 88 were undetected. Open loop tests were conducted to speed up the process because the closed loop system provided a layer of error detection and recovery. Of the 88 undetected faults, 15 were invert faults. Of the invert faults, 1.71 percent were detected while 4.37 percent of stuck-at faults were undetected, indicating that invert faults are less latent than stuck-at faults. Each of the 88 undetected faults were injected while executing the self-test program. All these faults were detected indicating that the processors "care" about these cases. All the micromemory and processor faults injected were detected. The 100 percent coverage of the self-test program indicates the value of running self-test in the background, if possible, every few minutes even though no errors are detected by the system in its present mode.

TABLE 2.6-1. EXPERIMENTAL RESULTS SUMMARY FROM FIIS

DEVICE	# FAULTS INJECTED	# FAULTS DETECTED	# FAULTS UNDETECTED	# DON'T CARES	PERCENT DETECTED	PERCENT UNDETECTED
Data Path	1535	1502	33	83	97.85	2.15
Control	1014	959	55	83	94.58	5.42
S-a-0/S-a-1	1670***	1597	73	123	95.63	4.37
Invert	879	864	15	43	98.29	1.71
2901	423	423	0	25	100.00	0.00
Micromemory	231	231	0	9	100.00	0.00
Input Pins	1805	1741	64	87	96.45	3.55
Output Pins	744	720	24	79	96.77	3.23
Self-Test	1687	1687	0	0	100.00	0.00
Totals	2549	2461**	88*	166	96.55**	3.45**

NOTE: (1) Injected faults do not include "don't cares"
(2) Percent "don't cares" = 6.11
(3) * = Detected by self-test
(4) ** = Excludes self-test
(5) *** = 791 S-a-0, 806 S-a-1

Investigations are under way at NASA Langley, using the advanced data acquisition environment shown in figure 2.6-2, to investigate the FTMP error detection and faulty unit isolation functions (Padilla 1989). Earlier tests disclosed that the FTMP occasionally disabled the wrong units. This behavior continued

until the system crashed after 4,000 to 6,000 faults were injected. These investigations uncovered reasons for the failures due to the fault identification procedure incorrectly identifying that only one Line Replaceable Unit (LRU) detected the error. This event triggers a software component, designed to handle Byzantine faults, which keeps track of the accuser and disables it after a second event. In this case one fault causes two LRUs to be identified as in error, one bad unit and one good unit.

As a result of these investigations, a new methodology for selection of the pins for fault injection is proposed (Padilla 1990). To investigate the abnormal behavior of FT systems, tests should be based on two principles:

- Faults should be injected in signals having high fan-outs (board enables and other control signals), thus maximizing the amount of interaction (number of potential errors).

- Information to determine the system state should be observable at all times in real time.

These principles, although simple in concept, require a completely new experimental environment. The data acquisition environment overburdened the original injection and test system. The new data acquisition system, figure 2.6-2, has three orders of magnitude increase in data rates over the old system.

Some of the initial experiments using the new system disclosed that LRUs not being faulted were being disabled by the fault management software. Although the system classified the events as caused by real permanent faults that occurred during the experiments, no faults could be found at the conclusion. All the supposed faulty units were reactivated either manually or by rebooting the system.

2.6.4 Limitations

Faults used in testing are often limited in type and number. Since real hardware is used for the system under test, faults may only be injected at connector pins. Internal gate nodes may not be tested.

The hardware and software must be available or sufficiently well defined that prototypes may be made.

The methodology for selection of pins for fault injection does not have universal acceptance. Systems architecture and design assumptions may limit the faults tested to those locations where designs have addressed error detection and correction. On the contrary, pins may be selected at random, perhaps testing unused gates or overlooking important vulnerabilities.

Physical FI has been used extensively in system validation, with faults inserted at the pin level as S-A-1 and inverted faults. With systems implemented in an SSI or medium scale integration (MSI), pin-level stuck-at faults closely resemble failures which have been observed to occur in such devices. But with LSI and VLSI realizations, failures may be remote from the input/output pins. At these higher levels of integration, FI seldom claims to portray physical

faults accurately. However, the hope is that they prove a first approximation to the metrics under study. The data is not yet available to verify this assertion. Additional information is presented with detailed examples in section 3.

Although faults may be remote from the boundaries, promising results have been reported with pin-level FI for LSI and VLSI devices. Schuette et al. (1986) inserted transient faults of the data, address, and control lines of an MC68000 bus, representing faults within data and control sections of the processor. With this FI technique, two error detection schemes were evaluated. Czeck et al. (1987) inserted faults in an FTMP triad by causing one unit to execute a special code, causing a trigger in the error detection mechanism. This method was able to duplicate some of the results presented by Lala and Smith (1983). Even so, McGough and Swern 1983 illustrated a gap between gate and pin-level fault injection.

3. ADDITIONAL FAULT INSERTION AND TEST EXAMPLES

3.1 Draper Labs FTMP Tests Using Fault Emulation

This section describes the results of a simulation test using fault insertion. This experiment was conducted in the NASA Langley AIRLAB Diagnostic Emulation (DE) facilities, and was described fully in Baker, Mangum, and Scheper (1988).

The primary objective of this experiment was to determine the diagnostic self-test sequences used to uncover latent faults in a logic network providing the key fault tolerance features for a flight control computer. In this experiment, using the setup shown in figure 3.1-1, more than 1,600 faults were injected into a logic gate-level emulation of the Data Communicator Interstage (C/I), a key component of the CSDL FTMP. For each fault injected, diagnostic sequences consisting of over 300 test vectors were supplied to the C/I model as inputs. For each vector in a test sequence, outputs from the C/I were compared with the outputs of a fault free C/I. If the outputs differed, a fault was considered to be detectable for the test vector. These results were analyzed to determine the effectiveness of the various test sequences, to identify latent faults, and to identify ways for improving performance of the diagnostic sequences.

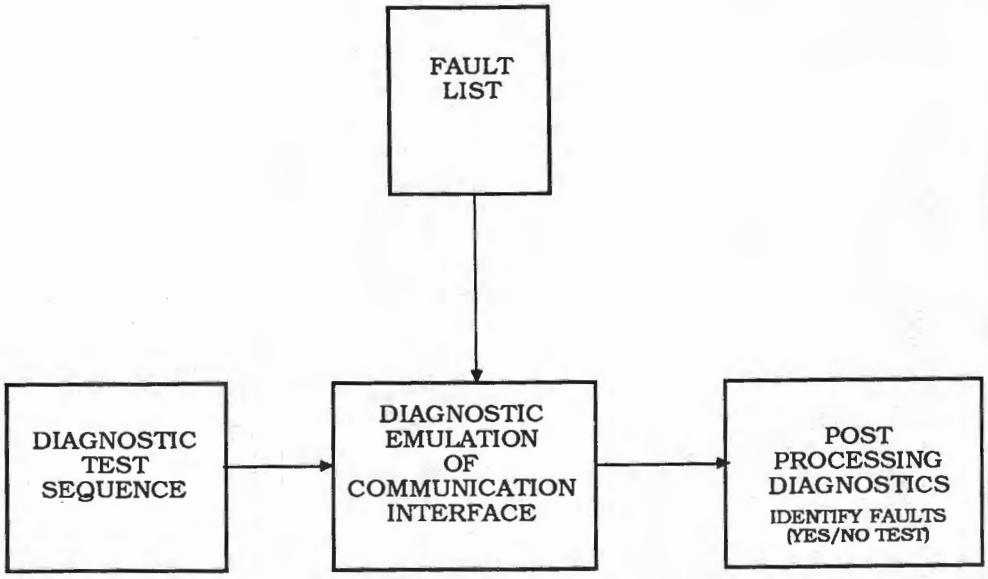


FIGURE 3.1-1. EXPERIMENT DIAGRAM

The C/I of the FTMP was selected because of the following reasons:

- The C/I network provides the critical features of input source congruency, error masking, error reporting, and system reconfiguration.
- The quadruplex C/I network used the full capacity of the AIRLAB DE.
- Documentation for the C/I design at the logic gate level was available.
- Results of these experiments could provide important information about the FTMP design.

The C/I provides the Fault Tolerant Processor (FTP) with certain key fault tolerance capabilities. These capabilities arise for circuitry to support correct replication of simplex source data in all good channels, to detect and mask errors in a single channel, and support reconfiguration by blocking out faulty channels.

The data exchange network associated with a quadruplex FTP is illustrated in figure 3.1-2. Each channel, A, B, C, or D, has transmit and receive registers which can be accessed by the input/output processor or the computation processor via a shared memory bus.

Simplex data (i.e., data available to one FTP channel) is distributed to all FTP channels by a "FROM x" exchange. This will cause the data word in the communicator transmit register of channel A to be sent to all other channels. Data transfers between communicators and interstages are byte serial with two bytes comprising a data word. Data sent to each communicator are distributed to associated interstages.

Each interstage replicates and passes a copy to each communicator. Communicators vote the data received and write the received data into the receive register. Receive registers are then read into the associated channel processor. In this manner simplex data from channel A is read into all good channels.

If each channel has the data required for fault masking, then a "FROM OWN" exchange can be performed. Errors detected during the voting process of an exchange are recorded in the Status Register of each communicator. Each status register can be read by its associated processor. These error reports are intended for use by the FTP in determining and isolating faulty channels.

A write-only Mask Register allows a channel to be excluded from the voting process.

The capability to execute exchanges which bypass the interstage and voter is provided for network testing.

The local data bus provides a large share of the communications. It is a 16-bit tristate bus communicating most of the signals required for redundancy and processor management.

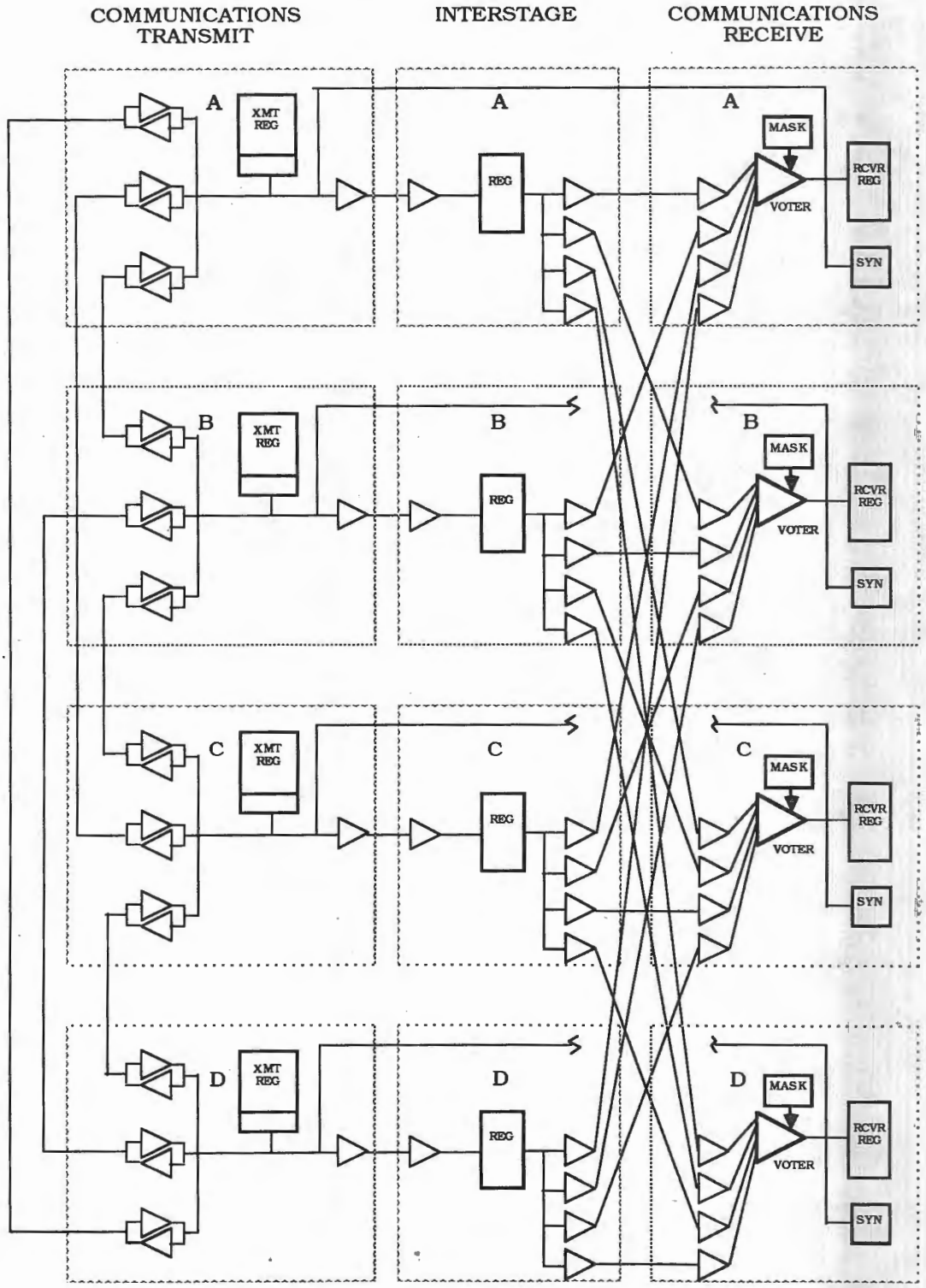


FIGURE 3.1-2. C/I EXCHANGE NETWORK

3.1.1 Gate-Level Model For Redundancy Communicator Interstage

During the analysis several assumptions and approximations were found necessary. Some of these proved limiting on the accuracy of the faulted performance prediction. Key features of the model and several limitations arising from these assumptions are discussed in the next paragraphs.

The local data bus was modeled using AND gates at each source enabled by the appropriate source enable line. The resulting model was functionally equivalent to a tristate bus. The OR function is a phantom gate in that there is no identifiable device in the network to which it can be associated. S-A-0 faults were inserted on this phantom gate to simulate the effects of certain bus faults.

The Transmit Register, Mask Register, and Receive Register are all implemented using D-type latches with tristate outputs. The storage portion of these circuits was modeled as D-flip-flops having tristate outputs. The source bus communicates data bytes from the Transmit Register to cross channel links and to the interstages. This bus was also modeled with AND gates and phantom OR gates similar to the local data bus.

The interstages consist of drivers, receivers, and D-type flip-flops. These devices were modeled using the diagnostic emulator gate and the same flip-flop primitives as above.

The cross channel links that distribute data from each communication source bus to all other communicator source buses consists of a bidirectional, tristate bus driver. The circuitry is diagrammed in figure 3.1-3. The model of this portion of the FTP required 384 connections and 96 phantom gates. A decision was made to simplify the circuit to 192 connections and zero phantom gates. This was based upon an incomplete analysis which indicated that the simpler circuit would adequately represent the network behavior. While data is handled by this model, the effects of faults are not properly treated. Consequently, certain gate faults were not found by the C/I diagnostics. In the discussion presented in Baker, Mangum, and Scheper (1988) these faults would have been discovered using the more complex model.

The Status register file is implemented with two 4x16 Random Access Memory (RAM) chips. Only 5x5 bits of the register in use were modeled. This required five five-bit D-flip-flops with address decoder and selection logic to direct the data to and from these registers.

The Mask transfer block is implemented with a 2048x8 bit programmable ROM. A feature of the diagnostic emulator was used to model this device. This feature allows storage of memory contents outside of the range of memory being modeled. In this function, bits are forced to the values contained in the ROM address. Two other programmable logic arrays are used in the FTP. These were modeled directly as logic equations. The adequacy of stuck-at faults for these devices is questionable since bridging faults cannot be represented with such a simple model. A more elaborate model was judged to be beyond the scope of the modeling effort.

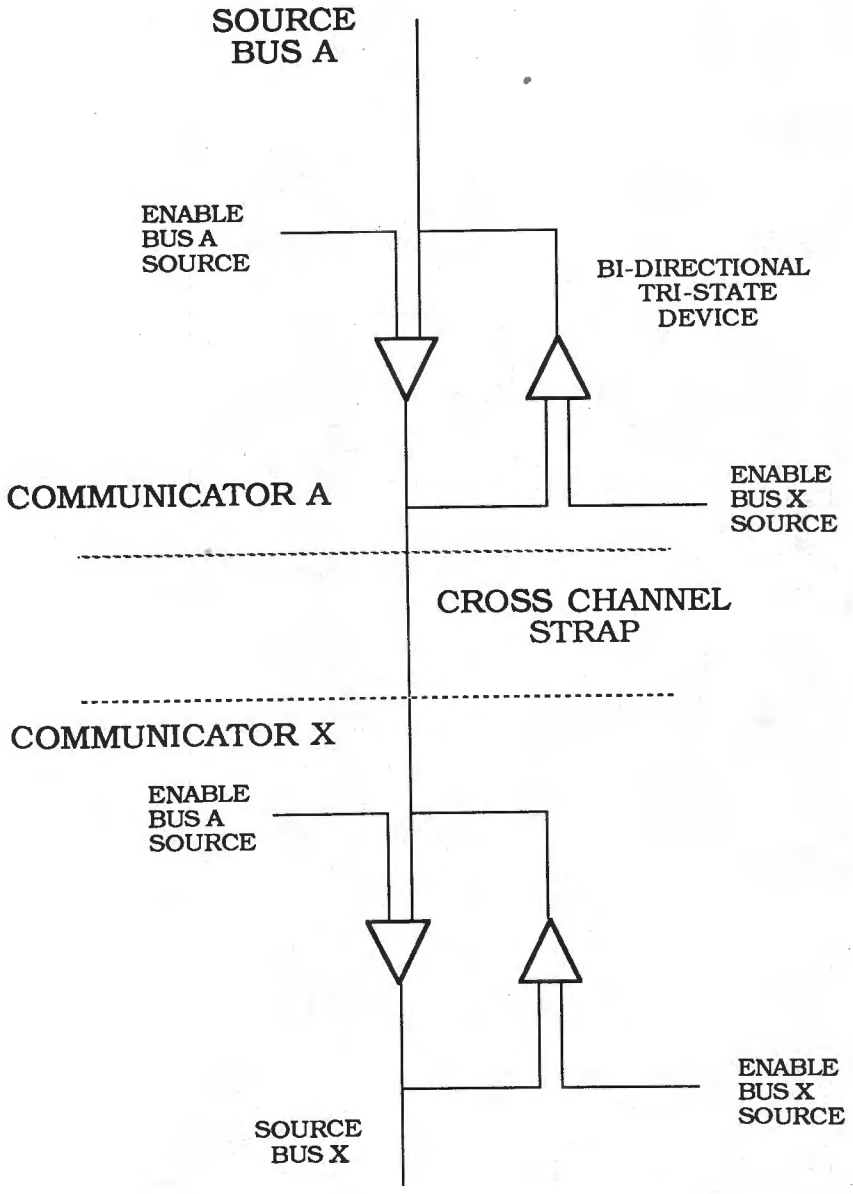


FIGURE 3.1-3. ACTUAL CIRCUITRY FOR CROSS CHANNEL LINKS

3.1.2 Fault Detection Effectiveness of Diagnostic Tests

Most of the effort was directed toward developing and validating the data C/I gate-level model and associated software. These areas of activity included:

- Development and testing of experiment support software.
- Activities associated with learning to use the diagnostic emulator.
- Development of a gate-level model of the C/I.
- Development of C/I sequences that would test the C/I in accordance with the CSDL C/I software description.
- Validation of the C/I model.

The system flow diagram for the diagnostic emulation experiment is shown in figure 3.1-4.

The C/I plays a critical role in FTP fault tolerance by ensuring that all good processors receive identical data values, and by masking errors and providing for reconfiguration after faults are detected. An important part of this operation is the use of diagnostic test sequences for the C/I. These sequences are used to uncover latent faults. The experiment was also designed to determine the effectiveness of proposed diagnostic test sequences.

A typical sequence of operation mixes the test sequences with voting and data transfer between processor channels. In operation, about nine transfers are required between the four processors; the test sequences using vectors consisting of self-test and two exchanges are intermixed with the normal transfers. A complete test requires approximately 2^{16} test vectors. Assuming that self-test is constrained to using only five percent of the processor through-put capacity, the self-test is constrained to use only five percent of the C/I throughput capacity. The self-test will require approximately two minutes to execute.

Estimates of the time needed to conduct a fault injection experiment using the diagnostic emulator are also of interest. Consider an experiment that requires the injection of 2,000 faults with the C/I self-test sequence being run for each of the faults. Assume that the NANO Fast QM-1 is used and that only self-test must be run (i.e., the simulation is 100 percent used for test sequences). The QM-1 will execute approximately 50,000 times slower than real time. The total simulation time required would be 19 years. If the VAX FORTRAN version of the DE were used, the simulation would require 3.8 centuries. These unacceptably long run times provided motivation for restricting the experiment. If the required simulation time could be reduced by a factor of about 100, then 200 hours of run time on the QM-1 would be needed. Such a simulation would be well within the feasible range.

A reduction in time was planned for and accomplished by noting that all of the transactions needed to share opinions in an actual FTP would not be needed in an experiment. The additional factor of 100 reduction was obtained by reducing

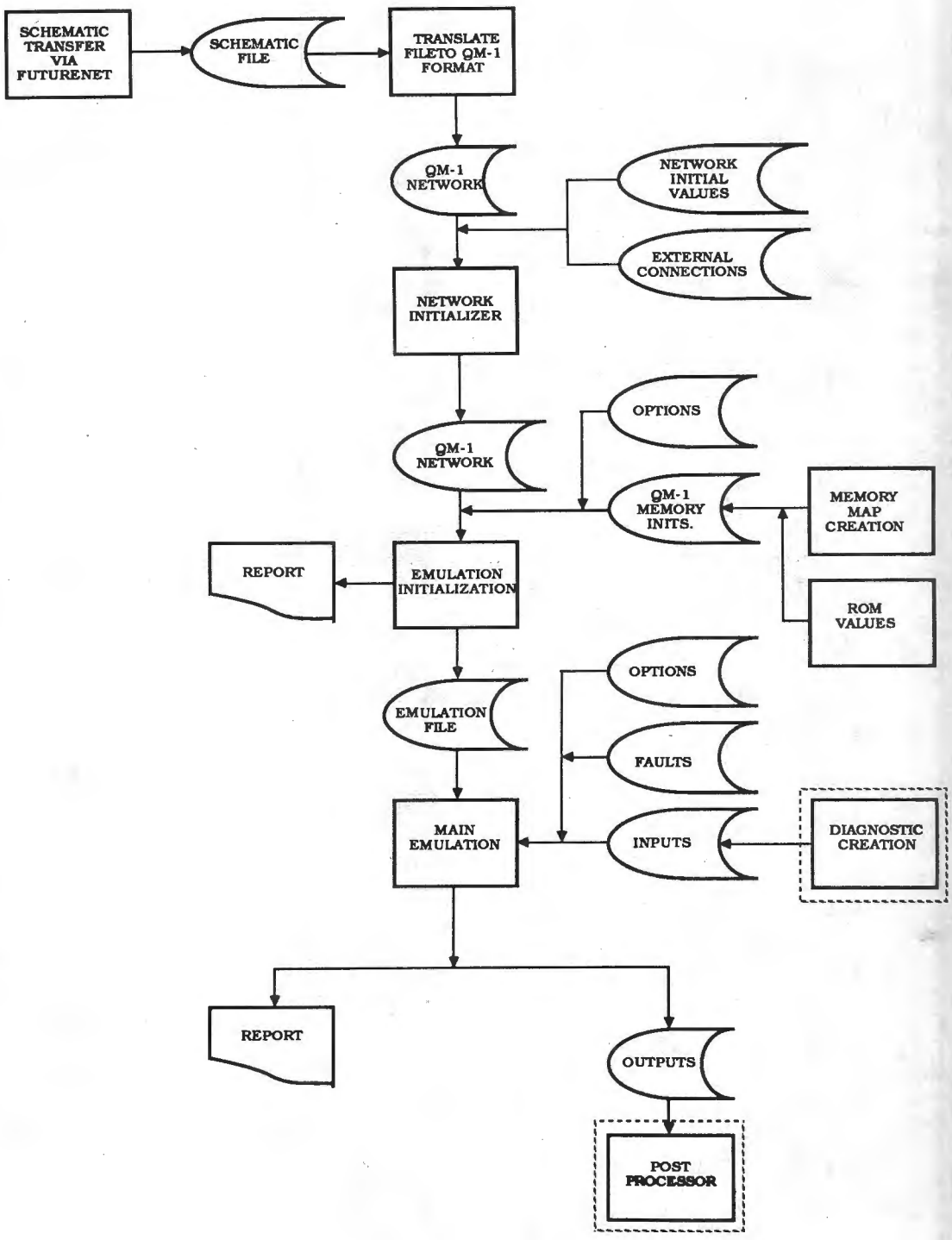


FIGURE 3.1-4. SYSTEM FLOW DIAGRAM

the number of test vectors. This was accomplished by eliminating the test vectors on the Programmable Array Logic (PAL) failure modes since these were modeled not to have errors. Further it was decided to test only the C/I vectors for the quadruplex and triplex configurations and only one for the PAL chips. The experimental tests used are shown in table 3.1-1. The C/I model is shown in figure 3.1-5.

3.1.3 Experimental Results on Hardware/Software

The experimental data presented are drawn from the test report by Baker, Mangum, and Scheper (1988). These data are not adjusted or modified to account for certain problems in the models. For each self-test, the percentage of faults detected is relative to the number of faults injected. It should be noted that any conclusions need to be drawn from the complete report and not this synopsis. Data shown is to illustrate the kinds of experimental results available and not the detailed conclusions.

TABLE 3.1-1. TESTS USED FOR EXPERIMENT

- VOTER TESTS

  Tested a single voter PAL chip (2 bits) in the quad and triplex configurations. Used both upper and lower bytes to give two test vectors per vote.

- MASK TRANSFORM TESTS

  Tested mask transform logic for quad and duplex configurations.

- CURRENT STATUS UPDATE TEST

  Tested the current status update PAL.

- PRESENCE TEST

  Equivalent to CSDL "Presence".

- BASIC TEST

  RTI test which tested a range of C/I functions beyond "Presence".

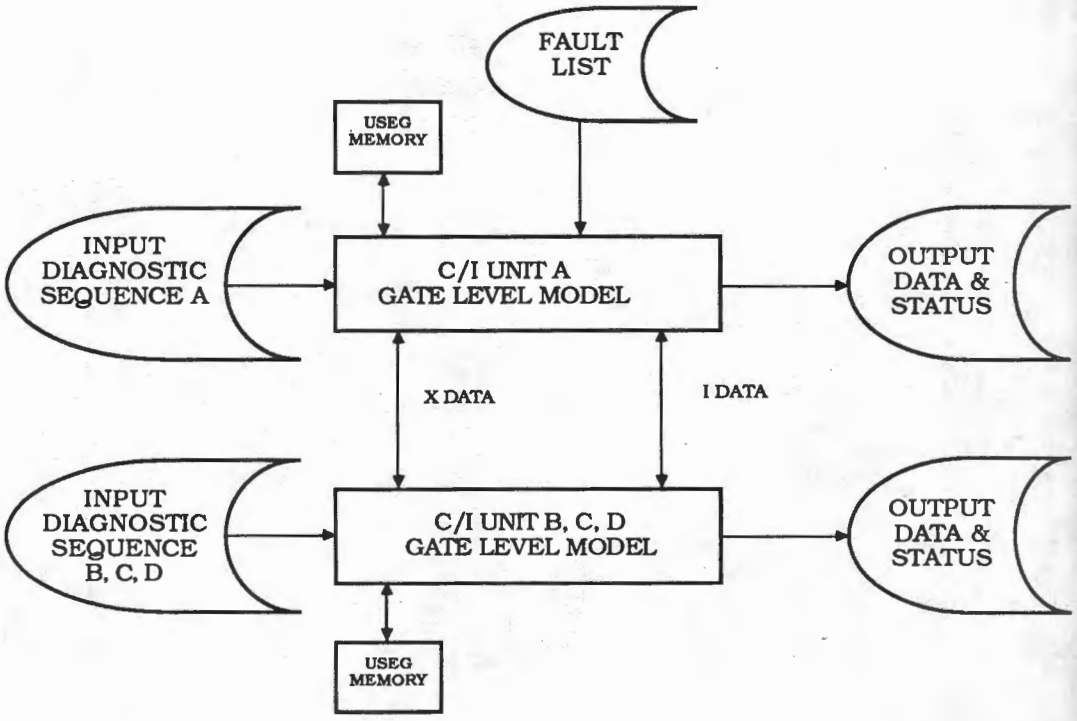


FIGURE 3.1-5. C/I MODEL FOR DIAGNOSTIC EMULATION

Figures 3.1-6 and 3.1-7 illustrate some of the detailed results in the report. The first shows the effectiveness of the diagnostic test run on the simulation. The second shows the performance of the non-voter versus the voter tests in detecting faults.

Table 3.1-2 summarizes the undetected faults for the 2,000 injected faults. If the results were adjusted for assessments of the undetected faults, and if the identification (ID) logic faults are inconsequential, the adjusted performance of the C/I self-tests would exceed 98 percent and could approach 100 percent coverage.

Much of the effort in this experiment was directed toward recapturing the C/I design, and validating the model derived from the recaptured design. Well established Computer Aided Design (CAD) tools will be required to expedite the modeling process to deal effectively with VLSI designs.

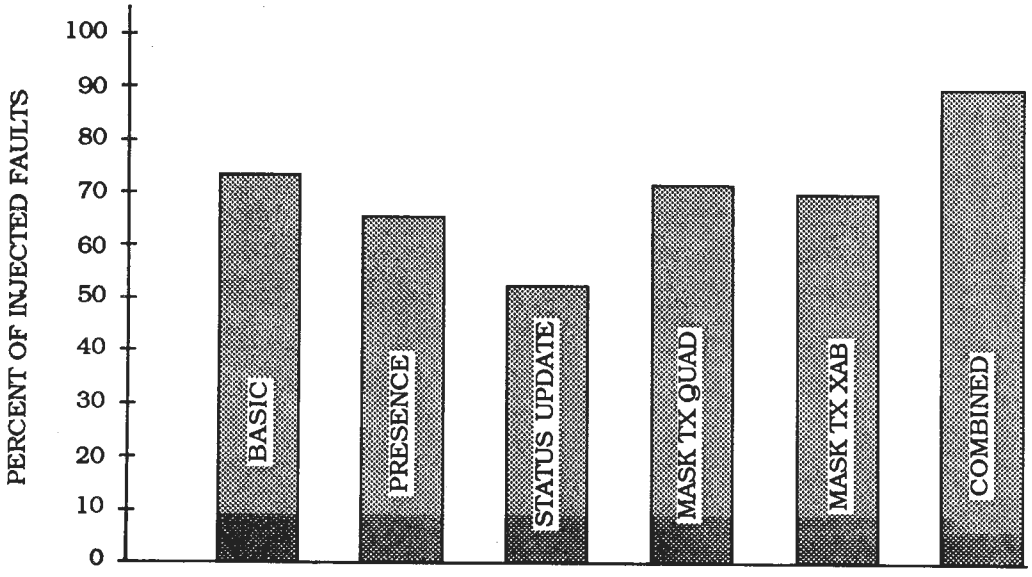


FIGURE 3.1-6. DIAGNOSTIC TEST PERFORMANCE

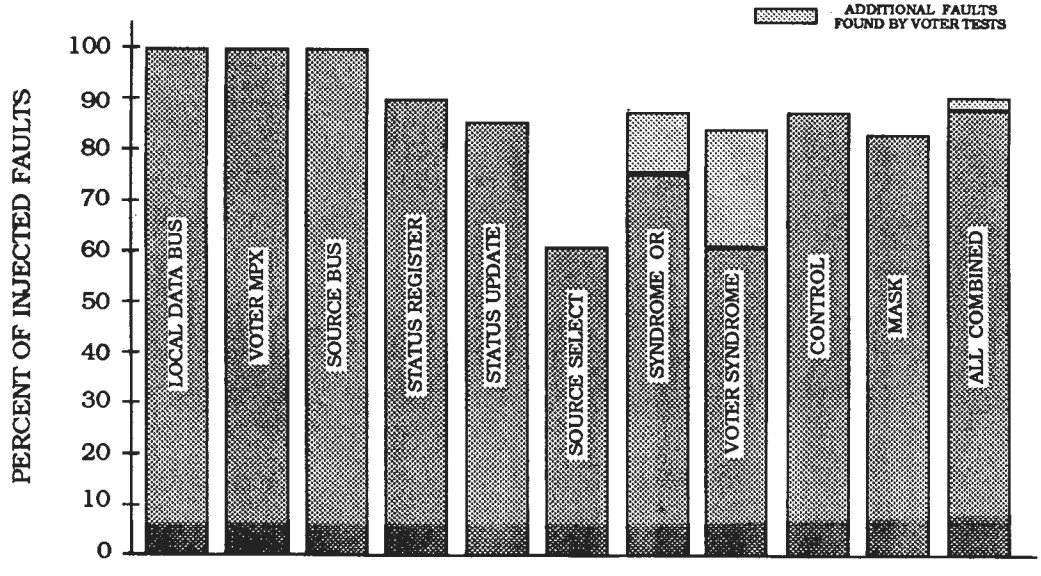


FIGURE 3.1-7. COMBINED PERFORMANCE OF ALL NON-VOTER TEST

TABLE 3.1-2. SUMMARY OF UNDETECTED FAULTS

FAULT SET	UNDETECTED FAULTS	SOURCE	GOAL	ESTIMATED UNDETECTABLE
Source Select	47	Cross channel model and ID logic		0
Control	26	ID logic and cross channel + product interface and reset		3
Voter Syndrome	19		Run Voter Quad Test	0
Status Update	16	Not known	Run Voter Quad Test	0-16
Syndrome Or.	14	Not known	Run Voter Quad Test	0-14
Status Reg.	15	Status reg. address decode	Modify Self-Test	0
Mask	4	ID logic		0
Local Data Bus	0			0
Voter Mux	0			0
Source Bus	0			0
Combined	141			3-33

3.2 Electromagnetic Induced Transient Injection Test Examples

3.2.1 Electromagnetic Transients and Fault Injection

It has been estimated that approximately 80 percent of sudden computer failures, or crashes, are caused by some form of electric transient (Iyer and Rosetti 1986). This statistic serves nicely to illustrate the crucial role that transient EM fields may play in flight system fault tolerance and reliability.

The avionic industry is experiencing a rapidly growing dependence on extensive digital control of complex and crucial flight systems. These systems necessarily demand high reliability and fault tolerance. Digital systems, however, are notoriously vulnerable to extraneous electric fields and transient pulses of voltage.

Modern aircraft are also utilizing more composite types of materials in their structural design. These materials offer high strength and lightweight structures which are very beneficial for increased fuel efficiency. However, these materials are ordinarily less efficient at reflecting and attenuating EM fields. Furthermore, a material joint between an all-metal part and a composite part can serve as an efficient penetration point for EM fields by disrupting the surface currents induced on the two materials. In short, modern aircraft electronic equipment will likely be subjected to significant EM induced transients in the wiring and equipment.

Guidelines are being developed for methods of determining upset susceptibility of redundant systems to EM transients. The studies of transient susceptibility and transient injection, discussed in this section, will lead to increased understanding of upset mechanisms and thereby improved tolerance strategies.

EM transient fields arise from a wide variety of sources. The most significant sources in terms of very high potential energy are lightning and high power Radio Frequency (RF) transmitters, such as for radio, microwaves, and radar.

Lightning pulses are similar to Nuclear Electromagnetic Pulses (NEMP), both having similar waveforms in their signature transient fields. Both may typically be characterized in the time domain by a double exponential waveform with a very rapid rise time, and a fall time from 10 to 100 times longer than the rise time. An example of this waveform is illustrated in figure 3.2-1.

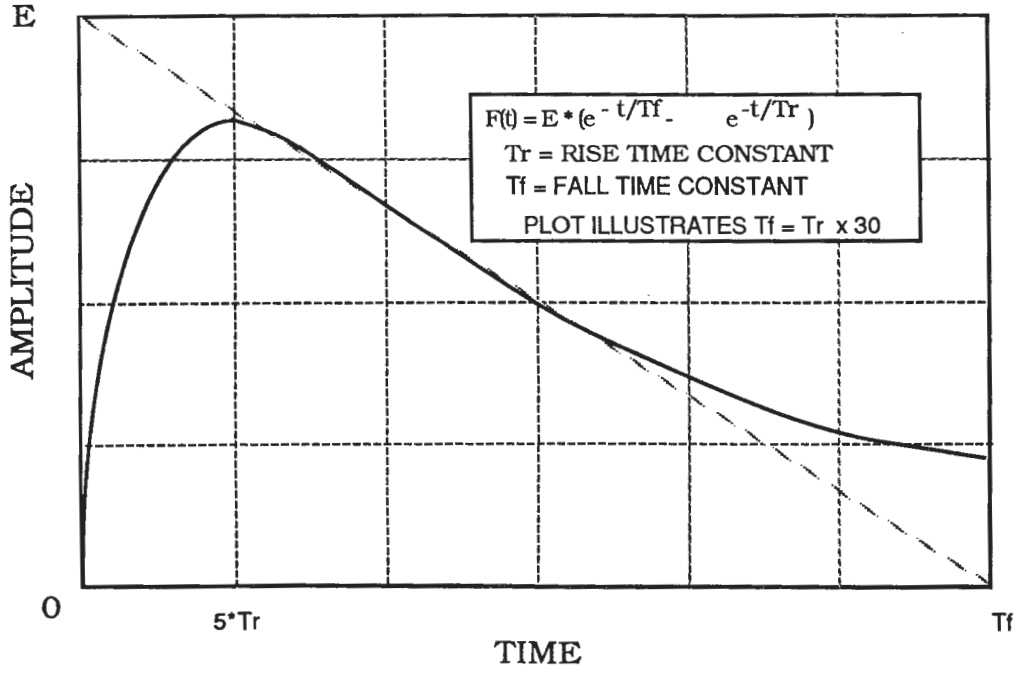


FIGURE 3.2-1. TIME-DOMAIN TRANSIENT WAVEFORM GENERATED BY LIGHTNING

These transients predominantly cover the frequency range from tens of kiloHertz (kHz) to hundreds of megaHertz (MHz). An adequate simulation waveform for laboratory and first order analysis of equipment response purposes is an exponentially damped sinusoid which reaches about 60 percent of maximum amplitude on the order of four cycles (SAE AE4L 1988, Rasch 1987, Cooley and Shortess 1988).

Aircraft are regularly exposed to High Energy RF (HERF) fields from radio, microwave, and radar. Effects on aircraft systems from these power transmitters is a function of the aircraft materials and electronics technology, as well as the power levels. Important factors include the transmitter modulation characteristics (AM, FM, pulsed, continuous wave) and geometry, such as the orientation of the aircraft in high field regions and the proximity of the source antenna.

Ordinarily, the voltage and current waveforms on an antenna will be the same as the EM field waveform. However, an exponentially damped sinusoidal EM field will induce an exponentially damped sinusoidal voltage/current onto a line or within a component. The coupling mechanisms for transients in aircraft wiring are often much more complex events. The aircraft structure and wiring acts like a complex antenna for remote EM sources. Complex RLC and M parameters govern the coupling from currents on the aircraft structure to critical wire and circuits within. Induced transient waveforms inside an aircraft may be derivative signals of complex filter functions of the incident EM fields. The most dominant coupling mechanisms include current or voltage induced directly by the EM field onto a wire or component (essentially acting as part of an antenna). In addition, important interactions include inductive or capacitive coupling from some conductive medium or aperture near a wire or component, and other interactions, such as fluctuations in the ground potential as the transient passes.

Methods for simulating or electrically reproducing the EM induced transients has to date been accomplished through laboratory bench test techniques. It is currently unknown what the implications may be of introducing a transient through only one specific wire in a test experiment. One feature of EM transients is that they may affect all the components of a redundant system at the same time. This massive injection of faults could effectively circumvent the fault detection and recovery process. It would seem likely that the system would be affected in parts. Some parts of the system may be most vulnerable to perhaps an inductive coupling of a current to a wire, whereas other parts of the system will have difficulty with a ground potential fluctuation. Designing a test program to address all of these issues is very challenging. Of course, economics and time are always the practical limitation of any test capability.

Another consideration is the timing of the transient injection. It may be logical to inject the faults using the same clock used for the rest of the testing system. However, if this timer is also driving the test subject, the result could be a synchronous injection of the fault occurring at the same time as changing system states. This would likely influence the data with an unrealistic bias. A technique for random asynchronous fault injection would

more closely resemble actual environments. Detail studies could benefit from different time phase injections relative to system clock pulses.

Experimental investigations of EM transient induced faults may be further divided into two realms of testing depending upon the level of system definition in the test. As discussed in section 2, tests may be performed entirely as simulations on computers.

A real or conceptualized system may be modeled down to the functional block, VLSI chip, or the transistor gate level using available circuit and logic simulators. The simulated system is then stepped in time through a series of tasks as fault signals are injected into circuit nodes using ideal voltage or current sources. This technique has the advantage of being able to completely control where faults are injected, and at precisely what time and amplitude. Simulations can trace exactly what the injection produces throughout the entire system. The limitation of this method is in the expense (both time and money) of the computer time required to step the entire system through any substantial amount of real time, not to mention analyzing the enormous amount of data which is generated. Furthermore, a software simulation can never precisely parallel the actions of a real system.

At the other extreme, hardware testing physically injects EM transient currents or voltages into a real system.

An actual system may be brought into the laboratory, and the system altered to allow transient injection and monitoring of pertinent system data. Care must be taken not to perturb the true functionality by loading the system's hardware with the transient injection equipment or the data monitoring equipment. The obvious advantage of the real system method is that a real system is observed in an actual functional environment while subjected to controlled EM transients. A very broad range of system states may be investigated in this manner over long periods of time. The disadvantages to this method are also numerous. It may be very difficult to access some pertinent data points; fault injection levels are much less controllable (although more realistic), and tracking fault effects through the system may be difficult.

3.2.2 Upset and Failure Detection

Component disruptions in FT systems must be handled through conventional fault tolerant methods such as redundancy and reconfiguration. Digital systems can be particularly vulnerable to EM transients. System functions can be disrupted in two ways:

- The components themselves may be permanently damaged by a high potential or current. It is known that charge accumulation levels above nine picocoulombs injected into a node within a CMOS microcomputer VLSI chip can cause permanent damage (Nichols et al. 1985). The speed and high magnitudes of many EM transients can easily lead to such charge accumulations.

- Component functions may be disrupted without damage. This is termed an "upset". The normal functionality of a system or component may be altered

by the passing of a transient. The loss of function may persist after the transient has gone, although no actual physical damage occurs.

A simple example of a serious disruption would be a logic gate temporarily stuck in a single state. This particular upset could cause a microprocessor to access an erroneous address space, program execution to branch to a subroutine intended for a completely different function, or incorrect mathematics to occur in an internal register.

Testing components and systems for EM transient fault tolerance is also a new area of research. A few pioneering efforts have been made on general purpose microprocessors executing simple codes, but these systems are not necessarily representative of actual avionic systems. Two recent works (Belcastro 1989 and Carreno, Choi, and Iyer 1989) provide sources for references to earlier studies. To date there are very few studies which have been performed on actual avionic systems; section 3.2.3 describes two such studies on an avionic electronic engine controller. These studies are detailed in Belcastro (1989) and Carreno, Choi, and Iyer (1989).

The two studies covered in section 3.2.3 are particularly beneficial in that each study was performed in a different way using the same electronic engine controller. One study completely utilized the software simulation approach and the other study utilized actual hardware systems. Aside from these strategic differences, the two testing methods have much in common. It is likely that future EM transient testing strategies will not deviate too far from these fundamental practices.

To monitor a system for fault occurrences, the tester needs to know the "normal" system function. The known "normal" function detail must match the same level of detail as the desired testing level of the target system. If the testing is to be done down to the transistor level in a logic system, then the "normal" function of the system at the transistor level must be known.

This knowledge can be gained in several ways as discussed in section 2. One extreme (all in software) is to run a testing algorithm through the system (simulated or actual) without the injection of any faults. The system can then be rerun with precisely the same algorithm with fault injection. The second extreme (all in hardware - used primarily for actual hardware testing) is to run two or more systems in parallel with one system remaining non-faulted as the reference comparison for normal operation.

The multiple real hardware method is limited by such factors as system size, cost, availability, and manufacturing repeatability. Using real hardware is usually faster and more convenient for fault detection testing since detection may occur in "real time" through direct comparison of system states. The single system method is often more economical, slower, and may provide better comparison data since exactly the same algorithm is run through the same system.

The types of system data collected during EM transient testing will probably not differ much between the two testing strategies. This system data will normally include control lines, address buses, data buses, and microprocessor status registers. The system control lines establish the operational mode of the

system. These will often give the most obvious indication of incorrect operation. (The control lines are also probably one of the most significant sources for EM transient conduction to the sensitive digital components.)

The system data buses should be monitored for signs of incorrect command modes and erroneous data accessed (or induced). This data may provide the earliest indication of upset or failure. Similarly, the address buses provide indication of invalid memory access and program execution. This is obviously crucial data for detecting and tracking upset/failure response. Microprocessor status registers may be helpful in tracking a fault occurrence and response. The microprocessor status registers typically include operational information flags, such as parity, configuration, self-tests, watchdog timer, and arithmetic overflow.

3.2.3 Digital Engine Controller Test Examples

Two complementary studies, Belcastro (1989) and Carreno, Choi, and Iyer (1989), serve as excellent examples for describing the proposed processes of determining EM transient fault behavior in avionic systems. Belcastro (1989) actually documents the plan for a test which had not been completed at the time of its publication. However, the testing should be complete by the time of this publication. Carreno, Choi, and Iyer (1989) documents in detail an actual test completed in late 1989. These are two of the first such tests to be performed in the area of EM transient upset detection in actual avionic systems. Both of these examples offer tremendous insight into the current state of the art in EM transient fault modeling, detection, and injection. It must certainly be anticipated that this type of work will greatly increase in the coming years.

Both tests utilize the same Hamilton Standard electronic engine control (EEC) unit. This controller is a commercial unit for Pratt & Whitney engines. A block diagram of the EEC system is shown in figure 3.2-2. The fault tolerant strategy utilizes a dual-channel system with an extensive self-testing algorithm to determine which channel is in control of the engine.

3.2.3.1 Physical Transient Insertion

The testing strategy of Belcastro (1989) tests real hardware using a single system run in a non-faulted condition. This run establishes normal mode reference data and ranges. Another run is then made in which the hardware is subjected to EM transient waveforms coupled inductively onto wiring in the EEC. The EEC simulation strategy and a functional block diagram of the test system are shown in figures 3.2-3 and 3.2-4, respectively.

The EEC hardware is altered for test to allow access of the data bus, address bus, and the control lines of the microprocessors in each channel. Digital data is recorded through a DAS8200 digital analysis system using up to 240 input capture lines.

Upset detection is defined as the occurrence of one of three conditions:

- Selected parameter values for engine speed, throttle resolve angle, and inlet air temperature out of range for n cycles.

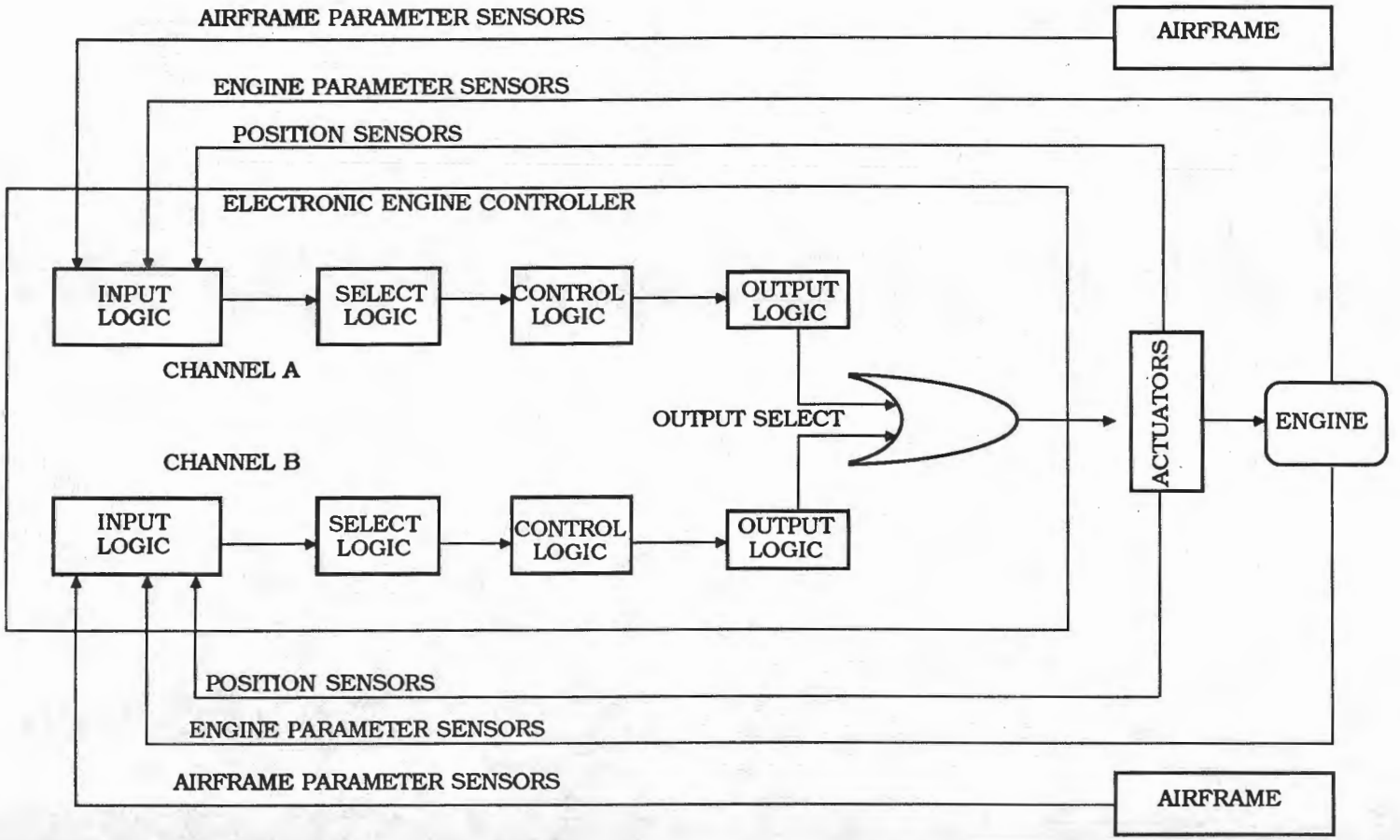


FIGURE 3.2-2. FUNCTIONAL BLOCK DIAGRAM OF THE HAMILTON STANDARD ELECTRONIC ENGINE CONTROLLER

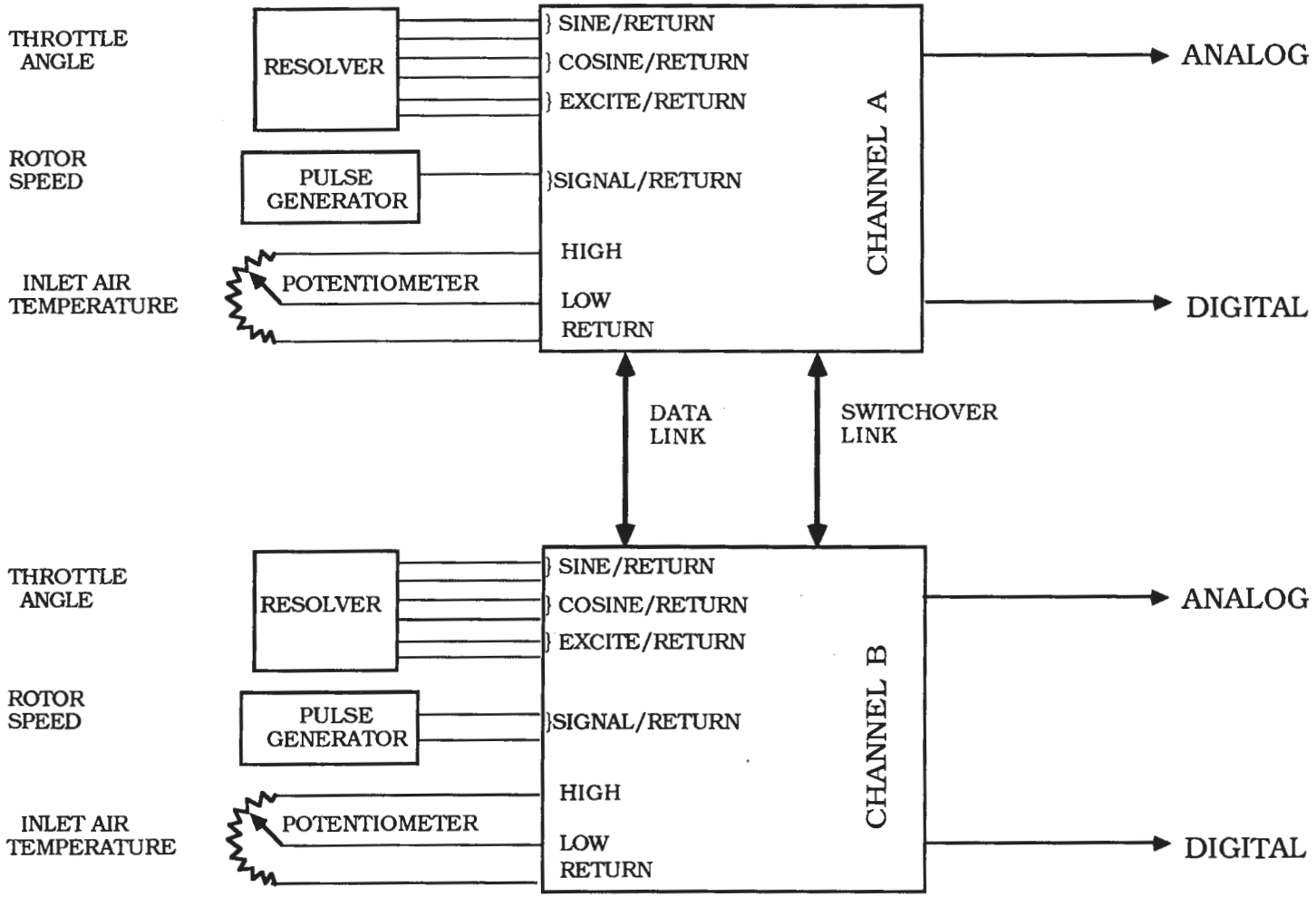


FIGURE 3.2-3. SIMULATION OF EEC INPUTS FOR EM TRANSIENT FAULT INJECTION

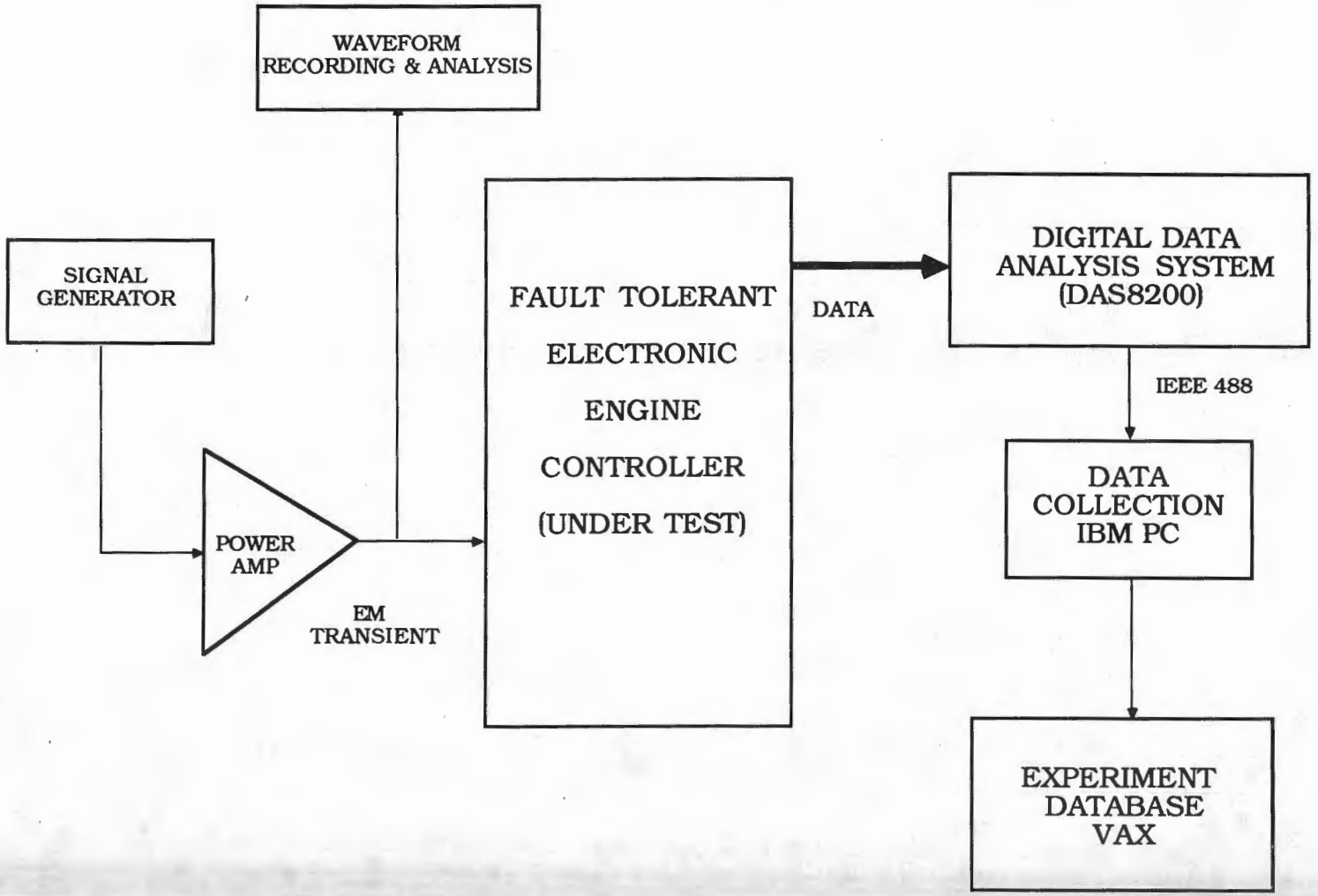


FIGURE 3.2-4 EEC INSTRUMENTATION FOR EM TRANSIENT FAULT INJECTION

- Control commands obtained from the data bus out of range for a given flight mode for n cycles.

- Invalid memory space accessed for n cycles.

Indication of any of these events in the monitored data will result in time-stamped data being recorded for that particular test run; that is, only data from upset/failure detection will be recorded.

EM transient injection is accomplished with an inductive coupler using an exponentially damped sinusoid transient. This test equipment is capable of delivering a 1,000 watt signal over a frequency range of 10 kHz to 220 MHz.

The EEC will be cycled through eight simulated flight conditions utilizing the simulation strategy depicted in figure 3.2-3 with nominal flight parameter values, including most actual engine parameters, stored in a ROM. The EEC will experience idle, acceleration, take-off, climb, cruise, deceleration, reverse, and partial power.

The objectives of initial testing are to help establish a methodology for upset detection strategies, to add to the EM transient fault tolerance database, and to further define characteristic induced-waveform threshold norms leading to upset conditions.

The results of the actual experiment outlined by Belcastro (1989) will complement the experiments reported by Carreno, Choi, and Iyer (1989).

3.2.3.2 Simulated Transient Pulse Injection

The study by Carreno, Choi, and Iyer (1989) approaches testing of the EEC from a software perspective. One channel of the EEC, excluding the interfaces, was simulated with the aid of an application program supplied by Hamilton Standard. Hamilton Standard also supplied a complete gate-level description of the entire 16-bit HS1602 microprocessor used in the EEC. The gate-level description was then further reduced to a CMOS transistor-level description of the gate-level modules. Capacitance loads in the circuit were calculated from metalization lengths used in the circuit layout.

The simulation was carried out using the SPLICE1 mixed-mode simulation program. This program is capable of performing both analog circuit analysis and logic simulation. SPLICE1 was modified to allow for specification of nodal EM transient injections (as opposed to reconfiguring the circuit model by adding a voltage or current source to the individual nodes one at a time). The SPLICE1 simulator was also modified to allow for manipulation of external data files in order to simulate access of memories (RAM, ROM). The program was expanded to output up to 80 nodes' data only when data changed, and to provide an extended trace facility which generates output for a node each time a gate driving that node is evaluated by SPLICE1. This latter feature is particularly helpful when tracing fault propagation through the nearly 4,000 nodes of the HS1602 microprocessor model.

The simulated EM transient fault was of the double exponential type illustrated in figure 3.2-1. A current source was used for the injection. This current source had a rise time constant of 0.05 nanoseconds and a fall time constant of 0.164 nanoseconds. The amplitude of the source varied so as to produce a well defined charge accumulation in the range of 0.5 to nine picocoulombs. This method was used because it was found that the total charge accumulation was the more significant factor in determining upset rather than amplitude alone.

The operation of the microprocessor was categorized on a six part functional modular level: watchdog self-test, multiplexer, decoder, control, countdown, and arithmetic logic unit (ALU). The microprocessor was stepped in one nanosecond increments (82 steps per 12.2 MHz clock cycle) through 74 instruction cycles which included a watchdog test, parity test, instruction set test, RAM test, ROM sum test, and test case parameter transfers to RAM.

Simulations were performed with "gate distances" of one, two, three, four, and five modeled to the transistor level. Gate distance is defined in figure 3.2-5. It was observed that for a combinatorial circuit a minimum of three gate distances are needed for accurate simulation, and at least four gate distances are needed if at least one latch (i.e. flip-flop) is within the three gate distance.

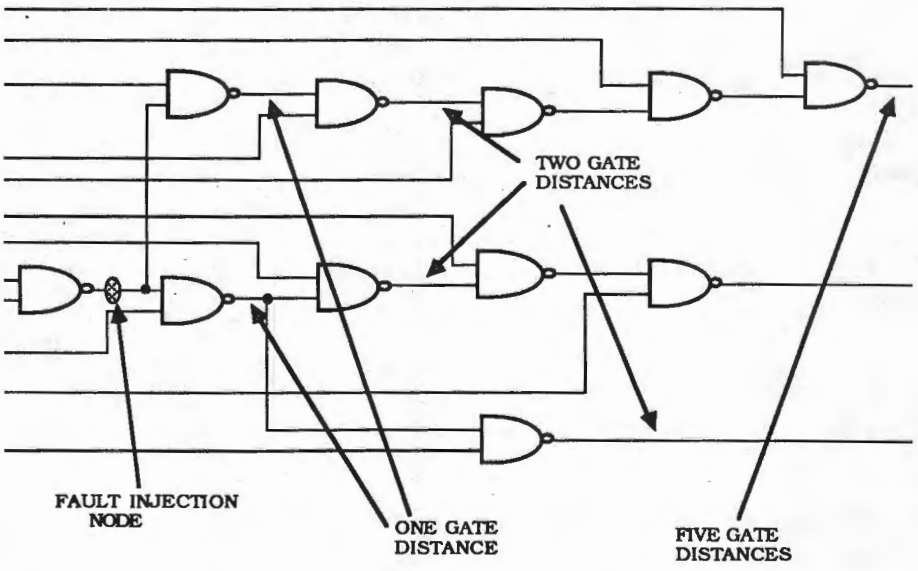


FIGURE 3.2-5. CLARIFICATION OF GATE DISTANCE DEFINITIONS

A non-faulted simulation run was used to generate the normal operation comparison data. This data was also compared to data obtained from an actual hardware unit including a comparison of such details as electrical delays, gate loading, etc. Correct digital data was observed in the simulation including

transitional times to within three nanoseconds of the actual system. These results suggest a high confidence in the modeling accuracy.

A total of 2,100 EM transient injections were performed by randomly selecting seven nodes from each of the six primary functional modules mentioned above. Charge accumulation levels of 0.5, one, two, three, four, five, six, seven, eight, and nine picocoulombs were injected at five different times during the program execution.

Errors were defined as first order if a logic error (difference from the comparison normal run) occurred one clock cycle after the transient injection. The error becomes second order if the logic difference persists two or more cycles after the transient fault injection. These definitions are also used to define pin errors if the device's pins were actually affected. A functional error was noted if the processor's operational function deviated from normal.

Table 3.2-1 summarizes the total error occurrence for the entire test as a function of the accumulated charge level induced by the simulated transient injection. Table 3.2-2 lists the occurrence of all first order errors (most sensitive detection) encountered within each of the six functional modules.

Both of these tables provide significant information about the sensitivity of the HS1602 EEC microprocessor. There is little sensitivity to transient charge accumulations of two or less picocoulombs. There are increasing errors with increasing charge accumulation between two and six picocoulombs. Charge levels above six picocoulombs do not increase upset errors. (Above nine picocoulombs permanent damage begins to occur.)

There is a very strong sensitivity of the microprocessor with respect to functionality. The watchdog self-test and the ALU are both very likely to be upset by this type of EM transient. The decoder, control, and countdown functions are about half as likely as the ALU and watchdog to suffer upset, whereas the multiplexer seems to be rather immune to the transient.

The results of this experiment offer a significant contribution to the growing database for EM transient injection experimentation in avionic systems. These results will be very interesting to compare with the results of the first test described in this section since the two experiments offer two very different perspectives on the same system.

It should be noted that significant computer resources are required. This experiment covers only a very small fraction of the whole system with respect to both components and function time. A simulation run of 12 two-input logic gates and two D-type flip-flops modeled at the transistor level running for 10 milliseconds requires about 130 hours of computer time on a MicroVAX II computer. This will obviously limit the simulation capability of complex avionic systems. However, the first test discussed will not likely produce the highly detailed results of this technique which provide definition of how the microprocessor upset occurs, and particularly in detail within the VLSI chip.

TABLE 3.2-1. ERRORS IN SIMULATED EM TRANSIENT ON EEC

INJECTION CHARGE LEVEL (Picocoulombs)	LOGIC ERRORS First Order	LOGIC ERRORS Second Order	PIN ERRORS First Order	FUNCTIONAL ERRORS
0.5	1	0	0	0
1.0	2	0	0	0
2.0	9	0	5	7
3.0	38	10	23	17
4.0	52	14	30	24
5.0	64	15	34	29
6.0	72	17	37	29
7.0	76	19	41	29
8.0	77	22	42	29
9.0	79	23	43	29

TABLE 3.2-2. ERRORS IN SIMULATED EM TRANSIENT ON MICROPROCESSOR MODULE

MICROPROCESSOR FUNCTIONAL MODULE	TOTAL FIRST ORDER ERROR INCIDENCE	ERROR INCIDENCE PERCENTAGE (Per Injection)
WATCHDOG	132	37.7%
MULTIPLEXER	0	0.0%
DECODER	68	19.4%
CONTROL	70	20.0%
COUNTDOWN	70	20.0%
ALU	130	37.1%

BIBLIOGRAPHY

Arnold, T. F., "The Concept of Coverage and Its Effect on the Reliability Model of a Repairable System", IEEE Transactions on Computers, Vol. C-22, pp. 251-254, March 1973.

Avizienis, A. and J. Laprie, "Dependable Computing: From Concepts to Design Diversity", Proceedings IEEE, 74(5):629-638, May 1986.

Avizienis, A. and D. Rennels, "Fault-Tolerance Experiments with the JPL Star Computer", COMPCON '72 Digest of Papers, pp. 321-324, September 1972

Baker, Robert, Scott Mangum, and Charlotte Scheper, A Fault Injection Experiment Using the Airlab Diagnostic Emulation Facility, NASA CR 178390, NASA Langley Research Center, March 1988.

Bavuso, Salvatore J. and Paul S. Miner, "Characterization of the Faulted Behavior of Digital Computers and Fault Tolerant Systems", Presented at the Hawaii International Conference on System Sciences, Honolulu, HI, January 1989.

Bavuso, Salvatore J., et al., Applications of the Hybrid Automated Reliability Predictor, NASA Technical Paper 2760, NASA Langley Research Center, December 1988.

Becher, Bernice, Diagnostic Emulation: Implementation and User's Guide, NASA CR 178391, NASA Langley Research Center, December 1987.

Belcastro, Celeste M., "Laboratory Test Methodology for Evaluating the Effects of Electromagnetic Disturbances on Fault-Tolerant Control Systems", NASA Technical Memo 101665, NASA Langley Research Center, November 1989.

Benson, J., D. Mulcare, and W. Larsen, Hardware Fault Insertion and Instrumentation System: Experimentation and Results, DOT/FAA/CT-86/34, FAA Technical Center, March 1987.

Butler, Ricky W. and Anna L. Martensen, The Fault-Tree Compiler (FTC), Program and Mathematics, NASA Technical Paper 2915, NASA Langley Research Center, July 1989.

Butler, Ricky W. and Allan L. White, SURE Reliability Analysis, Program and Mathematics, NASA Technical Paper 2764, NASA Langley Research Center, March 1988.

Carreno, Victor A., G. Choi, and R. K. Iyer, "Analog/Digital Simulation Study of Transient Induced Logic Errors and Upset Susceptibility of an Advanced Control System", NASA Technical Memo XXX, Draft, November 1989.

Carter, William C., et al., Design for Validation: An Approach to Systems Validation, NASA CR 181835, NASA Langley Research Center, May 1989.

Cohen, Gerald C., Charles W. Lee, and Daniel L. Palumbo, "Methods for Evaluating Integrated Airframe/Propulsion Control System Architectures", IEEE-Naecon, Dayton, OH, May 1987.

Cohen, Gerald C., et al., AIAA-89-2703, "Experiences With a Prevalidation Methodology for Designing Integrated/Propulsion Control System Architectures", AIAA/ASME/SAE/ASEE 25th Joint Propulsion Conference, Monterey, CA, July 1989.

Cooley, W. W. and D. Shortess, Lightning Simulation Test Technique Evaluation, DOT/FAA/CT-97/38, FAA Technical Center, October 1988.

Courtois, B., "Some Results About the Efficiency of Simple Mechanisms for the Detection of Microcomputer Malfunctions", 9th International Symposium of Fault Tolerant Computing, pp. 71-74, 1979.

CSDL, The Charles Stark Draper Laboratory, Inc., Document #43-334, CBD Announcement Ref SS017, Issue PSA-9214, November 1986.

Cullyer, W. J. and C. H. Pygot, "Application of Formal Methods to the VIPER Microprocessor", IEEE Proceedings, Vol 134, Pt. E, No. 3, May 1987.

Czeck, E. W., Z. Z. Segal, and D. P. Siewiorek, Software Implemented Fault Insertion an FTMP Example, Department of Computer Science, Carnegie-Mellon University, January 15, 1989.

Czeck, E., D. Sieworek, and Z. Segal, Software Implemented Fault Injection, an FTMP Example, NASA CR-178423, Carnegie Mellon University, October 1987.

Czeck, Edward W., Daniel P. Siewiorek, and Zary Z. Segal, Predeployment Validation of Fault-Tolerant Systems Through Software-Implemented Fault Insertion, NASA CR 4244, NASA Langley Research Center, July 1989.

Czeck, Edward W., et al., Software Implemented Fault Insertion: An FTMP Example, Department of Computer Science, Carnegie-Mellon University, January 1987.

Decouty, B., G. Michel, and C. Wagner, "An Evaluation Test of Fault Detection Mechanisms Efficiency", 10th International Symposium on Fault Tolerant Computing, pp. 225-227, 1980.

Dotson, Kelly J., "Examples of Nonconservatism in the CARE III Program", NASA Technical Memo 100524, NASA Langley Research Center, January 1988.

Dotson, Kelly J., Analysis and Testing of the SURE Program, NASA Technical Paper 2817, NASA Langley Research Center, August 1988.

Dzwonczyk, M. J., et al., Avionic Architecture Studies for the Entry Research Vehicle, NASA CR 181828, NASA Langley Research Center, May 1989.

Ferguson, F. J., Inductive Fault Analysis of VLSI Circuits, PhD Thesis, Carnegie-Mellon University, Electrical and Computer Engineering Department, October 1987.

Ferguson, F. J., J. P. Shen, and W. Maly, "Inductive Fault Analysis of MOS Integrated Circuits", IEEE Design and Test of Computers, 2(6):13-26, December 1985.

Finelli, G., "Characteristics of Fault Recovery Through Fault Injection on FTMP", IEEE Transactions on Reliability, R-36(2):164-170, June 1987.

Finelli, George B., AIAA-88-4436, "Results of Software Error-Data Experiments", AIAA/AHS/ASEE Aircraft Design, Systems and Operations Conference, NASA Langley Research Center, September 1988.

Goetz, F., "Design for Detection, an Attempt at Complete Fault Detection of State", Digest of Papers, COMPCON '78, pp. 325-328, 1972.

Iyer, R. K. and D. J. Rossetti, "A Measurement Model for Workload Dependence of CPU Errors", IEEE Transactions Computers, vol. C-35, pp. 511-519, June 1986.

Johnson S. J., "Evaluation of Fault-Tolerant Parallel Processor Architectures Over Long Space Missions", NASA Memorandum 4123, NASA, 1989.

Johnson, Sally C., "Assist User's Manual", NASA Technical Memo 87735, NASA Langley Research Center, August 1986.

Kurlack, R. and J. Chobot, "CPU Coverage Evaluation Using Automatic Fault Injection", 4th AIAA/IEEE Digital Avionic Systems Conference, pp. 294-300, 1981.

Lala, Jaynarayan H. and Basil T. Smith, III, Development and Evaluation of a Fault-Tolerant Multiprocessor (FTMP) Computer, Vol. II, FTMP Software, NASA CR 166072, NASA Langley Research Center, May 1983.

Marchall, P., "Updating Functional Fault Models for Microprocessors Internal Buses", 15th International Symposium of Fault Tolerant Computing, pp. 58-64, 1985.

Martensen, Anna L. and Salvatore J. Bavuso, "Tutorial and Hands-On Demonstration of a Fluent Interpreter for CARE III", NASA Technical Memo 4011, NASA Langley Research Center, November 1987.

McGough, John G., Digital Systems Validation Handbook Volume II, "Latent Faults", DOT/FAA/CT-88/10, 1988.

McGough, John G. and Fred L. Swern, Measurement of Fault Latency in a Digital Avionic Mini Processor, NASA CR 3462, Bendix Corporation, October 1981.

McGough, John G. and Fred L. Swern, Measurement of Fault Latency in a Digital Avionic Mini Processor, Part II, NASA CR 3651, NASA Langley Research Center, January 1983.

McGough, J. G., Swern, F., and S. Bavuso, "New Results in Fault Latency Modeling", Proceedings of the IEEE EASTCON Conference, pp. 299-306, August 1983.

Meissner, C. W., Jr., J. R. Dunham, and G. Crim, "NASA-LaRC Flight-Critical Digital Systems Technology Workshop", NASA Conference Publication 10028, NASA Langley Research Center, April 1989.

Migneault, Gerard E., "On the Diagnostic Emulation Technique and Its Use in the Airlab", NASA Technical Memo 4027, NASA Langley Research Center, October 1988.

MIL-HDBK-217E, "Reliability Prediction of Electronic Equipment", Distribution Statement A, DOD/AMSC, October 1986.

Nichols, W., et al., "Trends in Parts Susceptibility to Single Event Upset", IEEE Transactions Nuclear Science, vol. NS-32, no. 6, December 1985.

Padilla, Peter A., "In-Circuit Fault Injector User's Guide", NASA Technical Memo 100478, NASA Langley Research Center, June 1987.

Padilla, Peter A., "FTMP Data Acquisition Environment", NASA Technical Memo 100636, NASA Langley Research Center, July 1988.

Padilla, Peter A., "Fault Recovery Characteristics of the Fault Tolerant Multi-Processor", NASA Memo L-16630, NASA Langley Research Center, to be published as a NASA report 1990.

Rasch, N., User's Manual for AC-20-53A Protection of Airplane Fuel Systems Against Fuel Vapor Ignition Due to Lightning, DOT/FAA/CT-88/3, FAA Technical Center, October 1984.

Rothman, Elizabeth, et al., HARP: The Hybrid Automated Reliability Predictor, Introduction and Guide for Users, Version 6, Department of Computer Science, Duke University, NASA Langley Research Center, April 1989.

SAE Committee AE4L, Recommended Draft Advisory Circular: Protection of Aircraft Electrical/Electronic Systems Against the Indirect Effects of Lightning-Induced Transients, Report AE4L-87-3, Revision A, October, 1988.

Schuette, M., et al., "Experimental Evaluation of Two Concurrent Error Detection Schemes", 16th International Symposium of Fault Tolerant Computing, pp. 138-143, 1986.

Smith, T. Basil, III, and Jaynarayan H. Lala, Development and Evaluation of a Fault-Tolerant Multiprocessor (FTMP) Computer, Volume I, FTMP Principles of Operation, NASA CR 166071, NASA Langley Research Center, May 1983.

Strickland, Michael J. and Daniel L. Palumbo, Fault Tolerant System Performance Modeling, AIAA, Boeing, 1988.

Trivedi, Kishor S. and Robert M. Geist, A Tutorial on the CARE III Approach to Reliability Modeling, NASA CR 3488, Duke University, Prepared for NASA Langley Research Center, 1981.

NASA Conference Publication 2114, Working Group Meeting I, "Validation Methods for Fault-Tolerant Avionics and Control Systems", NASA Langley Research Center, March 1979.

NASA Conference Publication 2114, Working Group Meeting II, "Validation Methods Research for Fault-Tolerant Avionics and Control Systems", NASA Langley Research Center, December 1979.

Yang, X., W. York, and D. Siewiorek, "Fault Recovery of Triplicated Software on the Intel iAPX 432", Distributed Computing Systems, pp. 438-443, 1985.

GLOSSARY OF TERMS

BYZANTINE RESILIENCE. A fault tolerant process which is tolerant of intermittent faults that can send good information part of the time.

COVERAGE. The conditional probability of the system successfully recovering from a component fault and continuing to perform the intended functions correctly, given the presence of the fault. Coverage is the measure of effectiveness of a system's utilization of redundant hardware. Coverage can be qualified and applied to many different components of a system and phases of recovery process. Examples include, fault detection coverage, fault isolation coverage, latent fault coverage, sensor failure coverage, and memory failure coverage.

ELECTROMIGRATION. Drifting of metal atoms toward the cathode of a cathode ray tube.

FAILURE. The deviation of system behavior from specifications (arithmetic failure, storage failure, flight control function failure.)

FAILURE, HARD. Repeated use of the same input and initial conditions results in the same incorrect response.

FAILURE MECHANISM. Any situation that could produce an error condition. Examples of failure mechanisms include metal migration, voltage overstress, and lack of air-conditioning.

FAILURE, PERMANENT. Repeated use of the same input and initial conditions results in the same incorrect response.

FAILURE, SOFT. Repeated use of the same input and initial conditions does not result in the same incorrect response.

FAILURE, TEMPORARY. Repeated use of the same input and initial conditions does not result in the same incorrect response.

FAILURE, TRANSIENT. Repeated use of the same input and initial conditions does not result in the same incorrect response.

FAULT. The phenomenological reason for a failure (open wire, stuck-at fault, design fault, etc.). In general, any condition preventing a digital component from correctly changing state when directed to change by input parameters. For electrical components there is a one-to-one correspondence between faults and failures. The situation is not so simple with digital circuits. For if the circuit is S-A-1, any input causing a one output will be correctly processed; a little like the stopped clock that is correct twice per day. For a processor

having a million or so logic gates, it is not possible to test for all the combinations of input and output states.

FAULT, LATENT. A fault which has not yet caused a failure. (For example, a fault in a memory chip that is not being used for the foreground program or in this particular mode of the system is a latent fault.)

FAULT, STUCK-AT. A logic signal which remains at zero (S-A-0) or one (S-A-1).

FAULT TOLERANT SYSTEM. A system that continues to function although certain components may have faults.

SYSTEM RELIABILITY. The probability of performing a given function from the some initial time, t=0, to time t.

ACRONYMS AND ABBREVIATIONS

AC	Advisory Circular
AE4L	SAE Subcommittee (Lightning)
AIAA	American Institute of Aeronautics and Astronautics
AIRLAB	Avionics Integration Research Laboratory
ALU	Arithmetic Logic Unit
AM	Amplitude Modulated
AMSC	Document number prefix used by the Department of Defense
ASEE	American Society of Electrical Engineers
ASME	American Society of Mechanical Engineers
ATTR	Attribute
B-GLOSS	Gate Logic Software Simulator developed by Bendix
C/I	Communicator Interstage
CAD	Computer Aided Design
CAP	Collins Application Processor
CARE III	Computer Aided Reliability Evaluator
CBD	Commerce Business Daily
CMOS	Complementary Metal-Oxide Semiconductor
CPA	Central Processor - A
CPU	Central Processing Unit
CR	Contractor Report
CSDL	Charles Stark Draper Laboratory
CTA	Collins Test Adaptor
DE	Diagnostic Emulation
DEFN	Definition
DEV	Development
DMA	Direct Memory Access
DOD	Department of Defence
EEC	Electronic Engine Control
EM	Electromagnetic
EMI	Electromagnetic Interference
EMR	Electromagnetic Radiation
ESS	Electronic Switching System
EXCHNG	Exchange
EXP	Experiment
FAA	Federal Aviation Administration
FCC	Flight Control Computer
FCR	Fault Containment Regions
FCS	Flight Control System
FET	Field Effect Transistor
FI	Fault Insertion circuitry
FIAT	Fault Injection Automated Testing
FIIS	Fault Insertion and Instrumentation System
FIM	Fault Injection Manager
FIRE	Fault Injection Receptor
FM	Frequency Modulated

FMECA	Failure Modes and Effects Criticality Analysis
FT	Fault Tolerant
FTC	Fault Tree Compiler
FTMP	Fault Tolerant Multiprocessor
FTP	Fault Tolerant Processor
GE	General Electric
GEN	Generation
GGLOSS	Generalized Gate-Level Logic System Simulator
GLOSS	Gate Logic Software Simulator
GPC	General Purpose Computer
HARP	Hybrid Automated Reliability Predictor
HDBK	Handbook
HERF	High Energy Radio Frequency
I/O	Input/Output
IBM	International Business Machines
ICIS	Intercomputer Interface Sequencer
ICS	Intercomputer Sequencer
ID	Identification
IEEE	Institute of Electrical and Electronics Engineers
IGGLOSS	Gate Logic Software Simulator (improved version developed at NASA Langley)
IOPA	Input/Output Processor - Channel A
IOS	Input/Output Subsystem
JPL	Jet Propulsion Laboratories
KHz	KiloHertz
LAN	Local Area Network
LaRC	Langley Research Center
LRU	Line Replaceable Unit
LSI	Large Scale Integration
M	Mutual (when used with RLC)
MDICU	Modular Digital Interface Conversion Unit
MHz	MegaHertz
MIL	Military
MOS	Metal-Oxide Semiconductor
MPX	Multiplex
MSI	Medium Scale Integration
Mux	Multiplexed
Naecon	National Avionics and Electronics Conference
NAND	Not AND
NASA	National Aeronautics and Space Administration
NEMP	Nuclear Electromagnetic Pulse
OS	Operating System
PAL	Programmable Array Logic
PAWS	Padé Approximation With Scaling
PC	Personal Computer
PMS	Physical Message Switch
PROC	Processor
QUAD	Quadruple
RAM	Random Access Memory
RCVR	Receiver
RDFCS	Reconfigurable Digital Flight Control System
REG	Register

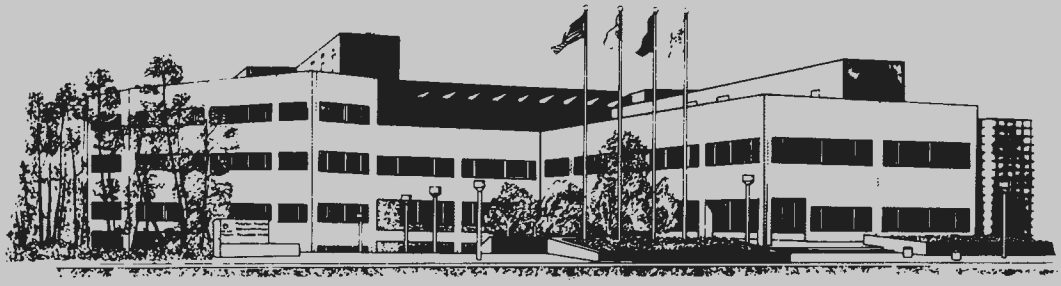
RF	Radio Frequency
RLC	Resistance/Inductance/Capacitance
ROM	Read Only Memory
RT	Remote Terminal
RTI	Remote Terminal Interface
S-A-0	Stuck-At-Zero
S-A-1	Stuck-At-One
S-GLOSS	Gate Logic Software Simulator developed by Stevens Institute
SAE	Society of Automotive Engineers
SEU	Single Event Upset
SHRD	Shared
SQL	Software Query Language
SSI	Small Scale Integration
STEM	Scaled Taylor Expansion Matrix
SURE	Semi-Markov Unreliability Range Evaluator
SYN	Synch
TTL	Transistor-Transistor Logic
TX	Transmit
UNIBUS	Universal Bus
USEG	Unsegmented
VLSI	Very Large Scale Integration
XAB	Transmit Compare A B
XMT	Transmit
XOR	Exclusive OR

U.S. Department of Transportation
Federal Aviation Administration

DOT/FAA/CT-88/10

HANDBOOK-VOLUME II
DIGITAL SYSTEMS VALIDATION

CHAPTER 6
DIGITAL DATA BUSES FOR AVIATION APPLICATIONS



PREPARED BY:

COMPUTER RESOURCE MANAGEMENT, INC.
950 HERNDON PARKWAY, SUITE 360
HERNDON, VIRGINIA 22070

PREPARED FOR:

FEDERAL AVIATION ADMINISTRATION
TECHNICAL CENTER
ATLANTIC CITY INTERNATIONAL AIRPORT, NEW JERSEY 08405

NOTICE

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The United States Government assumes no liability for the contents or use thereof.

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the objective of this report.

PREFACE

Digital data buses are used extensively in the current generation of civil aircraft. These buses are used in flight control and avionics applications to transfer data and to perform complex calculations. At the present time, the Federal Aviation Administration (FAA) has no published criteria or procedures for evaluating these complex systems, other than the general characteristics described by Federal Acquisition Regulation (FAR) 25.671, FAR 25.901, FAR 25.1309, and special conditions established by each unique aircraft type certification board.

This gap is being filled in two ways. First, standards have been developed for each major type of data bus, with the consequence that evaluation of a candidate data bus architecture can take place by comparing the "as-built" functional characteristics and system operation against the requirements published in the standards. Second, data collection, a necessary part of implementing digital data buses in new and derivative aircraft, has been an ongoing activity for the last 3 to 5 years. Information, data, and "lessons learned" from ongoing inspection, analysis, and documentation activities associated with certification testing for the Boeing 757/767 and other aircraft which use Aeronautical Radio, Incorporated (ARINC) 429, Military Standard (MIL-STD) 1553B, and General Aviation Manufacturers Association (GAMA) Avionics Standard Communication Bus (ASCB) data buses are providing guidelines and criteria for evaluation and certification of the next generation of aircraft.

This body of standards and experience is the topic of this tutorial. The first two sections ("Introduction" and "Data Bus Architecture and Topology") provide an overview of major characteristics of data buses in general. These sections include necessary background information for understanding data buses and why they are used. Descriptions of major characteristics applicable to all data buses are also included.

The third section ("Current Aviation Buses"), summarizes current and accepted standards and requirements for five of the most commonly used aviation-oriented data buses. This section is organized to enable efficient access to specifications required to evaluate documentation presented for data bus certification. The final two sections of the tutorial ("Bus Performance Considerations" and "Fiber Optic Data Bus for Avionics Integration") address major performance requirements, applicable to all data buses, which are not currently covered by the standards. These performance requirements cover data latency, failure modes, and factors related to the use of fiber optics.

TABLE OF CONTENTS

Section		Page
1.	INTRODUCTION	6-1
	1.1. Background	6-1
	1.2. Bus Requirements	6-4
2.	DATA BUS ARCHITECTURE AND TOPOLOGY	6-7
	2.1. Introduction	6-7
	2.2. The Individual Component	6-7
	2.3. Single Bus Architectures	6-9
	2.4. The System Level	6-11
	2.5. The Choice of an Architecture or Topology	6-18
	2.6. Protocol/Topology Relationships	6-18
3.	CURRENT AVIATION BUSES	6-23
	3.1. Introduction	6-23
	3.2. ARINC 429	6-23
	3.2.1. History	6-24
	3.2.2. ARINC 429 General Characteristics	6-24
	3.2.3. ARINC 429 Protocol: Command Response	6-25
	3.2.4. ARINC 429 Hardware Characteristics	6-26
	3.2.5. Major Documents	6-26
	3.3. MIL-STD-1553B and MIL-STD-1773	6-26
	3.3.1. History	6-26
	3.3.2. MIL-STD-1553B General Characteristics	6-27
	3.3.3. MIL-STD-1553B Protocol: Time Division Multiplex, Command/Response	6-29
	3.3.4. MIL-STD-1553B Hardware Characteristics	6-30
	3.3.5. MIL-STD-1773 Hardware Characteristics	6-30
	3.3.6. Major Documents	6-32
	3.4. The GAMA ASCB	6-33
	3.4.1. History	6-33
	3.4.2. GAMA ASCB General Characteristics	6-33
	3.4.3. GAMA ASCB Protocol: HDLC	6-34
	3.4.4. GAMA ASCB Hardware Characteristics	6-34
	3.4.5. Major Documents	6-34
	3.5. DATAC (ARINC Specification 629) Bus	6-35

TABLE OF CONTENTS (Continued)

Section		Page
	3.5.1. History	6-35
	3.5.2. DATAC General Characteristics	6-36
	3.5.3. DATAC Protocol: Collision Detection	6-37
	3.5.4. DATAC Hardware Characteristics	6-38
	3.5.5. Major Documents	6-38
3.6.	SAE High-Speed Data Buses	6-39
	3.6.1. History	6-39
	3.6.2. SAE General Characteristics	6-40
	3.6.3. SAE Protocol: Token Passing	6-40
	3.6.4. SAE Hardware Characteristics	6-42
	3.6.5. Major Documents	6-45
4.	BUS PERFORMANCE CONSIDERATIONS	6-47
4.1.	Introduction	6-47
4.2.	Issues of Latency	6-47
	4.2.1. Types of Latency	6-47
	4.2.2. Data Latency	6-47
	4.2.3. System Delays	6-50
4.3.	Failure Modes and Effects	6-51
4.4.	Transmission Error	6-53
4.5.	Reliability for Flight Safety	6-59
5.	FIBER OPTIC DATA BUS FOR AVIONICS INTEGRATION	6-61
5.1.	Elements of Fiber Optic Data Bus	6-61
5.2.	Optical Cables	6-62
5.3.	Optical Path	6-62
5.4.	Splices and Connectors	6-62
5.5.	Optical Couplers	6-62
5.6.	Size	6-63
5.7.	Reflections	6-63
5.8.	Connectors	6-63
5.9.	Splicing	6-64
5.10.	Technology - Optical Bus Transmitters and Receivers	6-64
5.11.	Maximizing Bus Efficiency	6-64
5.12.	Transmission Losses	6-64
5.13.	Receiver Losses	6-65
5.14.	Transmitter Losses	6-65
5.15.	Optical T/R Power Margin	6-66
5.16.	Topology Analysis	6-66
5.17.	Fiber Optic Network Based Losses	6-68

TABLE OF CONTENTS (Continued)

Section	Page
APPENDIX	
A - MODULATION AND SIGNALING METHODS	6-69
BIBLIOGRAPHY	6-74
GLOSSARY	6-76
ACRONYMS AND ABBREVIATIONS	6-78

LIST OF ILLUSTRATIONS

Figure		Page
2.1-1	INDIVIDUAL SUBSYSTEM STRUCTURE	6-8
2.3-1	SINGLE BUS ARCHITECTURES	6-10
2.4-1	TYPICAL AVIONIC HIERARCHIAL AND PARALLEL ARCHITECTURES	6-12
2.4-2	PRELIMINARY ARCHITECTURE OVERVIEW	6-13
2.4-4	INTEGRATION CONCEPTS FOR FLIGHT CRITICAL SUBSYSTEMS	6-17
3.2-1	GENERIC ARINC BUS LAYOUT	6-24
3.3-1	GENERIC MIL-STD-1553B BUS LAYOUT	6-28
3.4-1	GENERIC ASCB BUS LAYOUT	6-33
3.5-1	GENERIC DATAC BUS LAYOUT	6-36
3.6-1	GENERIC SAE LINEAR BUS LAYOUT	6-39
3.6-2	GENERIC SAE RING BUS LAYOUT	6-39
4.2-1	DATA LATENCY ILLUSTRATION IN HIERARCHIAL ARCHITECTURE	6-49
4.3-1	SIMPLE FAULT TREE EXAMPLE	6-55
4.5-1	SINGLE CHANNEL-DUAL OUTPUT (BUSES A AND B) BUS CONTROL	6-59
A-1	AMPLITUDE MODULATION	6-70
A-2	PHASE MODULATION	6-70
A-3	WAVEFORMS GENERATED BY FIVE DIFFERENT ENCODING SCHEMES FOR A FIXED BINARY SIGNAL	6-73

LIST OF TABLES

Table		Page
1.1-1	SUMMARY OF BUS CHARACTERISTICS	6-3
2.4-1	COMPARISON OF HIERARCHICAL AND PARALLEL AVIONICS BUS ARCHITECTURES	6-15
2.5-1	MEASURES OF MERIT	6-19
2.5-2	ATTRIBUTES	6-20
2.6-1	ALTERNATIVE TOPOLOGIES AND PROTOCOLS	6-21
3.2-1	ARINC 429 HARDWARE CHARACTERISTICS	6-27
3.3-1	MIL-STD-1553B HARDWARE CHARACTERISTICS	6-31
3.3-2	MIL-STD-1773 HARDWARE CHARACTERISTICS	6-32
3.4-1	GAMA ASCB HARDWARE CHARACTERISTICS	6-35
3.5-1	BOEING DATAC BUS CHARACTERISTICS	6-38
3.6-1	SAE RING BUS HARDWARE CHARACTERISTICS	6-43
3.6-2	SAE LINEAR BUS HARDWARE CHARACTERISTICS	6-44
4.3-1	DIGITAL DATA BUS SYSTEM FAILURE MODES AND EFFECTS	6-54
4.4-1	FAILURE CLASSES [MIL-STD-1553B]	6-56
4.4-2	TYPICAL ERROR-CORRECTION TECHNIQUES [MIL-STD-1553B]	6-57
4.5-1	POTENTIAL FAILURES RESULTING IN LOSS OF BUS CONTROL SINGLE CHANNEL - DUAL OUTPUT (BUSES A AND B)	6-60
5.1-1	OPTICAL BUS TECHNOLOGY LIMITS	6-61
5.17-1	STAR-COUPLED NETWORK LOSSES	6-68
A-1	IMPORTANT PARAMETERS OF ENCODING TECHNIQUES	6-72

1. INTRODUCTION

1.1. Background

Advances in digital technology have made possible greater processing power and memory capacity in smaller and smaller packages (Schmitter and Baues, 1984). Reductions in size mean that this processing capability can be located where it is needed, thus supporting the distribution of intelligence throughout the system network. In the specific case of modern civil aircraft, distributed intelligence has meant increasing numbers of distinct computer-controlled avionics, flight control, propulsion, and structural systems, with corresponding explosive growth in the volume of data being processed throughout the system.

As data processing becomes increasingly distributed throughout the aircraft, the need to support integration of this data and to allow access to it, as required, by both central and remote processors becomes apparent. Microprocessors, sensors, and servomechanisms must be appropriately interconnected to support communication and to enable fault tolerant designs to be implemented. Integrating aircraft subsystems offers numerous advantages, including the following (McSharry, 1983):

- Reduced crew workload through such capabilities as autoland and autopilot.
- Enhanced aircraft performance and capability.
- Increased hardware efficiencies (and resulting decreases in cost) through reduction in hardware duplication, complexity, and function.
- Improved flight safety through features such as flight envelope limiters.

Maximum use of common data allows the development of functional capabilities otherwise not possible. In addition, a common interface enables improvements in individual subsystems and flight tasks to be made with minimal disturbance of higher-order information transfer functions. Enhanced modularity simplifies subsystem interconnection so that retrofitting can be accomplished without the cost and complexity associated with modifying the entire system. Maintenance also becomes easier since dynamic testing of installed equipment can take place through the data bus interface by means of standardized maintenance equipment (Kiernan and Sims, 1980).

Prior to the mid-1970s, discrete subsystems passed information back and forth through analog point-to-point wire bundles. Traditionally, there has been considerable independence in the design of these subsystems. Components such as sensors were provided separately for each subsystem. However, advanced aircraft designs often require that these systems have significant interaction and a common data source. The combination of the need to integrate these

systems by function and to avoid unnecessary duplication of hardware provides the impetus for developing integration techniques and supporting architectures, which both reduce the overall costs and increase performance.

The analog wire approach could not match the demands for distributed processing. It increased weight (due to the large quantity of wiring required) and economic costs, while decreasing rapid access to a database shared by all subsystems. The system designers had to develop an alternative approach: the data bus. A data bus is a system for transferring data between discrete pieces of equipment in the same complex.

Commercial and transport aircraft currently employ only single-level data bus architectures. These architectures use one of the following: (1) centralized control, (2) bidirectional information transfer system, or (3) direct-connect, unidirectional information transfer system. Microprocessor-based flight control and avionics systems (as represented by the Boeing 757/767, the Lockheed L1011-500, and the Airbus A310/A320) use data bus architectures based on either the ARINC 429-5 or the MIL-STD-1553A/B specification and standards. These architectures use shielded, twisted pair wires for the transmission medium. They interconnect microprocessors that primarily use bit slice processors. The microprocessors provide the required internal processing speed (7-14 Megahertz (MHz) clock rate) and the inherent reliability, stability, and flexibility required for flight essential and flight critical control systems. In this generation of digital systems, the individual processors are run in a bit or frame synchronized manner. The data are exchanged between redundant computers via dedicated serial data buses (either wire or fiber optic). Internally, the data are exchanged by high-speed dedicated transfer buses/backplanes.

The extensive use of existing bus structures has proven that multiplexed data transfer systems can achieve a degree of integration. The present bus characteristics are ideally matched to many intraavionics subsystem data transfer requirements. These requirements include sensor data collection, central processing, and distribution of results to peripheral areas. Continued use of bus networks for the intrasubsystem data transfer will be required.

Unfortunately, current protocols and architectures may not provide the characteristics needed to support future overall system integration requirements. The next generation aircraft will have multiple information transfer systems which require interchange of data. These systems will communicate with one another through global memory storage interface units. Consequently, such aircraft will require total airframe/system integration (on a full-time/full authority basis). With subsystems integrated in this manner, a negative change in one could possibly result in erroneous data and information being propagated throughout the entire system.

A solution to this potential problem is the development and use of an information transfer system which will efficiently interconnect, in a hierarchical order, multilevel, multiplexed buses and bus architectures. With such an approach, software-intensive, fault-tolerant executive and operating systems can be created. These operating systems provide the processing of functions required of multisystem inputs using the local terminals. A high-speed, higher-order transfer system will probably employ contention or token-passing

protocols. These protocols provide each active unit (within the information transfer system structure) the capability of performing its own data processing autonomously. The responsibility for passing completed data onto the bus at the appropriate time belongs to each active unit.

Some of the more common subsystems are likely to be combined into logical units (components). New subsystems or groups of architecturally related functions will be implemented as common units. Each of the major systems will probably be designed as an integral unit. Each unit will have its own unique intra-multiplexed topological (bus) network. Each asynchronous information transfer function and topological network must be interconnected, using high bandwidth buses to create integrated data and management bases. Information flow can be directed and managed from these bases.

Data buses currently in use, such as ARINC 429, MIL-STD-1553B, and GAMA ASCB do not have the capability to handle such massive integration (see table 1.1-1). Consequently, new approaches must be developed for the interconnection of avionic subsystems to ensure the integrity of the data at all times. New bus standards are being developed to support higher-order data. The new standards use operational protocols that provide high-speed interconnection of subsystems and common sensors, sufficient subsystem independence, and fault tolerance. These protocols also distribute control of the common data bus at both the subsystem black box level and the aircraft/application level.

TABLE 1.1-1. SUMMARY OF BUS CHARACTERISTICS

	ARINC 429	MIL-STD-1553B	GAMA ASCB	(DATAC) ARINC 629	SAE Linear/ Ring
Maximum Bit Rate (Hz)	100K	1M	667K	1M-2M	10M-20M/ 50M
Bidirectional	No	Yes	Yes	Yes	Yes
Bus Controllers	No	Yes	Yes	No	No
Defined Data Formats	Yes	No	Yes	Yes	Yes
Low Cost Components	No	No	Yes	Yes No	No

The next generation of flight control and avionics systems architectures will be dramatically different from the current generation. It will include multiple microprocessors in each computing channel, more local processing within a processor, and the transfer of preprocessed data within the bus network. In addition, the system architectures will make use of 16/32 bit microprocessors,

which will use high-speed backplane buses (running at 20-50 MHz) for internal (processor-to-processor) interfaces and exchange of data and information. Furthermore, these processors and their fault-tolerant designs will make use of global memory and functional partitioning of executive and applications software to decrease the complexity and increase the reliability of the system.

Although such objectives have yet to be realized, two data bus standards, soon to be in commercial use, will allow incremental progression towards these goals: the Digital Autonomous Terminal Access Communication (DATAC) (ARINC 629) and the Society of Automotive Engineers (SAE) linear and ring bus architectures (see table 1.1-1). These buses will be able to support the integration of avionics, flight, and propulsion systems expected to be implemented digitally in current and future aircraft. This method of implementation will allow the necessary sharing of data between subsystems.

Maximizing data availability between subsystems is in conflict with the need to isolate these systems from propagation of failures from one system to another. Therefore, the integration solution must consider the balance between the need for and type of integration, versus the flight-safety and application-criticality of each subsystem. The solution must also consider various architectural implementations within the different aircraft configurations and applications.

1.2. Bus Requirements

The overall advantage of integrating flight-critical subsystems (flight and propulsion controls) with other avionics subsystems can be realized only if efficient, safe, and practical methods of subsystem communication can be implemented. Items to be considered in implementing subsystem communication include the following:

- Architecture topology.
- Bus interface design.
- Interaction with the host processor (controller) and data bus interface.
- Bus protocol.
- Hardware/software failure modes.
- Fault propagation potential.
- Protection mechanisms that prohibit fault introduction or allow detection and management of faults.

The trend in data bus evolution is to implement the following general characteristics:

- Real-time data integration - current data are essential to critical flight operations.

- Protection of subsystems from propagation of failures from other subsystems. There must be an appropriate balance between integration and isolation.

- Decentralized system control.

- Standard interfaces for all equipment connected to the bus.

- Continued reductions in the use of costly hardware/software elements required in centrally controlled, data transfer systems.

- Continued dispersion of microprocessors within subsystems necessitating the interchange of processed data between subsystems.

- Generation of an aircraft database, available to all subsystems, which includes all airframe/performance parameters.

- Maximal use of common sensor data and redundant data sources.

- Further standardization of hardware/software elements by the use of other standards for interchangeability between the avionic systems and aircraft.

This tutorial examines the characteristics of avionics buses in use today. It also reviews two bus concepts proposed for use in the near future. While none of these buses has all the qualities desired for the next generation, the newer network buses offer the greatest potential match with the technology anticipated for the next generation of airframes and avionics.

Data and information for avionics systems integration can be successfully transmitted using these existing or other proposed bus structures. However, each bus has limitations, which must be considered when assessing its suitability for use in a specific application.

Before addressing the specific performance characteristics of the five buses, the topic of bus architecture and topology needs to be considered to:

- Identify the influence of physical structure on data bus performance.

- Understand the impact of dispersed locations of applications processors on computational capability and system flexibility.

2. DATA BUS ARCHITECTURE AND TOPOLOGY

2.1. Introduction

Data bus architecture and topology refers to the physical layout of the bus, (i.e., the organization of individual components attached to the buses). There are three levels of organization:

- Individual component.

  The primary purpose of a data bus is to allow individual components to transmit and receive relevant data. Each component performs a function. During the time it performs the function, it may need to receive data from or send data to one or more other components. For example, a Flight Control Computer (FCC) requires sensor data as input, performs a set of functions or equations using that data, and then sends commands to actuators or other avionics systems. Each component, therefore, requires an interface compatible with the physical structure of the bus.

- Individual bus.

  An individual bus is a topological configuration that allows the exchange of information between two or more individual components. The individual bus can support a minimum of two components up to the maximum capacity authorized by the bus architecture or structure. A specific bus may integrate related subsystems, which perform similar tasks ("the navigation bus") or integrate all subsystems ("the system bus"). Consequently, the complexity of individual buses can vary.

- System level.

  When multiple buses are used, a system level is required to integrate the individual buses. The extent of this integration depends on the degree of overall system integration required and the demands of the overall architectural design. The system level bus ensures that data are distributed to all levels that need the data at the proper time.

Each of these levels is described in more detail below.

2.2. The Individual Component

The generic structure of the bus/box interface is shown in figure 2.1-1.

Bus isolation refers to the physical attachment of the component to the bus medium. Typically, this attachment is accomplished by either direct coupling or transformer coupling. Direct coupling is used with short lengths of wire (normally one foot or less) with the result that impedance mismatches are not

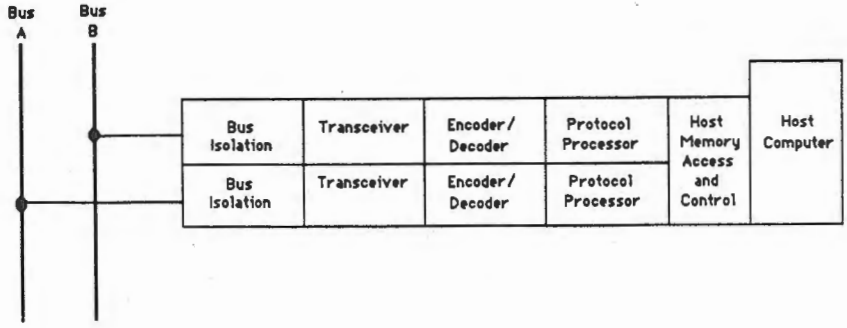


FIGURE 2.1-1. INDIVIDUAL SUBSYSTEM STRUCTURE

critical. Transformer coupling is used with longer lengths of wire (up to 20 feet) to achieve the specified signal-to-noise ratio and system error rate performance, and to maintain the proper impedance balance on both sides of the transformer over the longer lengths of connecting cable.

The transceiver is a combined transmitter and receiver. It is responsible for the interface between the digital data provided by the component and the analog format of the bus. The receiver element also has the task of monitoring the data sent along the bus until it detects "its name". Information intended for the component is detected and picked up by the receiver. The transmitter, in contrast, places information on the bus. Additional functions of the transceiver are to provide clock generation (ensuring that data are introduced to the bus at the correct time) and to evaluate received data for errors.

The encoder/decoder converts the digital data provided by the component into an analog format that is compatible with the bus. It also converts the analog data removed from the bus into a digital format. Since data buses support only serial data transfer while the processors expect parallel data, the encoder/decoder must provide serial/parallel data conversion.

The protocol processor adds or removes protocol information (such as error correcting codes) from transmitted messages. The processor is also responsible for checking the accuracy of the data coming from the bus. In evaluating data transmission protocols, the error detection and correction techniques, which could be used by the protocol processor, include Vertical Redundancy Check (VRC), Longitudinal Redundancy Check (LRC), and Cyclic Redundancy Check (CRC).

VRC appends one additional overhead bit (a "1" or a "0") to a data word to implement either odd or even parity. VRC does not detect double bit errors. LRC views a frame as a block of characters and appends an additional character consisting of the parity bit for each bit position in the character. Even when

used with VRC, some patterns of even number errors remain undetected. CRC generates a frame check sequence for a frame, which is exactly divisible by some predetermined number. The frame check sequence may be verified at both ends of the transmission. Only rare combinations of errors remain undetected with this system.

Forward error correction codes are used when the receiver alone corrects data errors. The codes are calculated and transmitted along with the data. For acceptable correction, data rates are reduced by at least 50 percent. Backward error correction (retransmission) is used to resend messages when the receiver signals the transmitter that an error occurred in the transmission.

In some cases, the protocol processor translates and implements the type of message (command, status, data) and the requirements for special actions. When a bus controller is not used, two protocols may be required. The first protocol is the data transmission protocol itself. The second gains access to the bus in order to transmit data.

Host memory access and control have the capability to interrupt the main processor as required, without waiting for the response cycle through a hard-wired Direct Memory Access (DMA) channel.

The host computer is the component that actually produces data. Its functions include computation, management, and instruction. Once the host computer has completed its functions, it may request data from the bus or send its own already-processed data to the bus.

2.3. Single Bus Architectures

To achieve orderly transmission and reception of data, rules must be established for individual components to gain access to the bus network. These rules depend on the basic topological structure of the bus network itself. To date, the following structures have been used:

- Point-to-point.
- Linear.
- Star.
- Ring.

Figure 2.3-1 (Schmitter and Bauss, 1984) shows three of the above structures. The point-to-point topology is not included in this figure. It is similar to the linear bus except that each component is individually connected (by one or more independent wires) to other components within the system. There is no competition in gaining access to the transmitting medium, because each component is physically linked by its own wire to every other component to which it transmits. On the other hand, if a hard-wire connection does not exist, the components cannot exchange data as they would in the case of a shared, common bus. With this concept, a single transmitter can broadcast to all receivers simultaneously as long as the receivers are connected to the same wire.

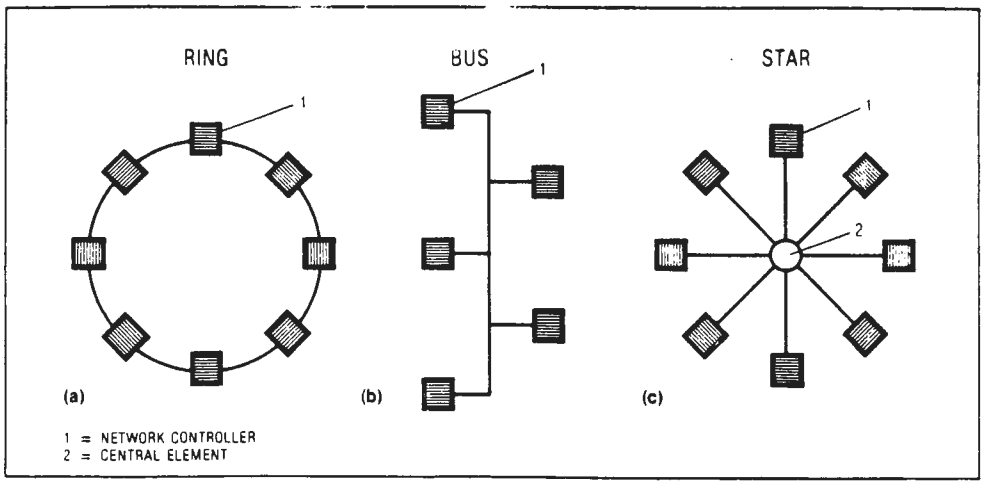


FIGURE 2.3-1. SINGLE BUS ARCHITECTURES
(from Schmitter and Baues, 1984)

With the linear bus concept, all components are attached to a common bus medium that contains multiplexed data operating in a half-duplex mode. A linear bus can be governed in either of two modes:

- By means of a centralized bus controller, which initiates information transfers to (from) specified components in a command (response) mode.
- Through decentralized operation using contention logic where a transmitter, which wishes to send data, waits until there is an opening on the bus.

The ring bus concept provides an easy method of joining together any number of stations, up to the specified limit of the addressing field, due to its use of simple point-to-point links.

With the ring bus, there are no restrictions with respect to the following:

- The minimum number of stations (above two).
- The physical or logical position of a station. (Note: the inherent token addressing dispenses with the need to have sequential addressing for speed.)

In addition, the point-to-point links allow the use of any medium defined in the specification - including the possible use of mixed media (i.e., wire and fiber optic).

The star bus topology always requires a central bus controller as the active element with peripheral processing accomplished by the other application processors (boxes) attached to the central element. Star topologies are

particularly suited for fiber-optic coupled systems as a result of the development of the star fiber optics couplers and the nature of the optics required.

2.4. The System Level

A single bus may integrate the processing of all the components comprising the vehicle system, or it may be used with components performing related functions. The former is an example of a parallel bus (see the right-hand side of figure 2.4-1). As an alternative, a hierarchical organization of multiple buses may be used (see the left-hand side of figure 2.4-1). Each set of components performing similar functions (e.g., navigation) is attached to the same bus. Integrated data produced from this bus is then passed on to a second bus, which brings together data from other low-level buses. By means of this hierarchical arrangement, progressive layers of buses, carrying ever higher levels of data, are formed. As an example, figure 2.4-2 provides a multilevel concept composed of four digital information transfer bus structures (Sensor, Management, Systems, Actuator) and one or more dedicated analog bus structures.

The sensor bus contains data that are time critical and necessary for critical system functions including:

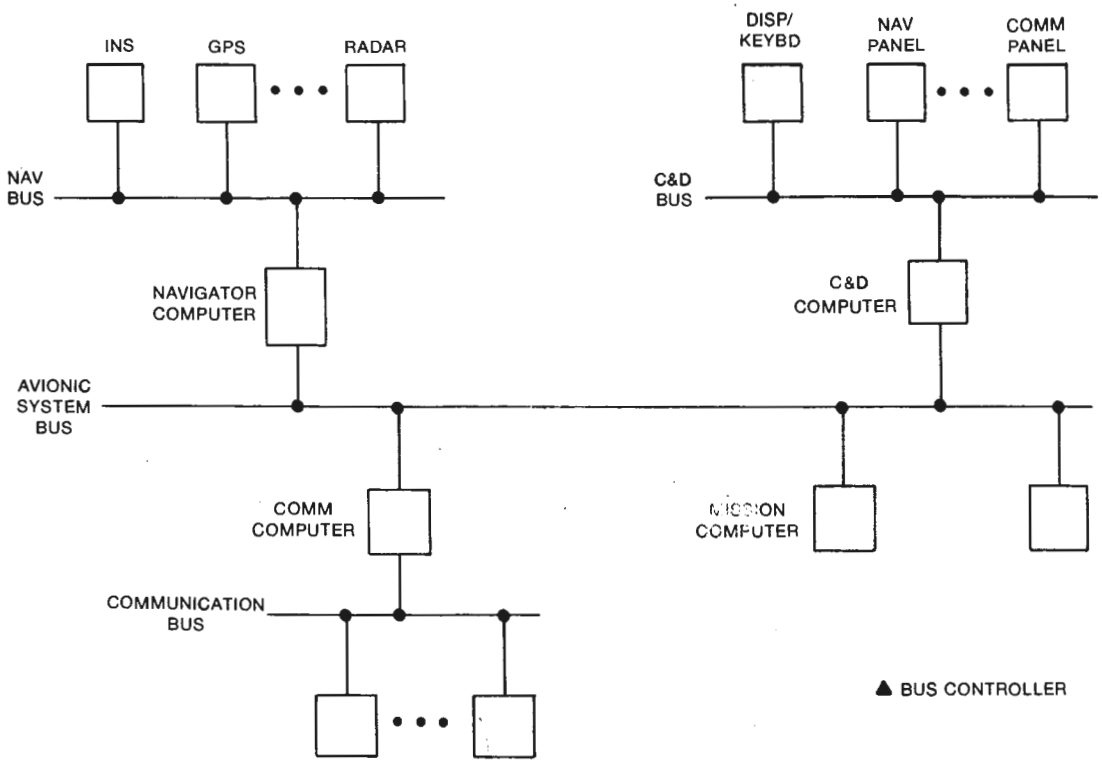
- Body accelerations and angular rates.
- Attitude angle and rates.
- Navigation and position (angles and deviations).
- Pilot inputs (column, wheel, throttle).
- Surface position (deflections and accelerations).

The data handled by the management bus are generally non-time-critical data that provide control information and system configuration. These data include the following:

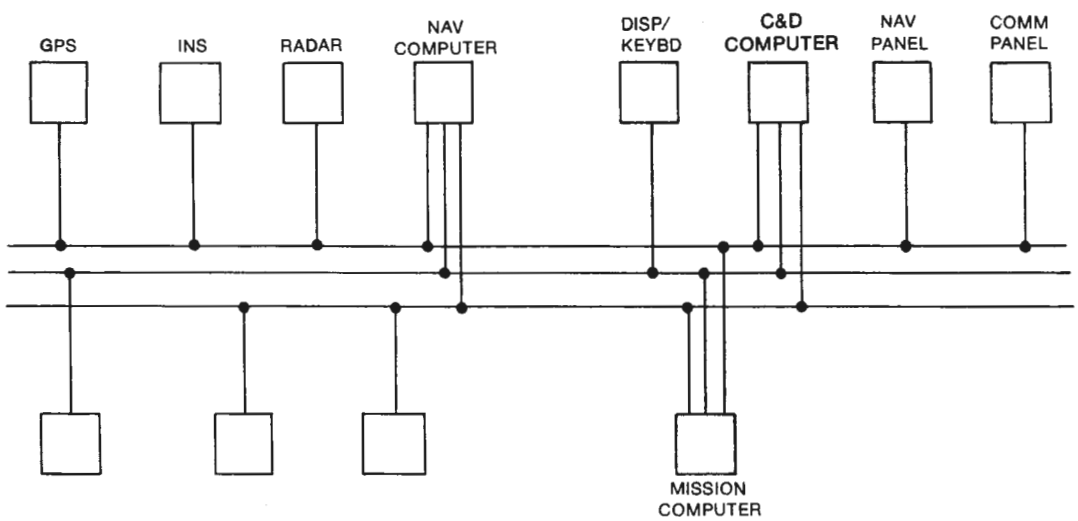
- Pilot selected parameters and modes.
- Initialization data.
- Reference angles.

The systems bus transfers time-critical data that are provided (by the aircraft avionics and flight controls systems) at a constant update rate to perform mission/flight-phase oriented and automatic functions. These data include the following:

- Auto-throttle position and rates.
- Autoload (deviations, deflections, and commands).
- Attitude reference/control.



Typical Avionic Hierarchical Architecture



Typical Avionic Parallel Architecture

FIGURE 2.4-1. TYPICAL AVIONIC HIERARCHIAL AND PARALLEL ARCHITECTURES

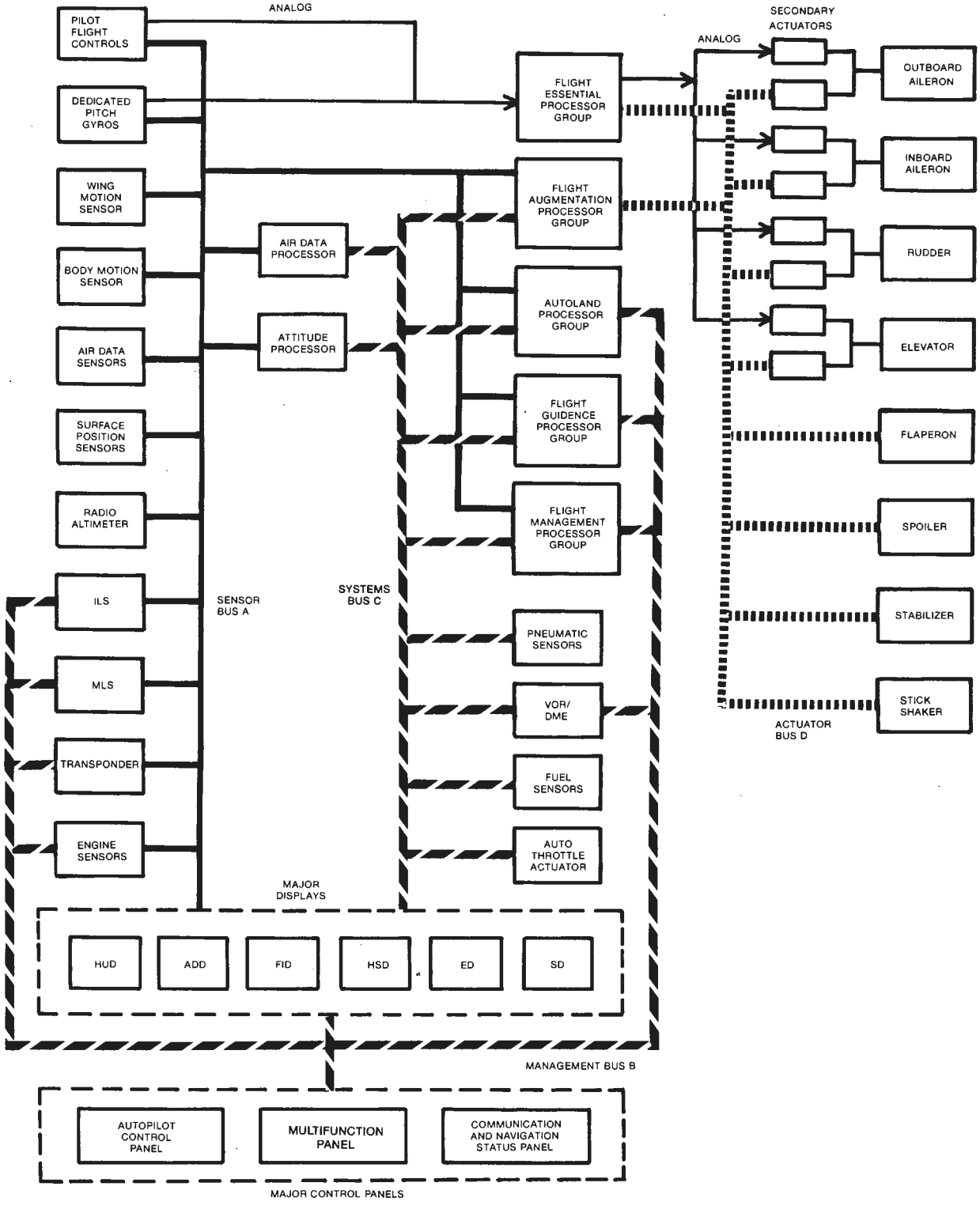


FIGURE 2.4-2. PRELIMINARY ARCHITECTURE OVERVIEW

- Flight management functions.
- Pneumatic (status/control).
- Fuel (flow/rate, quantities).

The actuator bus provides the necessary constant-update-rate data to command and feed control back to the surface controllers and tactile attitude warning devices. These data include the following:

- Deflection command/activator position (aileron, rudder, elevator, spoiler, stabilizer, etc.).
- Stability augmentation (gains/deflections).
- Stick shaker.

The analog (hard-wired) interconnections handle the flight-essential functions that include pitch rate sensors, pilot flight controls, and redundant actuators.

Early bus architectures used a parallel structure. Advances in digital technology make it possible to preprocess functions through distributed intelligence. Consequently, architectures more suitable to the sensors and actuators can be used. Onsite microprocessors can replace the central computer complex. As a result, the concept of hierarchically organized, multiple buses becomes attractive. One advantage of such an approach is that it enables easy differential treatment of critical and noncritical data simply by using different buses.

Both architectural arrangements are still in use. In general, hierarchical buses are more efficient. Data transmitted by hierarchical buses tends to receive more preprocessing than data transmitted via the parallel architecture. There is less chance of overloading system bus under hierarchical architectures. However, this also means that data tends to take longer to reach its final destination since it may have to pass through several buses.

Hierarchical buses support greater functional independence and isolation of individual subsystems. In contrast, parallel buses must carry more data but they offer a simpler architecture which allows greater reconfiguration capability.

Figure 2.4-3 (McSharry, 1983) provides two examples of the implementation of each architecture. In one case, integration is performed by means of a local bus, in the second by an avionics system bus. The avionics system bus sends out processed data while the local bus can transmit both processed and unprocessed data. The advantages and disadvantages of the hierarchical versus parallel approaches and the local bus versus system bus are summarized in table 2.4-1 (McSharry, 1983).

From the control system perspective, three integration alternatives are possible as shown in figure 2.4-4. All three cases make use of various combinations of

TABLE 2.4-1. COMPARISON OF HIERARCHICAL AND PARALLEL AVIONICS BUS ARCHITECTURES (adapted from McSharry, 1983)

Hierarchical Avionics Bus Architecture	Parallel Avionics Bus Architecture
• Local Bus	• Single Bus
Minimum data latency	Simpler
Lowest intersystem impact	Greater flexibility
Greater isolation	
• Avionic System Bus	• Multiple Buses
Information required at more than one local bus	Higher reliability levels
Highest inter-/intra-system impact	
Greater data latencies	

data bus structures (Sensor, Systems, Actuator, and Analog) previously referenced, and either use the FCC as a buffer between the avionics and flight control systems or connect directly to the control system bus(es) with other mission essential computers acting as buffers. In either case, the proposed architecture or topology and the attendant integration must be defined so that one of the following conditions exist.

- Isolation (in terms of fault propagation) is maximized by integration of functions and sensor signal requirements. This may be accomplished through the use of redundant avionics buses or buses dedicated to support avionics, flight control, and other mission dependent functions within the same bus structure. This approach, however, requires higher levels of system reliability to satisfy flight safety requirements.

- Data latency is minimized by use of separated structures in which critical sensor data co-exist with the flight control and mission dependent computation function on the same bus. It may also be minimized by making optional use of existing sensor redundancy with critical sensor data being

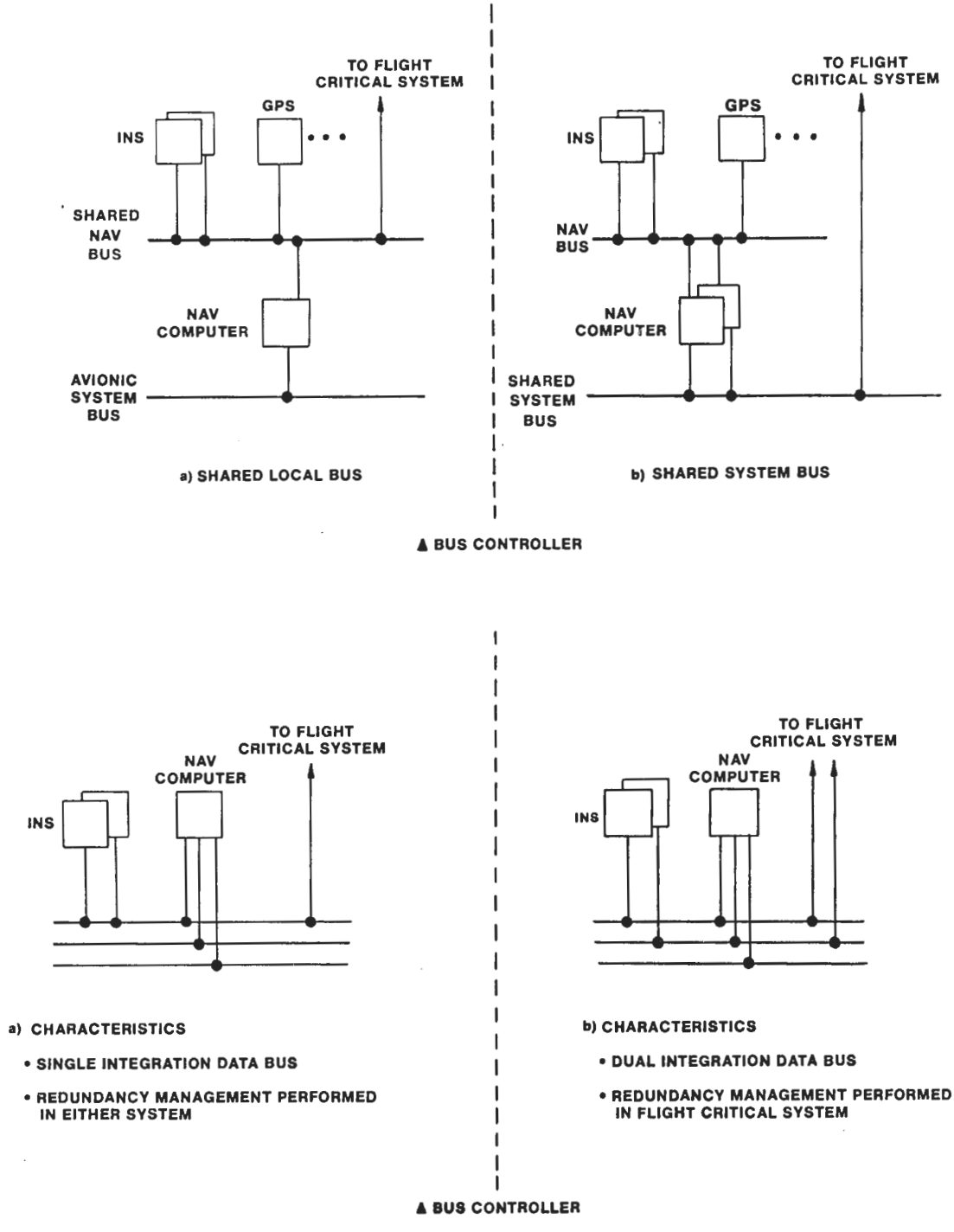


FIGURE 2.4-3. INTEGRATION CONCEPTS
(from McSharry 1983)

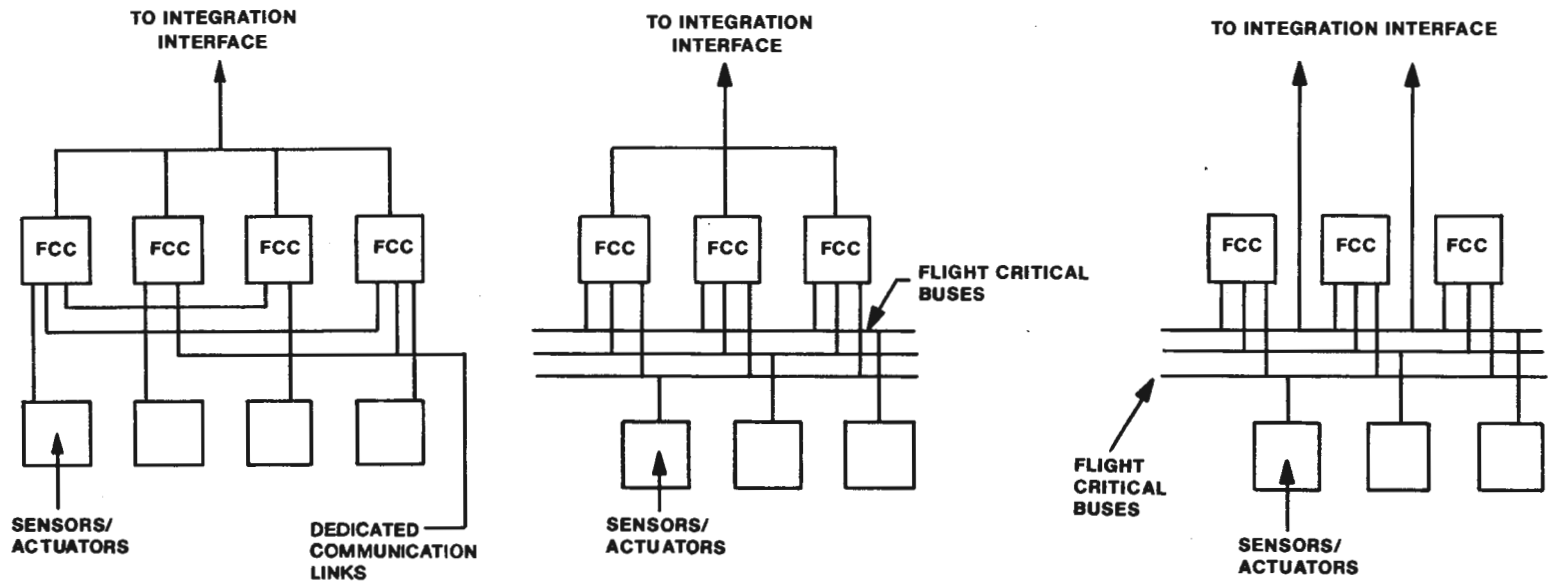


FIGURE 2.4-4. INTEGRATION CONCEPTS FOR FLIGHT CRITICAL SUBSYSTEMS

placed (through multiparty techniques) across the information transfer bus hierarchy. This approach reduces the reliability constraints on each of the various system functions. It can, however, introduce new potential failure points into the flight control and mission-dependent computation functions due to the increased complexity.

2.5. The Choice of an Architecture or Topology

The selection of a system architecture (including bus structure, topology, and integration concept) is based on the design requirements and the preference of the system designer, integrator, and implementor. In the concept design phase, a number of candidate architectural concepts, bus architectures, and topologies are postulated. Each is capable of satisfying system requirements within the constraints of the required performance, reliability, and safety criteria levels established by the relevant guidance documents (FARs, Advisory Circulars, and other accepted air worthiness practices). The selection of the final design for the information transfer system will ultimately be a function of selected system components, required interfaces, time-critical events and data, and various measure-of-merit attributes that drive the integrator's decisions.

The measures-of-merit and desirable bus attributes (as presented in tables 2.5-1 (McSharry, 1983) and 2.5-2 (Rich et al., 1983) are guidelines to be used by the system designer and integrator in assessing the integrity of the proposed information transfer system. Architecture, structure, protocol, and integration complexity must be considered to fully understand and evaluate the ultimate performance, reliability, safety, and airworthiness of the final design.

2.6. Protocol/Topology Relationships

The technical analyses leading to the selection of a topology and protocol for a given application require that topologies, protocols, media components, and configurations all be analyzed in terms of the constraints imposed upon the detailed system design, and the existing state of technology in each of these areas. Although a number of topologies and protocols currently exist, those most applicable to commercial transport are presented in table 2.6-1. As can be seen from this table, the choices for protocols are dependent upon the topologies, and the extent to which high-speed, distributed, fault-tolerant computation and passage of data are maximized in the system architecture. The topologies themselves are not likely to change in the near future, but the implementation, in terms of redundancy and hierarchical structure, will be designed to provide the critical data at the correct iteration rate with the least time delay at the lowest cost. As more and more aircraft systems are implemented with these concepts, the cost of integration (primarily Bus Interface Units (BIU)) will decrease and advances in electronics technology (at the silicon chip level) will allow additional variations which may not be feasible at the present time.

TABLE 2.5-1. MEASURES OF MERIT
(adapted from McSharry, 1983)

Measures-of-Merit	Quantitative Measure
Flight Safety - ability to maintain control of aircraft	Probability of loss of control
Mission/Flight Phase Reliability - ability to satisfy mission requirements	Probability of loss of mission/flight phase capability (i.e., Autoland, etc.)
Maintainability - time required to repair and frequency of repair	MTTR, MDT, MTBCF
Availability - ability to initiate a mission or flight phase activity/function including full-time, full-authority system (i.e., FADEC, PAS, Envelope Limiting)	Operational availability and inherent availability are usually presented as a percentage (i.e., 99%) or are given as an exposure time (i.e., two minutes).
Flexibility - ability to accommodate changes	Reconfiguration cost
Reconfigurability - ability to compute or perform mission or flight phase function in presence of failures	Dynamic reconfiguration time or redundancy default (fail safe/fail safe)
Computational Capability - throughput of system computers	Total instructions executed per second
Data Transfer Capability - ability to send messages in a timely manner and in presence of failures	Maximum data latency, % peak bus loading
Pilot Interface - ability to provide cognitive information to pilot	Qualitative
Cost - initial procurement and life cycle cost	Dollars

TABLE 2.5-2. ATTRIBUTES
(adapted from Rich et al., 1983)

Attribute	Quantitative Measure
Fault Tolerance	Probability of error occurring; Reconfiguration time; Probability of propagation
Efficiency	Available bandwidth
Simplicity	Presence/absence of complexity, and complexity metric rating
Data Integrity	Probability of correct data transfer, and number of retries
Synchronous/Asynchronous	Time to respond to emergency messages/interruption
Adaptable to new technology; Technology Insertion	Qualitative
Similarity to existing bus architectures/ structures/protocols	Qualitative
Deterministic	Qualitative

TABLE 2.6-1. ALTERNATIVE TOPOLOGIES AND PROTOCOLS

TOPOLOGIES	PROTOCOLS				
	COMMAND RESPONSE (ARINC 429-5)	CSMA/CD (DATAC ARINC 629)	TOKEN PASSING (SAE)	HDLC (ASCB)	TDM/CR (MIL-STD-1553B
Point-to-Point	X				
Linear Bus	X	X	X	X	X
Star					X
Ring			X		

3. CURRENT AVIATION BUSES

3.1. Introduction

The first two sections of this tutorial provided a brief overview of the history and likely evolution of aviation digital data buses. This section reviews the specifications for five current and near-future avionics data bus concepts. The three data buses currently in use are ARINC 429, MIL-STD-1553B, and GAMA ASCB.

The two near-future bus concepts are as follows:

- The proposed ARINC 629 (DATAC), which is expected to be certified in the 1990s on the 7J7 and other large commercial transport aircraft.

- The two proposed SAE (Avionics System Division) High-Speed Data Bus Architectures, which are expected to be the standard for civil and, perhaps, military aviation in the year 2000 and beyond.

These five bus concepts are the only buses expected to be certified by the FAA within the next 20 years. Each bus standard reflects an ongoing attempt to develop data buses that will enable greater integration and flexibility. The proliferation of diverse data buses is due to the different needs of the developers. For example, the ASCB bus was developed by GAMA to take advantage of the large number of avionics components being developed by the general aviation avionics industry. Their bus concept was based upon the availability of low-cost commercially-available components that could be easily integrated into the low cost equipment.

Each of the buses will be discussed in terms of the following:

- Major architectural and topological characteristics.

- Protocol used.

- Design specifications.

- Major documents describing the bus and its specifications which can be consulted for additional information.

A separate appendix (appendix A) has been included to explain and define the various modulations, signaling methods, and data code waveforms that are commonly used for these digital data buses.

3.2. ARINC 429

3.2.1. History

The ARINC 429 bus standard was the first standard developed for the transfer of digital data between avionics system elements in civil aircraft. Created during the mid to late 1970s by the air transport industry (the airlines, avionics manufacturers, airframe manufacturers, and the FAA) to meet the emerging need for distributed intelligence and communication between avionics subsystems, the ARINC bus quickly replaced analog point-to-point wiring systems which were prevalent at that time. Currently, the ARINC bus standard represents the type of bus used in most civil aircraft manufactured during the past decade. A schematic representing the basic organization of the ARINC bus is shown in figure 3.2-1.

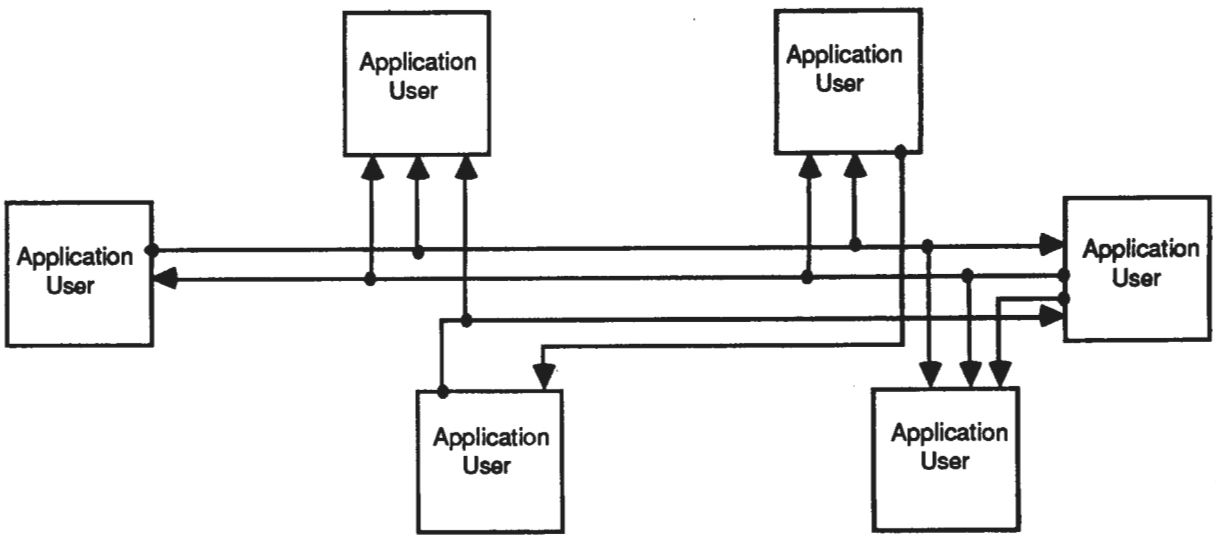


FIGURE 3.2-1. GENERIC ARINC BUS LAYOUT

3.2.2. ARINC 429 General Characteristics

General characteristics of the ARINC bus include the following:

- Relatively low-speed, high-reliability data transfer.
- Two data rate options (Note: High and low rate messages cannot be combined on the same bus):

  High-speed (100K bits per second (bps)) - Low-speed (12K to 14.5K bps).

- Unidirectional, asynchronous, serial data transfer.

- Data transfer by means of a single twisted, shielded pair of wires.

- Simplex data transfer only:

  Each component that needs to talk to other components has a wire going to the other components from its transmitter.

  Each component that needs to receive data from other components has a wire from that component to its receiver.

- Transmission by "broadcast" mode: receivers are not required to respond that data has been received (although they frequently do).

- Each transmitter able to send to up to 20 receivers.

- Data encoded in binary or binary-coded decimal.

- Data words composed of 32 bits, including label, word type, and a parity bit.

- Transmission accuracy determined by a single parity bit. Data reasonableness checks may also be performed.

3.2.3. ARINC 429 Protocol: Command Response

The ARINC 429 bus uses a Command Response (CR) protocol. When a transmitter is ready to send data to a receiver, it first sends a "request to send" word to the intended receiver by means of the bus that connects the transmitter to the receiver. If the receiver is ready, it will respond with a "clear to send" word sent by means of a separate bus that connects the receiver to the transmitter. The data then are sent immediately after transmission of a "data follows" initial word. A "final word", used for error control, signals completion of data transmission.

Once the data have been received, the receiver will process the data in search of errors, such as parity and file size. If no errors are found, the receiver may send a "data received OK" word to the transmitter. If an error is detected, the receiver requests the transmitter to retransmit the record containing the error.

Should the receiver not be ready to accept data, it sends back a modified "clear to send" that states its lack of readiness and may also specify the maximum number of records it is capable of receiving. After 200 millisecond (ms), a "request to send" is again transmitted. If, however, no response is received from the receiver, the transmitter will repeat a "request to send" after 50 ms. If the receiver fails to respond after four attempts, an alert is raised in the system containing the transmitter.

A single message, consisting of an initial word type, such as "data follows", and any data, is defined as a "record". Up to 126 data words may be sent in a single record. A single file may contain up to 127 records. Each record should contain one of the eight initial word types:

- Request to send.
- Clear to send.
- Data follows.
- Data received OK.
- Data received not OK.
- Synchronization lost.
- Header information (used to exchange file size information without actual transmission of file data).
- Poll ("I have no information for you, do you have anything for me?")

Synchronization occurs by means of a minimum of four bit-times which occur prior to each new word. Specific descriptions of these types of words, and the kind of information assigned to each bit, can be found in the ARINC 429 specification ("Mark 33 DITS").

3.2.4. ARINC 429 Hardware Characteristics

Table 3.2-1 presents a summary of hardware characteristics of the ARINC 429 data bus.

3.2.5. Major Documents

The characteristics of the ARINC 429 bus are described in ARINC Specification 429-5 ("Mark 33 DITS", April 1981).

3.3. MIL-STD-1553B and MIL-STD-1773

3.3.1. History

MIL-STD-1553B was originally developed by the military to support the integration of weapon systems. It has subsequently been used in commercial aviation for a number of systems including the Coast Guard Search and Rescue Helicopter HH-65A. Developed at the same time as ARINC 429, it represents the current generation of data buses developed for military and civilian aircraft. A schematic of the generic bus is shown in figure 3.3-1.

The proposed DOD-STANDARD 1773 DATA BUS (Fiber Optic Mechanization of an Aircraft Internal Time Division Command/Response Multiplex Data Bus) was developed for the following purposes:

- It seeks to preserve the multiplex bus techniques which have been standardized in MIL-STD-1553B, and
- It provides guidelines for the application of fiber optic transmission techniques to the MIL-STD-1553B interconnect.

TABLE 3.2-1. ARINC 429 HARDWARE CHARACTERISTICS

Transmission Medium	Twisted Shielded Pair
Characteristic Impedance	75 ± 5 Ohms
Main Bus Length	Not Specified
Media Connections	Direct Coupled
Modulation	Baseband (TDM)
Signaling Method	RZ Bipolar
Transmission Direction	Uni-Directional
Transmission Method	Asynchronous Broadcast
Transmission Order	LSB First
Data Rate	12-14.5 KHz or 100 KHz
Data Code	RZ Bipolar
Bit Error Rate	Not Specified
Word Error Rate	Not Specified
Topology	Serial Bus
Number of Terminals/Addresses	Less Than 20
Logical Addresses	Not Specified
Media Access	Point-to-Point
Data Link Control Protocol	Not Applicable
Error Detection	Odd Parity
Synchronization	Word
Word Size	32 Bits
Data Bits/Word	19 Bits
Words/Message (Min.-Max.)	1
Word Types	Not Specified
Intermessage Gap Time	4 Bit Times
Bus Frame Length	Not Specified
Bus Control Transfer Time	Not Applicable
Terminal Transmit Interface	Not Specified
Terminal Receive Interface	Less Than 20

3.3.2. MIL-STD-1553B General Characteristics

General characteristics of the MIL-STD-1553B bus include the following:

- Relatively high-speed (1 MHz bit rate), high-reliability data transfer.
- Single (redundant) pathway to connect each bus element (component) to the bus controller terminal. All equipment connected to the bus has access to information sent by means of the bus.

- Data transfer by means of a single twisted, shielded pair of wires.
- As many as 31 terminals per data bus.
- Bidirectional (half-duplex), asynchronous, serial data transfer.
- Data flow controlled by a bus controller. The controller may be a separate component or share a box with other functions.
- Redundant controllers optional (although only one is active at a time). The second controller is activated if the first controller malfunctions.
- Overhead penalty for multiple terminal and controller checking, testing, and switching functions.
- Command-response protocol.
- Data word size 20 bits. Blocks of up to 32 data words are allowed.
- Transmission accuracy determined by a single parity bit.
- Synchronization by a three-bit synchronizing field.

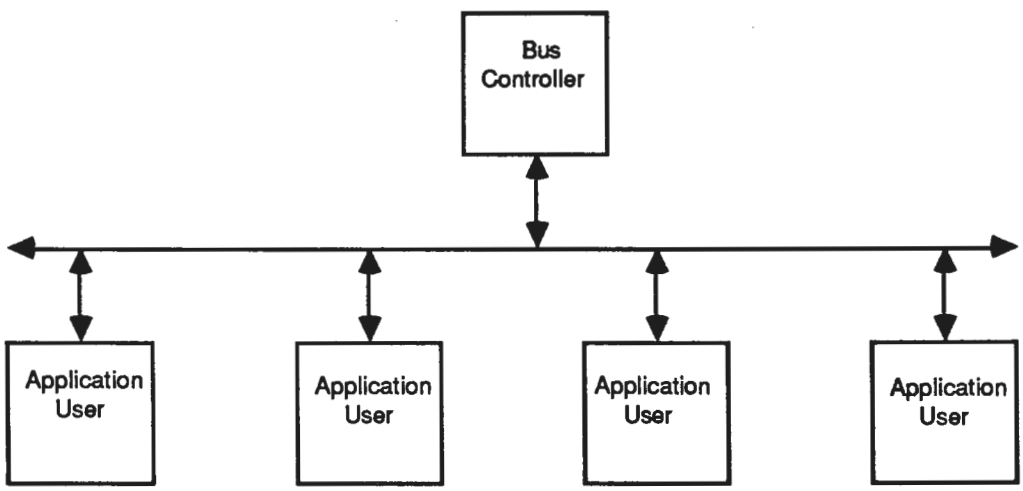


FIGURE 3.3-1. GENERIC MIL-STD-1553B BUS LAYOUT

3.3.3. MIL-STD-1553B Protocol: Time Division Multiplex, Command/Response

As the name suggests, data flow control for all transmissions on this bus uses a combination of Time Division Multiplexing (TDM) and command/response modes of operation. TDM refers to "the transmission of information from several signal sources through one communication system with different signal samples staggered in time to form a composite pulse train." (MIL-STD-1553 Designer's Guide, 1983, p. II-2.) With TDM, each component is assigned a pre-allocated timeslot. Consequently, an individual component knows when to transmit its data, how long it can transmit, and when to give up control of the bus to the next scheduled transmitter.

Pre-allocation of transmission slots requires that all message sequences be predefined to enable fixed schedules of data transfer. This is based on the definition of minor and major cycles. Minor and major cycles reflect the fastest and slowest iteration rates, respectively. The slowest iteration rate, which is the data required to be sampled least often, defines the time period for the major cycle. Over the course of a major cycle, all periodic bus events occur at least once and all periodic computations occur at least once. A frame is one major cycle. Within a frame, one or more minor cycles may occur as a function of the required frequency of the most rapidly transmitted periodic data.

The Multiplex Applications Handbook (1982, p. 5-4) offers the following example: "If the major frame is one second long and there are 64 minor cycles, then each minor cycle is 1/64 second or 15.625 ms long. Each periodic message would occur at least once each major frame, up to a maximum of 64 times. If a transaction needed to occur eight times per second, it must occur during one of the first eight minor cycles (64/8 = 8) and every eight minor cycles thereafter. The minor cycle in which the first message occurs is known as the 'phase', while the repetition rate is its 'period'."

Pre-allocation of timeslots means that responsibility for controlling the bus is distributed among all participating components. Since transfer of bus control can be almost instantaneous, little bus transmission time is lost. Consequently, bus utilization can, at least in theory, approach 100 percent (Rich et al., 1983). Practically speaking, this is not the case. Timeslotting is highly fault tolerant in that failure of the component currently controlling the bus does not cause a permanent disruption. Instead, the other components continue transmitting their data at their assigned slots. The only consequence is that the slot belonging to the failed component goes unfilled.

The timeslot protocol typically does not support 100 percent utilization because of problems with fault tolerance with respect to when individual message errors occur. Complete pre-allocation of slots does not allow for retransmission of messages that were not successfully transmitted initially - no slots are available for retries.

One approach for introducing message-level fault tolerance is to use timeslots that are larger than required for single-try message transmission. This leaves room for retransmission of a part of the message, if required. Clearly, this

reduces the amount of message traffic supported by the bus. Providing sufficient slot time for a single retransmission of a message has the major consequence of reducing overall bus capacity by 50 percent.

Note that the message retry capability represents an "unscheduled" data transmission event. Other types of unscheduled transmission can take place during those intervals when data are not already scheduled to be placed on the bus. The bus controller can command a receiver to transmit during these intervals as required. This is an example of how the command/response mode can be superimposed on the basic TDM mode. Using the command/response mode in conjunction with TDM has two advantages: intervals when no data are scheduled to be transmitted can be filled so as to make greater use of the bus's capacity, and the controller can respond to events that do not occur regularly.

3.3.4. MIL-STD-1553B Hardware Characteristics

Table 3.3-1 presents the characteristics of the MIL-STD-1553B bus.

3.3.5. MIL-STD-1773 Hardware Characteristics

Table 3.3-2 presents the characteristics of the MIL-STD-1773 bus, which is the fiber optic counterpart of MIL-STD-1553B. MIL-STD-1773 allows for five possible coupled architectures: reflective star, transmissive star, bidirectional T, unidirectional T, and bidirectional hybrid. The star coupler may be passive or active and can be embedded within the Line Replaceable Unit (LRU) or external to the LRU. Dual speed operation of the MIL-STD-1773 data bus is being examined by a number of vendors to make better use of the bandwidth possible in the bus.

Due to the need for compatibility with MIL-STD-1553B, the MIL-STD-1773 must operate in the time domain and use Manchester II encoding. Matching the Manchester II encoding scheme of MIL-STD-1553B with a fiber optic system results in the average optical power level during each sync code or information bit equaling one-half of the on-power level. Bilevel optical Manchester modulation does have an average optical power of zero when a message is not being transmitted. Consequently, there is a low-frequency component, which has a fundamental frequency equal to the message rate, often 10 Hz or less. Fiber optic receivers are usually ac-coupled to compensate for the photodetector's electrical signal levels, which are not very large in comparison with the magnitudes of amplified drift and offset voltages. Because of this, special signal processing is needed to offset the effect of the low-frequency component.

Several techniques have been developed for dealing with this low frequency component, but these are susceptible to noise from within the system. The result is that the transmitter sections must be more efficiently decoupled from sources of noise in their equipment and must be quieter when not transmitting. In addition, because of the low input power levels to the fiber-optic receivers, the front-end electrical signal levels are much lower in MIL-STD-1773 receivers than in those for MIL-STD-1553B. To obtain satisfactory performance with the greatly reduced signal level, careful shielding is required, as well as decoupling of electrical interference on subsystem lines entering the receiver.

TABLE 3.3-1. MIL-STD-1553B HARDWARE CHARACTERISTICS

Transmission Medium	Twisted Shielded Pair
Characteristic Impedance	70 to 85 Ohms @ 1 MHz
Main Bus Length	Not Specified
Media Connections	Transformer Coupled
Modulation	Baseband (TDM)
Signaling Method	Biphase Level
Transmission Direction	Bidirectional Half-Duplex
Transmission Method	Asynchronous
Transmission Order	MSB First
Data Rate	1 Megabit/Second
Data Code	Manchester II Biphase Level
Bit Error Rate	One Per 10^{12} Bits
Word Error Rate	One Per 10^{7} Words
Topology	Single Serial Bus (Redundant OK)
Number of Terminals/Addresses	31 Addresses/30 Subaddresses Each
Logical Addresses	Not Specified
Media Access	Command/Response
Data Link Control Protocol	Not Applicable
Error Detection	Odd Parity
Synchronization	Word
Word Size	20 Bits
Data Bits/Word	16 Bits
Words/Message (Min.-Max.)	1-32
Word Types	Command, Status, Data
Intermessage Gap Time	4 Microseconds
Bus Frame Length	Not Specified
Bus Control Transfer Time	Not Specified
Terminal Transmit Interface	Not Specified
Terminal Receive Interface	Not Specified

Since optical signals cannot assume negative values, the receiver outputs, which are complementary and thus never low at the same time, cannot be used to identify the no-message state in a MIL-STD-1773 system. As a result, the no-message state and the off state of a two-level Manchester II biphase bit cannot be distinguished. In MIL-STD-1773, it is considered good practice to design fiber-optic receivers with three output states, even though the receivers have only two input states. This is done for compatibility with the outputs of wire-based receivers.

TABLE 3.3-2. MIL-STD-1773 HARDWARE CHARACTERISTICS

Transmission Medium	Fiber Optic
Characteristic Impedance	Not Specified
Main Bus Length	Not Specified
Media Connections	Not Specified
Modulation	Baseband (TDM)
Signaling Method	Biphase Level, 2-State
Transmission Direction	Bidirectional Half-Duplex
Transmission Method	Asynchronous
Transmission Order	MSB First
Data Rate	Multiple Speed
Data Code	Manchester II Biphase Level
Bit Error Rate	One Per 10^{12} Bits
Word Error Rate	One Per 10^{7} Words
Topology	Single Serial Bus (Redundant OK)
Number of Terminals/Addresses	31 Addresses/30 Subaddresses Each
Logical Addresses	Not Specified
Media Access	Command/Response
Data Link Control Protocol	Not Applicable
Error Detection	Odd Parity
Synchronization	Word
Word Size	20 Bits
Data Bits/Word	16 Bits
Words/Message (Min.-Max.)	1-32
Word Types	Command, Status, Data
Intermessage Gap Time	Not Specified
Bus Frame Length	Not Specified
Bus Control Transfer Time	Not Specified
Terminal Transmit Interface	Not Specified
Terminal Receive Interface	Not Specified

3.3.6. Major Documents

The characteristics of the MIL-STD-1553B bus are described in The MIL-STD-1553 Multiplex Applications Handbook, dated February 1982.

The characteristics of the proposed DOD-STANDARD 1773 DATA BUS are described in a tutorial entitled FIBER OPTIC DATA BUSES, National Aerospace & Electronics Conference (NAECON), 19 May 1986. The specific subject is covered in the section entitled MIL-STD-1773 (C. R. Husbands, The Mitre Corporation).

3.4. The GAMA ASCB

3.4.1. History

The ASCB bus was developed, with the support of GAMA, to provide digital communications in business jets and commuter turboprop airplanes using low-cost commercially available components for the bus interface and an existing protocol (High-level Data Link Control (HDLC)). A schematic of the generic bus is shown in figure 3.4-1.

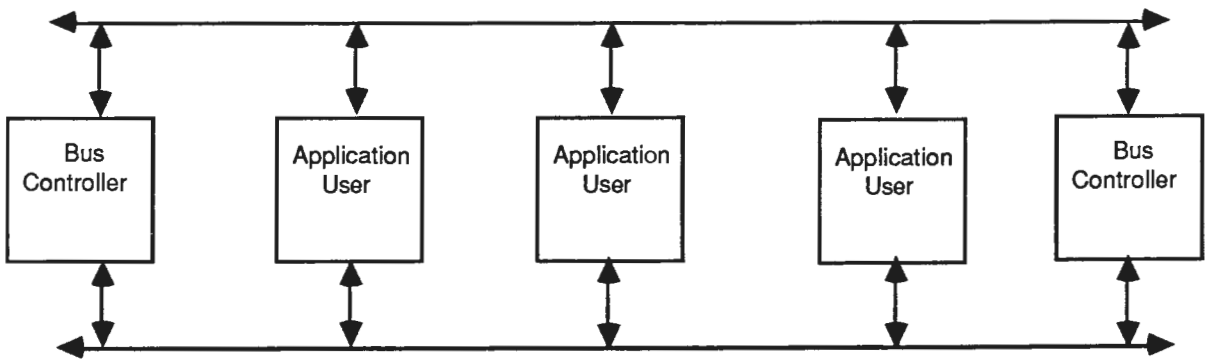


FIGURE 3.4-1. GENERIC ASCB BUS LAYOUT

3.4.2. GAMA ASCB General Characteristics

General characteristics of the ASCB bus include the following:

- Medium speed (0.67 MHz bit rate).
- Serial, bidirectional (half-duplex), redundant bus architecture.
- Redundant bus controllers (only one controller is active at a given time).
- Bus controller transmission on both buses simultaneously.
- Application user transmission on one bus; all application users listening to both buses.
- Support for up to 48 receivers.
- Broadcast format.
- Industry standard HDLC protocol.

- Word size of 16 bits, with up to 256 words per message.
- Low-cost interface components (transceivers, HDLC protocol chips, etc.).
- Multiple error-check capabilities, including data counters, CRC, and checksum.

3.4.3. GAMA ASCB Protocol: HDLC

With this protocol, transmission order is maintained by the active bus controller. An eight-frame timing cycle is used, with each frame lasting 25 ms. The bus controller requests the various users to transmit their data at appropriate intervals within the frame cycle, as required by the update rate needed for desired system performance. To prevent a user transmission from exceeding the frame time, an interlock is used.

The generic frame sequence is as follows. The active bus controller issues a start of frame word (the synchronization pattern) that announces the start of a new frame. This is followed by a frame control word, which contains information about the specific frame, including the frame number and controller identification. Any requests for user transmission complete the frame period. In this way, users can listen for information relevant to them and know when to expect that information to arrive. Similarly, each user knows when to transmit information: each user has a unique request address and response address. It listens for its address and transmits at the time stated in the frame. Users listen for their addresses on both buses. In response to a request for transmission, the user replies with its response address and the data it is required to transmit.

The transmitting user must have its data ready to transmit prior to its assigned frame. If it is not ready, the transmitter will not have enough data to fill out the frame. When this happens, the entire frame is aborted.

Synchronization is maintained by the unique eight-bit pattern that serves as both a "start-of-frame" word and an "end-of-frame" word. Each time this pattern occurs, users automatically resynchronize. This pattern, "01111110", is called a "flag". Consequently, no data sequence is allowed to contain six "1"s in a row. Whenever five successive "1"s are sent, the transmitter automatically replaces the final "1" with a false "0". The receiver then automatically replaces the false "0" with a "1". Adding and replacing these "1"s and "0"s has no effect other than to reduce the data transmission rate slightly.

3.4.4. GAMA ASCB Hardware Characteristics

Table 3.4-1 presents the characteristics of the GAMA ASCB.

3.4.5. Major Documents

The characteristics of the ASCB are described in the GAMA Specification dated February 1986.

TABLE 3.4-1. GAMA ASCB HARDWARE CHARACTERISTICS

Transmission Medium	Twisted Shielded Pair
Characteristic Impedance	125 Ohms
Main Bus Length	125 Feet
Media Connections	Transformer Coupled
Modulation	Baseband (TDM)
Signaling Method	Biphase Level
Transmission Direction	Bidirectional Half-Duplex
Transmission Method	Asynchronous
Transmission Order	LSB First
Data Rate	.67 MHz $\pm$ 0.05%
Data Code	Manchester II Biphase Level
Bit Error Rate	One Per 10^8 Bits
Word Error Rate	Not Specified
Topology	Dual Serial Bus
Number of Terminals/Addresses	48
Logical Addresses	Not Specified
Media Access	Not Specified
Data Link Control Protocol	HDLC (BOP)
Error Detection	CRC
Synchronization	Frame
Word Size	2 Bytes
Data Bits/Word	16 Bits
Words/Message (Min.-Max.)	1-256
Word Types	Not Specified
Intermessage Gap Time	8 Bit Times (Minimum)
Bus Frame Length	25 ms
Bus Control Transfer Time	50 ms
Terminal Transmit Interface	One Bus Only
Terminal Receive Interface	Both Buses

3.5. DATAC (ARINC Specification 629) Bus

3.5.1. History

The DATAC bus is being developed by Boeing in response to the ongoing changes in capabilities and complexities of avionics systems being implemented in the next generation commercial transport airplane. The DATAC bus concept is being further developed by Boeing and ARINC as ARINC Specification 629 (Multi-transmitter Data Bus) that is available (as of 1987) in draft form. The bus has already been implemented in the 7J7 for signaling, avionics, and airframe systems, and fly-by-wire. A guiding objective of the bus standard is to support ease of communication between subsystems yet avoid the problems associated with use of a central bus controller. Problems to be avoided include the potential

for single point failure and the need to change the bus controller when any aspect of the bus architecture is modified. A schematic of the generic bus is shown in figure 3.5-1.

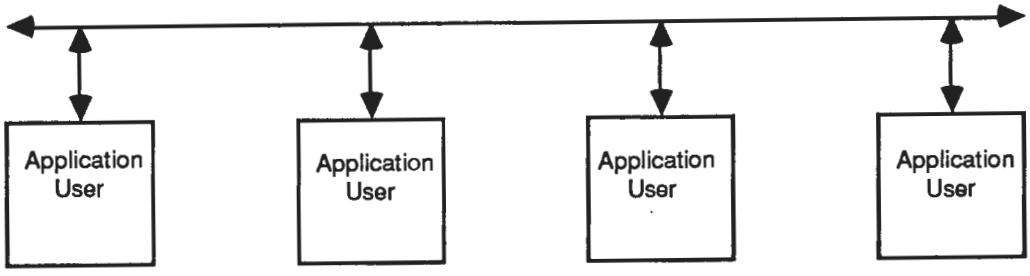


FIGURE 3.5-1. GENERIC DATAC BUS LAYOUT

3.5.2. DATAC General Characteristics

General characteristics of the DATAC bus include the following:

- High-speed data transfer (2 MHz bit rate).
- Shielded or unshielded twisted pair or fiber optic mode.
- Connection of as many as 120 terminals.
- Linear topology using carrier-sense multiple access/collision avoidance protocol.
- Multitransmitter implementation.
- Special current-mode coupling devices.
- Direct connection to bus through current-mode coupling devices.
- Self-monitored terminals to reduce the risk of bus failure.
- Modularity: can easily add and delete terminals without affecting overall bus functioning.
- Orderly transfer of data ensured by the protocol, rather than a bus controller.
- Carrier-Sense Multiple Access/Collision Avoidance Protocol.
- Compatibility with ARINC 429.

3.5.3. DATAC Protocol: Collision Detection

The DATAC protocol has no central bus controller. Instead, individual transmitters are allowed to transmit whenever they wish. Clearly, there is a strong likelihood that two or more transmitters will attempt to use the bus at the same time. Such collisions result in the transmission of distorted data from both transmitters. Consequently, both messages will have to be retransmitted. Even if a second attempt leads to successful transmission, rather severe inefficiencies in bus use will result since the time taken to send both messages could be many times longer than the lengths of the messages. Also, repeated collisions may result. As a result, use of the Collision Detection (CD) method, in its simplest form, tends to allow less than 20 percent utilization of bus capacity before bus stability becomes problematic (Rich et al., 1983). Higher utilizations radically increase the probability of collisions during a second attempt at message transmission. Once this occurs, the total traffic from the first collision, plus that from the second is thrust downstream in the overall message traffic, increasing the likelihood of additional collisions. In short, at some point the process begins cascading until all terminals in the network become involved and no successful transmissions can be performed.

Although the "pure" form of CD has some severe problems, modifications of this approach are more successful. One approach is to enable each transmitter to detect the presence of a message on the bus and, if the bus is "busy", to delay transmission of its message. This approach is known as Carrier Sense Multiple Access (CSMA). When used in conjunction with CD it is referred to as CSMA/CD and is the basis for the DATAC protocol.

With CSMA/CD the occurrence of interfering transmissions is restricted to that situation in which two terminals begin to transmit "so closely together in time" that neither has yet sensed the other's signal. This short time interval at the beginning of a message is referred to as the "collision window" and is simply due to the propagation delay of the network. The collision window is typically on the order of a microsecond in a wired network over short distances.

Requiring the following three events to occur before a transmitter is allowed to send a message again prevents collisions (Shaw, Herzog and Okubo, 1986, p. 223):

"(1) a frame time, common to all terminals on the bus, must have elapsed
(2) a sync gap, common for all terminals, must have existed on the bus
(3) a terminal gap, common for all terminals, must also have existed on the bus."

To protect against corrupted data, the receiver of the transmitting terminal monitors the transmission and checks that each label transmitted has been authorized, that each label contains the correct channel information, that the number of words allowed in that string has not been exceeded, and that the number of word strings in a message has not been exceeded. Any fault causes the transmitter to be inhibited for the remainder of that messages. It is allowed to try again on the next frame time. This continues until a certain number of successive tries are unsuccessful, at which time the terminal is permanently disabled.

3.5.4. DATAC Hardware Characteristics

Table 3.5-1 presents characteristics of the Boeing-developed DATAC (ARINC Specification 629) bus.

TABLE 3.5-1. BOEING DATAC BUS CHARACTERISTICS
(ARINC Specification 629)

Transmission Medium	Twisted Pair (Non-Shielded, Insulated)
Characteristic Impedance	Not Specified
Main Bus Length	<100 Meters (m)
Media Connections	Transformer Coupled (Current Mode)
Modulation	Baseband (TDM)
Signaling Method	Biphase Level
Transmission Direction	Bidirectional Half-Duplex
Transmission Method	Asynchronous Broadcast
Transmission Order	LSB First
Data Rate	2 MHz Bit Rate
Data Code	Manchester II Biphase Level
Bit Error Rate	One Per 10^{12} Bits
Word Error Rate	Not Specified
Topology	Single Serial Bus (Redundant OK)
Number of Terminals/Addresses	Up to 120
Logical Addresses	Not Specified
Media Access	Contention
Data Link Control Protocol	CSMA/Collision Avoidance
Error Detection	Odd Parity
Synchronization	Frame
Word Size	32 Bits
Data Bits/Word	16 Bits
Words/Message (Min.-Max.)	1-4096 (256 Words/String, 32 Str/Msg)
Word Types	Not Specified
Intermessage Gap Time	14 Bit Time Minimum (Terminal Dependent)
Bus Frame Length	50 ms
Bus Control Transfer Time	Not Specified
Terminal Transmit Interface	Not Specified
Terminal Receive Interface	Not Specified

3.5.5. Major Documents

The characteristics of the DATAC bus are described in draft ARINC Specification 629 (Multi-Transmitter Data Bus) Part 1 - Protocol Description and Part 2 - Label and Word String Definitions, both dated September 1987.

3.6. SAE High-Speed Data Buses

3.6.1. History

The newest data buses are being developed by the SAE and industry through the SAE Avionics System Division (formerly SAE/AE-9B High-Speed Data Bus Subcommittee). Under this organization, two competing bus standards are being developed: (a) High-Speed Ring Bus Standard; and (b) Linear Token Passing Bus (LTPB) Standard These two bus concepts are currently in the development stage with draft specifications having been produced (through 1988) and prototype hardware having been developed and tested. Schematics of the two buses are shown in figures 3.6-1 and 3.6-2.

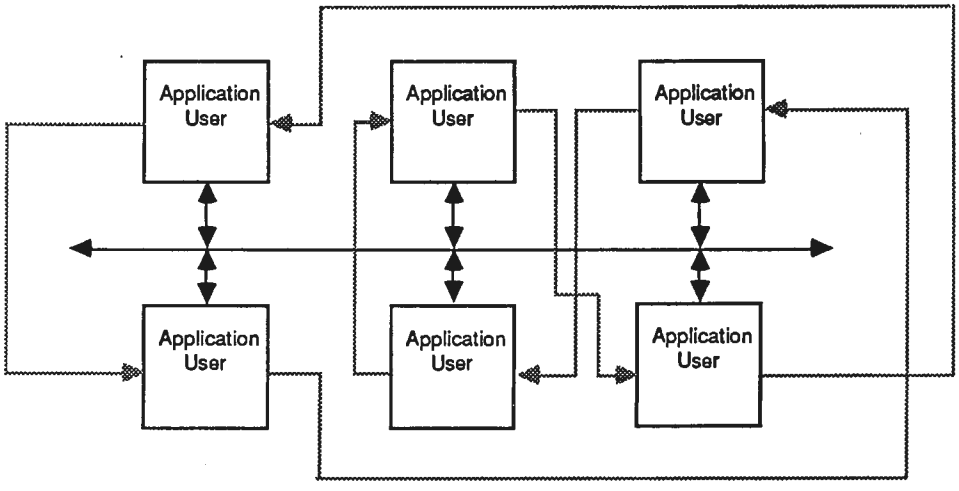


FIGURE 3.6-1. GENERIC SAE LINEAR BUS LAYOUT

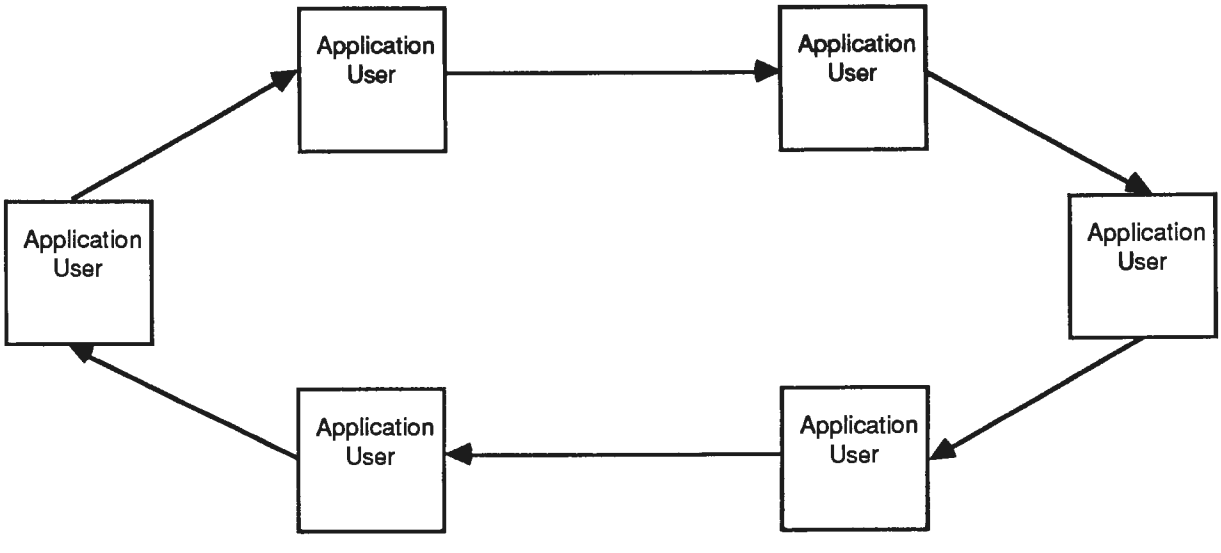


FIGURE 3.6-2. GENERIC SAE RING BUS LAYOUT

3.6.2. SAE General Characteristics

General characteristics of the SAE buses include the following:

- High-speed data transfer (20-40 MHz bit rate).
- Shielded or unshielded twisted pair, coaxial cable, or fiber optic media.
- Serial, bidirectional (half-duplex) data flow.
- Connection of as many as 128 physical terminals on each of four buses.
- Both broadcast and multicast modes.
- Modularity: can easily add and delete terminals without affecting overall bus functioning.
- Orderly transfer of data ensured by the protocol, rather than a bus controller.
- Word size of 16 bits (message lengths range from one word to 256 words for the linear bus; up to 4096 words for the ring bus).
- Logical addressing with 2^{16} addresses available.

3.6.3. SAE Protocol: Token Passing

Like the CD protocol, the token passing protocol also gives control of bus transmission to the transmitters themselves. Instead of a central bus controller, the currently-sending or last-sending station serves as the current controller. While acting controller, the transmitter provides the clock source for the entire bus, transmits its message, then passes a "token," a special message containing a data word which identifies the terminal which is to become the next controller. Each transmitter adds "1" to the token word before passing it on. The transmitter which receives the token becomes the new acting controller, repeating the same process.

When the last terminal controlling the bus completes operations, a message with a non-existent token number is created and transmitted. Since there is no terminal with that token number, a brief lapse occurs in the flow of data. The token-zero terminal is responsible for timing this data lapse and assuming control of the bus after the specified period of inactivity. Thus, the system automatically restarts itself with token zero whenever control lapses regardless of the number of terminals active in the system.

A terminal attempting to use an already active bus system simply monitors the bus for a few cycles to determine the highest token number currently in use. When this terminal identifies a token number to which no other terminal responds, it adopts this number and responds the next time that token number is issued. Since this terminal responds within the appropriate time-frame, the

token-zero terminal does not jump in. The token-zero terminal will restart the cycle after the new terminal has completed its operations and passed on a token to which no other terminal responds.

When a terminal has failed during a cycle and a token is offered to it, the token is ignored. No terminal takes control and bus activity ceases. The token-zero terminal detects the lack of activity and resumes control of the bus. All terminals with token numbers higher than the failed terminal are skipped. Each terminal in the system should be programmed to recognize and respond to this situation.

A terminal recognizes that something has failed in the network when more than two complete cycles have passed since the terminal received its last token offer. The terminal therefore decrements its token number by one. On the next cycle some terminal will recognize the previously ignored token and assume control. All higher numbered terminals have also decremented their token numbers and normal operation resumes. Until the ignored terminals decrement their tokens to rejoin the loop, abbreviated cycles will be run. Normal operations resume ignoring the failed terminal. If the unit recovers, it enters the loop as previously described.

This protocol works even if it is the token-zero terminal which fails, because when the token-one terminal discovers that it is being ignored, it decrements its token to zero and assumes the token-zero terminal functions. This implies that all terminals must have token-zero capabilities.

The token passing approach is highly tolerant of transmitter failure. But as was the case with the basic CD protocol, it is not as fault tolerant with respect to message errors. When a transmitter fails, not only is its data lost but also the data of all transmitters that follow the transmitter in the queue. Transmitter failure results in the transmission process starting over with the token zero terminal.

A potential problem arises because the protocol does not necessarily offer a good environment for managing unscheduled events since a terminal cannot transmit until the token is passed to it. The time between the generation of an unscheduled message and its transmission may be excessively long. Consequently, a true emergency message cannot be handled by the protocol without a scheme for assigning priorities, such as frequent polling of the source.

Another potential problem with this protocol is coping with message errors. For example, the time-out executed by the token zero terminal needs to be kept small in order to maintain high bus utilization. However, should a controller (due to extended error analysis,) pause too long before its next bus operation, the token zero terminal could interpret this as the end of a cycle and start the next cycle.

Consequently, resumption of operation by the pausing terminal could cause a collision with the traffic from the token zero terminal. When this happens, both colliding terminals believe they have failed. Should this occur, all bus activity stops until other bus transmitters recognize the problem and recover by adjusting their tokens accordingly. This may not fully resolve the problem

because when the two failed terminals attempt to rejoin the network, a collision is likely to occur once again. A collision can occur either between the two "failed" systems or between one of the "failed" systems and a correctly operating system.

Rich et al., 1983 suggest that concerns about such collisions could lead to extending the interval defined for token zero time-out. Rules for testing bus activity before initiating a new cycle need to be defined. Both these rules and the token zero time-out interval need to be considered when defining the time interval used by each terminal to decide when to decrement its token. This time interval should be longer than the maximum size and may be extended as a precaution.

The SAE High Speed Data Bus Committee, for both the linear and ring buses, has recognized these problems and has formulated solutions within the protocol.

3.6.4. SAE Hardware Characteristics

Table 3.6-1 lists the characteristics of the SAE High Speed Ring Bus (HSRB). This system is a physical token passing ring network which connects a set of stations in a closed loop topology with a serial transmission medium. While the semantics and protocol described are independent of the data transmission rate, the HSRB is intended to operate in the range ten million to one hundred million bits per second.

Ring access is by means of a token which passes from one station to the next to provide equitable access for any given priority of message. Data are transmitted in messages from one station to the next in a unidirectional manner, starting with the sending controlling station passing through the intermediate stations to the receiving addressed station. The receiving station copies the message for its bus interface unit (BIU) user and also retransmits it on the ring. The message passes through the remaining stations until it reaches the sending station that removes it from the ring. The controlling station must then transmit a token. A station is not permitted to transmit more than one message before issuing a token. Only a station that has claimed a valid token is authorized to transmit a message for its BIU user, all other stations are constrained to repeat each message as received on the ring with the exception of setting error reporting and priority reservation bits.

The ring bus offers superior throughput capability when compared with the linear bus due to short point-to-point media links between nodes. In the area of fault recovery and reliability, the ring is less attractive due to the need for failed node bypassing using either mechanical relays or fiber optic switches. Ring reconfiguration may take up to 25 ms when bypasses are activated. In addition, a limit must be placed on the number of consecutive nodes which may be bypassed, due to a lower power budget in the short point-to-point links and the relatively high losses inherent in the bypass devices (both wire and fiber optic).

Table 3.6-2 presents the characteristics of the SAE AS-2 (Formerly AE-9B) Linear Token Passing Bus (LTPB).

The AS-2 proposed LTPB protocol involves four simple states:

- Bus Initialization.
- Normal Token Passing.
- Station Insertion.
- Station Management.

The token is passed from the lowest physical address to the highest physical address and then back to the lowest.

TABLE 3.6-1. SAE RING BUS HARDWARE CHARACTERISTICS

Transmission Medium	50 MBPS Coax, 100 MBPS Fiber Optic
Characteristic Impedance	75 ohm Triax
Main Bus Length	300 m required, 1000 m desired (total station separation)
Media Connections	Optical or Transformer Coupling
Modulation	NRZ-I
Signaling Method	Biphase Level
Transmission Direction	Unidirectional
Transmission Method	Asynchronous Broadcast
Transmission Order	LSB First
Data Rate	10-1000 MBPS
Data Code	Unspecified
Bit Error Rate	One Per 10^{12} Bits
Word Error Rate	Not Specified
Topology	Ring - 2 to 128 Stations
Number of Terminals/Addresses	128 Physical/512 Subaddresses Each
Logical Addresses	2^{15} - Broadcast and Multicast
Media Access	Token Pass
Data Link Control Protocol	Token or Message Frame
Error Detection	CCITT-CRC-16
Synchronization	Frame
Word Size	16 Bits
Data Bits/Word	16 Bits
Words/Message (Min.-Max.)	1-4096
Word Types	4B5B Block Encoding
Intermessage Gap Time	Not Specified
Bus Frame Length	80K Bits
Bus Control Transfer Time	10 Million Data Bits/Second
Terminal Transmit Interface	4 Buses
Terminal Receive Interface	4 Buses

TABLE 3.6-2. SAE LINEAR BUS HARDWARE CHARACTERISTICS

Transmission Medium	Fiber Optic or Electrical Wire
Characteristic Impedance	50 ohms (electrical wire systems)
Main Bus Length	300 m required, 1000 m desired
Media Connections	Optical or Transformer Coupling
Modulation	NRZ
Signaling Method	Biphase Level
Transmission Direction	Bidirectional Half-Duplex
Transmission Method	Asynchronous Broadcast or Multicast
Transmission Order	LSB First
Data Rate	25, 50, or 100 Mbps (Optical Bus) 50 Mbps (Wire Bus)
Data Code	Manchester Biphase II
Bit Error Rate	One Per 10^{12} Bits
Word Error Rate	< 1 Every 4 Hours at BIR
Topology	1 to 4 Serial Linear Buses
Number of Terminals/Addresses	128 Physical/256 Subaddresses Each
Logical Addresses	2^{15}
Media Access	Token Pass
Data Link Control Protocol	Token or Message Frame
Error Detection	CCITT-CRC-16
Synchronization	Frame
Word Size	16 Bits
Data Bits/Word	16 Bits
Words/Message (Min.-Max.)	1-256 Required, 4K Desired
Word Types	Not Specified
Intermessage Gap Time	10 Bit Times
Bus Frame Length	Not Specified
Bus Control Transfer Time	Not Specified
Terminal Transmit Interface	4 Buses
Terminal Receive Interface	4 Buses

The worst case delay in the AS-2 LTPB is directly dependent on the maximum allowable message length. Message latency can be easily handled by implementation of system level message priorities.

In the linear bus, each component is assigned a logical token number (which may or may not correspond to the physical organization of components along the bus). A component gains access to the bus when it receives a token. On receiving the token, the component is allowed to use the bus for a pre-determined, maximum amount of time. This time limit for transmitting data is determined by the Token-Holding Timer (THT).

When the component completes its message (or times-out) it passes the token to the next higher numbered component. It is the responsibility of the passing component to verify receipt of the token by the receiving component. If, after two tries, the receiving component does not accept the token, the sending component increments the token number and tries again. This process is reiterated until all addresses are exhausted. The token is then set back to zero.

Stations are allowed admittance to the logical ring on a periodic basis. Each station contains a Ring Admittance Timer (RAT). When this timer expires and there is a gap between the local station's address and that of its successor then the token is passed to the sequential address following that of the local station. The normal token passing rules are then applied. Therefore, if any of the stations in the gap desire admittance they will be granted an opportunity during this time period.

3.6.5. Major Documents

The characteristics of the SAE High-Speed Data Buses are described in the two draft specifications: (a) HSRB Standard, draft dated March, 1987; and (b) LTPB Standard, draft dated April, 1987. Both of which are available from SAE, Warrendale, Pennsylvania.

To minimize the potential for error due to latency, the node data/status message could include a "time tag" generated when it is sent from the node select computer. When each successively higher level within the architectural hierarchy generates a message or command, it would automatically pass along the time tag of the node data/status message. When the message arrives at the various destinations, within the hierarchy, a comparison is made of the current and new node data/status values and the time tag to ascertain the validity of the command.

In general, the actual latency of a message within a given architecture is determined by the rate at which the bus structure (either autonomously or centrally controlled) allows a "sending" terminal the opportunity to "latch-on" to the bus in order to transmit its message/data. For a centrally controlled bus, to obtain the least possible (i.e., minimum) latency, the controlled bus could be configured to (1) continually poll the terminals within the bus structure, (2) respond to the service request bit in the terminal status word, and (3) initiate the terminal-to-terminal (or terminal-controller) message transfer.

An increase in the distribution of processing tasks to more specialized computers is occurring. Given this trend away from a central general purpose computer, an event-based scheduling scheme (i.e., scheduled or unscheduled system interrupt, such as a key press, a flag operation, etc.) may become a good alternative for some applications. When task scheduling is based on events rather than time, the latency of a message becomes more critical. The continuous polling technique is an effective way to reduce the message latency.

For an illustration of the event based scheduling, refer to the local display bus of figure 4.2-1. The display computer is normally operating in response to messages from the application computer. Its bus interface, which controls the local display bus, is continuously polling for keypad entry. When the keypad is pressed, a message is sent back to the display computer signaling an event to which the display computer must respond. The display computer will break out of its normal cycle, process the keypad message, and upon completion of this processing will have available keypad information that can be sent to other devices on the mission computer bus.

In this application two advantages are obtained from the event based scheduling and continuous polling. The latency of the message as it passes from a local bus to a higher level bus is minimized. Component faults in the communication system are identified early to provide time for management of the failure. For example, a simple management scheme would be to retransmit if the status response were not returned with the message error bits clear. A disadvantage of event based scheduling and continuous polling is that testing is more difficult due to the tester's inability to repeat a particular condition. When all scheduling is time based, then a repeatable test scenario can be generated. System response can therefore be accurately evaluated. When operation is based on asynchronous events, only a statistical comparison of results from multiple tests is valid.

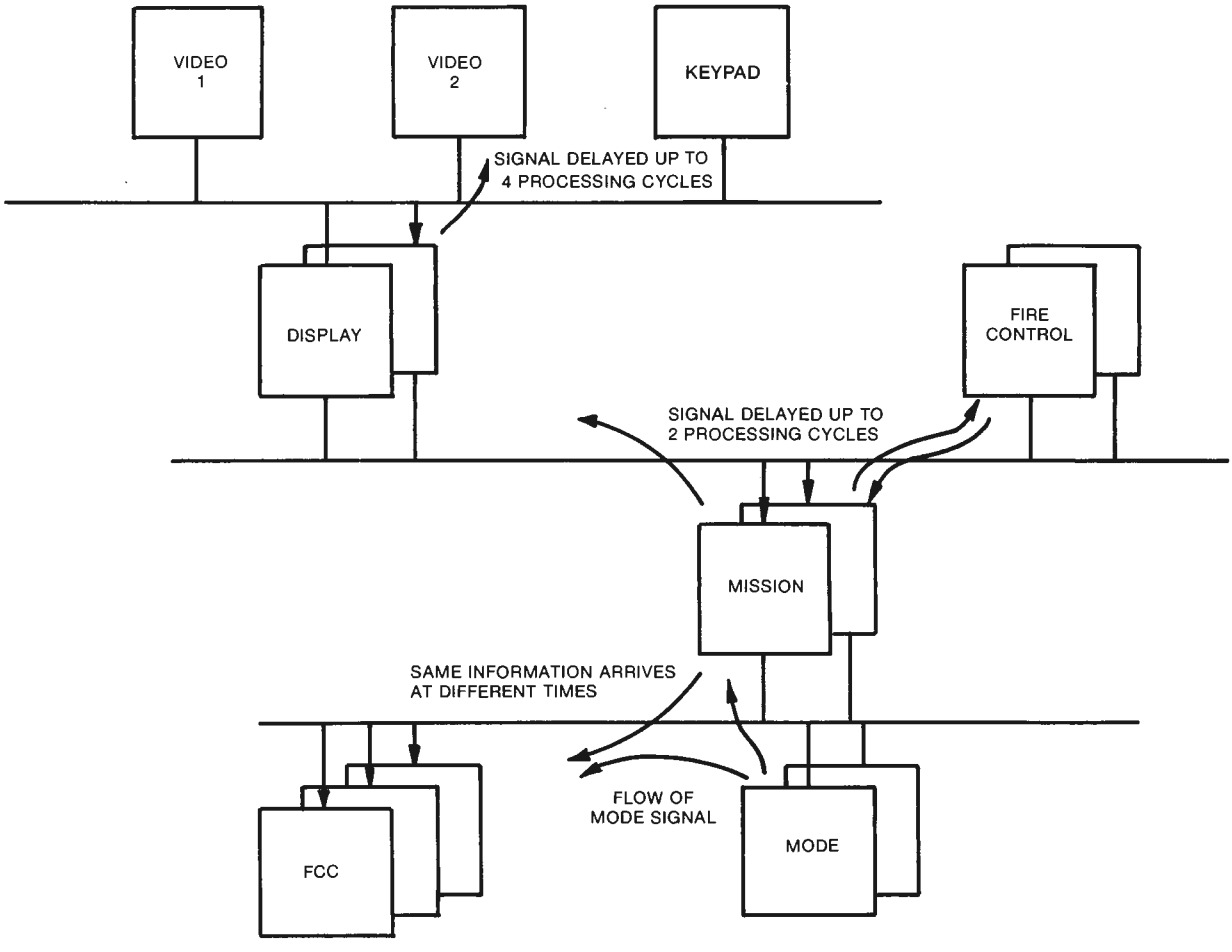


FIGURE 4.2-1. DATA LATENCY ILLUSTRATION IN HIERARCHIAL ARCHITECTURE

On a single hierarchical level, there are several protocols that can be used within the bus architecture framework. The choice of protocol affects message latency. For example, the use of a stationary master that polls all terminals on a regular basis provides minimum latency for a small number of terminals on the bus.

Alternatively when bus control is exchanged among a limited set of master computers, potentially greater latency is possible (depending upon the message table orientation of each master computer). If bus control capability exists in every terminal that may have a time critical message, the message latency will be in the range of several (2-4) ms. If continuous polling is done between message transmissions, latency improves; however, a large bandwidth penalty is paid. Continuous polling can only be used on buses with low activity levels.

4.2.3. System Delays

Average transfer delay is defined as the sum of delays resulting from queuing delay, access delay, and transport delay.

- Queuing Delay.

  Queuing delay is characterized by message arrivals and arrival rate. It is represented (characterized) as a Poisson Distribution. The mean queuing delay consists of the average delay incurred due to a message waiting for a previous message within the BIU to be serviced. The BIUs are effectively a single server queue, and therefore the queuing delay is a delay imposed on the user due to the BIU transmit buffer being full. This delay neglects the user/BIU message processing rate limitations and is dependent only on the message interarrival time as determined from the offered load.

- Access Delay.

  In the case of the CSMA/CD protocol, the mean access delay is determined by considering the two inherent access modes. The delay due to the random mode and the delay due to the ordered access mode are factored with the probabilities of being in their respective states and combined to equal the mean access delay.

  For the random access delay there are two components of delay: (1) delay due to the bus being busy, and (2) delay due to a collision. For the bus to appear busy, at least one other message must arrive before the message that encounters the media active state. Therefore, the probability of the bus being busy is the probability of two or more arrivals within the same time window.

  The probability of a collision can be described by the probability of two or more simultaneous arrivals. The delay due to a collision is determined by the time required to recognize contention, issue the jamming signal (approximately 1 microsecond), wait for the appropriate gap time, and then wait for the appropriate timeslot. Because the load distribution is assumed to be equal among the network BIUs, the average delay for the

timeslot count to reach the assigned time is one-half the total scan time for the timeslot sequence as determined by the loading conditions.

Looking at the access delay encountered by a message under the ordered access mode, two conditions are possible: (1) message ready before the timeslot arrives, or (2) message ready after the appropriate timeslot has passed. For an equal load distribution, the probability of each case is 0.5.

- Transport Delay.
  - The transport service time is determined by the transmission rate, the message length, and the overhead required for each transmitted packet. The overhead includes the following:
  - Gap time between messages.
  - Turnon time.
  - Packet encapsulation.
  - Acknowledge turnon.
  - Propagation delay.
  - Acknowledge message.

System level fault management is further facilitated by the monitoring of network statistics at each node. During operation, the BIUs collect the following statistics:

- Number of collisions.
- Number of collisions during own transmission.
- Number of packet rejects due to decoder buffer full.
- Number of successful transmissions.
- Number of unsuccessful transmissions.
- Number of data transmissions received.
- Number of status responses received.
- Number of commands received.

4.3. Failure Modes and Effects

Operation of digital aircraft requires the proper function of a number of interconnected systems, subsystems, and components within the framework of an integrated bus structure. Intermittent or erratic behavior or total failure of

one or more modules can impact the ability of the aircraft to perform its intended function. In some cases, the impact will be transparent as the fault is automatically detected, the failed module identified, and a redundant "like element" (similar or dissimilar) activated or "switched-to" automatically. Continued successive failures (or in the worst case, multiple simultaneous failures) could result in increased pilot workload, loss of function, or in the most severe case, the total loss of aircraft.

Because of the interactive relationships among systems and subsystems in aircraft, failures may propagate. Errors may affect not only the subsystem in which the failed module is embedded but also other subsystems to which erroneous data is transferred. This failure propagation potential between multiple systems/subsystems is greatly magnified by the differing levels of functional integration where data and information are exchanged between and among subsystems (using bus architectures and structures) as a requirement for normal operation.

Failures that could cause loss of essential mission capability or loss of aircraft must be protected against by using equipment redundancy, analytical redundancy, or functional redundancy to provide for continued operation after one or more failures. The redundancy may be applied at the system level (multiple buses or FCCs), at the sensor level (redundant Inertial Navigation System (INS), Altitude Heading Reference System (AHRS), Digital Air Data Computer (DADC), etc.), or at the module level (multiple similar or dissimilar microprocessors located in multiple processor subsystems). Failures that result only in some loss of function, restricted operation, or increased pilot workload, may or may not require redundancy. Redundancy requirements depend on the exact nature of the loss and the probability that such a loss will impact aircraft performance or aircraft flight safety. Failures that reduce the level of hardware redundancy or analytical redundancy, without loss of functional capability, may be able to be tolerated without performance degradation.

Failures can result from external disturbances or internal malfunction and can be either transient or permanent. Transient faults can often be ignored if the system is designed to tolerate such faults. In other cases, a transient fault can cause a more serious failure, such as the interruption of an instruction sequence in a computer, which in turn could cause a time-out or retry sequence, resulting in the completion of the computation using "stale" data. Permanent failures, on the other hand, must be recognized as such, and action must be taken to reconfigure around the failure.

Environmental effects can often be the cause of the failure. In the case of wire buses, heat, power supply surges (spikes), or low voltage levels could cause permanent or intermittent loss of an electronics unit or corruption of the bus data, which in turn would cause incorrect data and/or information to be passed to another unit in the hierarchy. Loss of electrical integrity (due to faulty shielding, grounding, or loss of cable integrity) could result in susceptibility to electromagnetic radiation, thus causing erroneous or erratic behavior.

In general, failures may exist in any one of the five functional elements relating to the integration of two or more subsystems. These functional elements include the following:

- Computers that process the data exchanged between subsystems.
- Data bus interfaces.
- Data bus(es).
- Input/output devices that govern the transmit/receive functions.
- System software.

Table 4.3-1 (McSharry, 1983) summarizes the potential faults which can cause intermittent or erratic behavior or even total failure of the networked architecture to communicate data and information to the various systems/subsystems within the structure.

The following considerations concerning the possible failures of digital data buses must be taken into account: First, transmission failures that may occur; second, effects on subsystems that are connected to the data bus by a bus controller or remote terminal failure; and third, effects of multiplex hardware failure. The navigation system must be self-contained and the aircraft must not become "lost" because of any type of transient. These safety requirements lead to subsystem requirements to store critical data in multiple locations and to recover rapidly from failures and upsets.

There are three failure modes, as follows: First, no transmission; second, incorrect transmission; and third, failure to relinquish control. A fault with these failure modes and some of the related causes are shown in figure 4.3-1.

4.4. Transmission Error

If the multiplex terminal hardware detects either an invalid word or a transmission discontinuity, the word and message are to be considered invalid. This message invalidation requirement may cause some systems (i.e., Electrical Multiplex (EMUX)) a problem. Since the EMUX systems usually have bit-oriented data rather than word or multiple word (message) oriented data, errors in a word following the reception of good data will invalidate good data.

Message completion failures should always be detected in a multiplex system. They are detected by the bus controller when the status word is suppressed or the message error flag in the status word is set. The message error flag removes ambiguity as to whether the error occurred before the message was validated by the remote terminal or in the response to the message.

Data transmission errors are handled by special error-handling interrupt software. The software will indicate whether (1) the command is to be retried, (2) the bus is to be used for the retry, and (3) the transmitted data (if any) should be invalidated. Tables 4.4-1 and 4.4-2 (MIL-STD-1553 Designer's Guide,

TABLE 4.3-1. DIGITAL DATA BUS SYSTEM FAILURE MODES AND EFFECTS
(Adapted from McSharry, 1983)

1.0 COMPUTER FAULTS	2.0 DATA BUS AND INTERFACE	3.0 INPUT/OUTPUT	4.0 SYSTEM REDUNDANCY/MANAGEMENT FAILURES
1.1 Computer Hardware Faults	2.1 Data Bus Hardware	3.1 Input/Output Device	4.1 Design Failure (Fault)
1.1.1 Central Processor Unit (CPU) Failure	2.1.1 Open/Short-Single Bus/Multiple Buses	3.1.1 Hardware Failures	4.1.1 Synchronization-Time Skews (Comparison Monitoring of Time Tag Data)
1.1.2 Memory Failure (Error)	2.1.2 Interface Unit Failure	3.1.1.1 Sensor, Sensor Output Transducer, Discrete Signal Input Failure	4.1.2 Generic Failure Modes - Power Interrupts, Software Design Faults, Common Mode Failures
1.1.3 Discrete and Analog Input/Output Failure	2.1.2.1 Failure to Respond	3.1.1.2 Induction of Noise (or Other Oscillary Disturbance) Due to EMI/EMC	
1.1.4 Clock Failure	2.1.2.2 Response to Wrong Address (Failure Diagnosis: Good = Defective)	3.1.2 Software Faults (Errors)	
1.2 Computer Software Failures (Faults)	2.1.3 Bus Controller Unit Failure - Site Selection Error	NOTE: Typical Failures Include:	
1.2.1 Failure to Isolate Faulty Entity	2.1.3.1 Multiple/Redundant Controller Selection - Hardware Arbitration	o Hardover to Max/Min Values o Steady Zero Values o Bias Offset o Reduced Dynamic Response o Sensitivity Change	
1.2.2 Isolation of Good Component as Failed	2.1.3.2 Multiple/Redundant Controller Selection - Software Algorithm Fault (Error)		
1.2.3 Processing Erroneous Data	2.1.3.3 Non-Centralized Controller Hardware Time-Out (Output Stage) Failure		
1.2.4 Failure to Complete Task - Exceed Task Time Allocation	2.1.4 Improper Decoding/Encoding of Address, Command, Data Word Structures		
1.2.6 Observability			

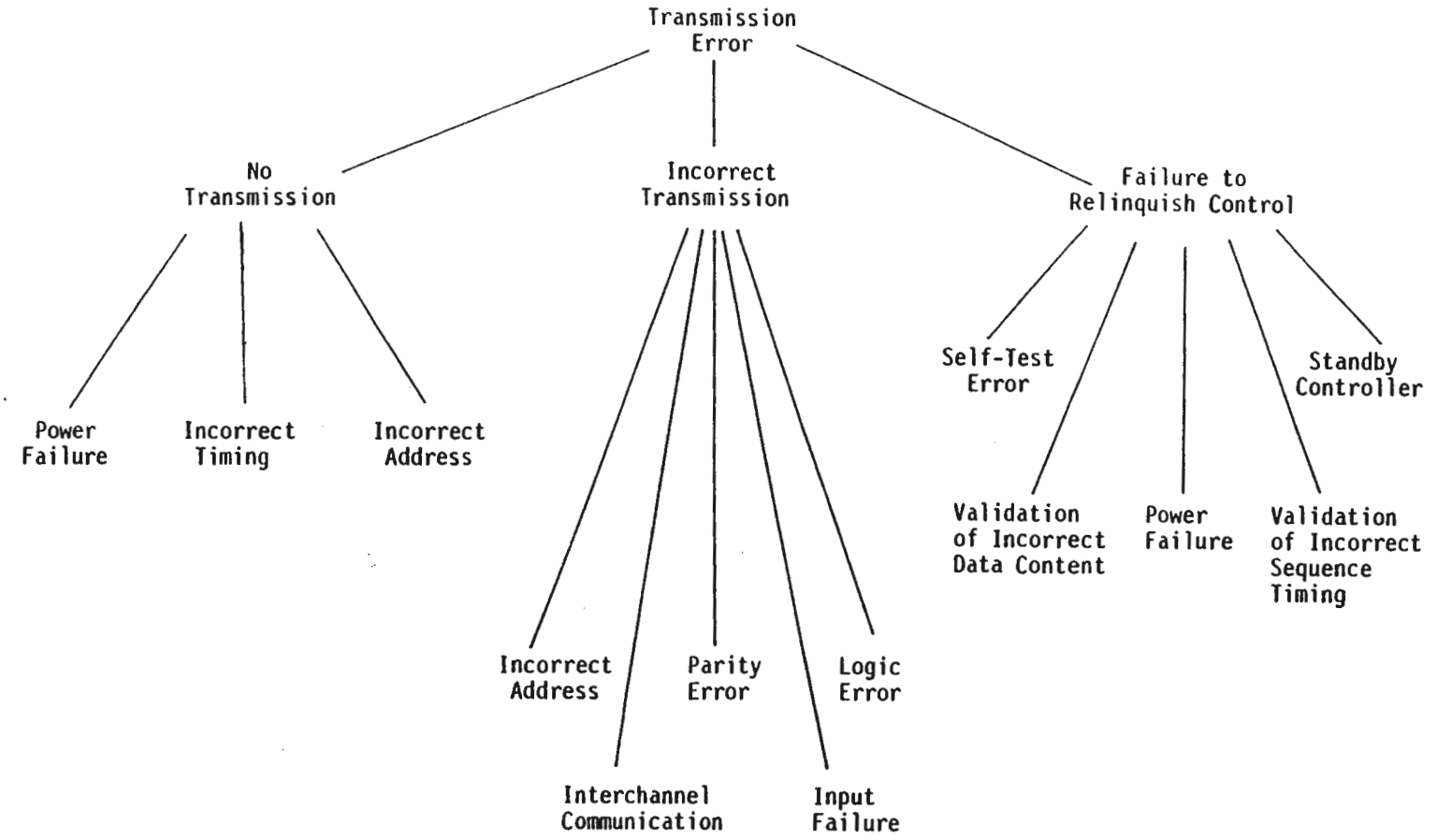


FIGURE 4.3-1. SIMPLE FAULT TREE EXAMPLE

TABLE 4.4-1. FAILURE CLASSES [MIL-STD-1553B]
(MIL-STD-1553 Designer's Guide, 1983)

ERROR IDENTIFICATION	BUS SYSTEM	SENSOR
a. Message error	Transmission from bus controller to to terminal was decoded with error condition by receiving remote terminal	--
b. Busy	Remote terminal unable to transmit or receive data at this time	Remote terminal and sensor unable to transmit or receive data at this time
c. Subsystem flag	--	Sensor failure preventing proper sensor actions
d. Terminal flag	Remote terminal failure preventing complete action by terminal	Remote terminal portion of sensor interface has failure preventing complete action by terminal
e. Parity error (incorrect odd parity)	Error in status word; data not usable	Error in status word; data not usable
f. Improper sync	Unknown problem -- ignore; continue to look for valid sync	Unknown problem -- ignore; continue to look for valid sync
g. Invalid manchester	Error in message -- ignore data in message	--
h. Improper number of data bits and parity	Error in message -- ignore data in message	--
i. Discontinuity of data words	Error in message -- ignore data in message	--
j. No status word response	Unknown problem -- requires further investigation	Unknown problem -- requires further investigation to achieve error

TABLE 4.4-2. TYPICAL ERROR-CORRECTION TECHNIQUES [MIL-STD-1553B]
(MIL-STD-1553 Designer's Guide, 1983)

ERROR IDENTIFICATION TYPES	ERROR CORRECTION TECHNIQUE
1. Bus system failures	
a. No status word response	Retry message on same bus n times
b. Message error	Retry messages on alternate bus n times
c. Parity error	Transmit status word mode code on each bus
d. Invalid Manchester	If necessary, transmit initiate self-mode code
e. Improper number of data bits and parity	Transmit BITE mode code
f. Discontinuity of data words	Analyze failure and determine corrective action, which may involve the following mode commands: Shut down transmitter Inhibit terminal flag bit Transmit reset remote terminal mode code
g. Busy	Retry message on same bus after a fixed delay time
h. Terminal flag	If necessary, transmit initiate self-test mode code Transmit BITE mode code Analyze failure and determine corrective action, which may involve the following mode commands: Shut down transmitter Inhibit terminal flag bit Transmit reset remote terminal mode code
i. Improper sync	Ignore and reset for valid sync
j. Subsystem flag	Retransmit the data (address/subaddress) to examine sensor BITE discretes or words
2. Sensor failure	
a. Discretes b. BITE data word(s)	Analyze failure and determine system-oriented corrective action

1983) show the error identification types and the corresponding failure classes and error correction techniques.

- No Transmission. The user should listen to the bus it transmits on for its request address. If no bus controller activity is detected, the user should transfer listening to the other bus for its request address. If no activity is detected on the other bus, the user should continue toggling between the buses in search of bus controller activity.

- Incorrect Transmission. The most serious failure for the bus controller is erroneous transmission. An independent frequency source should be used by the bus controller to provide monitoring and detection of transmission frequency faults. The two common types of transmissions are broadcast (which is sent on all of the channels) and CR (which is sent to a specific address). An error in a broadcast transmission has the potential for system failure if it is incorrectly validated at each of the addresses. An error in a CR has a more limited effect since it only involves one address. Each receiver should incorporate isolation provisions to ensure that the occurrence of any reasonably probable internal LRU or bus receiver failure does not cause any input bus to operate outside of its specification limits (both under voltage or over voltage).

- Failure to Relinquish Control. Subsystem or terminal failures may be detected using Built-In Test (BIT) circuitry. These failures are reported by the setting of the subsystem flag bit or the terminal flag bit in the status word. In aircraft, dual-redundant buses are used, so a terminal failure may be isolated to one bus. Depending on the capability of the remote terminal hardware, the transmit BIT word mode code can be a powerful diagnostic aid. For each fault, the action to be taken must be determined, designed for, and implemented by the system.

Subsystem or terminal failures can also be detected without the use of the optional terminal or subsystem flags. Bad data or nonvarying data from a subsystem may be interpreted as a subsystem failure. Repeated message completion failures to a remote terminal via all possible data paths could be considered as a loss of the terminal function. The system software should be used to detect these failures.

Bus controller operation in the event of failure is important to an integrated data bus system. The primary bus controller should relinquish bus control whenever it suffers a power interruption that might cause erroneous outputs. The primary bus controller should detect its own bus control processing faults and remove itself as controller in a fail-passive manner. Similarly, the backup bus controller should recognize invalid control messages or the absence of valid control messages and revert to active bus controller status. Monitoring techniques should provide coverage for both hardware faults and software errors. Any undetected fault in the primary bus controller, which results in continuous erroneous transmission, will make all standby controllers ineffective. The bus controller is structured such that two independent faults must occur in order to cause erroneous transmissions.

4.5. Reliability for Flight Safety

Flight safety requirements allow no more than one unrecoverable failure in the flight control subsystem per 10^9 flights. This failure rate is consistent with AC-25-13091 and is appropriate for integrated systems. The failure rate must encompass the entire flight control system including the necessary supportive electrical power, hydraulics, and any other subsystem used in the flight-critical capacity. When applied over the 2- and 3-hour mission duration of the aircraft, a maximum failure rate of approximately 5×10^{-6} failures per flight hour (for a 2-hour mission) and 3.3×10^{-6} failures per flight hour (for a 3-hour mission) can be allowed.

Figure 4.5-1 gives an example for the determination of the loss of bus control. The potential failures for the bus control example are given in table 4.5-1. The total failure rate must be equal to or less than the total allowable defined above. In the example, the loss of bus control, D, is

$$D = (E_1 + E_2 + E_3)(E_4 + E_5 + E_6) + E_7 + E_8 + E_9 + E_{10}$$

By substituting in the values from table 4.5-1, we obtain

$$D = 3.1012321 \times 10^{-5}$$

Therefore, in this example the data bus would fail to meet the reliability requirements for flight safety.

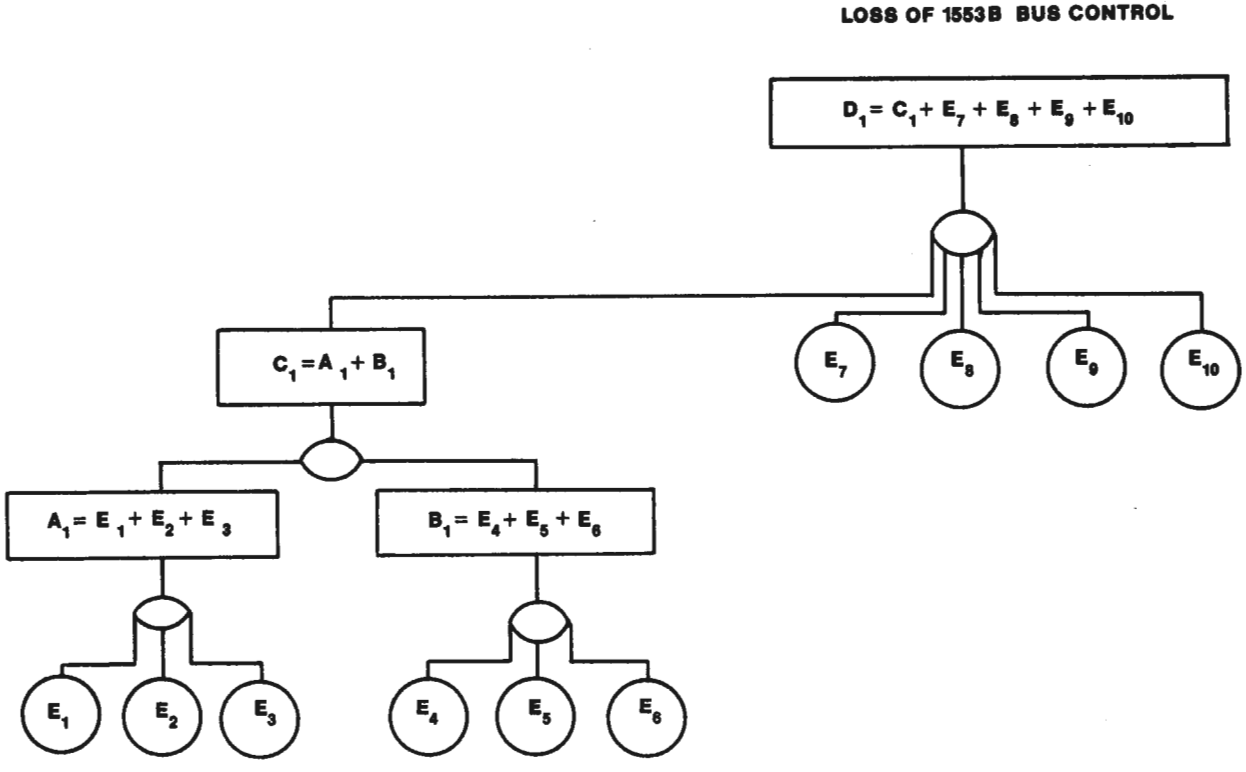


FIGURE 4.5-1. SINGLE CHANNEL-DUAL OUTPUT (BUSES A AND B) BUS CONTROL

TABLE 4.5-1. POTENTIAL FAILURES RESULTING IN LOSS OF BUS CONTROL SINGLE CHANNEL - DUAL OUTPUT (BUSES A AND B)

ERROR	ERROR SOURCE	FAILURE RATE
E_1	Bus A - Transformer Failure	10^{-6}
E_2	Bus A - Transceiver Failure	10^{-4}
E_3	Bus A - Decoder Failure	10^{-5}
E_4	Bus B - Transformer Failure	10^{-6}
E_5	Bus B - Transceiver Failure	10^{-4}
E_6	Bus B - Decoder Failure	10^{-5}
E_7	Single Encoder Failure	10^{-5}
E_8	Internal Control Logic Failure	10^{-5}
E_9	Interface Unit Failure	10^{-5}
E_{10}	Microprocessor System Failure	10^{-6}

5. FIBER OPTIC DATA BUS FOR AVIONICS INTEGRATION

5.1. Elements of Fiber Optic Data Bus

As stated in previous sections, the bus topology is the physical arrangement and interconnection of the various terminals. In a fiber optic bus, the elements utilized are optical couplers, fiber cable, connectors, and splices.

TABLE 5.1-1. OPTICAL BUS TECHNOLOGY LIMITS

COMPONENT	FACTORS
Couplers	Losses Number of Taps
Fiber	Fiber Type Modal Noise Connectors Splicing Reflections Cabling
Optical Source	Power Speed
Optical Receiver	Sensitivity Intermessage Dynamic Range Intermessage Response Time Clock Recovery
Processing/Interface Logic	Speed Power Consumption VHSIC/VLSI & GaAs
Topologies	Performance Reliability Flexibility Installation and Maintenance Cost

The design of these elements not only relates to system performance but also to system installation and maintenance. Because optical power losses occur whenever any of these components or functions are inserted in the optical path, performance is affected. Table 5.1-1 presents the components and factors that influence the limits of optical bus technology as it applies to buses used for avionics integration.

5.2. Optical Cables

Considerations involved in evaluating optical cables for a fiber optic data bus include fiber design (including modal noise and reflection effects) and cable type and construction.

5.3. Optical Path

Basically, the optical path is the fiber optic cable. In designing the proposed avionics architecture, the fiber cable must be selected for minimal loss (across the bus) and wide bandwidth. In addition, the fiber cable must be constructed for strength and endurance during the life of the bus architecture, ease of installation, and long term environmental performance.

5.4. Splices and Connectors

Interconnections between the fiber cable elements (controllers, remote terminals, junctions, etc.) can be made with either splices or connectors. Splices in the fiber cable are easier to incorporate and provide lower losses than connectors; however, splices are permanent. Connectors, on the other hand can be mated and unmated hundreds of times with virtually no degradation in performance. Therefore, in the development (and design) of a fiber optic based avionics architecture, optical couplers (connectors) should be utilized for bus interface connections to the physical bus. This will minimize downtime due to repair or change in the architectural structure when remote terminals are added or deleted. In the case of aircraft having pressurized bulkheads, several penetrations through these bulkheads may need to be made. At these penetrations, the fiber optic cable can be run straight through the bulkhead or an optical connector (coupler) can be used on each side of the bulkhead. The tradeoff, in this case, is between the ease of installation and rework using a connector (coupler) system or the lower loss and absence of reflections using a spliced or through cable.

5.5. Optical Couplers

The two basic types of optical coupling techniques that are considered for an optical data bus are star couplers and taps or tees.

In a transmissive star, N ports are designated as input ports, and N ports as output ports. The optical energy on any input port is split more or less equally among all output ports, with a splitting loss of 10 log N. Star couplers also have an insertion loss and a port-port variation of 1 to 3 Decibel (dB), each depending on the number of ports. Stars in excess of 100 ports have been fabricated; however, for minimal cost and port-port variations, the practical limit of current technology is 64 ports.

Directional couplers for tapping a transmitter and receiver onto a fiber optic bus are basically like a four-port transmissive star with an excess loss of 0.5 to 1 dB. Typically a tap into the receiver can be accomplished with a 90/10 or 95/5 split providing a 0.5 to 2.0 dB link throughput loss, respectively, and a 10 dB to 13 dB tap-off or reduction of the link power into the bus receiver. For tapping the transmitter into the bus, the throughput loss as well as the coupled transmitter power reduction is 3 dB in commercially available couplers.

5.6. Size

Of the available fiber options, the 100/140 micrometer (μm) or the 85/125 μm graded index fiber operating at a 0.05 μm signal length is optimum because:

- Their large core, high numerical aperture, and operation wavelength support many more modes, thus minimizing the modal noise limitation.
- Their large core enables greater Light Emitting Diode (LED) coupled power, thus extending the application of LEDs.
- Their core-clad geometry makes it easier to make low excess loss star couplers.

5.7. Reflections

Another consideration in the medium analysis is reflections. Reflections result from an index of refraction discontinuity at connectors, poor splices, or mismatched fiber types. For example, with a star coupler, the main signal passes through the link; however, part of the signal is first reflected at the star coupler dry connector (8 percent) and then again at the transmitter dry connector (8 percent). The resulting reflected signal is down 22 dB with respect to the main signal and delayed by as much as 1 microsecond (1 ns/m). This reflected signal becomes a problem if it overlaps the next bus transmission and shows up as noise superimposed on this data. Therefore, consideration must be given to minimize reflections.

5.8. Connectors

Optical connectors, which are suitable for use in a data bus, are low cost, easily installed, and typically low loss. The connector loss depends on the fiber size as well as the quality of the connector. For 100/140 μm fiber, losses vary from 0.5 to 1.5 dB depending on connector quality. Available multiway connectors have the advantage of simplifying a bulkhead penetration and provide quicker connect/disconnect of a multifiber cable. Although there is no fundamental reason for higher loss in a multiway connector, the losses in currently available connectors average approximately 0.5 to 1 dB more than the loss in a single fiber connector.

5.9. Splicing

For field installation, maintenance, and repair the elastomeric splicing system has been identified as the best currently available splicing technique.

5.10. Technology - Optical Bus Transmitters and Receivers

Fiber optic bus Transmitter/Receiver (T/R) design is driven by the goal of maximizing bus efficiency. This is necessary to fully utilize the benefits of the bus, minimizing "dead" time, and allowing transfer of significant quantities of data.

An efficient bus T/R is relatively easy to design. However, providing very quick transmitter power output stabilization and very short receiver settling time at the start of a message significantly increases the difficulty and complexity of the T/R design. A fast response clock recovery scheme is also critical in minimizing the amount of time used for non-data. In summary, the more time that is used to perform overhead functions, the less time there is to transmit data, and the lower the efficiency of the bus.

5.11. Maximizing Bus Efficiency

One of the principal considerations in maximizing bus efficiency revolves around the unique aspects of an optical transmission. Intensity modulation of an optical carrier provides a unipolar transmission channel, unlike electrical current transmission over wire which may be bipolar. Unipolar signaling causes a Direct Current (dc) shift between signal-on, and signal-off states, which will disturb the operation of conventional receiver amplifiers having ac coupling until the interstage coupling capacitances have had time to accommodate the shift. A similar dc shift occurs between small and large signals.

Thus to avoid a long settling time at the start of messages, receivers designed for data bus application either have a short ac coupling time constant to minimize the disturbance time, or employ dc coupling. If dc coupling is used, more complex circuitry is required for setting the data decision threshold for the received waveform. The shift in average power between signal and no-signal states also complicates laser optical source power stabilization, which is normally accomplished using average power feedback control.

5.12. Transmission Losses

Optical bus configurations have considerable and somewhat undefined transmission losses between source and detector, resulting from the couplers and connectors. When combined with source power and detector sensitivity variations, this gives rise to an uncertain received power level. A high-gain, wide-dynamic-range receiver is required. Since time is a premium, long term averaging is undesirable. Alternate methods for rapidly accommodating the dynamic range are required. This is a major concern for optical data bus receiver designers.

5.13. Receiver Losses

Three receiver types which provide simple, instantaneous adjustment to message levels are known. In the symmetrical clamp receiver, all signals are bit-by-bit clamped to the same low level. After amplification, decisions about the validity of data are based on the data being above a fixed threshold. Good dynamic range is achievable and start-of-message time constant delays are eliminated. This is not the case with conventional linear or limiting receivers. The technique operates well up to bit rates around 50 Million bits per second (Mbps). Above this level implementation problems arise. (The upper bit rate limit may be extended using lower capacitance hybrid construction.) This technique is a leading candidate for receivers operating at lower data rates.

A second fast response scheme uses a dc coupled receiver (to avoid ac coupling time constants) and a bit-by-bit adaptive threshold decision. The technique is ideally suited to very high data rate reception but dynamic range is limited by amplifier design. Optimum performance is limited by dc offset in the amplifier, which may be a limitation for wide temperature range operation.

High-bit rate reception may also be handled efficiently with a high-pass filtering receiver when the signal is any biphase code or other reduced low-frequency content code. This is because required coupling capacity time constants become small compared to the fixed bus intermessage dead time resulting from propagation delays. Appropriate filters have been designed with a linear phase response in the stop band, providing an intermessage response time as low as six bit times for Manchester coded data.

Conventional point-to-point system optical receivers have well defined sensitivity limits that may be calculated from the thermal and shot noise of the devices. For data bus receivers, a number of compromises in design are necessary to achieve fast response to messages. These generally result in less sensitivity. Similarly, wider dynamic range may generally be achieved in a receiver, which has a long period to adjust to changes in signal level, versus a receiver that is required to adjust almost instantaneously.

The receiver sensitivity is affected largely by the type of photodetector and by preamplifier design. A silicon avalanche photodetector offers greatest sensitivity (at 0.85 μm); therefore, preamplifier design is less critical. At 0.85 μm, a silicon PIN diode with a sensitive preamplifier has approximately 10 dB less sensitivity.

5.14. Transmitter Losses

For relatively low rate transmission, i.e., <10-50 Mbps, little difficulty exists in designing a transmitter circuit using LEDs. Data modulation may be dc coupled through to the LED and any data format or message length may be accommodated. Very high data rate transmission requires the use of a semiconductor laser diode to achieve the required modulation rate and sufficient launched optical power to provide reliable reception after the transmission losses. Lasers require a more complex driver circuit to ensure that the drive current compensates for temperature and aging of the source, and is correctly

prebiased during transmission to avoid data distortion resulting from lasing turnon delay. Effective compensation of the drive current requires feedback control of the launched signal. Feedback control commonly operates by stabilizing the average transmitted power in continuous transmission point-to-point systems. With the burst nature of transmission in a bus system, averaging is not as convenient and requires a long preamble for the laser power to initially stabilize.

Any data bus transmitter design must include an override control, which provides a positive curtailment of transmission in the event of a nonself-correcting fault. An external timeout circuit or protocol function controls this override function.

5.15. Optical T/R Power Margin

A key element in the design and optimization of any fiber optic link including a data bus is the system power budget analysis. Such an analysis is important not only to ensure that there is adequate optical power at any given receiver under all conditions, but also to ensure, particularly in a data bus, that there is not too much optical power at any given receiver.

There are three basic elements to a power budget analysis: system losses, optical source output power, and optical receiver sensitivity. The maximum allowable system loss can be derived for a transmitter combined with a realizable receiver. Output powers of -6 Decibels per meter (dBm) can be achieved with high radiance LEDs coupled to 100 μm core fiber with an Numerical Acceptance (NA) of 0.3.

5.16. Topology Analysis

Using the practical technical limits as discussed in the previous sections, an analysis of various fiber optic data bus topologies or configurations was performed to evaluate the number of terminals possible at various data rates.

The topologies examined included:

- Linear.

  Inbound-outbound (loop or ring).
  Bidirectional (open-ended).
  Active.

- Star.

  Transmissive.
  Reflective.
  Star-star.
  Active star-star.

- Hybrids.

  Star-loop.
  Loop-star.

Since active stars and active rings are essentially point-to-point links, bus losses are not the limiting factor on the number of terminals, nor is dynamic range a factor in receiver design.

For this initial, first-order analysis, the best case performance for splices, connectors, and couplers was assumed. This approach "brackets" the problem by defining the best possible performance of a particular topology implemented with currently available (or nearly available) technology.

A passive transmissive star bus is the most efficient topology, because the power from any transmitter is distributed evenly among all receivers. In addition, there is only one coupler-insertion loss between any given T/R pair.

The principal disadvantage of a bus with a single star is that the cables from all T/R modules must be run to the star. In an aircraft, this increases the initial installation cost due to the increased number of bulkhead penetrations required. In addition, there is little flexibility for adding new terminals at arbitrary locations. One solution to this is to provide a distributed bus topology such as a star-star or a star-linear topology. The performance of the star-star topology can be easily improved by adding a single repeater (or two for redundancy) at the central star.

Two hybrid topologies combining stars with a linear bus concept were investigated because they provided four separate nodes with the potential for improved performance over a simple linear bus. The first is a star-loop; the second a loop-star.

Initial analysis of these revealed very little reduction in bus loss over a simple linear loop, and therefore a detailed analysis was not performed. The loop-star or distributed star topology can be effective, however, with active repeaters between the stars.

The only viable passive topology for 128 terminals is a star; however, an active linear bus, active star, or active star-star are viable implementations for 128 terminals at 300 Mbps. The active star-star appears optimal because it:

- Minimizes cabling and bulkhead penetrations with four (or more) nodes for concentrated location of terminals while enhancing flexibility.
- Minimizes number of repeaters and therefore cost and maintenance.
- Prevents a single point failure that will disable the entire bus.
- Allows use of star couplers with 6-32 ports, reducing the cost and increasing the performance and reliability of the couplers.

5.17. Fiber Optic Network Based Losses

A typical set of requirements for an avionics multiplexed bus of a commercial transport could include anywhere from 32 to 128 terminals. Data rates could be in the 10-100 MHz (or Mbps) range. The bus probably would be bidirectional, using a broadcast type mode in which any terminal might transmit data to any other terminal in the network. Various topologies for such a bus have been discussed earlier; however, the most probable topology for such an architecture would be a star-coupled topology due to the fact that it can be implemented without the use of active repeaters. This results in higher reliability, lower maintenance, and reduced losses in the optical path.

Table 5.17-1 presents a typical loss budget calculated for an approximately 60-terminal, star-coupled transmission network. From this table, it can be seen that the bus network will require high optical output from the transmitter and high receiver sensitivity to assure that the integrity of the data is maximized. To insure the high integrity, the bus optical components will have to be selected to be consistent with simple straightforward system design at both the transmitter and receiver ends.

TABLE 5.17-1. STAR-COUPLED NETWORK LOSSES

COMPONENT	MINIMUM LOSS	MAXIMUM LOSS	COMMENT
Fiber	0.0 dB	1.0 dB	50 m. terminal to star maximum, 5 dB/km
Connectors	0.4 dB	8.0 dB	.1 dB to 1.0 dB each, 4 to 8 total terminal to terminal
Star Coupler	17.1 dB	21.1 dB	Typical
TOTAL	17.5 dB	30.1 dB	
Optical Dynamic Range: 12.6 dB			

APPENDIX A - MODULATION AND SIGNALING METHODS

Modulation changes the carrier signal's characteristics. An unmodulated carrier is a periodic signal containing no information. Modulation imposes information on the carrier. Analog carriers can be represented by the sine wave:

$$a = A \sin (2\pi f t + \phi)$$

where

a = voltage at time t
A = maximum amplitude
f = frequency
ϕ = phase

The carrier's amplitude, frequency, and phase can all be modified. Amplitude Modulation (AM) refers to modification of the carrier's amplitude for information transfer. Frequency Modulation (FM) refers to modification of the carrier's frequency. Phase modulation (ϕM) refers to modification of the carrier's phase.

- Frequency Modulation.

  FM is not commonly applied to information transfer in digital avionics systems.

- Amplitude Modulation.

  The simplest form of AM varies the magnitude of the signal (A) from zero to a fixed peak-to-peak voltage. The zero level represents a binary zero and the maximum voltage represents a binary one. Figure A-1 (Held, 1986) shows the correspondence between the AM signal and the digital data it represents. AM by itself is normally used for very low data rates. For higher transmission rates, AM may be combined with ϕM.

- Phase Modulation.

  ϕM varies the phase of the carrier signal. Figure A-2 shows an example of ϕM.

In terms of bus transmission of digital data, modulation techniques and signaling methods refer to the type of coding used to transmit information through the bus. All of the subsystems connected to the bus must use the same coding method. There are a number of codes that can be used, which differ with respect to advantages and disadvantages. This brief section of the tutorial looks at the coding methods likely to be used by aviation-oriented data buses.

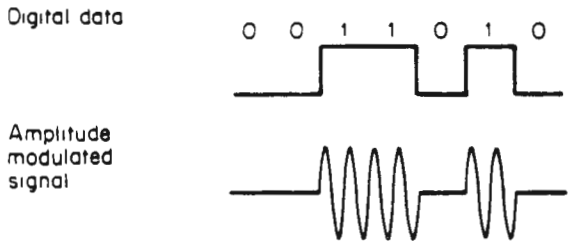


FIGURE A-1. AMPLITUDE MODULATION
(Held, 1986)

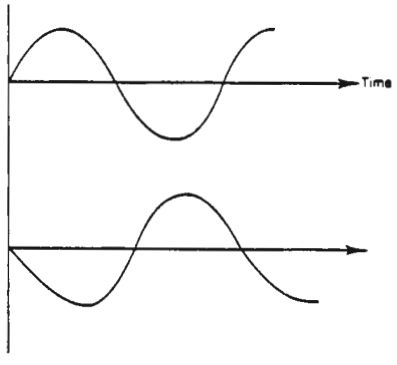


FIGURE A-2. PHASE MODULATION

Code is categorized as either single-density or double-density. Double-density codes include Delay Modulation (DM), Modified-Frequency Modulation (MFM), Group Code Recording (GCR), Zero Modulation (ZM), Enhanced Non-return to Zero (ENRZ), and Randomized Non-return to Zero (RNRZ), most of which are not appropriate for aircraft data buses. An exception is DM coding.

DM coding is complex both in hardware and functional characteristics. Its functional complexity stems from the requirement that there must be at least one signal transition for every two-bit interval and no more than one transition per bit. Because of this, some synchronization capability is still provided, but at a lower modulation and bandwidth requirement than required for the single-density codes. Consequently, it cannot be used effectively with buses requiring high data transfer rates.

The most common single-density codes are Non-return to Zero (NRZ); NRZ-Inverted (NRZ-I), which is sometimes referred to as NRZ-M; NRZ-Dual Level (NRZ-L) ratio; and biphase. Biphase covers several subcategories: Manchester II, FM, and Phase Encoding (PE). Since these single-density codes are self-clocking, the clock is represented by level transitions, which take place even if data transitions do not. NRZ, Return to Zero (RZ), and biphase are categorized by the suffixes L (level), M (mark), and S (space). An -L suffix indicates that data are represented by different levels; -M and -S suffixes indicate that data are represented by the presence or absence of transitions. In codes designated -M, a "1" (defined as a mark) occurs with a level transition; "0" is no transition. The converse is true for codes designated -S.

NRZ codes remain constant throughout a bit interval and either use absolute values of the signal elements or differential encoding where the polarity of adjacent elements is compared to determine the bit value. This method lacks independent synchronization and error-detection capabilities but provides efficient usage of the bandwidth.

RZ codes return to a binary "0" level at one-half the bit interval for binary "1" signals, requiring a higher bandwidth for an equivalent NRZ data rate.

Biphase codes include the Manchester and Differential Manchester techniques. At least one signal transition is required every bit interval, providing a self-clocking mechanism. The absence of the expected transitions may also be used for error detection. With two possible transitions per bit time, there is a corresponding increase in the bandwidth required.

Multilevel binary encoding schemes use more than two signal levels. One method is bipolar, which has no synchronization capability but does provide some error detection by requiring successive binary "1"s to be of opposite polarity.

Most of the aircraft data buses use biphase codes like Manchester II, which is self-clocking since the data and clock are included in a single serial data stream. In clocked systems, the clock defines the size of the data-bit cell; however, in nonself-clocking systems, speed fluctuations cause the data track to vary relative to the speed of the clock. Over a period of time, the clock will appear to speed up or slow down and will improperly define a data bit cell.

With self-clocking, everything stays synchronized. The mid-bit transitions of Manchester code help detect transmission errors.

Table A-1 summarizes the major features for some of the popular single and double-density codes. The encoded waveforms in figure A-3 illustrate patterns for an identical binary input produced by each form of encoding.

TABLE A-1. IMPORTANT PARAMETERS OF ENCODING TECHNIQUES

Code	Bandwidth f_l	Bandwidth f_h	Storage Efficiency	Self-Clocking	dc Presence	Band Speed Ratio	Preamble for Synch-ronization
NRZ	0	0.5f*	N.A.	No	Yes	N.A.	No
RZ	0.25f	1.0f	50%	No	Yes	4	Yes
NRZ-S	0	0.5f	80%	No	No	9	No
Ratio	0.75f	1.5f	33%	Yes	No	2	No
Biphase	0.5f	1.0f	100%	No	Yes	2	Yes
Double-density	N.A.	0.5f	100%	No	Yes	2	Yes

*Bandwidth in terms of the fundamental frequency of the data rate.
N.A. = Information Not Available

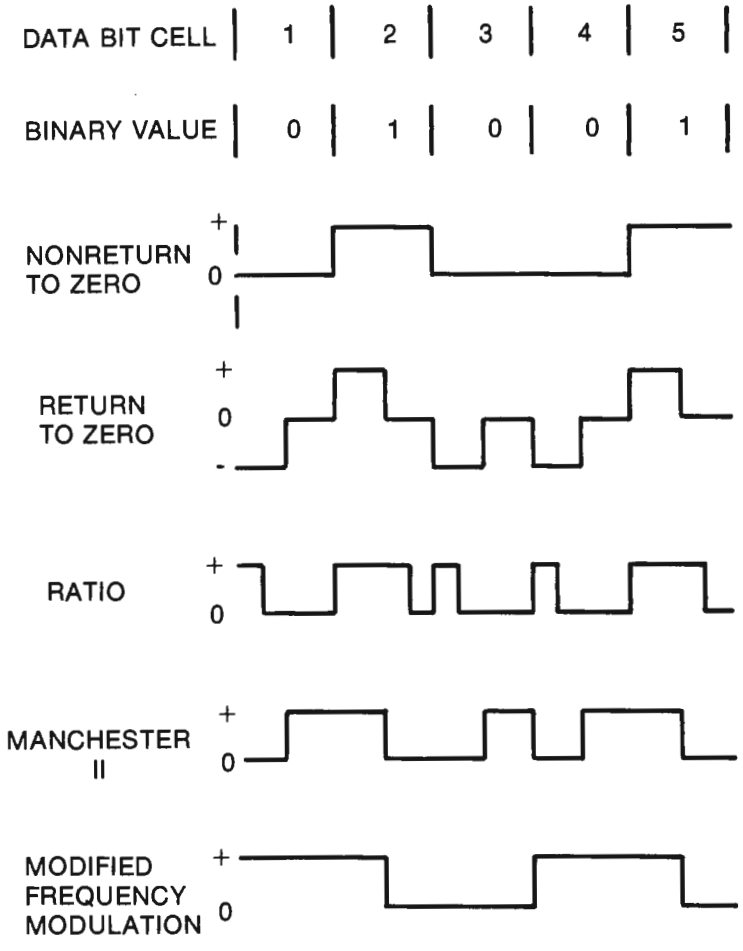


FIGURE A-3. WAVEFORMS GENERATED BY FIVE DIFFERENT ENCODING SCHEMES FOR A FIXED BINARY SIGNAL

BIBLIOGRAPHY

ARINC Specification 429-5, Mark 33 Digital Information Transfer System, April, 1981.

ARINC Specification 629 (Multi-Transmitter Data Bus), Part 1 - Protocol Description and Part 2 - Label and Word String Definitions, Drafts dated September 1987.

Aydin Vector. MIL-STD-1553: Theory and Application, 2/84-2M, Newtown, PA, 1984.

Eldredge, D., and E. F. Hitt, Digital System Bus Integrity, FAA Technical Report, DOT/FAA/CT-86/44, Atlantic City Airport, New Jersey, March 1987.

General Aviation Manufacturers Association Specification, Washington, DC, February 1986.

Held, G. Data Communications Networking Devices. John Wiley, New York, 1986.

ILC Data Device Corporation. MIL-STD-1553 Designer's Guide, New York, 1983.

Jennings, R. G. "Avionics Standard Communications Bus: Its Implementation and Usage," Proceedings of the AIAA/IEEE Seventh Digital Avionics Systems Conference, Fort Worth, Texas, October 1986.

Kiernan, J. J., and J. R. Sims, "Standardization in Military System Architecture," Standardization in Military Avionics Systems Architecture, Proceedings of an IEEE/AES-S Seminar, Dayton, Ohio, November 28, 1979.

MIL-STD-1553B, Aircraft Internal Time Division Command/Response Multiplex Data Bus, Specification, 1978.

McSharry, M. E. Architecture Tradeoffs for Bus-Integrated Control Systems, NADC-82041-60, Aircraft and Crew Systems Technology Directorate, Naval Air Development Center, Warminster, PA, October, 1983.

Rich, B. A., et al., Multibus Avionic Architecture Design Study (MAADS), AFWAL-TR-83-1141, AFWAL, Wright-Patterson Air Force Base, OH, October 1983.

Society of Automotive Engineers, SAE High-Speed Ring Bus Standard, (Draft) March, 1987.

SAE Linear Token Passing Bus Standard, (Draft) April, 1987.

Schmitter, E. J., and P. Baues, "The Basic Fault-Tolerant System," IEEE Micro, 1984.

SCI Systems, Inc. The MIL-STD-1553 Multiplex Applications Handbook, Huntsville, AL, February 1982.

Shaw, J. L., H. K. Herzog, and K. Okubo, "Digital Autonomous Terminal Access Communication," Proceedings of the AIAA/IEEE Seventh Digital Avionics Systems Conference, Fort Worth, Texas, October 1986.

Integrated Application of Active Controls (IAAC) Technology to an Advanced Subsonic Transport Project - ACT/CONTROL/GUIDANCE SYSTEM STUDY - VOLUME I. Final Report, NASA Contractor Report 165963, NASA Langley Research Center, Hampton, VA, December 1982.

ADDRESSING CAPACITY. The number of components addressable by the protocol used on a given data bus.

ASYNCHRONOUS MESSAGES. Electronic signals with transmission times that are not known a priori. These may include priority signals requiring immediate access to the bus.

BIT TIME. The time it would take to transmit one bit. Usually this is "blank" time when nothing is being transmitted. One nth of the bus speed (i.e., on a 1 kHz bus, the bit time is 10^{-3} seconds).

BLOCK TRANSFER. A data transfer mode allowing the transfer of variable length data blocks.

BROADCAST CAPABILITY. The capacity to transmit messages to all terminals simultaneously.

CAT IIIa LANDING. One of several landing categories defined in FAR 91. CAT IIIa implies the need for an instrument landing approach.

CENTRAL CONTROL. Control from one master, whether stationary or non-stationary.

COMMAND/RESPONSE. "Operation of a data bus system such that remote terminals receive and transmit data only when commanded to do so by the controller." (MIL-STD-1553 Designer's Guide, 1983, p. II-3).

CONTENT ADDRESSING. The system of identifying message recipients based on information embedded in the message. This is in contrast to destination terminal addresses.

DATA BUS. A system for transferring data between discrete pieces of equipment in the same complex.

DATA LATENCY. The delay from the time when a piece of information becomes available at a source terminal to the time it is received at the destination.

DATA LINK ASSURANCE OF RECEIPT. The guarantee of good data through the data link level.

DETERMINISTIC. A system where all parameters are known, as opposed to a statistical system where the outcome is subject to the laws of probability.

DISTRIBUTED CONTROL. Concurrent control from multiple points in the data bus system.

FAULT CONTAINMENT. The capacity of a system to prohibit errors and/or failures from propagating from the source throughout the system.

FAULT DETECTION. The capacity of a system to determine the occurrence of erroneous operation.

FAULT ISOLATION. The capacity of a system to isolate a failure to the required level so it can reconfigure.

FAULT TOLERANCE. The capability to endure errors and/or failures without causing total system failure.

INITIALIZATION. Setting the beginning parameters and values on system power-up. For redundant systems this includes setting the initial configuration of the system.

LABELED ADDRESSING. The system of identifying message recipients based on labels. This is in contrast to destination terminal addresses.

MESSAGE STRUCTURE. The organization of both protocol and data information in a message.

MONITORABILITY. The capacity of the protocol to be viewed passively to allow observation of the dynamics of the protocol.

NETWORK CONTROL STRATEGY. The solution proposed by the designer in addressing his specific problem (design flexibility).

NUMERICAL APERTURE. The angle of acceptance of light from a light source for a given fiber optic cable.

PARAMETERIZATION CAPABILITY. A measure of how well the attributes of the protocol can be described by parameters.

RECONFIGURATION. The capacity of a system to rearrange or reconnect the system elements or functions.

SYNCHRONOUS MESSAGES. Messages transmitted at a known a priori sequence and time or time interval.

SYSTEM INTEGRITY. The degree to which a system is dependable.

TESTABILITY. A measure of how well the protocol supports completeness of testing and the protocol's ability to produce repeatable or predictable results.

THROUGHPUT. The productivity of a data processing system as expressed in computing work per minute or hour.

ACRONYMS AND ABBREVIATIONS

ac	Alternating Current
AE	Avionics Equipment
AES-S	Aerospace and Electronic Systems Society
AFWAL	Air Force Wright Aeronautical Laboratory
AHRS	Attitude Heading Reference System
AIAA	American Institute for Aeronautics and Astronautics
AM	Amplitude Modulation
ARINC	Aeronautical Radio, Incorporated
ASCB	Avionics Standard Communications Bus
BIR	Benchmark Information Rate
BIT	Built-In Test
BITE	Built-In Test Equipment
BIU	Bus Interface Unit
bps	Bits Per Second
CCITT	Consultative Committee for International Telephone and Telegraph
CD	Collision Detection
CR	Command Response
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Multiple Access
CSMA/CD	Carrier Sense Multiple Access/Collision Detection
CT	Technical Center (designation used in FAA report numbering scheme)
DADC	Digital Air Data Computer
DATAC	Digital Autonomous Terminal Access Communication
dB	Decibel
dBm	Decibels per meter
dc	Direct Current
DITS	Digital Information Transfer System
DM	Delay Modulation
DMA	Direct Memory Access
DOT	Department of Transportation
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
EMUX	Electrical Multiplex
ENRZ	Enhanced Non-return to Zero
FAA	Federal Aviation Administration
FADEC	Full Authority Digital Engine Controller
FAR	Federal Acquisition Regulation
FCC	Flight Control Computer
FM	Frequency Modulation
GaAs	Gallium Arsenide
GAMA	General Aviation Manufacturers' Association
GCR	Group Code Recording
HDLC	High-Level Data Link Control
HSRB	High-Speed Ring Bus

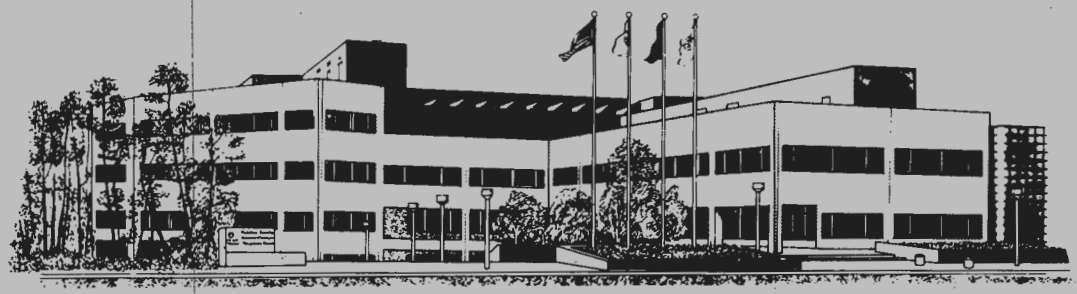
IEEE	The Institute for Electrical and Electronics Engineers, Incorporated
INS	Inertial Navigation System
K	Thousand
KHz	Kilohertz (kilocycles per second)
km	kilometer
LED	Light Emitting Diode
LRC	Longitudinal Redundancy Check
LRU	Line Replaceable Unit
LSB	Least Significant Bit
LTPB	Linear Token Passing Bus
m	meter
μm	Micrometer
M	Million
MAADS	Multibus Avionic Architecture Design Study
Mbps	Million bits per second
MDT	Mean Down Time
MFM	Modified-Frequency Modulation
MHz	Megahertz
MIL-STD	Military Standard
ms	Millisecond
MSB	Most Significant Bit
MTBCF	Mean Time Between Critical Failures
MTTR	Mean Time To Repair
NA	Numerical Acceptance
NADC	Naval Air Development Center
NAECON	National Aerospace & Electronics Conference
NRZ	Non-return to Zero
NRZ-I	Non-return to Zero Inverted
NRZ-L	Non-return to Zero Dual Level
nsec	Nanosecond
PAS	Pilot Assist System
PE	Phase Encoding
ϕM	Phase Modulation
RAT	Ring Admittance Timer
RNRZ	Randomized Non-return to Zero
RZ	Return to Zero
SAE	Society of Automotive Engineers
str	string
TDM	Time Division Multiplex
THT	Token-Holding Timer
T/R	Transmitter/Receiver
VHSIC	Very High-Speed Integrated Circuits
VLSI	Very Large Scale Integration
VLSIC	Very Large Scale Integrated Circuits
VRC	Vertical Redundancy Check
ZM	Zero Modulation

U.S. Department of Transportation
Federal Aviation Administration

DOT/FAA/CT-88/10

HANDBOOK-VOLUME II
DIGITAL SYSTEMS VALIDATION

CHAPTER 7
ANALYTICAL SENSOR REDUNDANCY



PREPARED BY:

COMPUTER RESOURCE MANAGEMENT, INC.
950 HERNDON PARKWAY, SUITE 360
HERNDON, VIRGINIA 22070

PREPARED FOR:

FEDERAL AVIATION ADMINISTRATION
TECHNICAL CENTER
ATLANTIC CITY INTERNATIONAL AIRPORT, NEW JERSEY 08405

NOTICE

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The United States Government assumes no liability for the contents or use thereof.

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the objective of this report.

TABLE OF CONTENTS

Section		Page
1.	INTRODUCTION	7-1
	1.1. Control Law Modification	7-2
	1.2. Fault Tolerant Design	7-2
	1.3. Scope	7-4
2.	OVERVIEW	7-5
	2.1. Analytical Redundancy Building Blocks	7-5
3.	ANALYTICAL REDUNDANCY - DESIGN APPROACH	7-11
	3.1. Observer/Signal Blender Filter	7-16
	3.2. Diagnostic Kalman Filters	7-16
	3.3. Super-Diagnostic Kalman Filter Design	7-16
	3.3.1. Multiple Hypothesis Tests	7-17
	3.3.2. Parameter Identification	7-17
	3.3.3. Generalized Likelihood Ratio Methods	7-17
	3.3.4. Modified Filter Design	7-18
	3.3.5. Jump Process Estimation	7-18
	3.4. Redundancy Management	7-18
4.	ANALYTICAL REDUNDANCY - RELIABILITY	7-21
5.	ANALYTICAL REDUNDANCY - FAULT DETECTION AND ISOLATION	7-23
	5.1. Multiple Trip Monitor	7-23
	5.2. Sequential Likelihood Ratio Tests	7-25
	5.3. Fault Modeling	7-27
6.	SUPPLEMENTAL WORKED EXAMPLES	7-29
	6.1. Altitude Kinematics	7-29
	6.2. Translational Kinematics	7-32
	6.3. Stability Augmentation System	7-33
BIBLIOGRAPHY		7-37
GLOSSARY		7-40
ACRONYMS		7-42

LIST OF ILLUSTRATIONS

Figure		Page
2-2	ROTATIONAL RELATIONSHIPS USED FOR ANALYTICAL REDUNDANCY	7-7
2.1-1	ANALYTICAL REDUNDANCY CONCEPT	7-8
2.1-2	ANALYTICAL SENSOR REDUNDANCY FAULT DETECTION LOGIC	7-9
3-1	EXAMPLES OF FAULT DETECTION USING DIAGNOSTIC FILTERS AND INTERCHANGEABLE SENSORS	7-12
3-2	EXAMPLES OF FAULT DETECTION USING DIAGNOSTIC FILTERS AND NONINTERCHANGEABLE SENSORS	7-13
3-3	EXAMPLE OF FAULT DETECTION WITH REVERSION MODES	7-15
5.1-1	MULTIPLE TRIP FAULT DETECTION MONITOR	7-24
5.1-2	SEQUENTIAL LIKELIHOOD FAULT DETECTION MONITOR	7-27
6.1-1	THIRD ORDER VERTICAL COMPLEMENTARY FILTER FOR ALTITUDE KINEMATICS	7-30
6.1-2	POSSIBLE REDUNDANCY MANAGEMENT FOR ALTITUDE KINEMATICS FAULT ISOLATION	7-31
6.2-1	EXAMPLE FILTER USED FOR TRANSLATIONAL KINEMATICS REDUNDANCY	7-34
6.3-1	EXAMPLE FILTERS USED FOR STABILITY AUGMENTATION SYSTEM	7-36

LIST OF TABLES

Table		Page
3-1	SENSOR REDUCTION COMPARISONS FOR ANALYTICAL REDUNDANCY BUILDING BLOCKS	7-14
4-1	SYSTEM RELIABILITY	7-22

1. INTRODUCTION

The advent of modern digital controls has greatly changed the control technology and hardware implementation of aircraft flight control. These Digital Flight Control Systems (DFCS) provide enhanced performance for stability and control augmentation over an extremely wide range of flight envelope parameters. They also offer improved capabilities to operate in differing aircraft configurations from takeoff with wheels and flaps down to a high speed clean configuration with low drag, thrust, and low g forces. In addition to the performance enhancements in Digital Flight Control (DFC), the DFC computer has the capability to perform other computations to effect improvements in reliability and fault tolerance. Elaboration of these later features is the major thrust of this portion of the handbook.

The DFC improves reliability due to two factors. The first is the increased reliability of digital computer hardware relative to older technology; this enhancement leads to tens of thousands of hours between failures. The second factor is the increased reliability of the sensors and actuators which leads to several hundreds of hours between failures. The principal factor affecting DFC system reliability is reliability of the sensors and actuators.

The overall reliability required in today's flight control systems dictates the need for 10^{-9} probability of failure in cases where stability augmentation is needed for safe flight. This requires an increase in the reliability of the DFC components. The reliability of DFC computers may be achieved by utilizing dual processors, error detection, and fault tolerant designs. An increase in reliability of sensors and actuators may be achieved by means of redundant elements (dual or triple).

Assessment of this situation leads to a conclusion that redundant actuators and linkages are an adequate solution for actuation, and that DFC computers are adequate. The term Analytical Redundancy (AR) refers to the scheme for increasing the reliability of sensors by using the DFC computer for Fault Detection (FD) and isolation and for analytically determining estimates of sensor values.

Reliable DFC systems are burdened with large numbers of sensors which are required for both quality and redundancy. The demand for control quality has expanded the types of sensors used in basic augmentation functions. The yaw damper currently requires sensor data concerning attitude, angle of attack, and airspeed. The pitch damper requires a complex multifunction control law along with data from several sensors to drive multiple surfaces. The complexity of the sensors has improved quality; however, designs that meet overall flight control system reliability requirements lead to triple and quadruple replication of system components.

High levels of fault tolerance are mandatory for flight-critical control systems. Current Fly-By-Wire (FBW) systems utilize redundant computers,

actuators, and sensors to achieve the necessary fault tolerance. Fault isolation (FI) is determined by some type of comparison or voting between like sensors. A one out of two sensor failure cannot be isolated by comparison techniques. In order to improve the reliability of a redundant set of sensors, one or more of the following must be implemented: increase the reliability of the existing sensors; add additional like sensors; synthesize the needed information; or isolate the remaining sensor if a failure occurs at the duplex level.

A large amount of sensor hardware is costly from a system standpoint. Sensors may act as antennas to increase electromagnetic susceptibility. The overall aircraft design may be constrained due to a large number of sensors. Therefore, there is a strong motivation to reduce the number of sensors. Significant reductions appear to be possible through a variety of methods. Recognized techniques for sensor reduction are control law modification and fault tolerant design. These methods may be used in combinations to optimize the number of sensors necessary for a particular application.

1.1. Control Law Modification

Control law modification ensures that operational requirements can be met with the minimum number of sensors and a better control system design. The control laws are modified to optimize the use of sensed data utilizing the analytical relationships which exist between sensors. The flight controller uses fewer sensed values in the basic single channel control laws, yet maintains the same performance as the controller which utilized all the sensor data. These modifications are application dependent.

Signal synthesis is one example of control law modification. The number of sensors required is reduced by synthesizing a signal or group of signals from a reduced set. The elements used to perform the synthesis are Kalman filters or Luenberger observers. Reduction in the number of sensors is a trade-off against increased complexity of the control laws; this is a critical factor for analog systems. The trade-off in digital systems occurs in system performance. The bandwidth and noise content of the synthesized signal must be equivalent to the sensor eliminated. These techniques must be applied with care because of potential physical limitations which cannot be practically overcome.

Integrated flight management is another example of control law modification. This method works on the principle that subsystems which use common sensor types are combined. An example is the combination of the navigator and the flight control. The signal quality derived from the navigator is high and the data could easily be used by the flight control. Issues here are cost (since the navigator is normally not flight critical) and multiple failures.

1.2. Fault Tolerant Design

The following fault tolerant design techniques exist to meet reliability requirements using a minimum number of sensors. These techniques use fewer sensors to achieve the same level of redundancy as a quad-redundant system.

- Skewed and special sensors.

- Sensor integration.
- In-line sensor monitoring.
- Analytical sensor redundancy.

A skewed sensor arrangement significantly reduces the number of sensors required for Redundancy Management (RM). Skewed sensors do not have to be placed on orthogonal axes. The x, y, and z components of the sensed quantity can be determined using an appropriate coordinate transformation matrix. If one of the skewed sensors fails, the three orthogonal components can still be determined with another sensor and its transformation matrix.

A quad-redundant dual fail-operative system with orthogonal gyros in three axes requires twelve gyros. The same properly configured system with skewed gyros requires only six to maintain the same redundancy requirements. Skewing accelerometers can similarly reduce the number of sensors from twelve to six. Potential limitations are differences in scale and resolution requirements for each axis. For the skewed configuration, each gyro must have the same specifications. Depending upon the gyro chosen, either the signal quality may be limited or the instrumentation cost may be greater. Weinstein (1978) and Kaniuka, et al., 1982 show adequate performance for this technique.

Special sensors measure linear combinations of variables. The measurements are skewed in measurement space rather than geometrically, i.e., the sensor measures both acceleration and body rate. A conventional quad-redundant system could be replaced by five special sensors. These sensors are typically unique to the application and have a higher cost. A multipurpose air data sensor which measured pitot pressure, static pressure, angle of attack, and angle of sideslip was implemented in the system described by Weinstein (1978) and Kaniuka, et al., 1982.

Subsystem sensor integration can also be used to reduce the number of sensors. This technique uses data from subsystems which are not functionally related for monitoring and tie-breaking. An example is on the space shuttle where derived rates from the navigation system are used in the primary flight control system for voting.

Fault tolerant design can be achieved through in-line monitoring. The state-of-the-art for servos and computers is near 99 percent self-test confidence. In-line self-test feasibility for sensors is limited for several reasons. The sensor input is unknown except for special test signals. There is no way to test for sensor installation errors. Any sensor self-test adds considerable complexity which may be undesirable. Typically, a self-test can only detect very large faults, particularly in a dynamic environment.

Analytical Redundancy provides a means for substituting software algorithms for hardware sensors. Sensors are eliminated by analysis using known relationships between sensed values. Various simulation studies have shown the effectiveness of utilizing AR to increase system reliability and to reduce the amount of sensor hardware.

1.3. Scope

This document details the uses and limitations of analytical sensor redundancy. Section 2 gives an overview of AR. The physical relationships which can be utilized are written in equation form. The basic concept and building blocks are discussed. Section 3 describes various algorithms and design tools used to implement the filters used in AR processing. Its feasibility and performance are discussed in section 4. Algorithms and methods used for FD and isolation are detailed in section 5. Examples of the filters, RM, and sensor relationships for three cases where analytical sensor redundancy is easily implemented are given in section 6.

2. OVERVIEW

The basic idea of AR is to use known geometric and dynamic relationships between different sensors in order to detect and isolate sensor failures. These relationships can be defined in translational and rotational equations as shown in figures 2-1 and 2-2. Figure 2-2 also shows a diagram of the Euler angle relationships.

Cunningham and Hartmann (1977) describe the application of these mathematical relationships to eleven different measured quantities, normal acceleration, lateral acceleration, angle of attack, true airspeed, altitude, P, Q, R, $\emptyset$, θ, and ψ.

Four categories of AR were utilized by Deckert, et al., 1978. Translational kinematics redundancy exists between the integrated accelerometer outputs, vertical and rate gyros, and the air data sensor outputs. Translational dynamics redundancy uses the aerodynamic forces on the aircraft measured by the accelerometers and stored coefficients to relate to aerodynamic forces calculated from air-data sensor data. Rotational kinematics redundancy relates the integrated outputs of the rate gyros and the vertical and directional gyro outputs. Altitude Kinematics (AK) redundancy is defined as the relationship between the altitude measured by the altimeter and the computed altitude from a double integration of the accelerometer and vertical gyro output data. These relationships are defined in the equations in figures 2-1 and 2-2. Examples of altitude and translational kinematics redundancy processing are given in sections 6.1 and 6.2.

2.1. Analytical Redundancy Building Blocks

Many specific approaches to AR have been developed, ranging from simple signal blenders to complex combinations of Kalman filters.

In general, AR is approached using one of two basic building blocks. A Diagnostic Filter (DF) is an algorithm which processes data from a family of N functionally related sensors in order to estimate signals and to assess the functioning of the sensors. It outputs signal estimates and an error flag which indicates that all sensors are functioning properly or that one of them has failed. A Super-Diagnostic Filter (SDF) performs the functions of a DF and can additionally isolate the specific faulted sensor. One can use assemblies of DFs to construct an SDF through truth table logic.

The concept of AR and associated FD are shown in figures 2.1-1 and 2.1-2.

Translational Equations

Inertial Velocity

$$\dot{X}_e = \cos\theta \cos\tau\, U + (\sin\phi \sin\theta \cos\tau - \cos\phi \sin\tau)V + (\cos\phi\sin\theta \cos\tau + \sin\phi \sin\tau)W$$

$$\dot{Y}_e = \cos\theta \sin\tau\, U + (\sin\phi \sin\theta \sin\tau + \cos\phi \cos\tau)V + (\cos\phi \sin\theta \cos\tau - \sin\phi \cos\tau)W$$

$$\dot{Z}_e = -\sin\theta\, U + \sin\phi \cos\theta\, V + \cos\phi \cos\theta\, W$$

Acceleration

$$\dot{U} = A_x - g \sin\theta - QW + RV$$

$$\dot{V} = A_y + g \cos\theta \sin\phi - RU + PW$$

$$\dot{W} = A_z + g \cos\theta \cos\phi - PV + QU$$

where X_e, Y_e, Z_e = earth referenced forward, lateral, and vertical positions
U, V, W = forward, lateral, and vertical body velocities, respectively
A_x, A_y, A_z = forward, lateral, and vertical body accelerations
g = acceleration due to gravity
θ, ϕ, τ = Euler pitch, roll, and yaw angles
P, Q, R = body roll, pitch, and yaw rates

FIGURE 2-1. TRANSLATIONAL RELATIONSHIPS USED IN ANALYTICAL REDUNDANCY

Rotational Equations

Euler Rates

$$\dot{\phi} = P + (Q \sin\phi + R \cos\phi) \tan\theta$$

$$\dot{\theta} = Q \cos\phi - R \sin\phi$$

$$\dot{\tau} = (Q \sin\phi + R \cos\phi) \sec\theta$$

Body Rates

$$P = \dot{\phi} - \dot{\tau} \sin\theta$$

$$Q = \dot{\theta} \cos\phi - \dot{\tau} \sin\phi \cos\theta$$

$$R = \dot{\tau} \cos\phi \cos\theta - \dot{\theta} \sin\phi$$

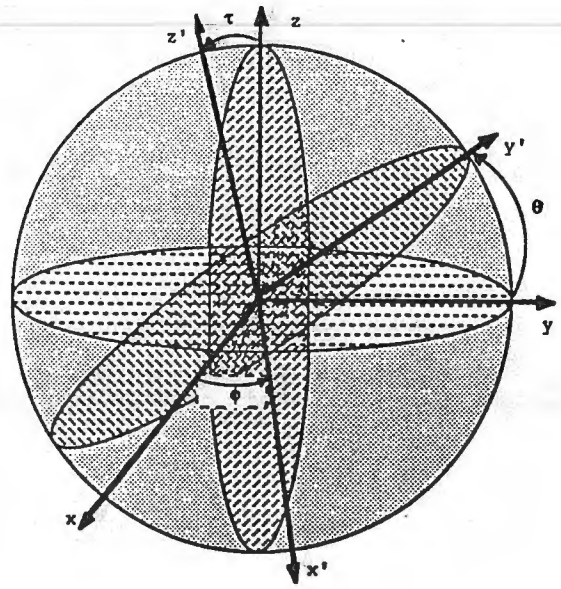
Moments

$$L = \dot{P} I_{xx} - \dot{R} I_{xz} + QR (I_{zz} - I_{yy}) - PQ I_{xz}$$

$$M = \dot{Q} I_{yy} + PR (I_{xx} - I_{zz}) + (P^2 - R^2) I_{xz}$$

$$N = \dot{R} I_{zz} - \dot{P} I_{xz} + PQ (I_{yy} - I_{xx}) + QR I_{xz}$$

where θ, ϕ, τ = Euler pitch, roll, and yaw angles
P, Q, R = body roll, pitch, and yaw rates
L, M, N = moment about the body x, y, and z axes
I_{xx}, I_{yy}, I_{zz} = moment of inertia about the x, y, and z axes
I_{xz} = cross moment of inertia (xz axis)



EULER ANGLE DIAGRAM

FIGURE 2-2. ROTATIONAL RELATIONSHIPS USED FOR ANALYTICAL REDUNDANCY

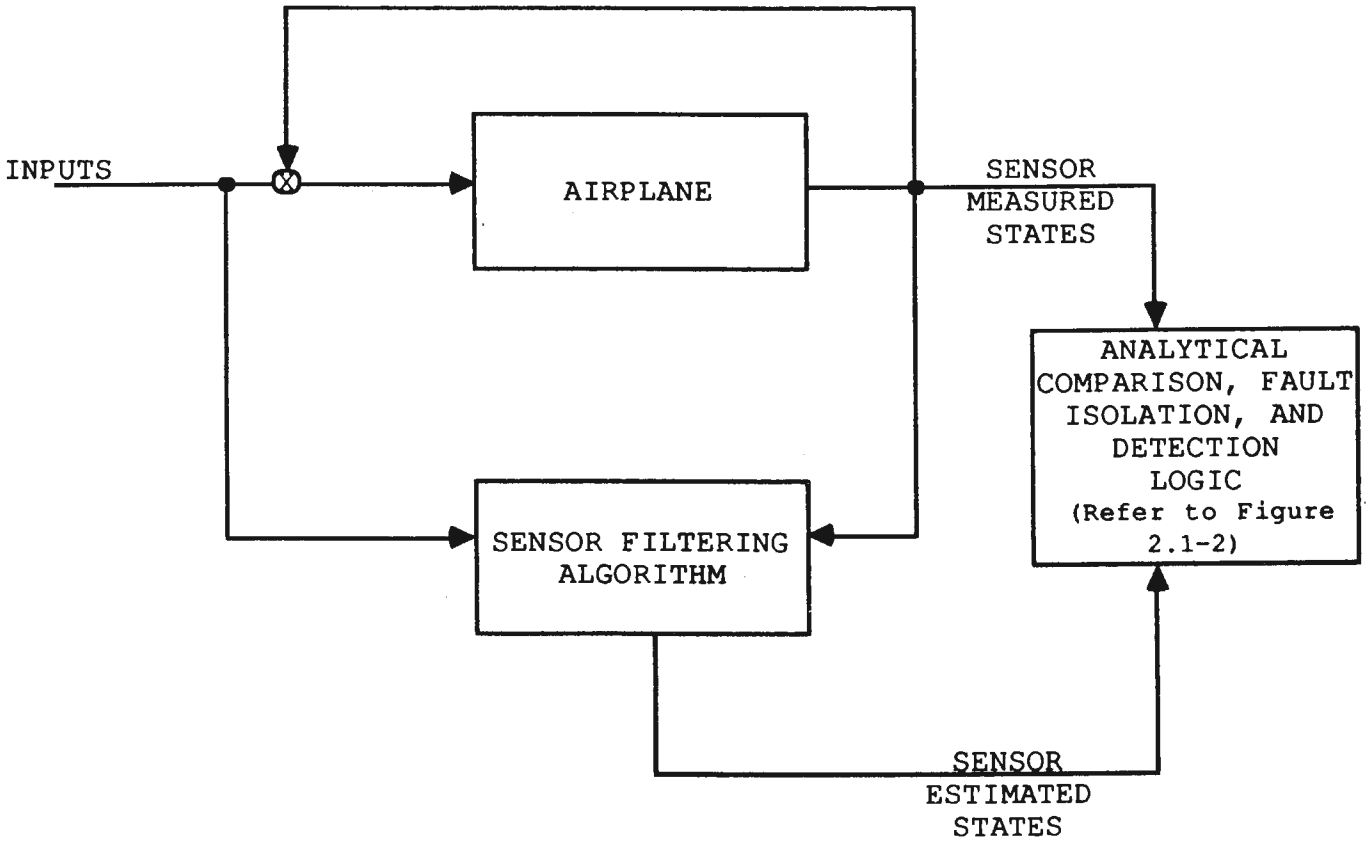


FIGURE 2.1-1. ANALYTICAL REDUNDANCY CONCEPT

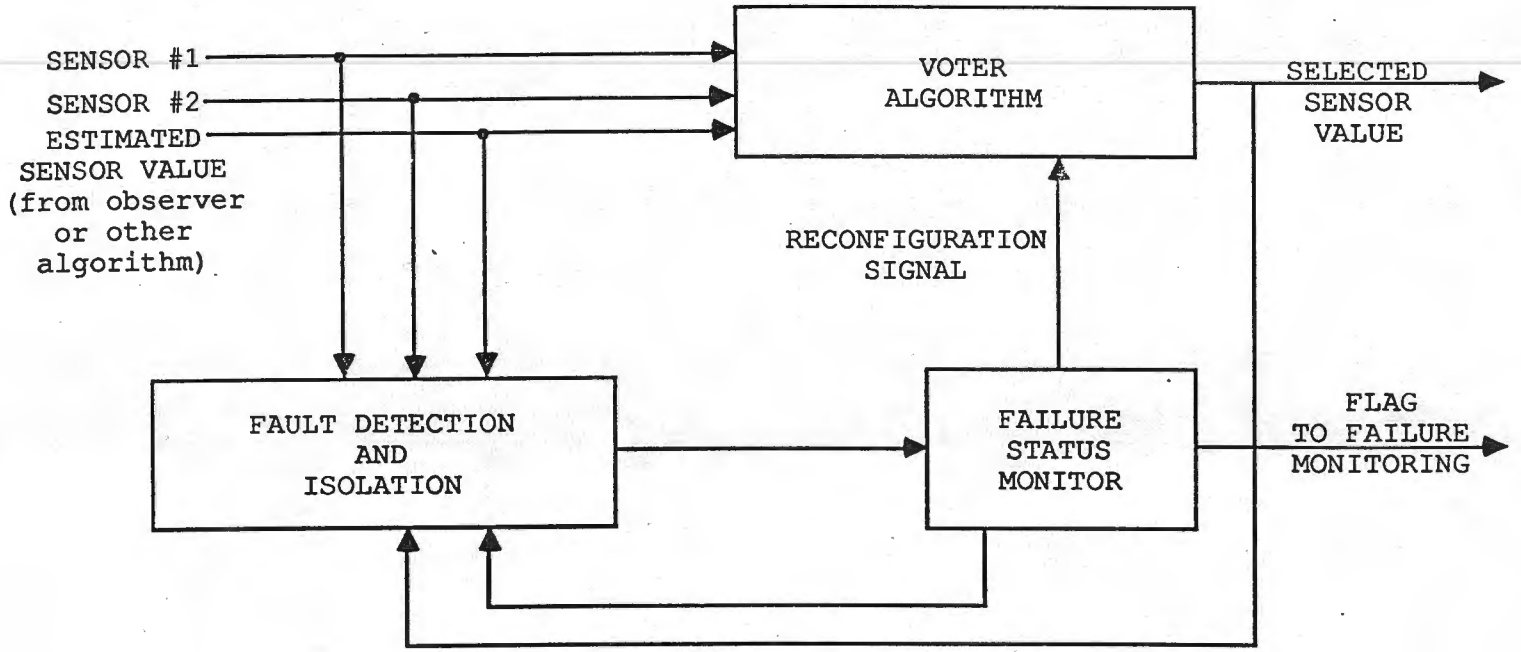


FIGURE 2.1-2. ANALYTICAL SENSOR REDUNDANCY FAULT DETECTION LOGIC

3. ANALYTICAL REDUNDANCY - DESIGN APPROACH

A literature survey was performed by Cunningham and Hartmann (1977) to examine currently available techniques that were able to detect and isolate faults in aircraft sensors and meet current onboard flight computer allocations. All such methods for failure detection were found to lie in three categories.

- Assemblies of DFs (see Meier, Ross, and Glaser, 1971; Maybeck, 1974; Hartmann, et al., 1975; and Clark, et al., 1975).
- Specific DF design techniques (see Kerr, 1974 and Mehra and Peschon, 1971).
- Explicit SDF design techniques.

The first two categories also include observer/blender techniques. Mulcare, Downing, and Smith (1987) and Deckert, et al., 1978 utilize these techniques.

The three filter types listed above can be combined in many ways to obtain useable AR results. The end product may vary considerably in accuracy and complexity. The structure of the filter depends on the overall RM approach, whether uniform redundancy or different levels of redundancy are required throughout the system.

Uniform redundancy schemes categorize sensors as interchangeable or noninterchangeable. Interchangeable sensors can substitute for each other when a failure occurs. Appropriate coordinate transformations must be accounted for, if necessary. If a noninterchangeable sensor fails, an identical one must replace it in order to maintain performance.

If N fully operational sensors are required to maintain aircraft operations, then a minimum of N+2 interchangeable sensors and 3N noninterchangeable sensors are required for dual fail-operational performance when using AR. In order to resolve conflicts using traditional voting techniques, N+3 and 4N sensors are needed to maintain the same performance.

Fault Detection using DFs with interchangeable sensors is illustrated in figure 3-1. The figure shows FD for N, the number of sensors required for safe flight, equal to 2 for fail-operational performance. The truth table shows that each column of the error flags is a unique binary word and that the sensor inputs to each filter are uniquely determined by each row of the table. The generalized way to assemble DFs for interchangeable sensors is given below.

- Determine the number of sensors required to maintain safe flight and meet operational requirements (N is the number of sensors necessary for safe flight).

N + 1 for fail-operational performance
N + 2 for dual fail-operational performance

- Determine the number of DFs (the brackets indicate to use the next largest integer of the enclosed expression).

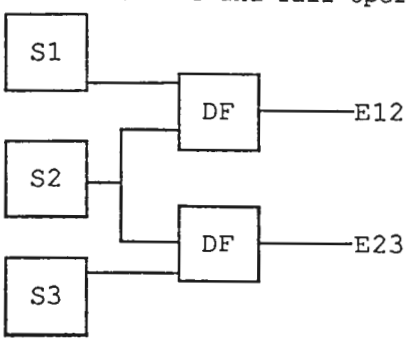
  $[\log_2 (N + 2)]$ for fail-operational performance
  $[\log_2 (N + 3)]$ for dual fail-operational performance

- Determine sensor inputs to each filter from the truth table.

The FD capability of a DF combined with comparators for noninterchangeable sensors can also be verified with a truth table. See figure 3-2 for an illustration of fail-operational performance with N, the number of sensors necessary to maintain safe flight, equal to 2. The generalized arrangement uses one filter and N comparators for fail-operational and one filter and 2N comparators for dual fail-operational. The truth table here contains FD for single sensor failures only. It is generally assumed that failures occur singly when designing FD algorithms. A single SDF is all that is needed in all cases to provide FD and isolation. This type of filter may be quite complex in order to achieve the same results as DFs.

INTERCHANGEABLE SENSORS

for N=2 and fail-operational performance



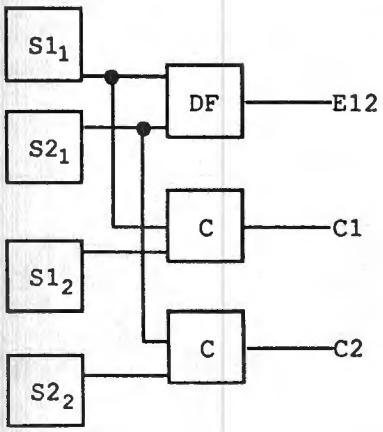
TRUTH TABLE

	SINGLE SENSOR FAILURE			
ERROR FLAGS	NONE	S1	S2	S3
E12	0	1	1	0
E23	0	0	1	1

FIGURE 3-1. EXAMPLES OF FAULT DETECTION USING DIAGNOSTIC FILTERS AND INTERCHANGEABLE SENSORS

NONINTERCHANGEABLE SENSORS

for N=2 and fail-operational performance



TRUTH TABLE

ERROR FLAGS	SINGLE SENSOR FAILURE				
	NONE	$S1_1$	$S2_1$	$S1_2$	$S2_2$
E12	0	1	1	0	0
C1	0	1	0	1	0
C2	0	0	1	0	1

C = Comparator
DF = Diagnostic Filter
N = # of sensors needed to maintain safe flight

FIGURE 3-2. EXAMPLES OF FAULT DETECTION USING DIAGNOSTIC FILTERS AND NONINTERCHANGEABLE SENSORS

The largest amount of sensor reduction is achieved for noninterchangeable sensors when using AR. In this case, AR implemented with one DF in combination with 2N comparators has the same benefit as one SDF. An SDF which is more complex will not be feasible. A comparison of the sensor reductions possible with these analytical techniques and traditional voting is given in table 3-1. The number of comparators and DFs needed for AR are also given. N is defined as the minimum number of sensors required to maintain safe flight.

There is a sensor reduction benefit for interchangeable sensors; however, it is a small one. AR provides little or no benefit to save one sensor. More DFs are needed to match the capabilities of a single SDF than for noninterchangeable sensors.

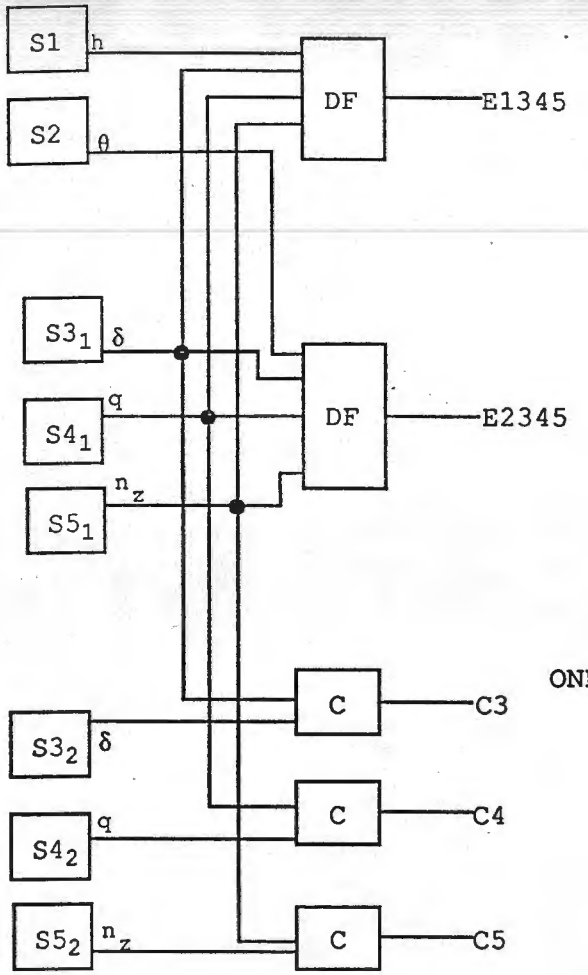
Current flight control systems normally do not have the same redundancy requirements for all sensors. A more realistic setup may have high levels of redundancy for some critical sensors and lower levels for others. The high levels of redundancy are known as reversion modes. While this may be application dependent, the basic principles can be used to obtain a failure detection scheme as shown in figure 3-3. In order to implement the reversion modes, sufficient filters are required to distinguish between four outer loop logic conditions: no failure; S1 failed; S2 failed; and $S3_1$, $S4_1$, or $S5_1$ failed. The error flags, E1345 and E2345, are used to obtain this information. The error flags, C3, C4, and C5, must be used to determine which of the inner loop sensors have failed.

TABLE 3-1. SENSOR REDUCTION COMPARISONS FOR ANALYTICAL REDUNDANCY BUILDING BLOCKS

FAIL-OPERATIONAL	NUMBER OF SDF'S	NUMBER OF DIAGNOSTIC FILTERS	NUMBER OF COMPARATORS	NUMBER OF SENSORS
INTERCHANGEABLE SENSORS				
DETECTION WITH SDF'S	1	0	0	N+1
WITH DF'S	0	$[\text{LOG}_2(N+2)]$	0	N+1
WITH VOTING	0	0	N+3	N+2
NONINTERCHANGEABLE SENSORS				
DETECTION WITH SDF'S	1	0	0	2N
WITH DF'S	0	1	N	2N
WITH VOTING	0	0	2N	3N
DUAL FAIL-OPERATIONAL	NUMBER OF SDF'S	NUMBER OF DIAGNOSTIC FILTERS	NUMBER OF COMPARATORS	NUMBER OF SENSORS
INTERCHANGEABLE SENSORS				
DETECTION WITH SDF'S	1	0	0	N+2
WITH DF'S	0	$[\text{LOG}_2(N+3)]$	0	N+2
WITH VOTING	0	0	N+4	N+3
NONINTERCHANGEABLE SENSORS				
DETECTION WITH SDF'S	1	0	0	3N
WITH DF'S	0	1	2N	3N
WITH VOTING	0	0	3N	4N

N = MINIMUM NUMBER OF SENSORS REQUIRED FOR SAFE FLIGHT

[ ] IMPLIES USE THE NEXT LARGEST INTEGER OF THE EXPRESSION WITHIN THE BRACKETS



TRUTH TABLE

ERROR FLAGS	SENSORS FAILED								
	NONE	S1	S2	$S3_1$	$S4_1$	$S5_1$	$S3_2$	$S4_2$	$S5_2$
E1345	0	1	0	1	1	1	0	0	0
E2345	0	0	1	1	1	1	0	0	0
C3	0	0	0	1	0	0	1	0	0
C4	0	0	0	0	1	0	0	1	0
C5	0	0	0	0	0	1	0	0	1

TWO FAIL-SAFE OUTER LOOPS (h), (θ)

ONE FAIL-OPERATIONAL INNER LOOP (δ, q, n_z)
(NON-INTERCHANGEABLE SENSORS)

- h = altitude
- θ = Euler pitch angle
- δ = control surface deflection
- q = body pitch rate
- n_z = normal acceleration
- C = Comparator
- DF = Diagnostic Filter

FIGURE 3-3. EXAMPLE OF FAULT DETECTION WITH REVERSION MODES

Three concepts encompass the categories of failure detection methods. These are examined as a result of various combinations of DFs. These concepts in order of increasing complexity are:

- Observer/signal blender filter.
- Diagnostic Kalman filters.
- Super-diagnostic Kalman filter design.

3.1. Observer/Signal Blender Filter

An observer/blender will model physical relationships using sensor outputs to provide FD for several sensors. It uses a specific DF design and has the least amount of complexity and computer requirements. Monitors may use a three trip level exceedance criterion to detect failures. Trip thresholds can be set to account for unmodeled dynamics and sensor anomalies.

The basic operation of the observer/blender is to define a consistent error function between a measured parameter and its derivative. Both values are measured for all but a few parameters, e.g., lateral acceleration. A relationship such as,

$$V_x = \dot{X}$$

is exact. For perfect measurements, there is no error. Realistically, the measured values of V_x and X are not true representations of V_x and X. There is an error associated with digital integration and sampling. Either V_x or X may contain faults. This implementation of AR must distinguish between the errors associated with measurements and digital sampling and errors associated with the sensors.

3.2. Diagnostic Kalman Filters

Diagnostic Kalman filters incorporate modeling of sensor anomalies (such as, bias, scale factors, and wind gust estimation) in the filter. FD is provided for more sensors than available with the observer/blender. This is achieved by using an assembly of DFs. This concept places moderate requirements on a computer.

3.3. Super-Diagnostic Kalman Filter Design

The super-diagnostic design utilizes Kalman filters as well as linear equations for pitch and lateral directional dynamic equations of motion. Both detection and isolation can be determined. An error signal is created for each sensor involved. The concept is built with an assembly of DFs. Some applications may be very complex.

The largest number of surveyed references are devoted to this topic. These filters are the most difficult to construct. They are constrained more than other techniques due to the amount of complexity which is feasible. Five methods have been examined for SDFs.

- Multiple hypothesis test.
- Parameter identification.
- Generalized likelihood ratio methods.
- Modified filter designs.
- Jump process estimation.

3.3.1. Multiple Hypothesis Tests

Each failure condition of a particular sensor is used to define a hypothesis. The entire range of failure conditions forms a set of hypotheses which are tested against observed data. One Kalman filter is required to test each hypothesis. The filter generates residuals, assuming that the hypothesis is true, which are summed into likelihood functions. The maximum likelihood then identifies the current valid hypothesis. This results in a large number of filters for even a few sensors and at present is too complex for advanced flight computers to utilize. Montgomery and Caglayan (1974), Montgomery and Price (1974), Athans and Willner (1973), Lainiotis (1971), and Buxbaum and Haddad (1969) contain more information.

3.3.2. Parameter Identification

Parameters of each sensor, such as, gain and bias, are defined as unknowns and estimated from sensor data. If the estimates deviate a great deal from nominal values, then a failure is declared. For any procedure used, a separate calculation similar to a Kalman filter is required to compare the nominal values with the estimated sensor parameters. This becomes as complex and inefficient as the previous technique. Mehra and Peschon (1971) and Stein and Hartmann (1976) have investigated this technique.

3.3.3. Generalized Likelihood Ratio Methods

In an attempt to reduce the complexity of this type of failure detection method, initial constraints were placed on the design of an algorithm. These constraints are that only one Kalman filter is available and that sensor failure detection must be decided through the monitoring of residuals from the single filter. This can be done for those failures which have recognizable signatures for correlation processing. The residuals are correlated with known signals, normalized by their expected no-failure value, and compared to predetermined threshold levels. The number of filters has been greatly reduced, however, the correlation functions can be fairly complex. See Willsky, Deyst, and Crawford (1974); Willsky and Jones (1974), Deyst and Deckert (1975), McAulay and Denlinger (1973), and Sanyal and Shen (1974).

3.3.4. Modified Filter Design

This method starts with a constraint of one Kalman filter. It does not allow the filter to remain a fixed element. An attempt is made to alter the filter gains to produce strong and easily recognized failure signatures. This concept is largely theoretical at this time. Details are given in Jones (1973) and Beard (1973).

3.3.5. Jump Process Estimation

This procedure is also theoretical. The basic idea is to represent failures as random jump processes in an otherwise known stochastic system. The time of occurrence and magnitude of the jumps are then estimated using optimal stochastic filtering theory. (See Sworder, 1972; Sworder and Robinson, 1973; Ratner and Luenberger, 1969; Pierce and Sworder, 1971; Davis, 1975; McGarty, 1974; and Chien, 1972.)

3.4. Redundancy Management

Redundancy Management will be optimized when included in a system design, but it can be incorporated into an existing system. Mulcare, Downing, and Smith (1987) modified an existing quadruplex DFCS to incorporate AR. Section 6.3 contains example filters which could be used to add AR to one of the functions, a Stability Augmentation System (SAS), implemented in this modification.

Issues concerned when designing and structuring the RM are:

- The number of success paths.
- The amount of dependency placed on monitoring quality.
- The tradeoff between fail safety and reliability.
- The cost effectiveness for increased reliability with added FD and isolation.
- The throughput constraints imposed by the flight computer. RM can be 60 to 80 percent of the required processing.

Performance qualities may vary for different RM structuring.

In order to maintain fail-safe sensor operations, the following three types of logic are needed.

- Comparison monitoring of dual like signals without an associated DF. Examples where this may be implemented are after a computer failure or a failure of an input to a DF.
- Monitoring the diagnostic error function after failure of one-out-of-two sensors in a dual set. If a pitch rate gyro fails, the relevant DF then monitors the remaining pitch rate gyro.

- Monitoring of a single sensor by its associated DF. This logic requires a failure declaration from both computers to avoid fault indication due to a dual sensor failure.

4. ANALYTICAL REDUNDANCY - RELIABILITY

The concept of AR, in order to be implemented in practice, must satisfy performance and reliability requirements of typical flights and must not overload the flight computer.

The traditional RM for a quadruplex set of sensors would isolate the first and second failed sensors by a majority vote. On the failure of the third sensor the set would be shut down. For a perfect monitoring process, the probability of total failure of the set is $4Q_s^3$, where Q_s is the probability of failure of one of the sensors within a certain time. The probability of any two sensors failing is $6Q_s^2$.

A rate gyro has a failure rate of about 10^{-4} per hour. This is one of the worst sensor failure rates. The probability of a quadruplex set of gyros failing would be 4×10^{-12}. This exceeds the typical flight control system requirements per sensor set of 10^{-9} by more than two orders of magnitude. For a triplex set of gyros, failure is defined as $3Q_s^2$ with a value of 3×10^{-8} which is less than the requirements by a factor of thirty.

The assumption of perfect monitoring is not valid. An additional factor, Q_m, should be included in the failure rate. Q_m defines the probability that the first failure is not detected and that both failures are alike. This issue of imperfect monitoring or coverage may cause failure of a quadruplex sensor set after only two failures, if it cannot distinguish between the two good and two failed sensors. The probability of failure here is $6Q_s^2\ Q_m$. No data exists for the term Q_m, but numbers on the order of 10^{-2} have been suggested. With imperfect monitoring, the probability of failure for a quadruplex set of rate gyros is 6×10^{-10} in one hour which is just within the requirements.

The diagnostics provided in an AR system must provide similar quality in monitoring. For triplex sensors configured for dual-fail-operational systems, the total failure rate is given in the following equation.

$$Q_s^3 + 3Q_s^2\ Q_d$$

The first term represents the probability for all three sensors failing. The second term is the combination of the probabilities of a dual sensor failure and the diagnostics failing to recognize the fault, Q_d. This is the dominant term. If it is set equal to the reliability requirement,

$$3Q_s^2\ Q_d = 10^{-9}$$

$$Q_d = 3.3 \times 10^{-2}$$

The value of Q_d indicates that the diagnostics must be 97 percent reliable. If this performance is feasible, then AR techniques are at least equal to those of quadruplex channel voting.

This example was devoted to just one set of sensors. Total system reliability includes all hardware: computers, servos, and sensors. A computer code, Computer Aided Redundant System Reliability Analysis (CARSRA), computes system reliabilities based on a specific configuration and sensor failure rates. Mulcare, Downing, and Smith (1987) used a revised version of this code to perform a system reliability assessment.

Table 4-1, utilizing parameters of an existing system (Cunningham and Hartmann, 1977), shows that servos, not sensors, tend to dominate reliability. By adding sensor redundancy, only a factor of two less than baseline system failure rate is obtained. With both sensor and servo redundancy, system failure rate decreased by more than an order of magnitude. Failure rates are dependent on system architecture. A system should be designed to have matched reliability.

TABLE 4-1. SYSTEM RELIABILITY

Probability (failures per flight hour)		
Failure Type	95% Sensor Redundancy	95% Sensor/Servo Redundancy
Either computer fails (95% effective self-test)	0.2×10^{-4}	0.2×10^{-4}
Servo failure in any axis	6.0×10^{-4}	0.3×10^{-4}
Gyro failure in any axis	0.3×10^{-4}	0.3×10^{-4}
Normal accelerometer failure	0.02×10^{-4}	0.02×10^{-4}
Total	6.52×10^{-4}	0.82×10^{-4}

5. ANALYTICAL REDUNDANCY - FAULT DETECTION AND ISOLATION

The type of FD monitor used to test various error signals is critical for performance. Important parameters are speed of response and an acceptable false alarm rate, one per 1000 hours of flight. Observers or DFs can operate with comparison monitors of dual sensors to isolate faults for a fail-operational capability. A fail-safe capability exists for all single sensors and for dual sensors after one failure.

Fault Detection monitors can be multiple trip monitors or a Sequential Likelihood Ratio Test (SLRT) of a defined parameter, error signals, or likelihood functions. The design goal of an FD monitor is FD within a minimum time after the occurrence of the fault while maintaining an acceptable false alarm rate. Another important goal is the number of missed alarms. This is addressed in the literature (Cunningham and Hartmann, 1977 and Deckert, et al., 1978) where data exists from flight simulation tests.

5.1. Multiple Trip Monitor

The FD logic in a multiple trip monitor typically requires three consecutive trips for a fault to be declared. The trip thresholds must be based on the performance of each sensor and known sensor and modeling inaccuracies.

The simplest test to check for an error signal is to compare its magnitude with predefined limits. This is illustrated in figure 5.1-1. The trip magnitude is typically placed at some multiple, m, of σ, the standard deviation of the error signal of an unfailed sensor. The mean of this error signal is zero. The value of m is chosen to yield an acceptable false alarm rate. This is shown in the following sequence of equations. The probability that there will be no false alarms during one hour of flight is given by a Poisson distribution. See Ross (1972) for more information on probability density functions, likelihood functions, and reliability.

$$P(\text{no false alarms}) = e^{-\delta t}$$

where δ = the false alarm rate (1 per 1000 flight hours)
t = the flight time (1 hour)

so that P(no false alarms) = 0.999. This is also equal to the probability that a sequence of random variables, the sampled error signals, will remain within the trip boundary of $m\sigma$.

$$P\left(\bigcap_{i=1}^{n} X_i\right) = e^{-\delta t}$$

For a one hour flight, sampling at 16 Hz, n will equal 57,600.

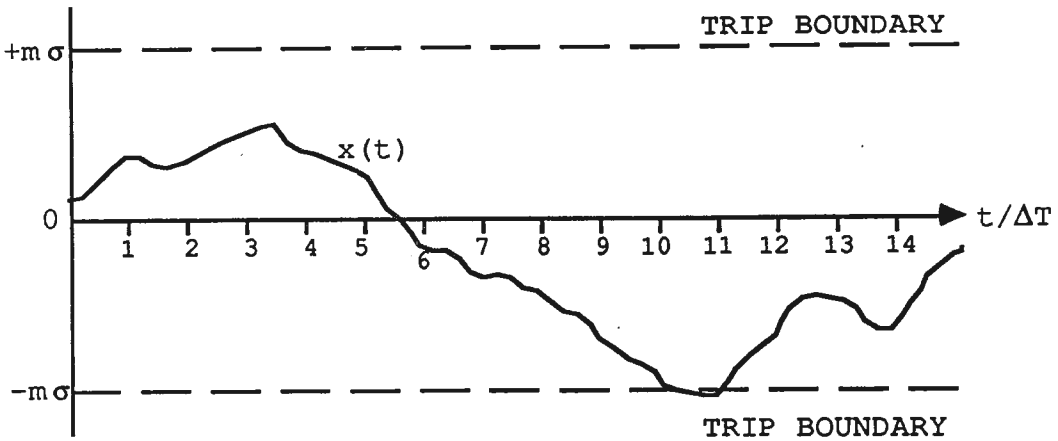


FIGURE 5.1-1. MULTIPLE TRIP FAULT DETECTION MONITOR

If the sequence of random numbers is independent, then

$$P(\bigcap_{i=1}^{n} X_i) = \prod_{i=1}^{n} P(X_i) = nP(X_i)$$

$$P(|X_i| > m\sigma) = (1 - e^{-\delta t}) / n$$

$$= 1.74 \times 10^{-8}$$

This would require m to be much larger than 4. Multiples of σ are not given in normal probability tables for numbers in this range. For three consecutive trips,

$$P(|X_i| > m\sigma; \text{ for three consecutive times}) = (1.74 \times 10^{-8})^{1/3}$$

$$= 2.6 \times 10^{-3}$$

This corresponds to $m \approx 3.2$. This multiple of σ was determined using a table containing probability values for a standard normal distribution.

For a completely dependent sequence,

$$P(|X_i| > m\sigma;\ \text{for three consecutive times}) \approx P(|X_i| > m\sigma;\ \text{one time})$$

$$\approx 1 - e^{-\delta t}$$

$$= 1.0 \times 10^{-3}$$

This corresponds to $m \approx 3.4$. This value was obtained from the same table used above.

The results of this analysis imply that the triple trip boundary needs to be set at 3.4σ to meet the false alarm rate for a given error signal.

The trip threshold can remain constant, adjusted based on the output of the sensor, or adjusted based on inputs from the pilot. An adjustable threshold can increase sensitivity to environmental disturbances, such as turbulence, and to aircraft in-flight maneuvering.

A multiple trip criterion (i.e., a fault is declared when a limit is exceeded n consecutive times) gives a much lower false alarm rate than a first trip monitor. The number of consecutive trips chosen for a monitor is a tradeoff between error insensitivity and the speed of FD. Choosing a multiple number of trips applies a built-in delay to FD.

5.2. Sequential Likelihood Ratio Tests

Likelihood ratio tests have been demonstrated to be a better FD scheme than the multiple trip monitor in Cunningham and Hartmann (1977). This algorithm provides quicker response to faults such as hardover, and more sensitivity to other smaller faults (bias and scale factor changes). However, there is more complexity to this monitoring scheme.

These tests are based on two hypothesized density functions of a random variable, $f_0(x)$ and $f_1(x)$. The likelihood ratio is expressed as a function, Γ_n, of error signal observations of that random variable.

$$\Gamma_n = \frac{f_0(e_1, e_2, \ldots, e_n)}{f_1(e_1, e_2, \ldots, e_n)}$$

The two hypothesized outcomes, H_0 and H_1, are no failure or sensor failure. These outcomes are accepted based on the following rules.

- Accept H_0 if $\Gamma_n \leq A$.
- Accept H_1 if $\Gamma_n \geq B$.
- No decision if $A < \Gamma_n < B$.

where A = 1/B. A typical value for B is 20,000.

H_0 implies that the expected value of a given error signal is 0 and there is no fault. H_1 implies the mean of the error signal has shifted unacceptably to μ_1 and a sensor fault should be declared. For actual monitoring, a combined test is used where μ_1 may equal $\pm$ mσ. Figure 5.2-1 illustrates the bounds for H_0 and H_1.

This can be defined mathematically using the following equations. The functions, f_0 and f_1, are assumed to have a normal distribution with a standard deviation of σ.

$$f_0(e_i) = c \exp[-\sum_{i=1}^{n} (e_i - \mu_0)^2 / 2\sigma^2]$$

$$f_1(e_i) = c \exp[-\sum_{i=1}^{n} (e_i - \mu_1)^2 / 2\sigma^2]$$

Using the no decision criteria, taking natural logarithms, and substituting 0 for μ_0 and $\pm m\sigma$ for μ_1,

$$\ln A < \sum_{i=1}^{n} [(e_i \mp m\sigma)^2 - (e_i)^2] / 2\sigma^2 < \ln B$$

which is equivalent to

$$nm\sigma/2 - \sigma \ln B/m < \sum_{i=1}^{n} e_i < nm\sigma/2 + \sigma \ln B/m, \text{ for } \mu_1 = +m\sigma$$

and

$$-nm\sigma/2 - \sigma \ln B/m < \sum_{i=1}^{n} e_i < \sigma \ln B/m - nm\sigma/2, \text{ for } \mu_1 = -m\sigma$$

A fault is determined when $\left| \sum_{i=1}^{n} e_i \right| \leq \sigma \ln B/m + nm\sigma/2$.

No fault is determined when $\left| \sum_{i=1}^{n} e_i \right| \geq nm\sigma/2 - \sigma \ln B/m$.

Otherwise, no decision is made.

The test sequence begins by initializing n to 1 and updates the summation each cycle. The sequence restarts when H_0 is accepted or there is no decision for a fixed period of time. Frequent restarts are desirable because summations over long intervals have less sensitivity to new data.

Likelihood functions of like sensors, i.e., with identical statistical error characteristics, can be compared and used to isolate the faulted sensor. The likelihood functions are L_{n1} and L_{n2} for sensors 1 and 2, respectively. The hypotheses here are that H_0 implies that sensor 1 has failed, and H_1 implies that sensor 2 has failed. The mean of H_0 is μ_0 which is equal to $+m\sigma$, and the mean of H_1 is μ_1 which is equal to $-m\sigma$.

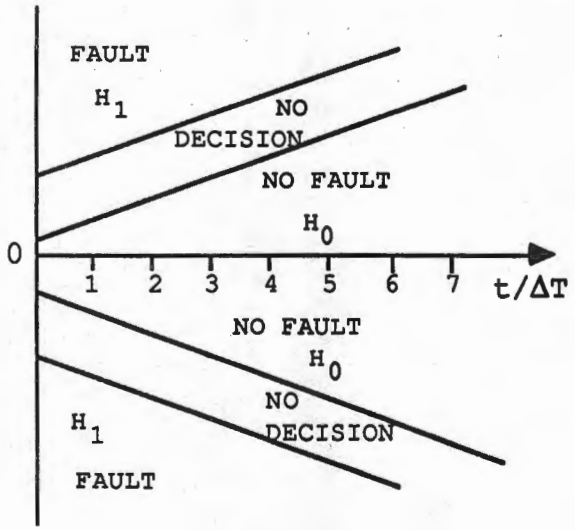


FIGURE 5.2-1. SEQUENTIAL LIKELIHOOD FAULT DETECTION MONITOR

Decisions are made using the difference of the likelihood functions based on the following rules.

$$\Delta L_n = L_{n1} - L_{n2}$$

- Accept H_0 if $\Delta L_n \geq \sigma \ln B/2m$.
- Accept H_1 if $\Delta L_n \geq \sigma \ln A/2m$.
- No decision if $\sigma \ln A/2m < \Delta L_n < \sigma \ln B/2m$.

Because the no fail hypothesis is not included here, this particular test can proceed only when a fault has already been declared. This test can be initiated after one trip using comparison monitoring. If a fault is declared (three consecutive trips), then the likelihood comparison test is used to determine which sensor is faulted. Because the likelihood functions can only increase in a fault situation, the plus or minus direction of the delta likelihood function provides the information necessary to determine which sensor has failed.

5.3. Fault Modeling

Sensor error and fault modeling require careful treatment for FD because typical sensor models are not sufficient and fault characteristics may not be

well-defined. A key issue is modeling sensor characteristics, including noise in an operational environment. Possible errors superimposed on the sensor output include bias, resolution, scale factor, null output, alignment error, hysteresis, environmental effects, response dynamics, and unmodeled sensor inputs. Errors may occur if the model does not include pilot inputs. Mode specific error signals may occur for selected modes, e.g., altitude or heading hold. A determination of the importance of each type of error is dependent on the type and percentage of occurrence of failure modes for each sensor. Modeling these errors may be a difficult task, because data is not easily obtained or does not exist on some of these parameters.

Faults can be classed as open-loop or closed-loop faults. An open-loop fault is one in which the sensor output is completely independent of its input. Examples of this category are hardover (at maximum output), zero output, output stuck at an intermediate value, and a randomly varying output. Closed-loop faults consist of the correct sensor output with errors superimposed. These faults are misalignment, scale factor changes, noise, or others listed in the previous paragraph.

Simulations should subject the FD algorithms to the highest probability faults. Sensor hardover, dead sensor, dynamic response reduction, scale factor changes, and bias shifts represent 90 to 95 percent of expected failures in typical rate gyros and accelerometers. These two sensor types have the highest failure rates compared to other sensors. Other faults should be defined dependent on the sensor.

Some techniques are limited in the type of failures they look for. Deckert, et al., 1978 limited their technique to looking for bias failures. They felt that most other failures could be found with the same technique. As yet, no one has addressed oscillatory faults which may or may not be important.

As defined by Cunningham and Hartmann (1977), the observer/blender and diagnostic Kalman filter worked well for FD for most sensors included in the study. Super-diagnostic techniques were found to have performance concerns and developmental needs. Problems arose in detection of accelerometer soft failure, dynamic response faults, and FI based on more than one filter detecting a given fault. (The difficulty arose when both filters did not detect the failure at the same time.)

A comparison of monitors shows that the SLRT of residual mean values performed well relative to the multiple trip monitor. SLRT caught hardover failures sooner. Frequently scale factor changes would escape the multiple trip monitor, but not the SLRT.

One way to enhance a particular FD algorithm is to utilize the fact that there are physical limitations as to how much the aircraft state can change from one sample to the next. This can minimize the effects of extremely large sensor failures. Desai, Deckert, and Deyst (1979) added a self-test in conjunction with an FD algorithm which provisionally failed a sensor when an abrupt change in output was maintained for three sample intervals.

6. SUPPLEMENTAL WORKED EXAMPLES

6.1. Altitude Kinematics

Altitude Kinematics refers to the redundancy between changes in altitude measured by the barostatic pressure sensors and changes in altitude computed from the vertical acceleration measured by the accelerometers. The vertical direction is determined via Euler attitude angles: roll (ϕ) and pitch (θ). Since the accelerometers sense gravity, the gravitational component must be subtracted in computing vertical acceleration. Vertical acceleration is computed as:

$$A_v = A_x \sin\theta - (A_y \sin\phi + A_z \cos\phi) \cos\theta - g$$

where A_x, A_y, A_z are measured accelerations compensated to the center of mass.

A third order complementary filter (illustrated in figure 6.1-1) is used to store $\dot{h}$ (vertical velocity) prior to a failure detection and possible corruption. The value of $\dot{h}$ can then be used to compare altitude deviation from slightly prior to the failure detection by keeping a moving window of data.

The altitude residues for both sensors would be:

$$\text{Residue \#1} = \Sigma(\text{current } h_1 - \text{previous } h_1 - \dot{h}\, dt)$$

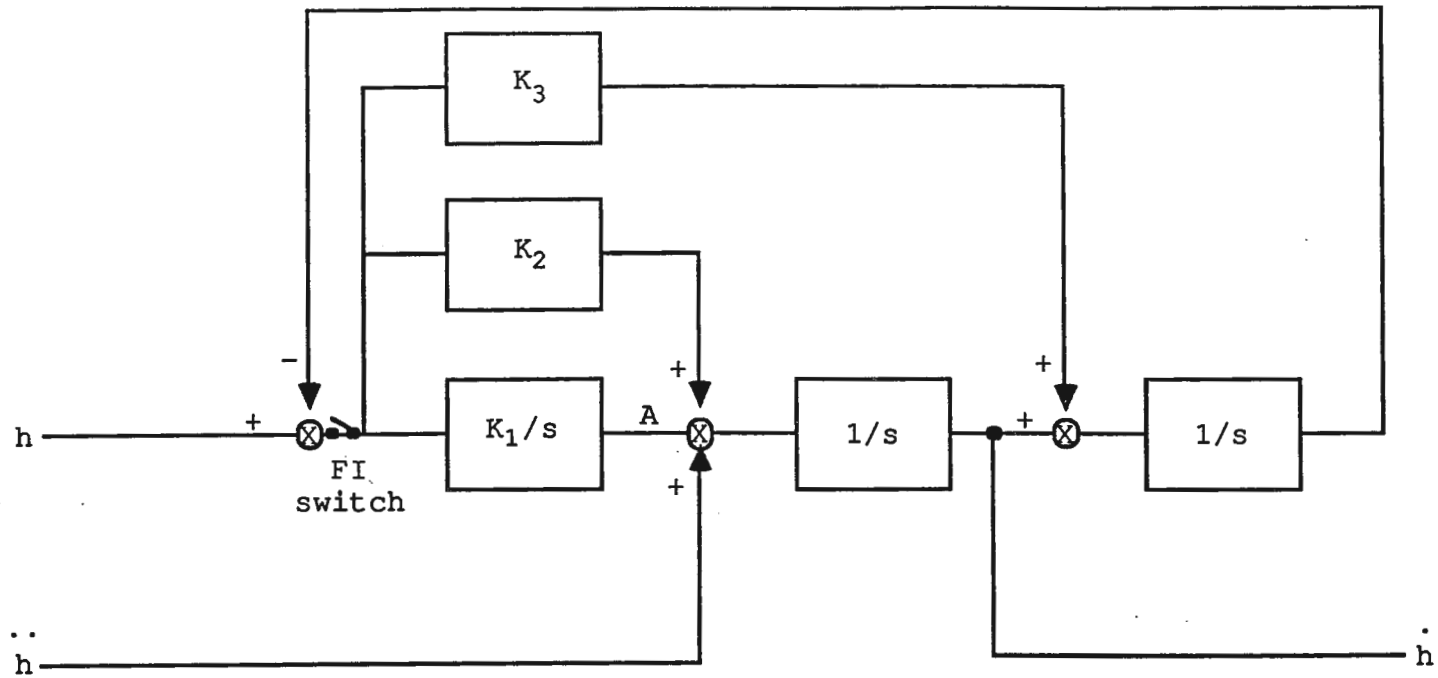
$$\text{Residue \#2} = \Sigma(\text{current } h_2 - \text{previous } h_2 - \dot{h}\, dt)$$

using h measurements from sensors 1 and 2, respectively, where dt is the integration interval.

Both residues are then evaluated by an FI algorithm. The algorithm may be a multiple trip threshold exceedance or a statistical hypothesis algorithm (e.g., SLRT).

During the FI period, the $\dot{h}$ used is only inertially updated via the switch shown in figure 6.1-1, i.e., $\dot{h} = \int \ddot{h}$. The switch opens (FI begins) when the absolute magnitude of $(h_1 - h_2)$ exceeds a threshold N times. The filter is not updated if any sample exceeds the threshold to avoid $\dot{h}$ corruption. Figure 6.1-2 demonstrates the RM processing.

Error sources must be evaluated to avoid a false FI decision. The third order filter has a state to estimate steady-state errors on $\ddot{h}$ to avoid a parabolic error growth when calculating both residues. An error analysis must be made to determine the maximum time that can be allowed for FI.



h is the average of two baro-static pressure altitudes

$\ddot{h}$ is an estimate of error of $\ddot{h}$, subsequently added to each $\ddot{h}$ calculation in the fault isolation residue equations to effectively cancel the error.

FIGURE 6.1-1 THIRD ORDER VERTICAL COMPLEMENTARY FILTER FOR ALTITUDE KINEMATICS

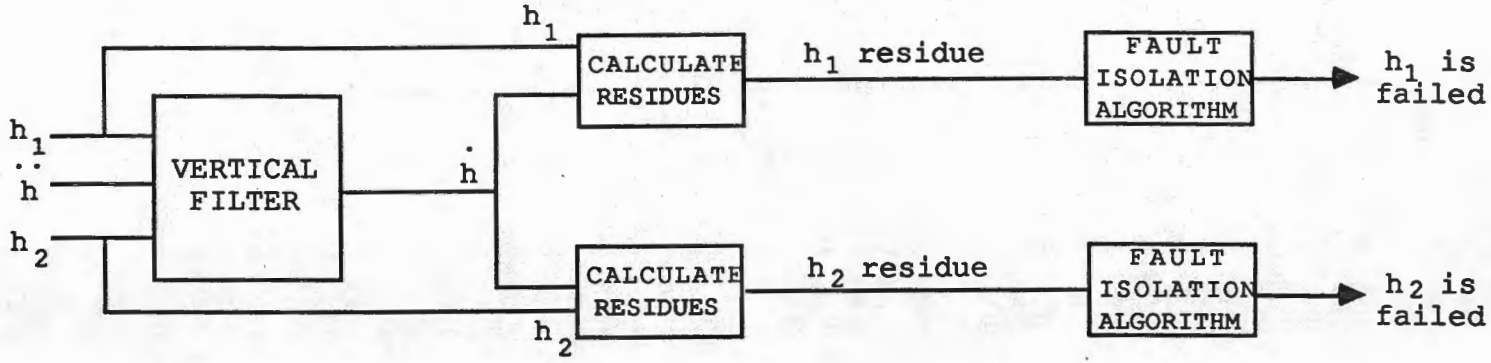


FIGURE 6.1-2. POSSIBLE REDUNDANCY MANAGEMENT FOR ALTITUDE KINEMATICS FAULT ISOLATION

If the time expires without an FI decision, then the following logic can be used:

Is there still a fault?

The answer is yes, if the absolute magnitude of $(h_1 - h_2)$ is greater than the threshold. Then isolate the sensor with the worst error history, i.e., the largest residue magnitude.

Otherwise, the answer is no. Reset the RM logic and restart the vertical filter, closing the FI switch and clearing the FD flag, if one is present.

6.2. Translational Kinematics

A translational kinematic relationship can be used to determine an airspeed sensor fault by means of the related inertial acceleration along the x-axis, such that

$$V_x = \int \dot{U}\, dt \qquad \text{or} \qquad \Delta V_x = \dot{U}\, dt$$

$$V_x = V_t \cos\beta \cos\alpha \approx V_t \qquad \text{for small angles of } \alpha \text{ and } \beta$$

where V_x = x-axis airspeed velocity

V_t = true airspeed

α = angle of attack

β = angle of sideslip

U = inertial velocity along the x-axis

An estimate of $\dot{V}_x$ can be obtained using the following equation.

$$\dot{V}_x = A_x - g \sin\theta - QW + RV + \delta_x$$

where Q = pitch rate

W = z-axis velocity

R = yaw rate

V = y-axis velocity

δ_x = wind acceleration along the x-axis

A_x = acceleration measured by axial accelerometer

Q and R are measured gyro values. V and W can be calculated as follows.

$V = V_t \sin\beta$ = y-axis velocity

$W = V_t \cos\beta \sin\alpha$ = z-axis velocity

The only term unaccounted for is δ_x, wind acceleration. It must be carefully analyzed as an error source in the FI decision. As long as the FI period is not too long relative to this error source, the decision should not be wrong.

Another way to alleviate this error source is to use a moving window to calculate the standard deviation of the airspeed residues (prior to FD) to determine the threshold size for FI. Wind turbulence associated with wind acceleration increases the threshold, thereby protecting from a false FI.

A filter is used to get a good estimate of any steady-state error on $\dot{V}_x$ as shown in figure 6.2-1. The term a is an estimate of error on $\dot{V}_x$, subsequently added to each V_x calculation in the FI residue equations to effectively cancel the error.

V_{tavg} is the average of the two true airspeed sensors. When an FD occurs the filter is no longer updated. FD is accomplished by the means discussed in section 6.1.

The airspeed residues are calculated as:

$$V \text{ Residue \#1} = \Sigma(\text{current } V_{t1} \text{ measured} - \text{previous } V_{t1} \text{ measured} - \dot{V}_x \, dt)$$

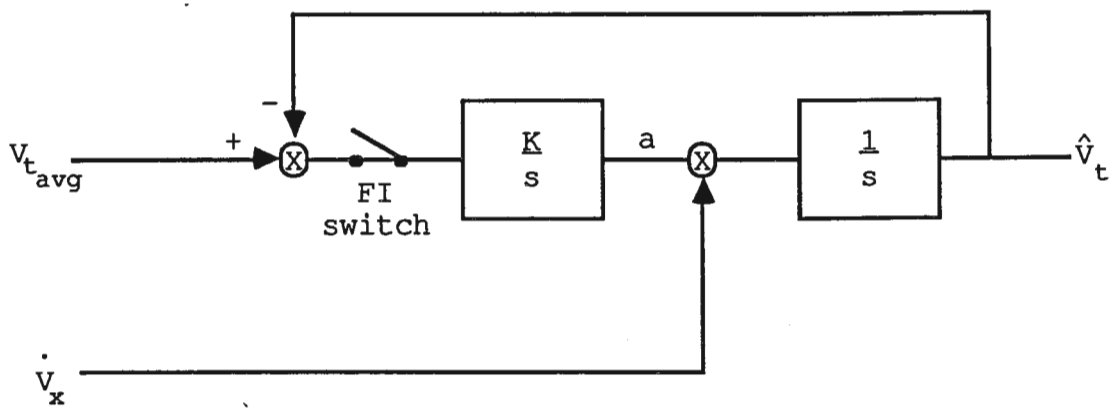
where dt is the update rate. This equation is duplicated for each airspeed sensor measurement. The two residues are then evaluated by an FI algorithm as discussed in section 6.1.

Similar comments concerning error analysis and adaptive threshold solutions apply for this example also. The standard deviation of the average of the sensors could be calculated with a moving window prior to an FD.

6.3. Stability Augmentation System

A large number of advanced aircraft are not statically stable. This is a concern because the airframe is no longer inherently stable. As an aircraft's center of gravity moves aft, the wing has a decreasing capacity to generate a nose down moment in response to an increase in angle of attack. At the neutral point, there is no pitch-axis response due to a change in angle of attack. When the center of gravity is farther aft than the neutral point, a pitch-up moment is generated from an increase in angle of attack which may make the aircraft very difficult to fly.

A Relaxed Static Stability (RSS) aircraft requires an SAS to restore stability to reasonable levels. If the SAS function is critical, it makes sense to implement AR rather than to increase the number of sensors. This function is dependent on feedback from inertial sensors, pitch angle, pitch rate, and vertical acceleration, or air data sensors, true airspeed, and angle of attack, to prevent a rapid pitch-axis divergence.



a is an estimate of error on $\dot{V}_x$, subsequently added to each $\dot{V}_x$ calculation in the fault isolation residue equations to effectively cancel the error.

FIGURE 6.2-1. EXAMPLE FILTER USED FOR TRANSLATIONAL KINEMATICS REDUNDANCY

An estimate of pitch rate and vertical acceleration can be determined with the following equations.

$$Q_E = Q_m - \dot{\theta}\cos\phi_m + \dot{\tau}\sin\phi_m\cos\theta_m$$

$$A_{zE} = A_{zm} - U_m(Q_m - \dot{\alpha}_m) - g\cos\theta_m\cos\phi_m \text{ assuming that } V = 0$$

where Q = pitch rate

A_z = vertical acceleration

U = x-axis velocity

g = acceleration due to gravity

ϕ = roll angle

θ = pitch angle

τ = yaw angle

α = angle of attack $\approx W/U$ where W is the vertical velocity.

The subscript m denotes measured quantities and the subscript E denotes an estimated value.

Figure 6.3-1 shows one way to implement filters for these two estimated values. FD and isolation is as described in section 6.1.

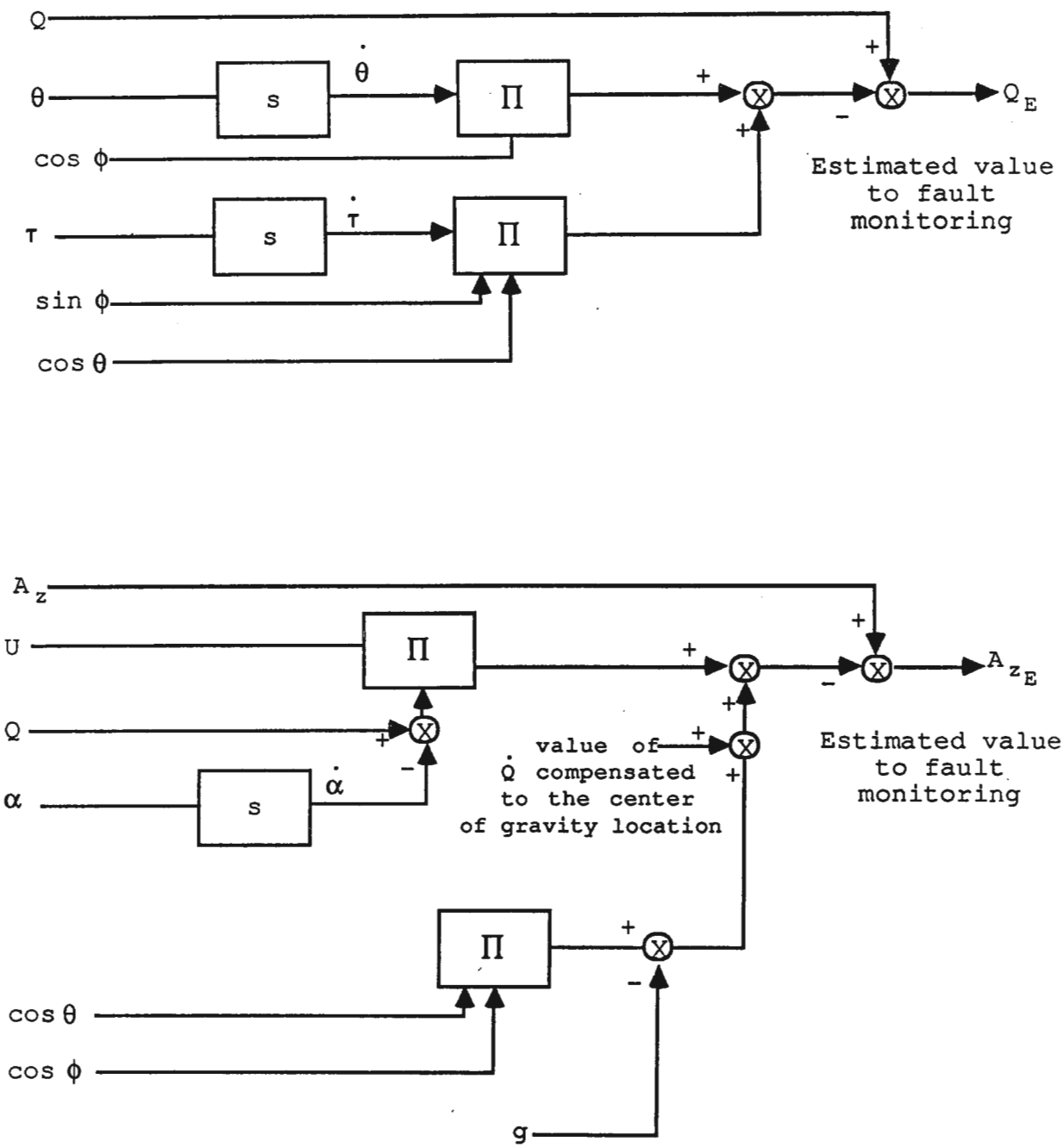


FIGURE 6.3-1. EXAMPLE FILTERS USED FOR STABILITY AUGMENTATION SYSTEM

BIBLIOGRAPHY

Athans, M. and D. Willner, "A Practical Scheme for Adaptive Aircraft Flight Control Systems," Symposium on Parameter Estimation Techniques and Applications in Aircraft Flight Testing, NASA Flight Research Center, Edwards Air Force Base, April 24-25, 1973.

Beard, R. V., Failure Accommodation in Linear Systems Through Self-Reorganization, Report MVT-71-1, Man-Vehicle Laboratory, Cambridge, MA, February 1973.

Buxbaum, P. J. and R. A. Haddad, "Recursive Optimal Estimation for a Class of Nongaussian Processes," Proceedings of Symposium on Computer Processing in Communications, Polytech Institute of Brooklyn, April 8-10, 1969.

Chien, T. T., An Adaptive Technique for a Redundant-Sensor Navigation System, Report T-560, Draper Labs, Cambridge, MA, February 1972.

Clark, R. N., et al., "Detecting Instrument Malfunctions in Control Systems," IEEE Transactions on Aerospace and Electronic Systems, Vol. AES-11, No. 4, July 1975.

Cunningham, T., and G. Hartmann, Fault Tolerant Digital Flight Control with Analytical Redundancy, Technical Report AFFDL-TR-77-25, May 1977.

Davis, M. H. A., "The Application of Nonlinear Filtering to Fault Detection in Linear Systems," IEEE Transactions on Automatic Control, Vol. AC-20, No. 2, April 1975.

Deckert, J. C., et al., Reliable Dual- Redundant Sensor Failure Detection and Identification for the NASA F-8 DFBW Aircraft, NASA Contractor Report 2944, February 1978.

Desai, M. N., J. C. Deckert, and J.J. Deyst, "Dual Sensor Failure Identification Using Analytical Redundancy," Journal of Guidance and Control, Vol. 2, No. 3, May-June 1979.

Deyst, J. J. and J. C. Deckert, "RCS Jet Failure Identification for the Space Shuttle," Proceedings of IFAC 1975, Cambridge, MA, August 1975.

Hartmann, G. L., et al., Digital Adaptive F-8C Control Laws, Final Report NAS 1-13358, July 1975.

Jones, H. L., Failure Detection in Linear Systems, PhD Thesis, Department of Aeronautics and Astronautics, MIT, Cambridge, MA, September 1973.

Kaniuka, W. W., et al., New Flight Control Technologies for Future Naval Aircraft, Report No. NADC-82240-60, September 1982.

Kerr, T. H., "A Two Ellipsoid Overlap Test for Real-Time Failure Detection and Isolation by Confidence Regions," IEEE Conference on Decision and Control, Phoenix, AZ, November 1974.

Lainiotis, D. G., "Joint Detection, Estimation, and System Identification," Information and Control, Vol. 19, No. 1, August 1971.

Maybeck, P. S., "Failure Detection Through Functional Redundancy," AFFDL-TR-74-3, January 1974.

McAulay, R. J. and E. Denlinger, "A Decision-Directed Adaptive Tracker," IEEE Transactions on Aerospace and Electronic Systems, Vol. AES-9, March 1973.

McGarty, T. P., "State Estimation with Faulty Measurements: An Application of Bayesian Outlier Rejection," Proceedings of the Fifth Symposium on Nonlinear Estimation and Its Applications, San Diego, CA, September 1974.

Mehra, R. K. and J. Peschon, "An Innovations Approach to Fault Detection and Diagnosis in Dynamic Systems," Automatica, Vol. 7, 1971.

Meier, L., D. W. Ross, and M. B. Glaser, "Evaluation of the Feasibility of Using Internal Redundancy to Detect and Isolate On-Board Control Data Instrumentation Failures," AFFDL-TR-70-172, January 1971.

Montgomery, R. C. and A. K. Caglayan, "A Self-Reorganizing Digital Flight Control System for Aircraft," AIAA 12th Aerospace Sciences Meeting, Washington, D.C., January 30-February 1, 1974.

Montgomery, R. C. and D. B. Price, "Management of Analytical Redundancy in Digital Flight Control Systems for Aircraft," AIAA Mechanics and Control of Flight Conference, Anaheim, CA, August 5-9, 1974.

Mulcare, D. B., L. E. Downing, and M. K. Smith, Analytical Sensor Redundancy Assessment, DOT/FAA/CT-86/32, Draft Report, June 1987.

Pierce, B. D. and D. D. Sworder, "Bayes and Minimax Controllers for a Linear System with Stochastic Jump Parameters," IEEE Transactions on Automatic Control, Vol. AC-16, No. 4, August 1971.

Ratner, R. S. and D. G. Luenberger, "Performance-Adaptive Renewal Policies for Linear Systems," IEEE Transactions on Automatic Control, Vol. AC-14, No. 4, August 1969.

Ross, S. M., "Introduction to Probability Models," Academic Press, 1972.

Sanyal, P. and C. N. Shen, "Bayes' Decision Rule for Rapid Detection and Adaptive Estimation Scheme with Space Applications," IEEE Transactions on Automatic Control, Vol. AC-19, June 1974.

Stein, G. and G. L. Hartmann, F-8C Adaptive Control Extensions, NASA Contract NAS 1-13383, June 1976.

Sworder, D. D. and V. G. Robinson, "Feedback Regulators for Jump Parameter Systems with State and Control Dependent Transition Rates," IEEE Transactions on Automatic Control, Vol. AC-18, No. 4, August 1973.

Sworder, D. D., "Bayes' Controllers with Memory for a Linear System with Jump Parameters," IEEE Transactions on Automatic Control, Vol. AC-17, February 1972.

Weinstein, W., Feasibility and Design Studies of an Integrated Sensory Subsystem (ISS) for Advanced V/STOL Aircraft, Report No. NADC-76259-30, March 1978.

Willsky, A. S., J. J. Deyst, and B. S. Crawford, "Adaptive Filtering and Self-Test Methods for Failure Detection and Compensation," Proceedings of the 1974 JACC, Austin, Texas, June 19-21, 1974.

Willsky, A. S. and H. L. Jones, "A Generalized Likelihood Ratio Approach to State Estimation in Linear Systems Subject to Abrupt Changes," Proceedings of the 1974 Conference on Decision and Control, Phoenix, AZ, November 1974.

GLOSSARY

ANALYTICAL REDUNDANCY. The use of software algorithms which use known mathematical relationships between different sensors for sensor failure detection and replace most of additional redundant sensor hardware.

CONTROL LAW. The physical relationship between various sensors and control surfaces.

COVERAGE. The percent confidence level of a given analytical redundancy fault detection and isolation algorithm for all types of faults.

DIAGNOSTIC FILTER. An analytical algorithm which processes data from N functionally related sensors. The data are used to estimate some sensor outputs and assess the correct functioning of the sensors.

DUAL FAIL-OPERATIONAL. A reliability requirement placed on a system which requires the system to be operational after two failures have occurred.

FAIL-OPERATIONAL. A reliability requirement placed on a system which requires the system to be operational after a single failure has occurred.

FAIL-SAFE. A reliability requirement placed on a system which requires that safe flight not be hindered even after a failure.

FALSE ALARM. The declaration of a fault by a fault detection monitor or algorithm when there is no fault.

FAULT DETECTION. The determination that a sensor is faulted by using a software algorithm.

FAULT ISOLATION. The determination that a particular sensor is faulted by using a software algorithm.

FAULT TOLERANCE. Accommodation of sensor hardware faults based on some type of comparator scheme.

FLIGHT-CRITICAL. A description of functions whose failure would contribute to or cause a failure condition preventing the continued safe flight and landing of the aircraft.

MISSED ALARM. The failure of a fault detection monitor or algorithm to detect a fault when there is a sensor fault.

MULTIPLE TRIP MONITOR. A fault detection algorithm which declares a fault after the sensor output has exceeded a predefined threshold N times.

OBSERVER. An algorithm which models physical relationships between sensor data and uses the data to provide fault detection for one or more sensors. This is also known as a Luenberger observer or a signal blender.

REDUNDANCY MANAGEMENT. The computer processing which is needed to implement fault detection and isolation algorithms.

REVERSION MODE. The high level of redundancy in a system having different redundancies requirements for some sensors. Critical sensors may have a high level of redundancy while other sensors have low levels.

SENSOR. An instrument which measures a particular physical parameter. The data output may be digital or analog and is utilized by the flight computer.

SEQUENTIAL LIKELIHOOD RATIO TEST. A fault detection algorithm which is based on two hypothesized density functions of no fault or sensor fault.

SEQUENTIAL PROBABILITY RATIO TEST. See sequential likelihood ratio test.

SUPER-DIAGNOSTIC FILTER. An algorithm which provides all the capabilities of a diagnostic filter. Additionally, it can isolate a specific faulted sensor. At the current time, this is the most complex technique used to implement analytical redundancy.

ACRONYMS

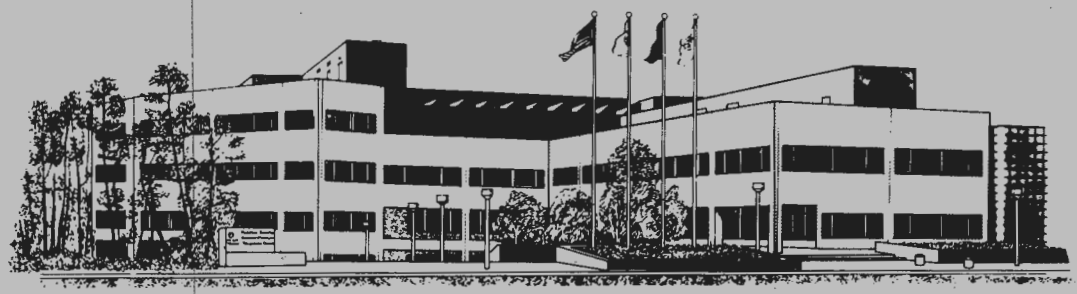
AFFDL	Air Force Flight Dynamics Laboratory
AK	Altitude Kinematics
AR	Analytical Redundancy
C	Comparator
CARSRA	Computer Aided Redundant System Reliability Analysis
DF	Diagnostic Filter
DFC	Digital Flight Control
DFCS	Digital Flight Control System
DOT	Department of Transportation
FAA	Federal Aviation Administration
FBW	Fly-By-Wire
FD	Fault Detection
FI	Fault Isolation
NADC	Naval Air Development Center
NASA	National Aeronautics and Space Administration
RM	Redundancy Management
RSS	Relaxed Static Stability
SAS	Stability Augmentation System
SDF	Super-Diagnostic Filter
SLRT	Sequential Likelihood Ratio Test
SPRT	Sequential Probability Ratio Test

U.S. Department of Transportation
Federal Aviation Administration

DOT/FAA/CT-88/10

HANDBOOK-VOLUME II
DIGITAL SYSTEMS VALIDATION

CHAPTER 7
ANALYTICAL SENSOR REDUNDANCY



PREPARED BY:

COMPUTER RESOURCE MANAGEMENT, INC.
950 HERNDON PARKWAY, SUITE 360
HERNDON, VIRGINIA 22070

PREPARED FOR:

FEDERAL AVIATION ADMINISTRATION
TECHNICAL CENTER
ATLANTIC CITY INTERNATIONAL AIRPORT, NEW JERSEY 08405

NOTICE

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The United States Government assumes no liability for the contents or use thereof.

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the objective of this report.

TABLE OF CONTENTS

Section		Page
1.	INTRODUCTION	7-1
	1.1. Control Law Modification	7-2
	1.2. Fault Tolerant Design	7-2
	1.3. Scope	7-4
2.	OVERVIEW	7-5
	2.1. Analytical Redundancy Building Blocks	7-5
3.	ANALYTICAL REDUNDANCY - DESIGN APPROACH	7-11
	3.1. Observer/Signal Blender Filter	7-16
	3.2. Diagnostic Kalman Filters	7-16
	3.3. Super-Diagnostic Kalman Filter Design	7-16
	3.3.1. Multiple Hypothesis Tests	7-17
	3.3.2. Parameter Identification	7-17
	3.3.3. Generalized Likelihood Ratio Methods	7-17
	3.3.4. Modified Filter Design	7-18
	3.3.5. Jump Process Estimation	7-18
	3.4. Redundancy Management	7-18
4.	ANALYTICAL REDUNDANCY - RELIABILITY	7-21
5.	ANALYTICAL REDUNDANCY - FAULT DETECTION AND ISOLATION	7-23
	5.1. Multiple Trip Monitor	7-23
	5.2. Sequential Likelihood Ratio Tests	7-25
	5.3. Fault Modeling	7-27
6.	SUPPLEMENTAL WORKED EXAMPLES	7-29
	6.1. Altitude Kinematics	7-29
	6.2. Translational Kinematics	7-32
	6.3. Stability Augmentation System	7-33
BIBLIOGRAPHY		7-37
GLOSSARY		7-40
ACRONYMS		7-42

LIST OF ILLUSTRATIONS

Figure		Page
2-2	ROTATIONAL RELATIONSHIPS USED FOR ANALYTICAL REDUNDANCY	7-7
2.1-1	ANALYTICAL REDUNDANCY CONCEPT	7-8
2.1-2	ANALYTICAL SENSOR REDUNDANCY FAULT DETECTION LOGIC	7-9
3-1	EXAMPLES OF FAULT DETECTION USING DIAGNOSTIC FILTERS AND INTERCHANGEABLE SENSORS	7-12
3-2	EXAMPLES OF FAULT DETECTION USING DIAGNOSTIC FILTERS AND NONINTERCHANGEABLE SENSORS	7-13
3-3	EXAMPLE OF FAULT DETECTION WITH REVERSION MODES	7-15
5.1-1	MULTIPLE TRIP FAULT DETECTION MONITOR	7-24
5.1-2	SEQUENTIAL LIKELIHOOD FAULT DETECTION MONITOR	7-27
6.1-1	THIRD ORDER VERTICAL COMPLEMENTARY FILTER FOR ALTITUDE KINEMATICS	7-30
6.1-2	POSSIBLE REDUNDANCY MANAGEMENT FOR ALTITUDE KINEMATICS FAULT ISOLATION	7-31
6.2-1	EXAMPLE FILTER USED FOR TRANSLATIONAL KINEMATICS REDUNDANCY	7-34
6.3-1	EXAMPLE FILTERS USED FOR STABILITY AUGMENTATION SYSTEM	7-36

LIST OF TABLES

Table		Page
3-1	SENSOR REDUCTION COMPARISONS FOR ANALYTICAL REDUNDANCY BUILDING BLOCKS	7-14
4-1	SYSTEM RELIABILITY	7-22

1. INTRODUCTION

The advent of modern digital controls has greatly changed the control technology and hardware implementation of aircraft flight control. These Digital Flight Control Systems (DFCS) provide enhanced performance for stability and control augmentation over an extremely wide range of flight envelope parameters. They also offer improved capabilities to operate in differing aircraft configurations from takeoff with wheels and flaps down to a high speed clean configuration with low drag, thrust, and low g forces. In addition to the performance enhancements in Digital Flight Control (DFC), the DFC computer has the capability to perform other computations to effect improvements in reliability and fault tolerance. Elaboration of these later features is the major thrust of this portion of the handbook.

The DFC improves reliability due to two factors. The first is the increased reliability of digital computer hardware relative to older technology; this enhancement leads to tens of thousands of hours between failures. The second factor is the increased reliability of the sensors and actuators which leads to several hundreds of hours between failures. The principal factor affecting DFC system reliability is reliability of the sensors and actuators.

The overall reliability required in today's flight control systems dictates the need for 10^{-9} probability of failure in cases where stability augmentation is needed for safe flight. This requires an increase in the reliability of the DFC components. The reliability of DFC computers may be achieved by utilizing dual processors, error detection, and fault tolerant designs. An increase in reliability of sensors and actuators may be achieved by means of redundant elements (dual or triple).

Assessment of this situation leads to a conclusion that redundant actuators and linkages are an adequate solution for actuation, and that DFC computers are adequate. The term Analytical Redundancy (AR) refers to the scheme for increasing the reliability of sensors by using the DFC computer for Fault Detection (FD) and isolation and for analytically determining estimates of sensor values.

Reliable DFC systems are burdened with large numbers of sensors which are required for both quality and redundancy. The demand for control quality has expanded the types of sensors used in basic augmentation functions. The yaw damper currently requires sensor data concerning attitude, angle of attack, and airspeed. The pitch damper requires a complex multifunction control law along with data from several sensors to drive multiple surfaces. The complexity of the sensors has improved quality; however, designs that meet overall flight control system reliability requirements lead to triple and quadruple replication of system components.

High levels of fault tolerance are mandatory for flight-critical control systems. Current Fly-By-Wire (FBW) systems utilize redundant computers,

actuators, and sensors to achieve the necessary fault tolerance. Fault isolation (FI) is determined by some type of comparison or voting between like sensors. A one out of two sensor failure cannot be isolated by comparison techniques. In order to improve the reliability of a redundant set of sensors, one or more of the following must be implemented: increase the reliability of the existing sensors; add additional like sensors; synthesize the needed information; or isolate the remaining sensor if a failure occurs at the duplex level.

A large amount of sensor hardware is costly from a system standpoint. Sensors may act as antennas to increase electromagnetic susceptibility. The overall aircraft design may be constrained due to a large number of sensors. Therefore, there is a strong motivation to reduce the number of sensors. Significant reductions appear to be possible through a variety of methods. Recognized techniques for sensor reduction are control law modification and fault tolerant design. These methods may be used in combinations to optimize the number of sensors necessary for a particular application.

1.1. Control Law Modification

Control law modification ensures that operational requirements can be met with the minimum number of sensors and a better control system design. The control laws are modified to optimize the use of sensed data utilizing the analytical relationships which exist between sensors. The flight controller uses fewer sensed values in the basic single channel control laws, yet maintains the same performance as the controller which utilized all the sensor data. These modifications are application dependent.

Signal synthesis is one example of control law modification. The number of sensors required is reduced by synthesizing a signal or group of signals from a reduced set. The elements used to perform the synthesis are Kalman filters or Luenberger observers. Reduction in the number of sensors is a trade-off against increased complexity of the control laws; this is a critical factor for analog systems. The trade-off in digital systems occurs in system performance. The bandwidth and noise content of the synthesized signal must be equivalent to the sensor eliminated. These techniques must be applied with care because of potential physical limitations which cannot be practically overcome.

Integrated flight management is another example of control law modification. This method works on the principle that subsystems which use common sensor types are combined. An example is the combination of the navigator and the flight control. The signal quality derived from the navigator is high and the data could easily be used by the flight control. Issues here are cost (since the navigator is normally not flight critical) and multiple failures.

1.2. Fault Tolerant Design

The following fault tolerant design techniques exist to meet reliability requirements using a minimum number of sensors. These techniques use fewer sensors to achieve the same level of redundancy as a quad-redundant system.

- Skewed and special sensors.

- Sensor integration.
- In-line sensor monitoring.
- Analytical sensor redundancy.

A skewed sensor arrangement significantly reduces the number of sensors required for Redundancy Management (RM). Skewed sensors do not have to be placed on orthogonal axes. The x, y, and z components of the sensed quantity can be determined using an appropriate coordinate transformation matrix. If one of the skewed sensors fails, the three orthogonal components can still be determined with another sensor and its transformation matrix.

A quad-redundant dual fail-operative system with orthogonal gyros in three axes requires twelve gyros. The same properly configured system with skewed gyros requires only six to maintain the same redundancy requirements. Skewing accelerometers can similarly reduce the number of sensors from twelve to six. Potential limitations are differences in scale and resolution requirements for each axis. For the skewed configuration, each gyro must have the same specifications. Depending upon the gyro chosen, either the signal quality may be limited or the instrumentation cost may be greater. Weinstein (1978) and Kaniuka, et al., 1982 show adequate performance for this technique.

Special sensors measure linear combinations of variables. The measurements are skewed in measurement space rather than geometrically, i.e., the sensor measures both acceleration and body rate. A conventional quad-redundant system could be replaced by five special sensors. These sensors are typically unique to the application and have a higher cost. A multipurpose air data sensor which measured pitot pressure, static pressure, angle of attack, and angle of sideslip was implemented in the system described by Weinstein (1978) and Kaniuka, et al., 1982.

Subsystem sensor integration can also be used to reduce the number of sensors. This technique uses data from subsystems which are not functionally related for monitoring and tie-breaking. An example is on the space shuttle where derived rates from the navigation system are used in the primary flight control system for voting.

Fault tolerant design can be achieved through in-line monitoring. The state-of-the-art for servos and computers is near 99 percent self-test confidence. In-line self-test feasibility for sensors is limited for several reasons. The sensor input is unknown except for special test signals. There is no way to test for sensor installation errors. Any sensor self-test adds considerable complexity which may be undesirable. Typically, a self-test can only detect very large faults, particularly in a dynamic environment.

Analytical Redundancy provides a means for substituting software algorithms for hardware sensors. Sensors are eliminated by analysis using known relationships between sensed values. Various simulation studies have shown the effectiveness of utilizing AR to increase system reliability and to reduce the amount of sensor hardware.

1.3. Scope

This document details the uses and limitations of analytical sensor redundancy. Section 2 gives an overview of AR. The physical relationships which can be utilized are written in equation form. The basic concept and building blocks are discussed. Section 3 describes various algorithms and design tools used to implement the filters used in AR processing. Its feasibility and performance are discussed in section 4. Algorithms and methods used for FD and isolation are detailed in section 5. Examples of the filters, RM, and sensor relationships for three cases where analytical sensor redundancy is easily implemented are given in section 6.

2. OVERVIEW

The basic idea of AR is to use known geometric and dynamic relationships between different sensors in order to detect and isolate sensor failures. These relationships can be defined in translational and rotational equations as shown in figures 2-1 and 2-2. Figure 2-2 also shows a diagram of the Euler angle relationships.

Cunningham and Hartmann (1977) describe the application of these mathematical relationships to eleven different measured quantities, normal acceleration, lateral acceleration, angle of attack, true airspeed, altitude, P, Q, R, $\emptyset$, θ, and ψ.

Four categories of AR were utilized by Deckert, et al., 1978. Translational kinematics redundancy exists between the integrated accelerometer outputs, vertical and rate gyros, and the air data sensor outputs. Translational dynamics redundancy uses the aerodynamic forces on the aircraft measured by the accelerometers and stored coefficients to relate to aerodynamic forces calculated from air-data sensor data. Rotational kinematics redundancy relates the integrated outputs of the rate gyros and the vertical and directional gyro outputs. Altitude Kinematics (AK) redundancy is defined as the relationship between the altitude measured by the altimeter and the computed altitude from a double integration of the accelerometer and vertical gyro output data. These relationships are defined in the equations in figures 2-1 and 2-2. Examples of altitude and translational kinematics redundancy processing are given in sections 6.1 and 6.2.

2.1. Analytical Redundancy Building Blocks

Many specific approaches to AR have been developed, ranging from simple signal blenders to complex combinations of Kalman filters.

In general, AR is approached using one of two basic building blocks. A Diagnostic Filter (DF) is an algorithm which processes data from a family of N functionally related sensors in order to estimate signals and to assess the functioning of the sensors. It outputs signal estimates and an error flag which indicates that all sensors are functioning properly or that one of them has failed. A Super-Diagnostic Filter (SDF) performs the functions of a DF and can additionally isolate the specific faulted sensor. One can use assemblies of DFs to construct an SDF through truth table logic.

The concept of AR and associated FD are shown in figures 2.1-1 and 2.1-2.

Translational Equations

Inertial Velocity

$$\dot{X}_e = \cos\theta \cos\tau\, U + (\sin\phi \sin\theta \cos\tau - \cos\phi \sin\tau)V + (\cos\phi\sin\theta \cos\tau + \sin\phi \sin\tau)W$$

$$\dot{Y}_e = \cos\theta \sin\tau\, U + (\sin\phi \sin\theta \sin\tau + \cos\phi \cos\tau)V + (\cos\phi \sin\theta \cos\tau - \sin\phi \cos\tau)W$$

$$\dot{Z}_e = -\sin\theta\, U + \sin\phi \cos\theta\, V + \cos\phi \cos\theta\, W$$

Acceleration

$$\dot{U} = A_x - g \sin\theta - QW + RV$$

$$\dot{V} = A_y + g \cos\theta \sin\phi - RU + PW$$

$$\dot{W} = A_z + g \cos\theta \cos\phi - PV + QU$$

where X_e, Y_e, Z_e = earth referenced forward, lateral, and vertical positions
U, V, W = forward, lateral, and vertical body velocities, respectively
A_x, A_y, A_z = forward, lateral, and vertical body accelerations
g = acceleration due to gravity
θ, ϕ, τ = Euler pitch, roll, and yaw angles
P, Q, R = body roll, pitch, and yaw rates

FIGURE 2-1. TRANSLATIONAL RELATIONSHIPS USED IN ANALYTICAL REDUNDANCY

Rotational Equations

Euler Rates

$$\dot{\phi} = P + (Q \sin\phi + R \cos\phi) \tan\theta$$

$$\dot{\theta} = Q \cos\phi - R \sin\phi$$

$$\dot{\tau} = (Q \sin\phi + R \cos\phi) \sec\theta$$

Body Rates

$$P = \dot{\phi} - \dot{\tau} \sin\theta$$

$$Q = \dot{\theta} \cos\phi - \dot{\tau} \sin\phi \cos\theta$$

$$R = \dot{\tau} \cos\phi \cos\theta - \dot{\theta} \sin\phi$$

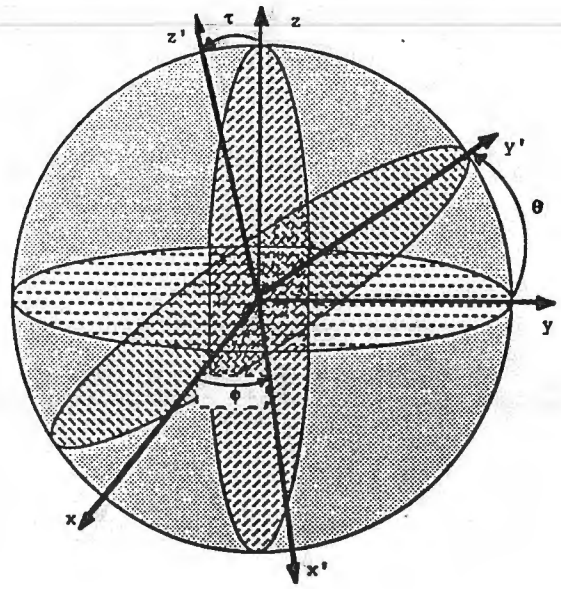
Moments

$$L = \dot{P} I_{xx} - \dot{R} I_{xz} + QR (I_{zz} - I_{yy}) - PQ I_{xz}$$

$$M = \dot{Q} I_{yy} + PR (I_{xx} - I_{zz}) + (P^2 - R^2) I_{xz}$$

$$N = \dot{R} I_{zz} - \dot{P} I_{xz} + PQ (I_{yy} - I_{xx}) + QR I_{xz}$$

where θ, ϕ, τ = Euler pitch, roll, and yaw angles
P, Q, R = body roll, pitch, and yaw rates
L, M, N = moment about the body x, y, and z axes
I_{xx}, I_{yy}, I_{zz} = moment of inertia about the x, y, and z axes
I_{xz} = cross moment of inertia (xz axis)



EULER ANGLE DIAGRAM

FIGURE 2-2. ROTATIONAL RELATIONSHIPS USED FOR ANALYTICAL REDUNDANCY

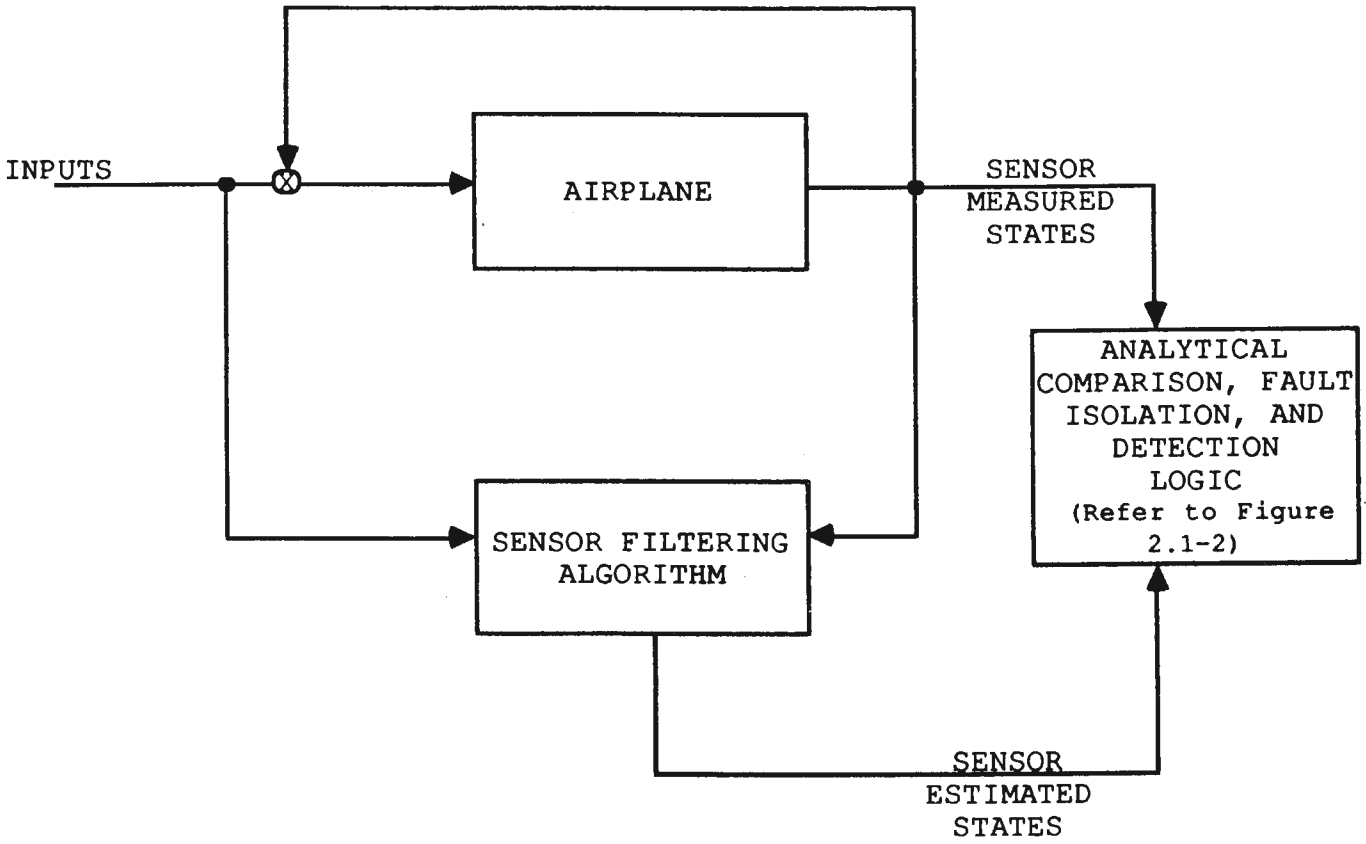


FIGURE 2.1-1. ANALYTICAL REDUNDANCY CONCEPT

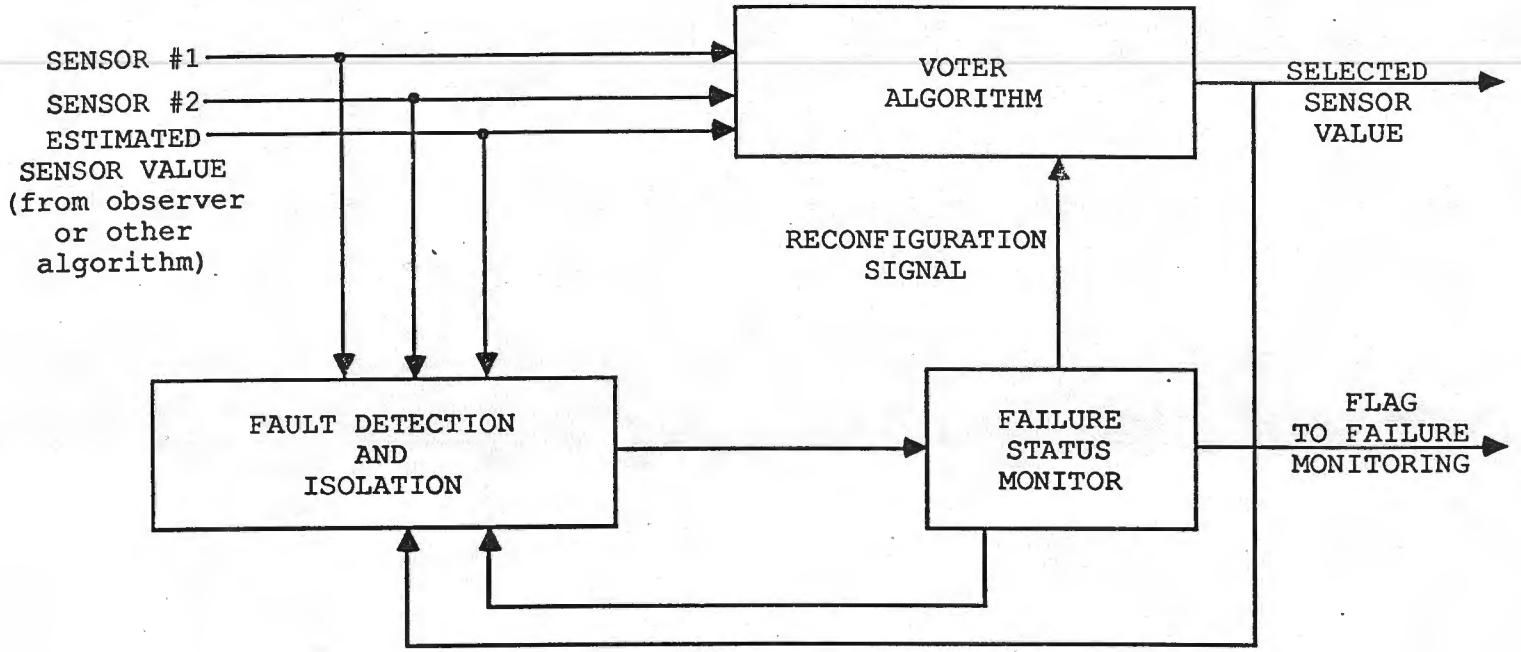


FIGURE 2.1-2. ANALYTICAL SENSOR REDUNDANCY FAULT DETECTION LOGIC

3. ANALYTICAL REDUNDANCY - DESIGN APPROACH

A literature survey was performed by Cunningham and Hartmann (1977) to examine currently available techniques that were able to detect and isolate faults in aircraft sensors and meet current onboard flight computer allocations. All such methods for failure detection were found to lie in three categories.

- Assemblies of DFs (see Meier, Ross, and Glaser, 1971; Maybeck, 1974; Hartmann, et al., 1975; and Clark, et al., 1975).
- Specific DF design techniques (see Kerr, 1974 and Mehra and Peschon, 1971).
- Explicit SDF design techniques.

The first two categories also include observer/blender techniques. Mulcare, Downing, and Smith (1987) and Deckert, et al., 1978 utilize these techniques.

The three filter types listed above can be combined in many ways to obtain useable AR results. The end product may vary considerably in accuracy and complexity. The structure of the filter depends on the overall RM approach, whether uniform redundancy or different levels of redundancy are required throughout the system.

Uniform redundancy schemes categorize sensors as interchangeable or noninterchangeable. Interchangeable sensors can substitute for each other when a failure occurs. Appropriate coordinate transformations must be accounted for, if necessary. If a noninterchangeable sensor fails, an identical one must replace it in order to maintain performance.

If N fully operational sensors are required to maintain aircraft operations, then a minimum of N+2 interchangeable sensors and 3N noninterchangeable sensors are required for dual fail-operational performance when using AR. In order to resolve conflicts using traditional voting techniques, N+3 and 4N sensors are needed to maintain the same performance.

Fault Detection using DFs with interchangeable sensors is illustrated in figure 3-1. The figure shows FD for N, the number of sensors required for safe flight, equal to 2 for fail-operational performance. The truth table shows that each column of the error flags is a unique binary word and that the sensor inputs to each filter are uniquely determined by each row of the table. The generalized way to assemble DFs for interchangeable sensors is given below.

- Determine the number of sensors required to maintain safe flight and meet operational requirements (N is the number of sensors necessary for safe flight).

N + 1 for fail-operational performance
N + 2 for dual fail-operational performance

- Determine the number of DFs (the brackets indicate to use the next largest integer of the enclosed expression).

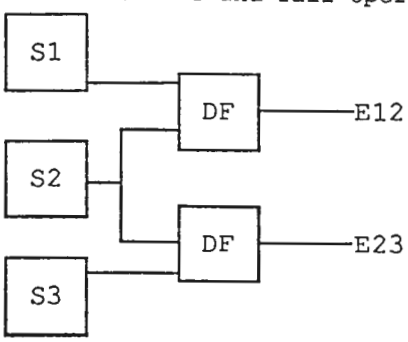
  $[\log_2 (N + 2)]$ for fail-operational performance
  $[\log_2 (N + 3)]$ for dual fail-operational performance

- Determine sensor inputs to each filter from the truth table.

The FD capability of a DF combined with comparators for noninterchangeable sensors can also be verified with a truth table. See figure 3-2 for an illustration of fail-operational performance with N, the number of sensors necessary to maintain safe flight, equal to 2. The generalized arrangement uses one filter and N comparators for fail-operational and one filter and 2N comparators for dual fail-operational. The truth table here contains FD for single sensor failures only. It is generally assumed that failures occur singly when designing FD algorithms. A single SDF is all that is needed in all cases to provide FD and isolation. This type of filter may be quite complex in order to achieve the same results as DFs.

INTERCHANGEABLE SENSORS

for N=2 and fail-operational performance



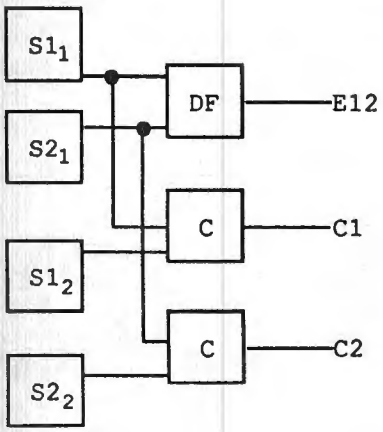
TRUTH TABLE

	SINGLE SENSOR FAILURE			
ERROR FLAGS	NONE	S1	S2	S3
E12	0	1	1	0
E23	0	0	1	1

FIGURE 3-1. EXAMPLES OF FAULT DETECTION USING DIAGNOSTIC FILTERS AND INTERCHANGEABLE SENSORS

NONINTERCHANGEABLE SENSORS

for N=2 and fail-operational performance



TRUTH TABLE

ERROR FLAGS	SINGLE SENSOR FAILURE				
	NONE	$S1_1$	$S2_1$	$S1_2$	$S2_2$
E12	0	1	1	0	0
C1	0	1	0	1	0
C2	0	0	1	0	1

C = Comparator
DF = Diagnostic Filter
N = # of sensors needed to maintain safe flight

FIGURE 3-2. EXAMPLES OF FAULT DETECTION USING DIAGNOSTIC FILTERS AND NONINTERCHANGEABLE SENSORS

The largest amount of sensor reduction is achieved for noninterchangeable sensors when using AR. In this case, AR implemented with one DF in combination with 2N comparators has the same benefit as one SDF. An SDF which is more complex will not be feasible. A comparison of the sensor reductions possible with these analytical techniques and traditional voting is given in table 3-1. The number of comparators and DFs needed for AR are also given. N is defined as the minimum number of sensors required to maintain safe flight.

There is a sensor reduction benefit for interchangeable sensors; however, it is a small one. AR provides little or no benefit to save one sensor. More DFs are needed to match the capabilities of a single SDF than for noninterchangeable sensors.

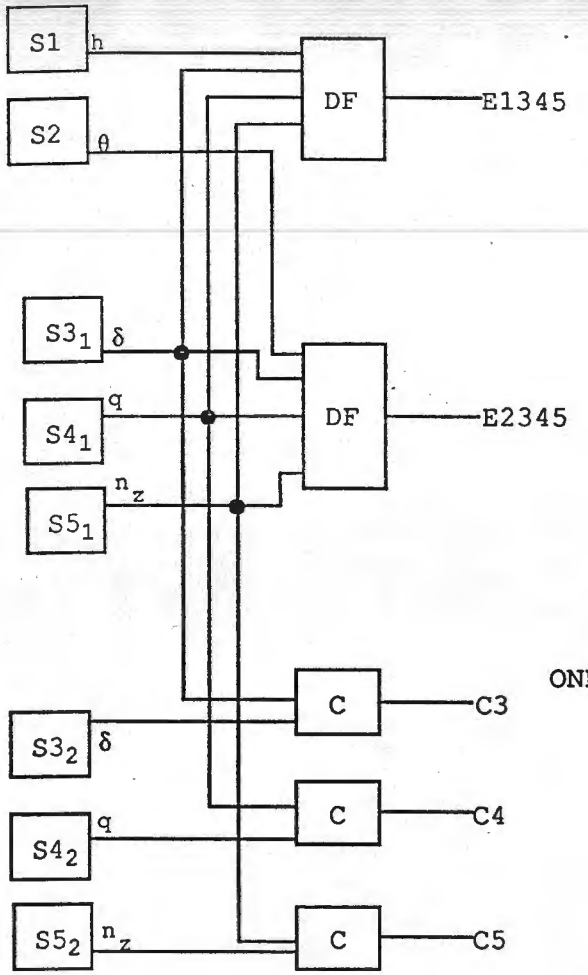
Current flight control systems normally do not have the same redundancy requirements for all sensors. A more realistic setup may have high levels of redundancy for some critical sensors and lower levels for others. The high levels of redundancy are known as reversion modes. While this may be application dependent, the basic principles can be used to obtain a failure detection scheme as shown in figure 3-3. In order to implement the reversion modes, sufficient filters are required to distinguish between four outer loop logic conditions: no failure; S1 failed; S2 failed; and $S3_1$, $S4_1$, or $S5_1$ failed. The error flags, E1345 and E2345, are used to obtain this information. The error flags, C3, C4, and C5, must be used to determine which of the inner loop sensors have failed.

TABLE 3-1. SENSOR REDUCTION COMPARISONS FOR ANALYTICAL REDUNDANCY BUILDING BLOCKS

FAIL-OPERATIONAL	NUMBER OF SDF'S	NUMBER OF DIAGNOSTIC FILTERS	NUMBER OF COMPARATORS	NUMBER OF SENSORS
INTERCHANGEABLE SENSORS				
DETECTION WITH SDF'S	1	0	0	N+1
WITH DF'S	0	$[\text{LOG}_2(N+2)]$	0	N+1
WITH VOTING	0	0	N+3	N+2
NONINTERCHANGEABLE SENSORS				
DETECTION WITH SDF'S	1	0	0	2N
WITH DF'S	0	1	N	2N
WITH VOTING	0	0	2N	3N
DUAL FAIL-OPERATIONAL	NUMBER OF SDF'S	NUMBER OF DIAGNOSTIC FILTERS	NUMBER OF COMPARATORS	NUMBER OF SENSORS
INTERCHANGEABLE SENSORS				
DETECTION WITH SDF'S	1	0	0	N+2
WITH DF'S	0	$[\text{LOG}_2(N+3)]$	0	N+2
WITH VOTING	0	0	N+4	N+3
NONINTERCHANGEABLE SENSORS				
DETECTION WITH SDF'S	1	0	0	3N
WITH DF'S	0	1	2N	3N
WITH VOTING	0	0	3N	4N

N = MINIMUM NUMBER OF SENSORS REQUIRED FOR SAFE FLIGHT

[ ] IMPLIES USE THE NEXT LARGEST INTEGER OF THE EXPRESSION WITHIN THE BRACKETS



TRUTH TABLE

ERROR FLAGS	SENSORS FAILED								
	NONE	S1	S2	$S3_1$	$S4_1$	$S5_1$	$S3_2$	$S4_2$	$S5_2$
E1345	0	1	0	1	1	1	0	0	0
E2345	0	0	1	1	1	1	0	0	0
C3	0	0	0	1	0	0	1	0	0
C4	0	0	0	0	1	0	0	1	0
C5	0	0	0	0	0	1	0	0	1

TWO FAIL-SAFE OUTER LOOPS (h), (θ)

ONE FAIL-OPERATIONAL INNER LOOP (δ, q, n_z)
(NON-INTERCHANGEABLE SENSORS)

- h = altitude
- θ = Euler pitch angle
- δ = control surface deflection
- q = body pitch rate
- n_z = normal acceleration
- C = Comparator
- DF = Diagnostic Filter

FIGURE 3-3. EXAMPLE OF FAULT DETECTION WITH REVERSION MODES

Three concepts encompass the categories of failure detection methods. These are examined as a result of various combinations of DFs. These concepts in order of increasing complexity are:

- Observer/signal blender filter.
- Diagnostic Kalman filters.
- Super-diagnostic Kalman filter design.

3.1. Observer/Signal Blender Filter

An observer/blender will model physical relationships using sensor outputs to provide FD for several sensors. It uses a specific DF design and has the least amount of complexity and computer requirements. Monitors may use a three trip level exceedance criterion to detect failures. Trip thresholds can be set to account for unmodeled dynamics and sensor anomalies.

The basic operation of the observer/blender is to define a consistent error function between a measured parameter and its derivative. Both values are measured for all but a few parameters, e.g., lateral acceleration. A relationship such as,

$$V_x = \dot{X}$$

is exact. For perfect measurements, there is no error. Realistically, the measured values of V_x and X are not true representations of V_x and X. There is an error associated with digital integration and sampling. Either V_x or X may contain faults. This implementation of AR must distinguish between the errors associated with measurements and digital sampling and errors associated with the sensors.

3.2. Diagnostic Kalman Filters

Diagnostic Kalman filters incorporate modeling of sensor anomalies (such as, bias, scale factors, and wind gust estimation) in the filter. FD is provided for more sensors than available with the observer/blender. This is achieved by using an assembly of DFs. This concept places moderate requirements on a computer.

3.3. Super-Diagnostic Kalman Filter Design

The super-diagnostic design utilizes Kalman filters as well as linear equations for pitch and lateral directional dynamic equations of motion. Both detection and isolation can be determined. An error signal is created for each sensor involved. The concept is built with an assembly of DFs. Some applications may be very complex.

The largest number of surveyed references are devoted to this topic. These filters are the most difficult to construct. They are constrained more than other techniques due to the amount of complexity which is feasible. Five methods have been examined for SDFs.

- Multiple hypothesis test.
- Parameter identification.
- Generalized likelihood ratio methods.
- Modified filter designs.
- Jump process estimation.

3.3.1. Multiple Hypothesis Tests

Each failure condition of a particular sensor is used to define a hypothesis. The entire range of failure conditions forms a set of hypotheses which are tested against observed data. One Kalman filter is required to test each hypothesis. The filter generates residuals, assuming that the hypothesis is true, which are summed into likelihood functions. The maximum likelihood then identifies the current valid hypothesis. This results in a large number of filters for even a few sensors and at present is too complex for advanced flight computers to utilize. Montgomery and Caglayan (1974), Montgomery and Price (1974), Athans and Willner (1973), Lainiotis (1971), and Buxbaum and Haddad (1969) contain more information.

3.3.2. Parameter Identification

Parameters of each sensor, such as, gain and bias, are defined as unknowns and estimated from sensor data. If the estimates deviate a great deal from nominal values, then a failure is declared. For any procedure used, a separate calculation similar to a Kalman filter is required to compare the nominal values with the estimated sensor parameters. This becomes as complex and inefficient as the previous technique. Mehra and Peschon (1971) and Stein and Hartmann (1976) have investigated this technique.

3.3.3. Generalized Likelihood Ratio Methods

In an attempt to reduce the complexity of this type of failure detection method, initial constraints were placed on the design of an algorithm. These constraints are that only one Kalman filter is available and that sensor failure detection must be decided through the monitoring of residuals from the single filter. This can be done for those failures which have recognizable signatures for correlation processing. The residuals are correlated with known signals, normalized by their expected no-failure value, and compared to predetermined threshold levels. The number of filters has been greatly reduced, however, the correlation functions can be fairly complex. See Willsky, Deyst, and Crawford (1974); Willsky and Jones (1974), Deyst and Deckert (1975), McAulay and Denlinger (1973), and Sanyal and Shen (1974).

3.3.4. Modified Filter Design

This method starts with a constraint of one Kalman filter. It does not allow the filter to remain a fixed element. An attempt is made to alter the filter gains to produce strong and easily recognized failure signatures. This concept is largely theoretical at this time. Details are given in Jones (1973) and Beard (1973).

3.3.5. Jump Process Estimation

This procedure is also theoretical. The basic idea is to represent failures as random jump processes in an otherwise known stochastic system. The time of occurrence and magnitude of the jumps are then estimated using optimal stochastic filtering theory. (See Sworder, 1972; Sworder and Robinson, 1973; Ratner and Luenberger, 1969; Pierce and Sworder, 1971; Davis, 1975; McGarty, 1974; and Chien, 1972.)

3.4. Redundancy Management

Redundancy Management will be optimized when included in a system design, but it can be incorporated into an existing system. Mulcare, Downing, and Smith (1987) modified an existing quadruplex DFCS to incorporate AR. Section 6.3 contains example filters which could be used to add AR to one of the functions, a Stability Augmentation System (SAS), implemented in this modification.

Issues concerned when designing and structuring the RM are:

- The number of success paths.
- The amount of dependency placed on monitoring quality.
- The tradeoff between fail safety and reliability.
- The cost effectiveness for increased reliability with added FD and isolation.
- The throughput constraints imposed by the flight computer. RM can be 60 to 80 percent of the required processing.

Performance qualities may vary for different RM structuring.

In order to maintain fail-safe sensor operations, the following three types of logic are needed.

- Comparison monitoring of dual like signals without an associated DF. Examples where this may be implemented are after a computer failure or a failure of an input to a DF.
- Monitoring the diagnostic error function after failure of one-out-of-two sensors in a dual set. If a pitch rate gyro fails, the relevant DF then monitors the remaining pitch rate gyro.

- Monitoring of a single sensor by its associated DF. This logic requires a failure declaration from both computers to avoid fault indication due to a dual sensor failure.

4. ANALYTICAL REDUNDANCY - RELIABILITY

The concept of AR, in order to be implemented in practice, must satisfy performance and reliability requirements of typical flights and must not overload the flight computer.

The traditional RM for a quadruplex set of sensors would isolate the first and second failed sensors by a majority vote. On the failure of the third sensor the set would be shut down. For a perfect monitoring process, the probability of total failure of the set is $4Q_s^3$, where Q_s is the probability of failure of one of the sensors within a certain time. The probability of any two sensors failing is $6Q_s^2$.

A rate gyro has a failure rate of about 10^{-4} per hour. This is one of the worst sensor failure rates. The probability of a quadruplex set of gyros failing would be 4×10^{-12}. This exceeds the typical flight control system requirements per sensor set of 10^{-9} by more than two orders of magnitude. For a triplex set of gyros, failure is defined as $3Q_s^2$ with a value of 3×10^{-8} which is less than the requirements by a factor of thirty.

The assumption of perfect monitoring is not valid. An additional factor, Q_m, should be included in the failure rate. Q_m defines the probability that the first failure is not detected and that both failures are alike. This issue of imperfect monitoring or coverage may cause failure of a quadruplex sensor set after only two failures, if it cannot distinguish between the two good and two failed sensors. The probability of failure here is $6Q_s^2\ Q_m$. No data exists for the term Q_m, but numbers on the order of 10^{-2} have been suggested. With imperfect monitoring, the probability of failure for a quadruplex set of rate gyros is 6×10^{-10} in one hour which is just within the requirements.

The diagnostics provided in an AR system must provide similar quality in monitoring. For triplex sensors configured for dual-fail-operational systems, the total failure rate is given in the following equation.

$$Q_s^3 + 3Q_s^2\ Q_d$$

The first term represents the probability for all three sensors failing. The second term is the combination of the probabilities of a dual sensor failure and the diagnostics failing to recognize the fault, Q_d. This is the dominant term. If it is set equal to the reliability requirement,

$$3Q_s^2\ Q_d = 10^{-9}$$

$$Q_d = 3.3 \times 10^{-2}$$

The value of Q_d indicates that the diagnostics must be 97 percent reliable. If this performance is feasible, then AR techniques are at least equal to those of quadruplex channel voting.

This example was devoted to just one set of sensors. Total system reliability includes all hardware: computers, servos, and sensors. A computer code, Computer Aided Redundant System Reliability Analysis (CARSRA), computes system reliabilities based on a specific configuration and sensor failure rates. Mulcare, Downing, and Smith (1987) used a revised version of this code to perform a system reliability assessment.

Table 4-1, utilizing parameters of an existing system (Cunningham and Hartmann, 1977), shows that servos, not sensors, tend to dominate reliability. By adding sensor redundancy, only a factor of two less than baseline system failure rate is obtained. With both sensor and servo redundancy, system failure rate decreased by more than an order of magnitude. Failure rates are dependent on system architecture. A system should be designed to have matched reliability.

TABLE 4-1. SYSTEM RELIABILITY

Probability (failures per flight hour)		
Failure Type	95% Sensor Redundancy	95% Sensor/Servo Redundancy
Either computer fails (95% effective self-test)	0.2×10^{-4}	0.2×10^{-4}
Servo failure in any axis	6.0×10^{-4}	0.3×10^{-4}
Gyro failure in any axis	0.3×10^{-4}	0.3×10^{-4}
Normal accelerometer failure	0.02×10^{-4}	0.02×10^{-4}
Total	6.52×10^{-4}	0.82×10^{-4}

5. ANALYTICAL REDUNDANCY - FAULT DETECTION AND ISOLATION

The type of FD monitor used to test various error signals is critical for performance. Important parameters are speed of response and an acceptable false alarm rate, one per 1000 hours of flight. Observers or DFs can operate with comparison monitors of dual sensors to isolate faults for a fail-operational capability. A fail-safe capability exists for all single sensors and for dual sensors after one failure.

Fault Detection monitors can be multiple trip monitors or a Sequential Likelihood Ratio Test (SLRT) of a defined parameter, error signals, or likelihood functions. The design goal of an FD monitor is FD within a minimum time after the occurrence of the fault while maintaining an acceptable false alarm rate. Another important goal is the number of missed alarms. This is addressed in the literature (Cunningham and Hartmann, 1977 and Deckert, et al., 1978) where data exists from flight simulation tests.

5.1. Multiple Trip Monitor

The FD logic in a multiple trip monitor typically requires three consecutive trips for a fault to be declared. The trip thresholds must be based on the performance of each sensor and known sensor and modeling inaccuracies.

The simplest test to check for an error signal is to compare its magnitude with predefined limits. This is illustrated in figure 5.1-1. The trip magnitude is typically placed at some multiple, m, of σ, the standard deviation of the error signal of an unfailed sensor. The mean of this error signal is zero. The value of m is chosen to yield an acceptable false alarm rate. This is shown in the following sequence of equations. The probability that there will be no false alarms during one hour of flight is given by a Poisson distribution. See Ross (1972) for more information on probability density functions, likelihood functions, and reliability.

$$P(\text{no false alarms}) = e^{-\delta t}$$

where δ = the false alarm rate (1 per 1000 flight hours)
t = the flight time (1 hour)

so that P(no false alarms) = 0.999. This is also equal to the probability that a sequence of random variables, the sampled error signals, will remain within the trip boundary of $m\sigma$.

$$P\left(\bigcap_{i=1}^{n} X_i\right) = e^{-\delta t}$$

For a one hour flight, sampling at 16 Hz, n will equal 57,600.

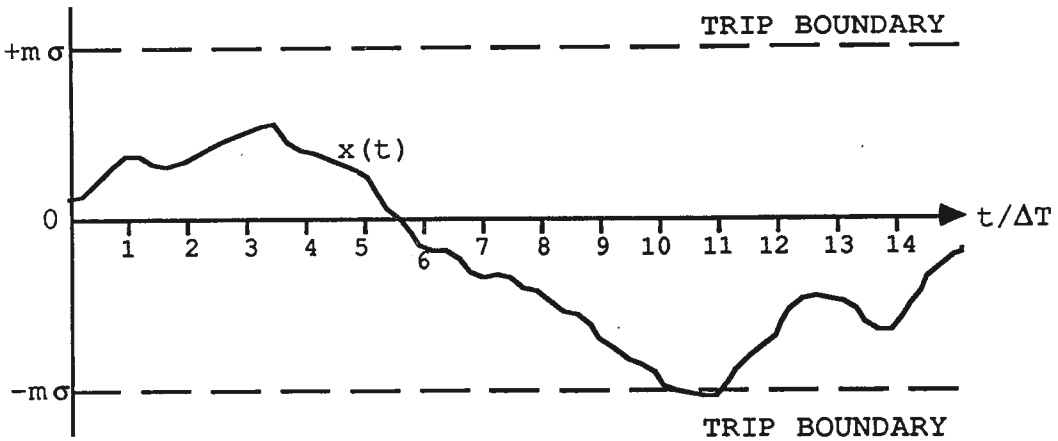


FIGURE 5.1-1. MULTIPLE TRIP FAULT DETECTION MONITOR

If the sequence of random numbers is independent, then

$$P(\bigcap_{i=1}^{n} X_i) = \prod_{i=1}^{n} P(X_i) = nP(X_i)$$

$$P(|X_i| > m\sigma) = (1 - e^{-\delta t}) / n$$

$$= 1.74 \times 10^{-8}$$

This would require m to be much larger than 4. Multiples of σ are not given in normal probability tables for numbers in this range. For three consecutive trips,

$$P(|X_i| > m\sigma; \text{ for three consecutive times}) = (1.74 \times 10^{-8})^{1/3}$$

$$= 2.6 \times 10^{-3}$$

This corresponds to $m \approx 3.2$. This multiple of σ was determined using a table containing probability values for a standard normal distribution.

For a completely dependent sequence,

$$P(|X_i| > m\sigma;\ \text{for three consecutive times}) \approx P(|X_i| > m\sigma;\ \text{one time})$$

$$\approx 1 - e^{-\delta t}$$

$$= 1.0 \times 10^{-3}$$

This corresponds to $m \approx 3.4$. This value was obtained from the same table used above.

The results of this analysis imply that the triple trip boundary needs to be set at 3.4σ to meet the false alarm rate for a given error signal.

The trip threshold can remain constant, adjusted based on the output of the sensor, or adjusted based on inputs from the pilot. An adjustable threshold can increase sensitivity to environmental disturbances, such as turbulence, and to aircraft in-flight maneuvering.

A multiple trip criterion (i.e., a fault is declared when a limit is exceeded n consecutive times) gives a much lower false alarm rate than a first trip monitor. The number of consecutive trips chosen for a monitor is a tradeoff between error insensitivity and the speed of FD. Choosing a multiple number of trips applies a built-in delay to FD.

5.2. Sequential Likelihood Ratio Tests

Likelihood ratio tests have been demonstrated to be a better FD scheme than the multiple trip monitor in Cunningham and Hartmann (1977). This algorithm provides quicker response to faults such as hardover, and more sensitivity to other smaller faults (bias and scale factor changes). However, there is more complexity to this monitoring scheme.

These tests are based on two hypothesized density functions of a random variable, $f_0(x)$ and $f_1(x)$. The likelihood ratio is expressed as a function, Γ_n, of error signal observations of that random variable.

$$\Gamma_n = \frac{f_0(e_1, e_2, \ldots, e_n)}{f_1(e_1, e_2, \ldots, e_n)}$$

The two hypothesized outcomes, H_0 and H_1, are no failure or sensor failure. These outcomes are accepted based on the following rules.

- Accept H_0 if $\Gamma_n \leq A$.
- Accept H_1 if $\Gamma_n \geq B$.
- No decision if $A < \Gamma_n < B$.

where A = 1/B. A typical value for B is 20,000.

H_0 implies that the expected value of a given error signal is 0 and there is no fault. H_1 implies the mean of the error signal has shifted unacceptably to μ_1 and a sensor fault should be declared. For actual monitoring, a combined test is used where μ_1 may equal $\pm$ mσ. Figure 5.2-1 illustrates the bounds for H_0 and H_1.

This can be defined mathematically using the following equations. The functions, f_0 and f_1, are assumed to have a normal distribution with a standard deviation of σ.

$$f_0(e_i) = c \exp[-\sum_{i=1}^{n} (e_i - \mu_0)^2 / 2\sigma^2]$$

$$f_1(e_i) = c \exp[-\sum_{i=1}^{n} (e_i - \mu_1)^2 / 2\sigma^2]$$

Using the no decision criteria, taking natural logarithms, and substituting 0 for μ_0 and $\pm m\sigma$ for μ_1,

$$\ln A < \sum_{i=1}^{n} [(e_i \mp m\sigma)^2 - (e_i)^2] / 2\sigma^2 < \ln B$$

which is equivalent to

$$nm\sigma/2 - \sigma \ln B/m < \sum_{i=1}^{n} e_i < nm\sigma/2 + \sigma \ln B/m, \text{ for } \mu_1 = +m\sigma$$

and

$$-nm\sigma/2 - \sigma \ln B/m < \sum_{i=1}^{n} e_i < \sigma \ln B/m - nm\sigma/2, \text{ for } \mu_1 = -m\sigma$$

A fault is determined when $\left| \sum_{i=1}^{n} e_i \right| \leq \sigma \ln B/m + nm\sigma/2$.

No fault is determined when $\left| \sum_{i=1}^{n} e_i \right| \geq nm\sigma/2 - \sigma \ln B/m$.

Otherwise, no decision is made.

The test sequence begins by initializing n to 1 and updates the summation each cycle. The sequence restarts when H_0 is accepted or there is no decision for a fixed period of time. Frequent restarts are desirable because summations over long intervals have less sensitivity to new data.

Likelihood functions of like sensors, i.e., with identical statistical error characteristics, can be compared and used to isolate the faulted sensor. The likelihood functions are L_{n1} and L_{n2} for sensors 1 and 2, respectively. The hypotheses here are that H_0 implies that sensor 1 has failed, and H_1 implies that sensor 2 has failed. The mean of H_0 is μ_0 which is equal to $+m\sigma$, and the mean of H_1 is μ_1 which is equal to $-m\sigma$.

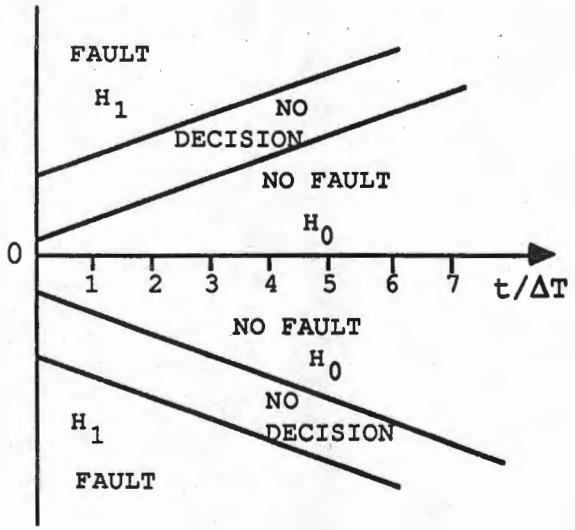


FIGURE 5.2-1. SEQUENTIAL LIKELIHOOD FAULT DETECTION MONITOR

Decisions are made using the difference of the likelihood functions based on the following rules.

$$\Delta L_n = L_{n1} - L_{n2}$$

- Accept H_0 if $\Delta L_n \geq \sigma \ln B/2m$.
- Accept H_1 if $\Delta L_n \geq \sigma \ln A/2m$.
- No decision if $\sigma \ln A/2m < \Delta L_n < \sigma \ln B/2m$.

Because the no fail hypothesis is not included here, this particular test can proceed only when a fault has already been declared. This test can be initiated after one trip using comparison monitoring. If a fault is declared (three consecutive trips), then the likelihood comparison test is used to determine which sensor is faulted. Because the likelihood functions can only increase in a fault situation, the plus or minus direction of the delta likelihood function provides the information necessary to determine which sensor has failed.

5.3. Fault Modeling

Sensor error and fault modeling require careful treatment for FD because typical sensor models are not sufficient and fault characteristics may not be

well-defined. A key issue is modeling sensor characteristics, including noise in an operational environment. Possible errors superimposed on the sensor output include bias, resolution, scale factor, null output, alignment error, hysteresis, environmental effects, response dynamics, and unmodeled sensor inputs. Errors may occur if the model does not include pilot inputs. Mode specific error signals may occur for selected modes, e.g., altitude or heading hold. A determination of the importance of each type of error is dependent on the type and percentage of occurrence of failure modes for each sensor. Modeling these errors may be a difficult task, because data is not easily obtained or does not exist on some of these parameters.

Faults can be classed as open-loop or closed-loop faults. An open-loop fault is one in which the sensor output is completely independent of its input. Examples of this category are hardover (at maximum output), zero output, output stuck at an intermediate value, and a randomly varying output. Closed-loop faults consist of the correct sensor output with errors superimposed. These faults are misalignment, scale factor changes, noise, or others listed in the previous paragraph.

Simulations should subject the FD algorithms to the highest probability faults. Sensor hardover, dead sensor, dynamic response reduction, scale factor changes, and bias shifts represent 90 to 95 percent of expected failures in typical rate gyros and accelerometers. These two sensor types have the highest failure rates compared to other sensors. Other faults should be defined dependent on the sensor.

Some techniques are limited in the type of failures they look for. Deckert, et al., 1978 limited their technique to looking for bias failures. They felt that most other failures could be found with the same technique. As yet, no one has addressed oscillatory faults which may or may not be important.

As defined by Cunningham and Hartmann (1977), the observer/blender and diagnostic Kalman filter worked well for FD for most sensors included in the study. Super-diagnostic techniques were found to have performance concerns and developmental needs. Problems arose in detection of accelerometer soft failure, dynamic response faults, and FI based on more than one filter detecting a given fault. (The difficulty arose when both filters did not detect the failure at the same time.)

A comparison of monitors shows that the SLRT of residual mean values performed well relative to the multiple trip monitor. SLRT caught hardover failures sooner. Frequently scale factor changes would escape the multiple trip monitor, but not the SLRT.

One way to enhance a particular FD algorithm is to utilize the fact that there are physical limitations as to how much the aircraft state can change from one sample to the next. This can minimize the effects of extremely large sensor failures. Desai, Deckert, and Deyst (1979) added a self-test in conjunction with an FD algorithm which provisionally failed a sensor when an abrupt change in output was maintained for three sample intervals.

6. SUPPLEMENTAL WORKED EXAMPLES

6.1. Altitude Kinematics

Altitude Kinematics refers to the redundancy between changes in altitude measured by the barostatic pressure sensors and changes in altitude computed from the vertical acceleration measured by the accelerometers. The vertical direction is determined via Euler attitude angles: roll (ϕ) and pitch (θ). Since the accelerometers sense gravity, the gravitational component must be subtracted in computing vertical acceleration. Vertical acceleration is computed as:

$$A_v = A_x \sin\theta - (A_y \sin\phi + A_z \cos\phi) \cos\theta - g$$

where A_x, A_y, A_z are measured accelerations compensated to the center of mass.

A third order complementary filter (illustrated in figure 6.1-1) is used to store $\dot{h}$ (vertical velocity) prior to a failure detection and possible corruption. The value of $\dot{h}$ can then be used to compare altitude deviation from slightly prior to the failure detection by keeping a moving window of data.

The altitude residues for both sensors would be:

$$\text{Residue \#1} = \Sigma(\text{current } h_1 - \text{previous } h_1 - \dot{h}\, dt)$$

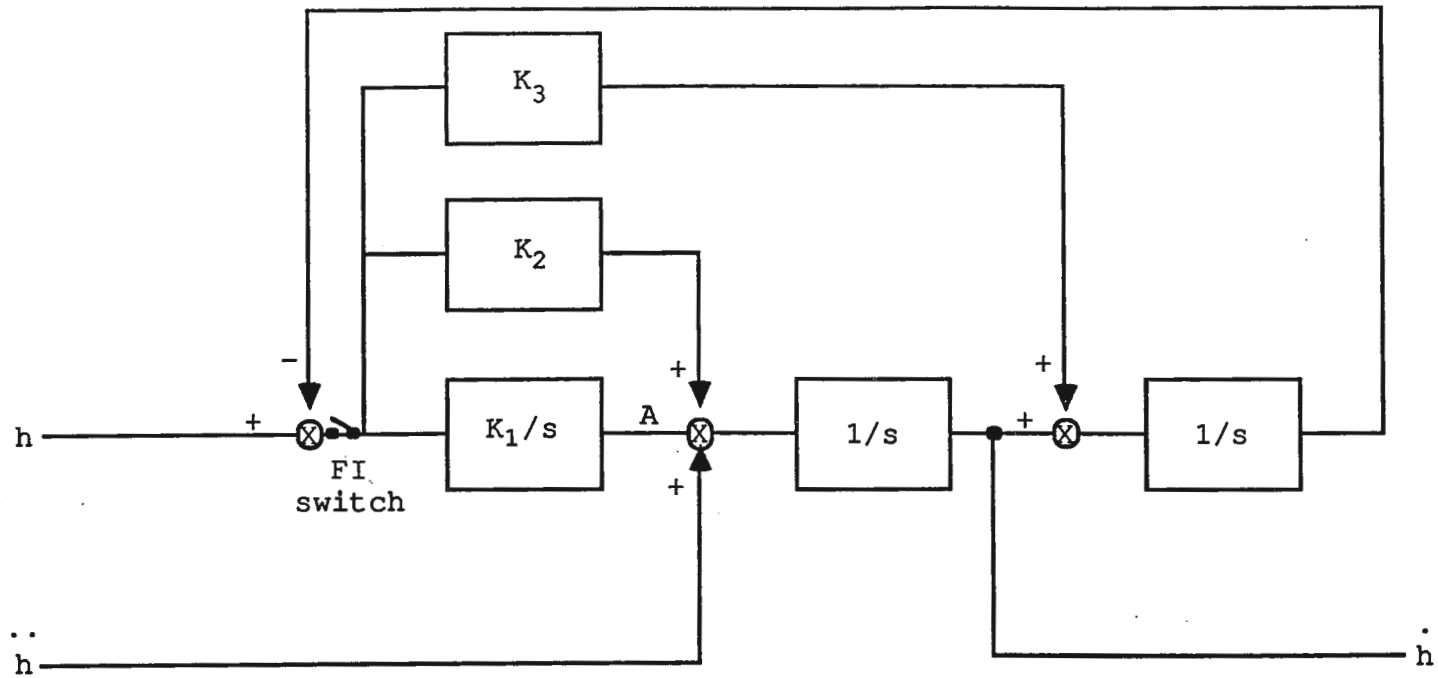
$$\text{Residue \#2} = \Sigma(\text{current } h_2 - \text{previous } h_2 - \dot{h}\, dt)$$

using h measurements from sensors 1 and 2, respectively, where dt is the integration interval.

Both residues are then evaluated by an FI algorithm. The algorithm may be a multiple trip threshold exceedance or a statistical hypothesis algorithm (e.g., SLRT).

During the FI period, the $\dot{h}$ used is only inertially updated via the switch shown in figure 6.1-1, i.e., $\dot{h} = \int \ddot{h}$. The switch opens (FI begins) when the absolute magnitude of $(h_1 - h_2)$ exceeds a threshold N times. The filter is not updated if any sample exceeds the threshold to avoid $\dot{h}$ corruption. Figure 6.1-2 demonstrates the RM processing.

Error sources must be evaluated to avoid a false FI decision. The third order filter has a state to estimate steady-state errors on $\ddot{h}$ to avoid a parabolic error growth when calculating both residues. An error analysis must be made to determine the maximum time that can be allowed for FI.



h is the average of two baro-static pressure altitudes

$\ddot{h}$ is an estimate of error of $\ddot{h}$, subsequently added to each $\ddot{h}$ calculation in the fault isolation residue equations to effectively cancel the error.

FIGURE 6.1-1 THIRD ORDER VERTICAL COMPLEMENTARY FILTER FOR ALTITUDE KINEMATICS

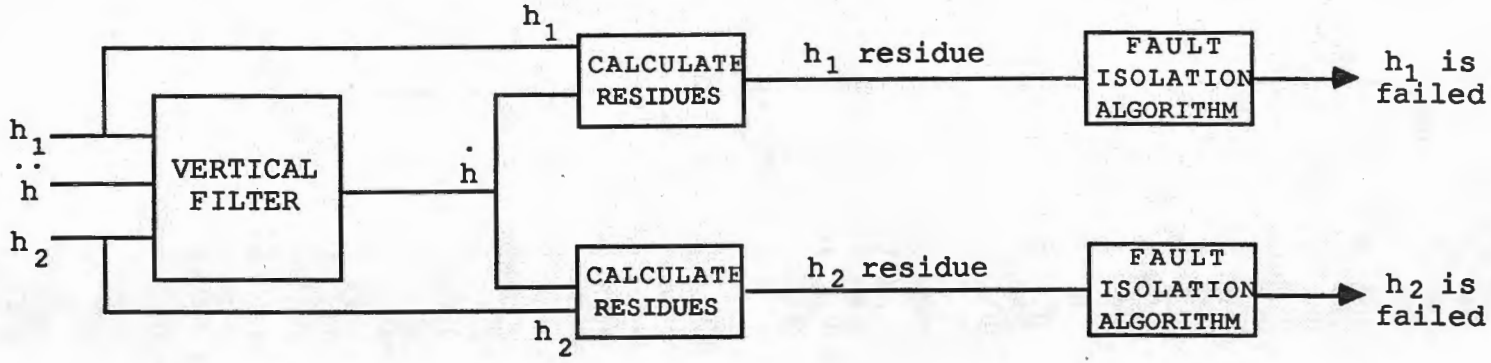


FIGURE 6.1-2. POSSIBLE REDUNDANCY MANAGEMENT FOR ALTITUDE KINEMATICS FAULT ISOLATION

If the time expires without an FI decision, then the following logic can be used:

Is there still a fault?

The answer is yes, if the absolute magnitude of $(h_1 - h_2)$ is greater than the threshold. Then isolate the sensor with the worst error history, i.e., the largest residue magnitude.

Otherwise, the answer is no. Reset the RM logic and restart the vertical filter, closing the FI switch and clearing the FD flag, if one is present.

6.2. Translational Kinematics

A translational kinematic relationship can be used to determine an airspeed sensor fault by means of the related inertial acceleration along the x-axis, such that

$$V_x = \int \dot{U}\, dt \qquad \text{or} \qquad \Delta V_x = \dot{U}\, dt$$

$$V_x = V_t \cos\beta \cos\alpha \approx V_t \qquad \text{for small angles of } \alpha \text{ and } \beta$$

where V_x = x-axis airspeed velocity

V_t = true airspeed

α = angle of attack

β = angle of sideslip

U = inertial velocity along the x-axis

An estimate of $\dot{V}_x$ can be obtained using the following equation.

$$\dot{V}_x = A_x - g \sin\theta - QW + RV + \delta_x$$

where Q = pitch rate

W = z-axis velocity

R = yaw rate

V = y-axis velocity

δ_x = wind acceleration along the x-axis

A_x = acceleration measured by axial accelerometer

Q and R are measured gyro values. V and W can be calculated as follows.

$V = V_t \sin\beta$ = y-axis velocity

$W = V_t \cos\beta \sin\alpha$ = z-axis velocity

The only term unaccounted for is δ_x, wind acceleration. It must be carefully analyzed as an error source in the FI decision. As long as the FI period is not too long relative to this error source, the decision should not be wrong.

Another way to alleviate this error source is to use a moving window to calculate the standard deviation of the airspeed residues (prior to FD) to determine the threshold size for FI. Wind turbulence associated with wind acceleration increases the threshold, thereby protecting from a false FI.

A filter is used to get a good estimate of any steady-state error on $\dot{V}_x$ as shown in figure 6.2-1. The term a is an estimate of error on $\dot{V}_x$, subsequently added to each V_x calculation in the FI residue equations to effectively cancel the error.

V_{tavg} is the average of the two true airspeed sensors. When an FD occurs the filter is no longer updated. FD is accomplished by the means discussed in section 6.1.

The airspeed residues are calculated as:

$$V \text{ Residue \#1} = \Sigma(\text{current } V_{t1} \text{ measured} - \text{previous } V_{t1} \text{ measured} - \dot{V}_x \, dt)$$

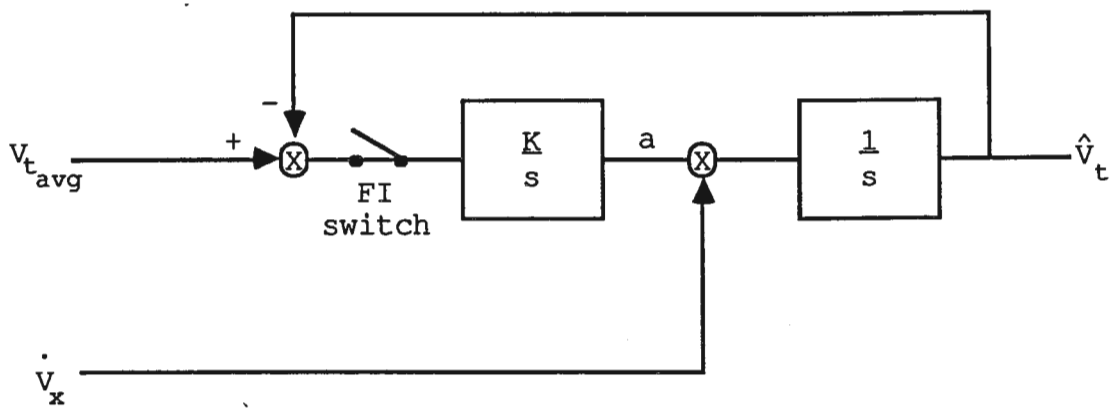
where dt is the update rate. This equation is duplicated for each airspeed sensor measurement. The two residues are then evaluated by an FI algorithm as discussed in section 6.1.

Similar comments concerning error analysis and adaptive threshold solutions apply for this example also. The standard deviation of the average of the sensors could be calculated with a moving window prior to an FD.

6.3. Stability Augmentation System

A large number of advanced aircraft are not statically stable. This is a concern because the airframe is no longer inherently stable. As an aircraft's center of gravity moves aft, the wing has a decreasing capacity to generate a nose down moment in response to an increase in angle of attack. At the neutral point, there is no pitch-axis response due to a change in angle of attack. When the center of gravity is farther aft than the neutral point, a pitch-up moment is generated from an increase in angle of attack which may make the aircraft very difficult to fly.

A Relaxed Static Stability (RSS) aircraft requires an SAS to restore stability to reasonable levels. If the SAS function is critical, it makes sense to implement AR rather than to increase the number of sensors. This function is dependent on feedback from inertial sensors, pitch angle, pitch rate, and vertical acceleration, or air data sensors, true airspeed, and angle of attack, to prevent a rapid pitch-axis divergence.



a is an estimate of error on $\dot{V}_x$, subsequently added to each $\dot{V}_x$ calculation in the fault isolation residue equations to effectively cancel the error.

FIGURE 6.2-1. EXAMPLE FILTER USED FOR TRANSLATIONAL KINEMATICS REDUNDANCY

An estimate of pitch rate and vertical acceleration can be determined with the following equations.

$$Q_E = Q_m - \dot{\theta}\cos\phi_m + \dot{\tau}\sin\phi_m\cos\theta_m$$

$$A_{zE} = A_{zm} - U_m(Q_m - \dot{\alpha}_m) - g\cos\theta_m\cos\phi_m \text{ assuming that } V = 0$$

where Q = pitch rate

A_z = vertical acceleration

U = x-axis velocity

g = acceleration due to gravity

ϕ = roll angle

θ = pitch angle

τ = yaw angle

α = angle of attack $\approx W/U$ where W is the vertical velocity.

The subscript m denotes measured quantities and the subscript E denotes an estimated value.

Figure 6.3-1 shows one way to implement filters for these two estimated values. FD and isolation is as described in section 6.1.

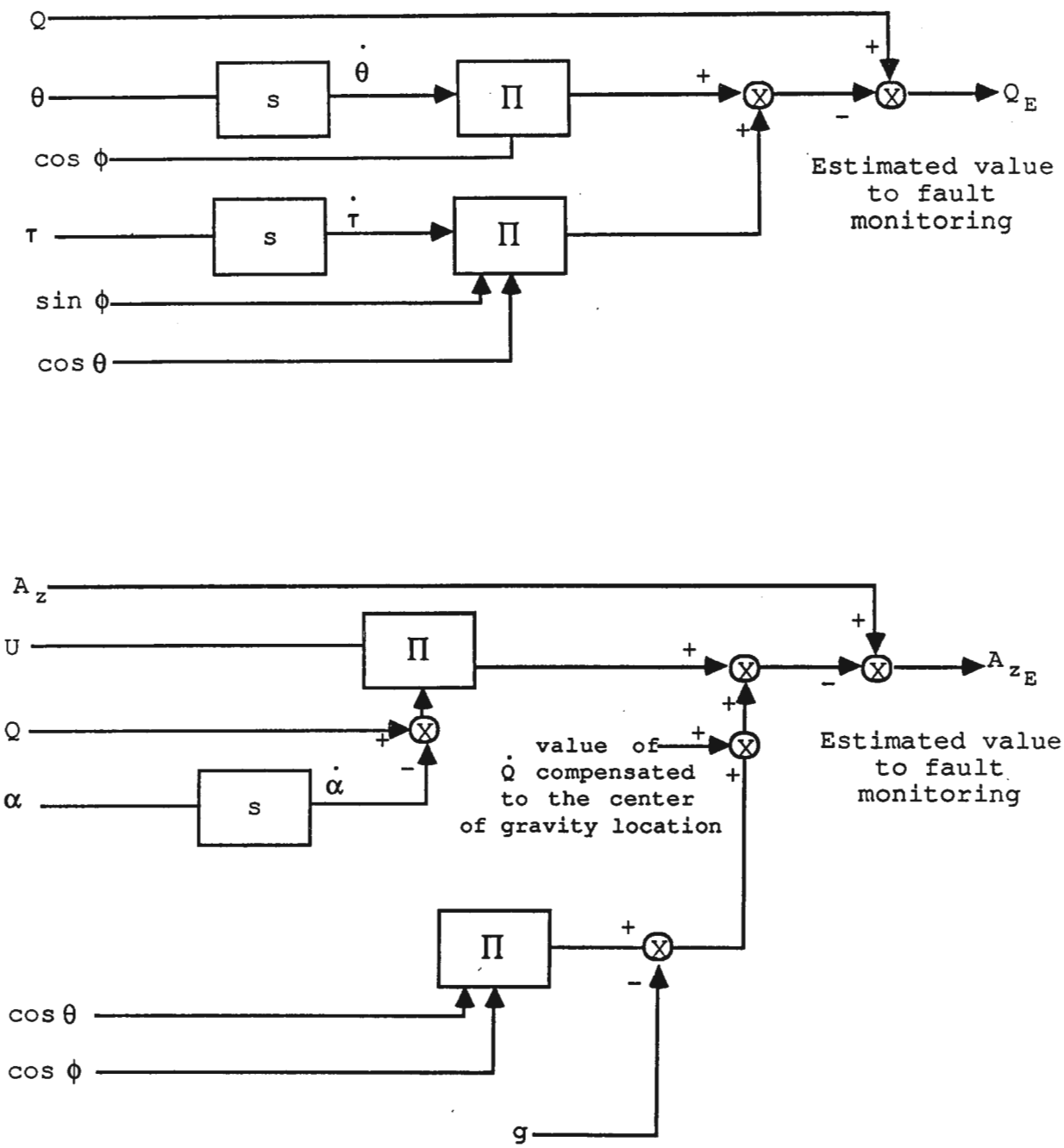


FIGURE 6.3-1. EXAMPLE FILTERS USED FOR STABILITY AUGMENTATION SYSTEM

BIBLIOGRAPHY

Athans, M. and D. Willner, "A Practical Scheme for Adaptive Aircraft Flight Control Systems," Symposium on Parameter Estimation Techniques and Applications in Aircraft Flight Testing, NASA Flight Research Center, Edwards Air Force Base, April 24-25, 1973.

Beard, R. V., Failure Accommodation in Linear Systems Through Self-Reorganization, Report MVT-71-1, Man-Vehicle Laboratory, Cambridge, MA, February 1973.

Buxbaum, P. J. and R. A. Haddad, "Recursive Optimal Estimation for a Class of Nongaussian Processes," Proceedings of Symposium on Computer Processing in Communications, Polytech Institute of Brooklyn, April 8-10, 1969.

Chien, T. T., An Adaptive Technique for a Redundant-Sensor Navigation System, Report T-560, Draper Labs, Cambridge, MA, February 1972.

Clark, R. N., et al., "Detecting Instrument Malfunctions in Control Systems," IEEE Transactions on Aerospace and Electronic Systems, Vol. AES-11, No. 4, July 1975.

Cunningham, T., and G. Hartmann, Fault Tolerant Digital Flight Control with Analytical Redundancy, Technical Report AFFDL-TR-77-25, May 1977.

Davis, M. H. A., "The Application of Nonlinear Filtering to Fault Detection in Linear Systems," IEEE Transactions on Automatic Control, Vol. AC-20, No. 2, April 1975.

Deckert, J. C., et al., Reliable Dual- Redundant Sensor Failure Detection and Identification for the NASA F-8 DFBW Aircraft, NASA Contractor Report 2944, February 1978.

Desai, M. N., J. C. Deckert, and J.J. Deyst, "Dual Sensor Failure Identification Using Analytical Redundancy," Journal of Guidance and Control, Vol. 2, No. 3, May-June 1979.

Deyst, J. J. and J. C. Deckert, "RCS Jet Failure Identification for the Space Shuttle," Proceedings of IFAC 1975, Cambridge, MA, August 1975.

Hartmann, G. L., et al., Digital Adaptive F-8C Control Laws, Final Report NAS 1-13358, July 1975.

Jones, H. L., Failure Detection in Linear Systems, PhD Thesis, Department of Aeronautics and Astronautics, MIT, Cambridge, MA, September 1973.

Kaniuka, W. W., et al., New Flight Control Technologies for Future Naval Aircraft, Report No. NADC-82240-60, September 1982.

Kerr, T. H., "A Two Ellipsoid Overlap Test for Real-Time Failure Detection and Isolation by Confidence Regions," IEEE Conference on Decision and Control, Phoenix, AZ, November 1974.

Lainiotis, D. G., "Joint Detection, Estimation, and System Identification," Information and Control, Vol. 19, No. 1, August 1971.

Maybeck, P. S., "Failure Detection Through Functional Redundancy," AFFDL-TR-74-3, January 1974.

McAulay, R. J. and E. Denlinger, "A Decision-Directed Adaptive Tracker," IEEE Transactions on Aerospace and Electronic Systems, Vol. AES-9, March 1973.

McGarty, T. P., "State Estimation with Faulty Measurements: An Application of Bayesian Outlier Rejection," Proceedings of the Fifth Symposium on Nonlinear Estimation and Its Applications, San Diego, CA, September 1974.

Mehra, R. K. and J. Peschon, "An Innovations Approach to Fault Detection and Diagnosis in Dynamic Systems," Automatica, Vol. 7, 1971.

Meier, L., D. W. Ross, and M. B. Glaser, "Evaluation of the Feasibility of Using Internal Redundancy to Detect and Isolate On-Board Control Data Instrumentation Failures," AFFDL-TR-70-172, January 1971.

Montgomery, R. C. and A. K. Caglayan, "A Self-Reorganizing Digital Flight Control System for Aircraft," AIAA 12th Aerospace Sciences Meeting, Washington, D.C., January 30-February 1, 1974.

Montgomery, R. C. and D. B. Price, "Management of Analytical Redundancy in Digital Flight Control Systems for Aircraft," AIAA Mechanics and Control of Flight Conference, Anaheim, CA, August 5-9, 1974.

Mulcare, D. B., L. E. Downing, and M. K. Smith, Analytical Sensor Redundancy Assessment, DOT/FAA/CT-86/32, Draft Report, June 1987.

Pierce, B. D. and D. D. Sworder, "Bayes and Minimax Controllers for a Linear System with Stochastic Jump Parameters," IEEE Transactions on Automatic Control, Vol. AC-16, No. 4, August 1971.

Ratner, R. S. and D. G. Luenberger, "Performance-Adaptive Renewal Policies for Linear Systems," IEEE Transactions on Automatic Control, Vol. AC-14, No. 4, August 1969.

Ross, S. M., "Introduction to Probability Models," Academic Press, 1972.

Sanyal, P. and C. N. Shen, "Bayes' Decision Rule for Rapid Detection and Adaptive Estimation Scheme with Space Applications," IEEE Transactions on Automatic Control, Vol. AC-19, June 1974.

Stein, G. and G. L. Hartmann, F-8C Adaptive Control Extensions, NASA Contract NAS 1-13383, June 1976.

Sworder, D. D. and V. G. Robinson, "Feedback Regulators for Jump Parameter Systems with State and Control Dependent Transition Rates," IEEE Transactions on Automatic Control, Vol. AC-18, No. 4, August 1973.

Sworder, D. D., "Bayes' Controllers with Memory for a Linear System with Jump Parameters," IEEE Transactions on Automatic Control, Vol. AC-17, February 1972.

Weinstein, W., Feasibility and Design Studies of an Integrated Sensory Subsystem (ISS) for Advanced V/STOL Aircraft, Report No. NADC-76259-30, March 1978.

Willsky, A. S., J. J. Deyst, and B. S. Crawford, "Adaptive Filtering and Self-Test Methods for Failure Detection and Compensation," Proceedings of the 1974 JACC, Austin, Texas, June 19-21, 1974.

Willsky, A. S. and H. L. Jones, "A Generalized Likelihood Ratio Approach to State Estimation in Linear Systems Subject to Abrupt Changes," Proceedings of the 1974 Conference on Decision and Control, Phoenix, AZ, November 1974.

GLOSSARY

ANALYTICAL REDUNDANCY. The use of software algorithms which use known mathematical relationships between different sensors for sensor failure detection and replace most of additional redundant sensor hardware.

CONTROL LAW. The physical relationship between various sensors and control surfaces.

COVERAGE. The percent confidence level of a given analytical redundancy fault detection and isolation algorithm for all types of faults.

DIAGNOSTIC FILTER. An analytical algorithm which processes data from N functionally related sensors. The data are used to estimate some sensor outputs and assess the correct functioning of the sensors.

DUAL FAIL-OPERATIONAL. A reliability requirement placed on a system which requires the system to be operational after two failures have occurred.

FAIL-OPERATIONAL. A reliability requirement placed on a system which requires the system to be operational after a single failure has occurred.

FAIL-SAFE. A reliability requirement placed on a system which requires that safe flight not be hindered even after a failure.

FALSE ALARM. The declaration of a fault by a fault detection monitor or algorithm when there is no fault.

FAULT DETECTION. The determination that a sensor is faulted by using a software algorithm.

FAULT ISOLATION. The determination that a particular sensor is faulted by using a software algorithm.

FAULT TOLERANCE. Accommodation of sensor hardware faults based on some type of comparator scheme.

FLIGHT-CRITICAL. A description of functions whose failure would contribute to or cause a failure condition preventing the continued safe flight and landing of the aircraft.

MISSED ALARM. The failure of a fault detection monitor or algorithm to detect a fault when there is a sensor fault.

MULTIPLE TRIP MONITOR. A fault detection algorithm which declares a fault after the sensor output has exceeded a predefined threshold N times.

OBSERVER. An algorithm which models physical relationships between sensor data and uses the data to provide fault detection for one or more sensors. This is also known as a Luenberger observer or a signal blender.

REDUNDANCY MANAGEMENT. The computer processing which is needed to implement fault detection and isolation algorithms.

REVERSION MODE. The high level of redundancy in a system having different redundancies requirements for some sensors. Critical sensors may have a high level of redundancy while other sensors have low levels.

SENSOR. An instrument which measures a particular physical parameter. The data output may be digital or analog and is utilized by the flight computer.

SEQUENTIAL LIKELIHOOD RATIO TEST. A fault detection algorithm which is based on two hypothesized density functions of no fault or sensor fault.

SEQUENTIAL PROBABILITY RATIO TEST. See sequential likelihood ratio test.

SUPER-DIAGNOSTIC FILTER. An algorithm which provides all the capabilities of a diagnostic filter. Additionally, it can isolate a specific faulted sensor. At the current time, this is the most complex technique used to implement analytical redundancy.

ACRONYMS

AFFDL	Air Force Flight Dynamics Laboratory
AK	Altitude Kinematics
AR	Analytical Redundancy
C	Comparator
CARSRA	Computer Aided Redundant System Reliability Analysis
DF	Diagnostic Filter
DFC	Digital Flight Control
DFCS	Digital Flight Control System
DOT	Department of Transportation
FAA	Federal Aviation Administration
FBW	Fly-By-Wire
FD	Fault Detection
FI	Fault Isolation
NADC	Naval Air Development Center
NASA	National Aeronautics and Space Administration
RM	Redundancy Management
RSS	Relaxed Static Stability
SAS	Stability Augmentation System
SDF	Super-Diagnostic Filter
SLRT	Sequential Likelihood Ratio Test
SPRT	Sequential Probability Ratio Test

U.S. Department of Transportation
Federal Aviation Administration

DOT/FAA/CT-88/10

HANDBOOK-VOLUME II DIGITAL SYSTEMS VALIDATION

CHAPTER 8
ESTIMATION AND MODELING FOR REAL-TIME SOFTWARE RELIABILITY MODELS



PREPARED BY:

COMPUTER RESOURCE MANAGEMENT, INC.
950 HERNDON PARKWAY, SUITE 360
HERNDON, VIRGINIA 22070

PREPARED FOR:

FEDERAL AVIATION ADMINISTRATION
TECHNICAL CENTER
ATLANTIC CITY INTERNATIONAL AIRPORT, NEW JERSEY 08405

NOTICE

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The United States Government assumes no liability for the contents or use thereof.

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the objective of this report.

TABLE OF CONTENTS

Section			Page
1.	INTRODUCTION		8-1
	1.1. What is Software Reliability		8-1
	1.2. The Scope of this Tutorial		8-1
2.	TOPIC HISTORY		8-3
	2.1. Introduction		8-3
	2.2. Achieving System Reliability Estimates		8-5
3.	THE SOFTWARE DEVELOPMENT PROCESS		8-9
	3.1. Relationship Between the Software Development Process and Software Reliability Estimation		8-9
	3.2. Software Testing		8-11
4.	FAULT-INTOLERANT RELIABILITY MODELS		8-13
	4.1. Introduction		8-13
	4.1.1. Step 1:	Evaluating the Failure Data Acquired From Testing	8-13
	4.1.2. Step 2:	Curve Fitting	8-16
	4.1.3. Step 3:	Model Selection	8-19
	4.1.4. Step 4:	Statistical Computation to Obtain Parameter Estimates (from Goel, 1983)	8-19
	4.1.5. Step 5:	Computation of Performance Statistics	8-25
	4.1.6. Step 6:	Application of Predicted Fault Rates	8-25
	4.2. Computation of a Time-Between-Failures Model		8-27
	4.2.1. Step 3:	Model Selection	8-27
	4.2.2. Step 4:	Statistical Computation to Obtain Parameter Estimates	8-27
	4.2.3. Step 5:	Computation of Performance Statistics	8-30
5.	FAULT-TOLERANT SOFTWARE		8-33
	5.1. Introduction		8-33
	5.2. Principles of Fault Tolerance		8-33
	5.3. Fault-tolerant Software Reliability Models		8-34
	5.4. A Hierarchical Software Reliability Model		8-35
	5.4.1. Software Components		8-35
	5.4.2. Inputs to the Software Reliability Model		8-38

TABLE OF CONTENTS

Section			Page
	5.4.3.	Outputs from the Software Reliability Model	8-38
	5.4.4.	Model Representation	8-38
	5.4.5.	The Modeling Process	8-38
	5.4.6.	Detailed Function Blocks	8-40
	5.4.7.	Function Block States	8-42
	5.4.8.	System Reliability	8-44
	5.4.9.	General Equations	8-44
	5.4.10.	N-version Software Calculations	8-44
	5.4.11.	Recovery Block Calculations	8-45
	5.4.12.	Examples of Potential Models: Simple, High-Level Model Example	8-45
	5.4.13.	Simple, Detailed Level Model Examples	8-47
	5.4.14.	Complex, High-Level Model Example	8-49
	5.4.15.	Detailed Worked Examples	8-51
	5.4.16.	Accuracy Constraints	8-52
	5.4.17.	Accuracy	8-53
	5.4.18.	Accuracy of the Hybrid N-version Software	8-53
	5.4.19.	Accuracy of the Recovery Block	8-53
	5.4.20.	Safety	8-53
	5.4.21.	Availability	8-54
	5.4.22.	Response to Undesired Events	8-55
	5.4.23.	Assumptions	8-56
5.5.	Scott et al.'s Model		8-57
	5.5.1.	Introduction	8-57
	5.5.2.	Recovery Block	8-57
	5.5.3.	The Recovery Block Reliability Model	8-58
	5.5.4.	N-version Programming	8-59
	5.5.5.	The N-version Programming Reliability Model	8-60
	5.5.6.	Consensus Recovery Block	8-60
	5.5.7.	The Consensus Recovery Block Reliability Model	8-61
	5.5.8.	Verification	8-62

APPENDICES

A -	ASSUMPTIONS UNIQUE TO FAULT COUNT SOFTWARE RELIABILITY MODELS	8-63
B -	ASSUMPTIONS UNIQUE TO TIME-BETWEEN-FAILURE SOFTWARE RELIABILITY MODELS	8-67
C -	WORKED EXAMPLE FOR THE GOEL-OKUMOTO ON-HOMOGENEOUS POISSON PROCESS MODEL	8-69
D -	WORKED EXAMPLE SHOWING COMPUTATIONS FOR THE JELINSKI-MORANDA DE-EUTROPHICATION MODEL	8-79
E -	WORKED EXAMPLES FOR THE N-VERSION SOFTWARE RELIABILITY DESIGN TECHNIQUES	8-87

TABLE OF CONTENTS

Section	Page
F - WORKED EXAMPLES FOR THE RECOVERY BLOCK SOFTWARE RELIABILITY DESIGN TECHNIQUES	8-99
BIBLIOGRAPHY	8-113
GLOSSARY	8-115
ACRONYMS AND ABBREVIATIONS	8-116

LIST OF ILLUSTRATIONS

Figure		Page
2.1-1	HARDWARE AND SOFTWARE COST TRENDS	8-4
3.1-1	PHASES OF SOFTWARE DEVELOPMENT	8-10
4.1-1	FLOW FOR COMPLETING THE SOFTWARE RELIABILITY ESTIMATION PROCESS	8-14
4.1-2	OBSERVED FAILURES PER HOUR	8-16
4.1-3	ACTUAL VERSUS PREDICTED FAILURES PER HOUR	8-18
4.1-4	PREDICTED VERSUS ACTUAL NUMBER OF REMAINING FAULTS	8-26
5.4-1	SIMPLE BLOCK DIAGRAM EXAMPLE	8-41
5.4-2	OVERALL REPRESENTATION OF THE FUNCTIONAL CONSTRUCTS	8-41
5.4-3	SIMPLE BLOCK DIAGRAM EXAMPLE	8-46
5.4-4	DETAILED DIAGRAM FOR THE SINGLE VERSION SOFTWARE FUNCTION BLOCK OR THE DECISION ALGORITHM FUNCTION BLOCK	8-47
5.4-5	EQUIVALENT DIAGRAM INDICATING THE FUNCTIONAL STRUCTURE TO BE USED IN THE DETAILED DIAGRAM FOR A SINGLE VERSION SOFTWARE FUNCTION BLOCK	8-48
5.4-6	COMPLEX, HIGH LEVEL MODEL EXAMPLE	8-50
E-1	GENERAL FORMAT FOR N-VERSION SOFTWARE	8-87
E-2	N-VERSION SOFTWARE WITH ACCEPTANCE TESTS	8-91
E-3	N-VERSION SOFTWARE IN WHICH THE OUTPUTS ARE SUBJECTED TO AN ACCEPTANCE TEST IF THE DECISION ALGORITHM FAILS	8-94
F-1	GENERAL FORMAT OF A BACKWARD RECOVERY BLOCK	8-99
F-2	GENERAL FORMAT OF A FORWARD RECOVERY BLOCK	8-102
F-3	ALTERNATE FORMAT FOR A FORWARD RECOVERY BLOCK	8-108

LIST OF TABLES

Table		Page
2.2-1	COMPARISON OF HARDWARE AND SOFTWARE PROPERTIES	8-6
4.1-1	FAILURES IN ONE HOUR (EXECUTION TIME) INTERVALS AND CUMULATIVE FAILURES	8-15
4.1-2	FAULT-INTOLERANT SOFTWARE RELIABILITY MODEL ASSUMPTIONS	8-17
4.1-3	FAULT-INTOLERANT SOFTWARE RELIABILITY MODELS	8-19
4.1-4	ASSUMPTIONS FOR THE GOEL-OKUMOTO NON-HOMOGENEOUS POISSON PROCESS MODEL	8-20
4.1-5	PERFORMANCE MEASURES FOR TIME-BETWEEN-FAILURE MODELS	8-21
4.1-6	PERFORMANCE MEASURES FOR FAULT-COUNT MODELS	8-22
4.2-1	ASSUMPTIONS FOR THE JELINSKI-MORANDA DE-EUTROPHICATION MODEL	8-28
5.4-1	RELIABILITY MODEL MENU SELECTIONS	8-39
5.4-2	CORRELATION OF FAULTS WITH THE DETAILED PORTIONS OF THE SINGLE VERSION SOFTWARE FUNCTION BLOCK	8-42
5.4-3	CORRELATION OF FAULTS WITH THE DETAILED PORTIONS OF THE DECISION ALGORITHM FUNCTION BLOCK	8-43
5.4-4	TRUTH TABLE OF FUNCTION BLOCK STATES	8-43
E-1	RELIABILITY VALUES FOR THE SOFTWARE COMPONENTS IN FIGURE	8-88
E-2	RELIABILITY VALUES FOR THE SOFTWARE COMPONENTS IN FIGURE E-2	8-91
E-3	RELIABILITY VALUES FOR THE SOFTWARE COMPONENTS IN FIGURE E-3	8-94
F-1	RELIABILITY VALUES FOR THE SOFTWARE COMPONENTS IN FIGURE F-1	8-100
F-2	RELIABILITY VALUES FOR THE SOFTWARE COMPONENTS IN FIGURE F-2	8-102
F-3	ACCURACY EFFECTS WHEN FEWER THAN n ALTERNATES ARE USED	8-107
F-4	RELIABILITY VALUES FOR THE SOFTWARE COMPONENTS IN FIGURE F-3	8-108
F-5	COMPARISON OF RELIABILITY VALUES WHEN FEWER THAN n ALTERNATES ARE USED	8-111

1. INTRODUCTION

1.1. What is Software Reliability

Industry concern for software reliability is growing as increasingly complicated computer software is being used to perform critical tasks. Software reliability is "the probability that a given software program operates for some time period, without a software error, on the machine for which it was designed given that it is used within design limits." (Dickson et al., cited by Gephart et al., 1978, p. 2). A number of methods have been proposed for improving software quality and predicting its reliability. Until recently, software developers assumed that the testing strategy selected would be sufficient to detect all faults or failures in the software prior to its implementation. However, rapid increases in software complexity have made it impossible to thoroughly test the software within the constraints of reasonable time and budget.

There are two possible approaches to the problem of software reliability. The first approach is to use reliability models that predict the number of software faults expected and the likelihood that these faults will cause a system failure. Such predictions are based upon data acquired through initial testing of the software. This information can then be used to guide the testing process and to provide a reliability confidence level. The objective of this approach is to find and remove all faults in the software. Fault-intolerant (or single-version software) is software to which these models can be applied.

A second approach assumes that software faults are likely to occur. Finding each fault, however, may not be possible. Rather than attempting to remove all faults, this approach tries to develop safeguards within the software that will support recovery from software failures or at least minimize the consequences of such failures. Reliability models have been developed in response to such fault-tolerant software. These models are used to predict the reliability of this type of software and to determine the types and number of safeguards that need to be included in a given type of software. Fault-tolerant reliability models are used even before the software has been coded. Consequently, results from the application of these models can guide the subsequent design and development of the actual software. This tutorial looks at both fault-tolerant and fault-intolerant reliability models that are appropriate for use with complex real-time software such as that used in avionics and flight control systems.

1.2. The Scope of this Tutorial

This tutorial provides aircraft certification specialists with the background needed to understand and evaluate documents that describe measures taken to assess software reliability submitted as part of the certification process for advanced digital avionics and flight control systems. It should be noted, however, that the development of software reliability methodologies is still in

its infancy. As a result, there is little agreement as to how and when these models should be applied, and which models are most appropriate for specific use situations. The reader who seeks hard and fast rules will be disappointed to discover that such rules simply do not exist. Consequently, this tutorial can only present the concept of reliability models, define their processes and goals, and describe those models which are considered acceptable and likely to be applied.

The unquestionable need for such models necessitates their use even though questions of theory and application remain. Specific guidelines and recommendations which can be directly applied to evaluating certification documents remains a future objective.

This tutorial is divided into five major sections:

- "Introduction" and "Topic History" provide overviews of software reliability: Why it is used and what it tries to accomplish. Two broad categories of software reliability models are also introduced.

- "The Software Development Process" briefly describes the stages involved in software development, the opportunities for faults to be introduced into the software, and limitations in current testing approaches intended to remove faults. This section is intended to serve as a framework for understanding when reliability models are used and how they compare with some existing approaches to supporting high software quality.

- "Fault-intolerant Software Reliability Models" looks at reliability models which are used with real-time, single-version or fault-intolerant software.

- "Fault-tolerant Reliability Models" addresses reliability models designed specifically for use with fault-tolerant software.

- The appendices contain worked examples demonstrating how the most important models (representing both the fault-tolerant and fault-intolerant categories) are used. Following the appendices are supplemental materials, including references, which can serve as a basis for further study in this area, and a glossary of terms.

2. TOPIC HISTORY

2.1. Introduction

Traditionally, improvements in aircraft design have been aimed at more efficient aerodynamic designs and propulsion systems (Larsen, et al., 1984). Recently, however, the greatest advancements have been in the development of digital electronic systems and devices for flight control and avionics systems. Examples of such systems include stability and control augmentation systems, active control systems, advanced displays, aircraft/crew interface, and aircraft handling and gust-load alleviation systems. Such systems rely on extensive software to monitor their functioning and maintain appropriate system conditions. With this growing emphasis on critical software functioning comes the requirement for methods to evaluate the reliability of this software.

Although reliable software is important for all applications, it is imperative for flight-critical software. Since in-use software testing of flight-critical systems is not a safe option, other methods must be implemented to demonstrate that the software can be relied upon to perform its intended function.

Hardware reliability assessment has been in use for more than 20 years. Hardware reliability models and procedures have proved useful and have achieved acceptance as validation tools. Although software reliability assessment has been around almost as long - 15 years - acceptance within the industry has been slow. Acceptance and implementation are expected to improve for many reasons, including:

- The recognition of a growing reliance on software to perform critical tasks. (A trend that is likely to continue.)
- The increasing use of software to perform tasks traditionally performed by hardware.
- The increased software development and operational costs compared to hardware costs.

Computer hardware costs are closely linked to technological advances in integrated circuit technology. Hardware advances pose a challenge to the software designer to develop software that will effectively utilize complex, innovative hardware. The complexity of such software has outgrown the technology for designing and testing it. Workable reliability assessment methods are therefore critical.

As figure 2.1-1 (Shooman, 1984) shows, hardware manufacturing costs have been decreasing over the past 30 years as hardware technology advances. Software costs, however, continue to increase.

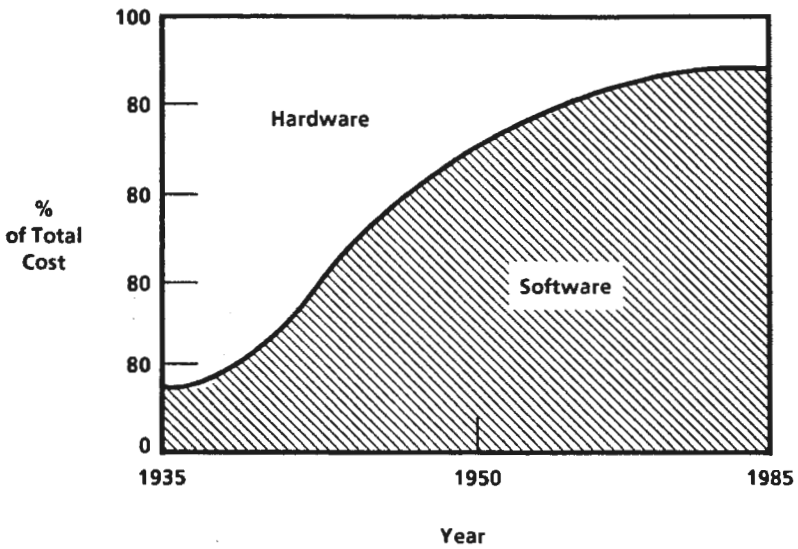


FIGURE 2.1-1. HARDWARE AND SOFTWARE COST TRENDS (from Shooman, 1984)

Software costs are roughly proportional to the number of lines of code in the program. Consequently, greater software complexity is accompanied by higher labor costs. Greater complexity also increases the likelihood of errors being made in the design and coding of software. Not surprisingly, the bulk of software development costs are devoted to testing and validation, a cost which also grows with increases in software complexity.

Concerns with software reliability can be traced back to two pressures: The need to reduce development costs, with the testing and validation stage as a major target; and the increasing likelihood of errors due to more complex software.

Attempts to reduce software errors conflict with attempts to reduce software costs. Obvious ways of reducing errors include devoting more time to the early stages of software design, such as development of software specifications and requirements (to enhance software quality) and thorough testing of the software. Both approaches are labor intensive and extremely costly, and can have a major impact on software development time. Therefore, there is a need to find other methods which promise effective but less expensive solutions.

Two approaches to offering more cost-effective quality control are described in this tutorial. The first approach is single-version software reliability modeling. The second approach is the development of fault-tolerant software.

- Single-version software reliability modeling serves two important functions (Goel, 1983). The modeling provides (1) a useful decision tool for software development planning, and (2) some assurance of how well the software can be expected to perform over time.

Based upon the results of these models, educated decisions can be made concerning how staff time and other costs can be allocated to produce the highest quality software. Predictions of the number of remaining faults enable software developers to determine the additional costs required for incremental enhancements in software reliability.

The cost of finding remaining faults rises very quickly. As faults are removed, remaining faults become more difficult to locate. At some point removing additional faults is no longer cost effective. Justification for decisions about reasonable reliability levels can be based, in part, on objective measures (i.e., performance parameters) provided by software reliability models.

- Fault-tolerant software has safeguards built into the software to enable it to detect and recover from faults, avoiding system failures or minimizing their impact.

  Examples of these safeguards include N-version Software (NVS) and Recovery Block (RB) methodology. NVS refers to incorporating multiple versions of the same software module, which have been developed by different coders, into the software program. When one version fails, a second version can be tried. The process continues until a version is successfully run.

  The RB method refers to providing components which may be alternated with a faulty component. Software reliability models have also been developed for fault-tolerant software. Such models provide the same benefits as single-version reliability models. Results of these models can be used for software development planning and assurance of software reliability.

2.2. Achieving System Reliability Estimates

Software reliability models attempt to achieve the same objectives as their hardware reliability counterparts: To provide reliability estimates that can contribute to estimation of overall system reliability. Initial attempts at software reliability modeling were based upon hardware reliability constructs. However, inherent differences between hardware and software have resulted in software reliability models assuming a somewhat different approach. This is especially true for single-version, fault-intolerant software models.

Hardware reliability models are developed on the basis of known failure rates for the components or modules comprising the system. These failure rates can be estimated using known values from Department of Defense (DOD) MIL-HDBK-217 or relying on past experience with similar hardware components. In the case of software reliability estimation, there are no failure rate standards. Although software is constructed by integrating existing pieces or modules into a newly integrated package, fault data do not exist for these entities.

There are other differences between software and hardware which prevent the simple application of hardware models to software estimation. A major difference lies in the source of faults. In the case of hardware, faults are introduced during the manufacturing stage, when each individual module is built. Each time a module is manufactured, there is a possibility that faults in the parts or

parts or entire assemblage may result. Faults may also appear during use as the module ages and wears. In contrast, software faults occur primarily during the design and development process. Copying the software does not produce new faults or wear out the software.

The primary consequence of these differences can be seen in failure rate curves viewed over the life cycle of the system. Hardware failures will start high and then drop as the faults are fixed. The failure rate will then remain constant until modules begin to wear out, at which time the failure rate will begin to increase again. Software, in contrast, will start high and drop, achieving a similar constant failure rate as is found in hardware. However, there will not be a subsequent increase over time since the software does not wear out. Consequently, software exhibits reliability growth. Other differences between hardware and software are summarized in table 2.2-1 (Krauson and Baker, 1982).

TABLE 2.2-1. COMPARISON OF HARDWARE AND SOFTWARE PROPERTIES (from Krauson and Baker, 1982)

Property	Characteristic	
	Hardware	Software
Ratio of Errors	Errors cause a relatively large (60% to 80%) portion of system failures	Errors cause a relatively small (20% to 40%) but significant portion of system failures
Parts Interchangeability	Many components are identical	Each module has a unique function and implementation
Fault Replication	Different copies of identical components have different defects	All copies of the same master have the same faults
Usage	Usage causes wear defects	Usage has no effect on faults

In the past, the inability to apply hardware reliability models to software has resulted in the assumption that software reliability has a value of 1.0. This value was then used to compute system reliability. In recent years, the increasingly critical role of software in overall system performance has made

this assumption untenable. Software reliability must be either obtained from exhaustive testing or estimated, rather than assumed, in order to properly assess system reliability.

These differences have affected the development of fault-intolerant and fault-tolerant software reliability models in unique ways. Assessment of single-version, fault-intolerant software reliability involves predicting the expected number of faults in the software as a whole. Since there are no a priori standards for expected number of faults, as are available for hardware, an initial testing phase must take place. Based upon the number of faults detected during testing, statistical methods can then be used to predict the fault distribution and estimate the remaining number of errors. Testing can then continue, if required, until remaining faults are found and corrected.

Fault-tolerant reliability methods are more similar to hardware methods. Within fault-tolerant software systems are individual software modules, such as each of the multiple versions of a given software module (NVS), RBs, and other "pieces." These pieces are conceptually similar to individual hardware modules. The similarity ends here in that currently there are no fault standards for these individual software modules. Rather, reliability for these modules must be estimated using testing and other procedures. Ideally, a database, which will include module reliability data, will be compiled over the next five to ten years as more experience is gained using these techniques. Overall software reliability is then determined through combining individual module reliabilities, as is the case with hardware.

3. THE SOFTWARE DEVELOPMENT PROCESS

3.1. Relationship Between the Software Development Process and Software Reliability Estimation

Gephart et al., 1978 have developed a generic model of software error rate as a function of software life cycle. Figure 3.1-1 (Gephart et al., 1978) shows how error rate varies with the phase of the software development cycle. Phase I, the design and initial testing period, involves designing equations and algorithms; preparing flowcharts; and writing, testing, and integrating individual software modules. Types of errors likely to occur during this phase include typographical errors, syntax errors, and logic errors. Typically, initial system integration testing is performed at the end of this phase.

"Advanced" testing, involving the program as a whole, takes place during the second phase, and includes subjecting the software to inputs thought to be typical of the use environment. By the end of this phase, the software is actually fielded. Phase III, an operational phase, involves only limited maintenance. Two types of errors will be removed in Phase III: (1) Those which have critical impact on system performance, and (2) Those whose removal is cost-effective. Phase IV involves operational use and, typically, no maintenance is performed.

Because Phase I is the design phase, the error rate is likely to be quite high and perhaps erratic. As the software becomes more structured and integrated, and extensive testing is performed, error rates will become more predictable and show a declining rate during Phase II as testing and debugging efforts continue. During Phase III, the error rate will continue to decline, eventually leveling off. Typically, error data used for reliability assessment are acquired during Phases II and III, allowing identification of error distributions and predicting the expected number of errors. The accuracy of reliability assessment depends, not surprisingly, on the quality of the error data obtained during these phases.

Since most of the errors have been identified and corrected during the first three phases, Phase IV errors tend to be infrequent and occur as a function of using a previously untested path as a result of an unanticipated situation. These errors are difficult if not impossible to find and correct. Some errors are never corrected, because they cannot be found within the constraints of reasonable time and cost. Also, an error may not always be successfully corrected, and the process of correcting one error may introduce new errors. Therefore, the error rate in Phase IV tends to be constant and mostly nonexistent.

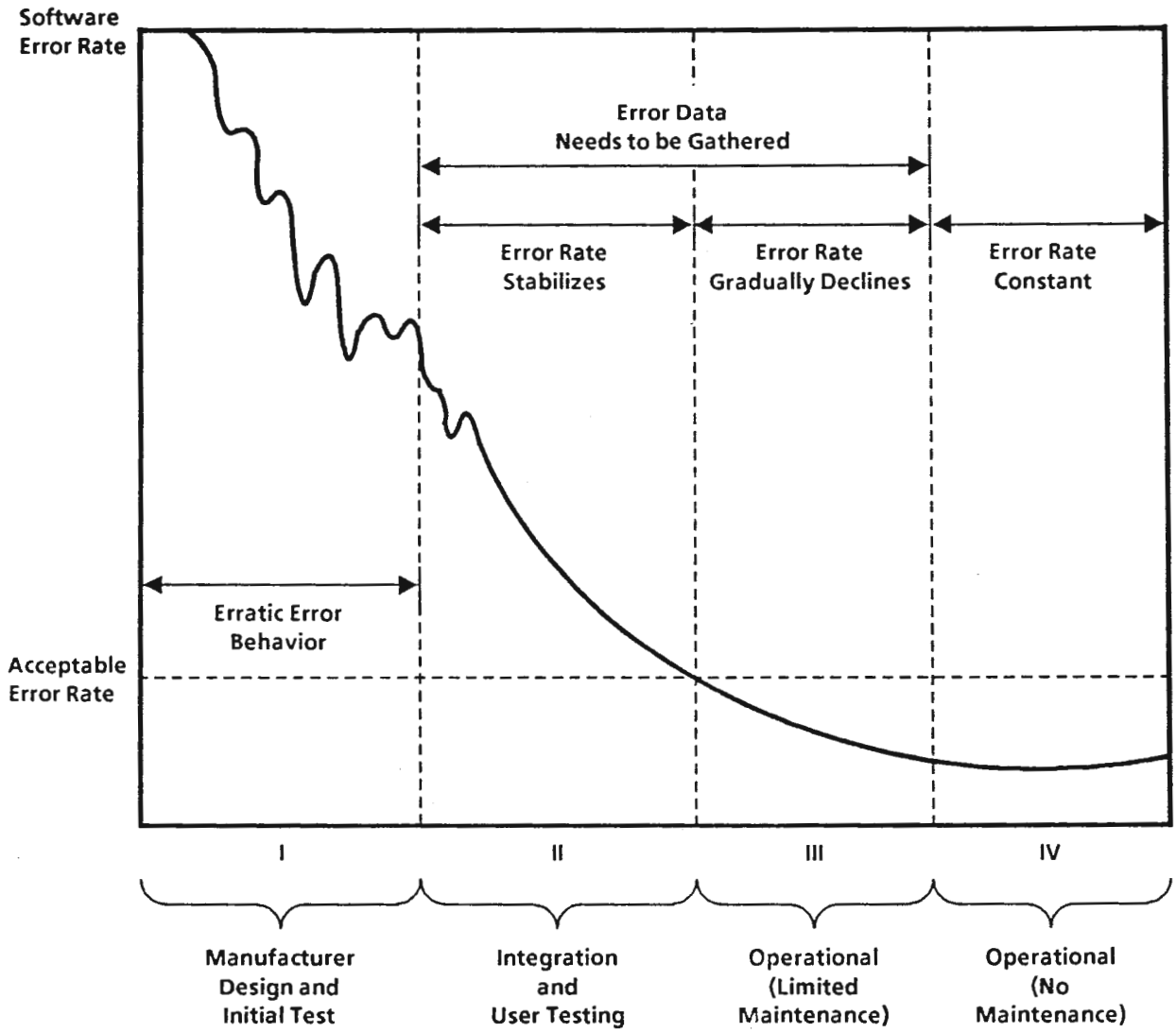


FIGURE 3.1-1. PHASES OF SOFTWARE DEVELOPMENT
(Gephart et al., 1978)

3.2. Software Testing

Software testing, used primarily during Phases II and III to determine the existence of errors in the coded software, is a somewhat heuristic process: Testing effectiveness is dependent upon the tester's ability to define test selection rules which, in turn, can be translated into tests that are sufficient for verifying the software's correctness. Test cases can be generated based upon the structure of the software (statements, control flow, etc.) or the functions identified in the functional specification. In either case, testing will reveal the presence of only those errors it was designed to identify; it will not detect those errors which are outside the testing boundaries.

A potential solution to this problem is the development of a metric (or methodology) which can be used to predict (based on data obtained during testing) the overall reliability of the software, including the expected number of errors in the program. This is the approach taken by software reliability models which have been specifically developed to address these problems. It is the subject of the next portion of this tutorial.

4. FAULT-INTOLERANT RELIABILITY MODELS

4.1. Introduction

In this section, the assumptions and characteristics of one category of software reliability models, those designed for single-version, fault-intolerant software, are presented. These models, which are based mainly on the failure history of the software, are time-dependent. The time-dependent approach uses either the times between failures (time-between-failure models) or the number of failures observed in a sequence of test time intervals (error-count models) to estimate the reliability of the software undergoing testing.

Both classes of time-dependent models are intended to be used during the software testing and debugging phase to provide either a reliability estimate or some other useful parameter (e.g., number of remaining errors, test reliability, and confidence in the program) which in turn yields some measure of the correctness of the program. Application of these models relies on methods of testing, verification, code walk-through, etc. to first provide the raw data (either as time between failures or as number of failures per individual test interval). Models are then used to estimate the expected performance statistics based on the use of maximum likelihood estimation (or least squares approximation) statistics and the underlying distribution of the data. This process, based on Goel's (1983) description and example data, is explained in more detail below and is summarized in figure 4.1-1.

4.1.1. Step 1: Evaluating the Failure Data Acquired From Testing

The initial step in selecting and applying a reliability model is to examine the test error data in order to identify the underlying data distribution. Identification of the data distribution is crucial because this distribution determines (1) whether the data can appropriately be modeled by time-dependent models; and (2) which class of time-dependent models (time-between-failure or failure-count) should be used. To ensure that the actual data distribution is attained, biases in the testing process need to be accounted for in the modeling process. For example, the actual distribution may be distorted if more and more functions were added during the subsystem or system test, or if test case severity changed during testing. Such biases may be compensated for by use of statistical procedures, such as normalizing the data to meet the steady-state system assumption of the models.

To demonstrate how the reliability modeling process occurs, an example will be shown. The data to be used are given as times between failures and are shown in table 4.1-1 (Goel, 1983). These data are also plotted in figure 4.1-2 (Goel, 1983).

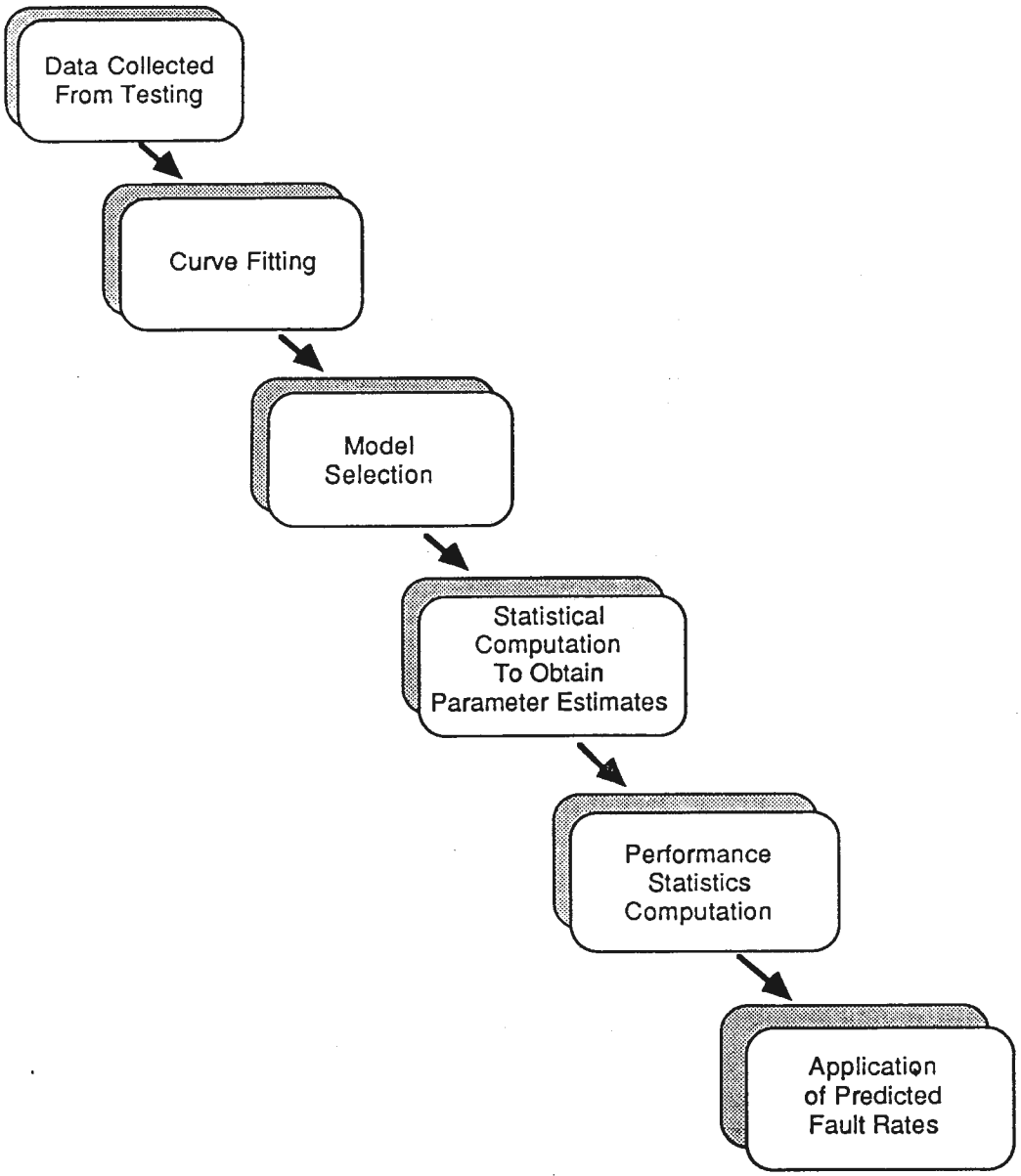


FIGURE 4.1-1. FLOW FOR COMPLETING THE SOFTWARE RELIABILITY ESTIMATION PROCESS

TABLE 4.1-1. FAILURES IN ONE HOUR (EXECUTION TIME) INTERVALS AND CUMULATIVE FAILURES (from Goel, 1983)

Hour	No.	Cum.
1	27	27
2	16	43
3	11	54
4	10	64
5	11	75
6	7	82
7	2	84
8	5	89
9	3	92
10	1	93
11	4	97
12	7	104
13	2	106
14	5	111
15	5	116
16	6	122
17	0	122
18	5	127
19	1	128
20	1	129
21	2	131
22	1	132
23	2	134
24	1	135
25	1	136

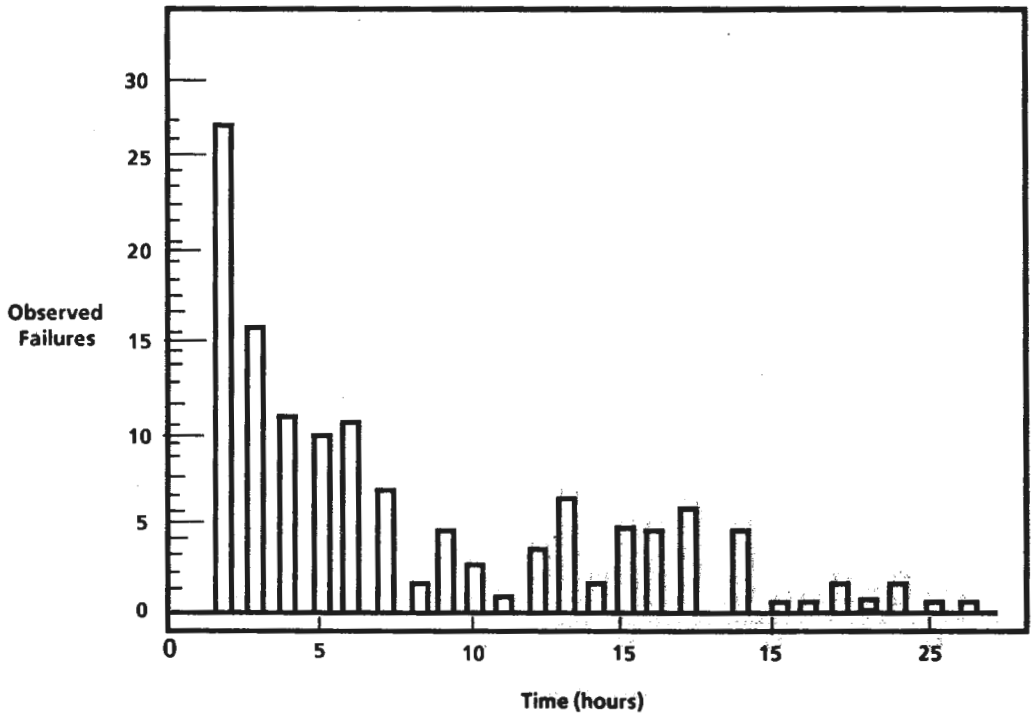


FIGURE 4.1-2. OBSERVED FAILURES PER HOUR (from Goel, 1983)

4.1.2. Step 2: Curve Fitting

After the data have been plotted, examined, and normalized (as required), decisions about the applicability of a class of models or the selection of a specific model can be made based upon the apparent distribution of the data. The two classes of models (time-between-failure, failure count) require different assumptions, as outlined in table 4.1-2. In addition, each class of model assumes a specific statistical distribution.

In the case of time-between-failure models, the test data must follow an exponential distribution since these models assume that the failure interval is constant and proportional to the current fault content of the program. The constant failure rate assumption means that for a given system of software, a given failure can be expected to occur during the next time interval regardless of when during testing the time interval is selected. However, successive times between failure are expected to increase in duration as faults are removed from the software. (Note that for a given set of test data this may not be entirely true, since the failure times are random variables and the resultant data are subject to statistical fluctuations.) Thus, for this given class of models, the assumed underlying distribution is expected to reflect the overall improvement in software quality as faults are detected and removed. The parameters of the distribution depend upon the number of faults remaining in the system, in a given interval, after the ith failure.

TABLE 4.1-2. FAULT-INTOLERANT SOFTWARE RELIABILITY MODEL ASSUMPTIONS (from Goel, 1983)

Time-Between-Failure Models	Error-Count Models
• Independent times between failures	• Testing intervals are independent of each other
• Equal probability of the exposure of each fault	• Testing during intervals reasonably homogeneous
• Embedded faults are independent of each other	• Numbers of faults detected during non-overlapping intervals are independent of each other
• Immediate fault removal, perfect fault removal, no new faults introduced during correction	

Failure count models attempt to model the number of failures in a given testing interval. As faults are removed from the system, these models assume that the number of failures observed per unit time will decrease and the curve representing cumulative number of failures versus time will eventually level off. Typically, duration of time intervals is fixed a priori and the number of failures in each interval is a random variable. Consequently, the underlying data must be distributed as a poisson distribution in that the failure rate (number of failures per unit time) is expected to decrease over time, thus demonstrating an increase in program reliability.

The data used in the example demonstrate a monotonically decreasing failure rate (number of failures per hour) and appear to conform to a poisson distribution (see figure 4.1-3; Goel, 1983). Consequently, a fault-count model is appropriate for these data.

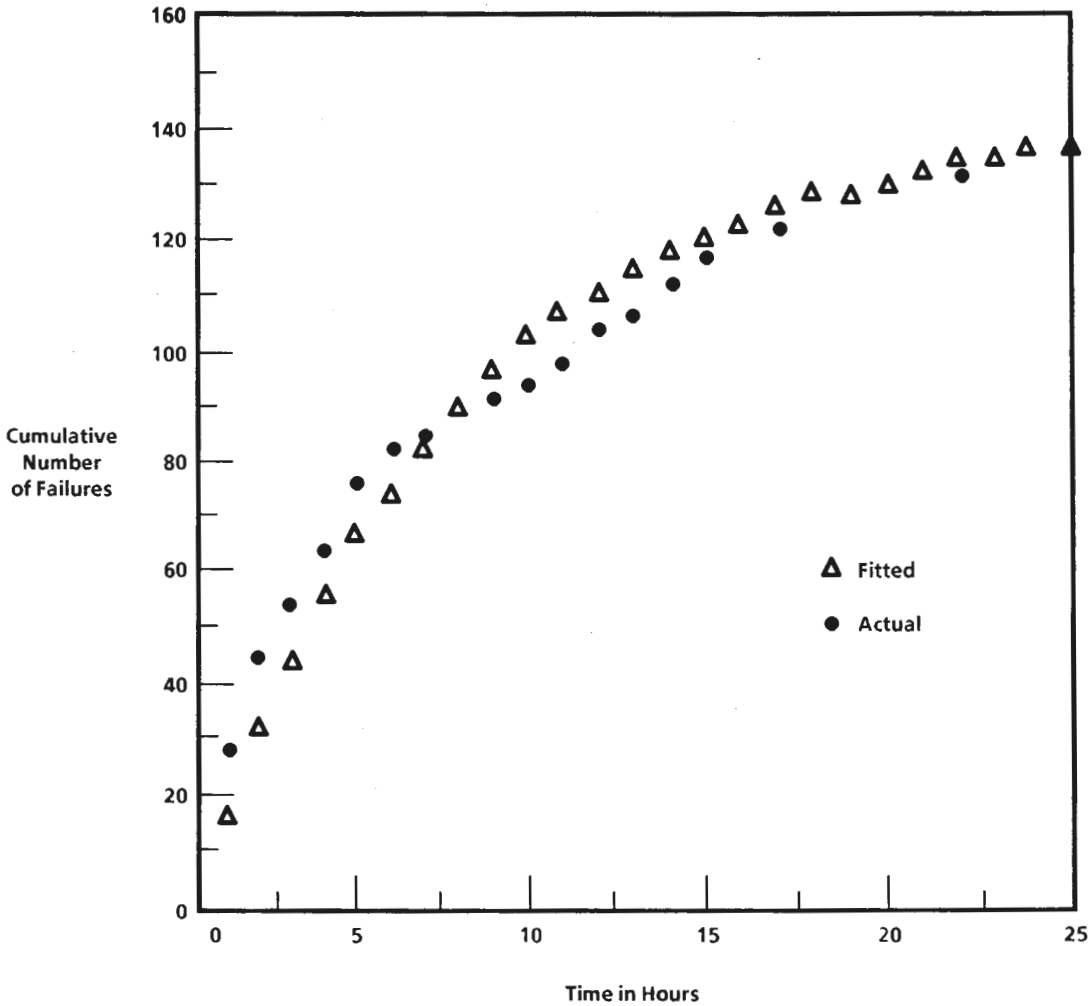


FIGURE 4.1-3. ACTUAL VERSUS PREDICTED FAILURES PER HOUR (from Goel, 1983)

4.1.3. Step 3: Model Selection

At this point, a specific fault-count model can be selected based upon the results obtained from the previous two steps. Listed in table 4.1-3 (Goel, 1983) are current time-dependent models appropriate for real-time applications. (See Prater, Hitt, and Eldredge, 1986, for a discussion of model requirements for real-time applications.) Included are both time-between-failure and failure-count models.

TABLE 4.1-3. FAULT-INTOLERANT SOFTWARE RELIABILITY MODELS (from Goel, 1983)

Time-Between-Failure Models	Error-Count Models
• Jelinski-Moranda De-Eutrophication Model	• Goel-Okumoto Non-Homogeneous Poisson Process (NHPP) Model
• Geometric De-Eutrophication	• Geometric Poisson Model
• Generalized Imperfect Debugging Model	• Extended Jelinski-Moranda Model
• Bug-Proportional Model	• Modified Geometric De-Eutrophication Model
	• Shooman Exponential Model

Selection of an appropriate model requires that the assumptions of the model be met by characteristics of the data. These assumptions are listed for each model in appendices A and B. (Goel, 1983, provides a useful discussion of the validity of these assumptions.)

For the current example, it was determined that the assumptions associated with the Goel-Okumoto model best corresponded to the data. The assumptions for this model are listed in table 4.1-4.

4.1.4. Step 4: Statistical Computation to Obtain Parameter Estimates (from Goel, 1983)

Shown in tables 4.1-5 and 4.1-6 are the various parameters and equations required for reliability estimation for each of the models. The parameters are used by the model to define the shape of the predicted fault distribution and to calculate the performance parameters (such as expected number of errors and expected time to the nth failure) which are given by the equations.

TABLE 4.1-4. ASSUMPTIONS FOR THE GOEL-OKUMOTO NON-HOMOGENEOUS POISSON PROCESS MODEL (from Goel, 1983)

- Initial fault content:

  Expected number of software faults to be eventually detected is an unknown fixed quantity.

  Actual number of faults to be observed is a random variable.

- Independence of faults:

  Each failure is caused by one fault and each of them is equally likely to cause a failure during testing.

  Number of software faults detected during non-overlapping testing intervals is independent of each other.

- Fault removal process:

  Fault removal time is negligible.

  No new faults are introduced during the fault removal process.

- Intensity function:

  Expected number of software faults detected during $(t, t + \Delta t)$ is proportional to the expected number of software faults undetected by time t, i.e.,

EQN 20

where

$m(t)$ = expected number of software faults detected by time t,

a = expected number of software faults to be eventually detected,

b = constant of proportionality.

The intensity function or the fault-detection rate $\lambda(t)$ is a decreasing function of t and is given by

EQN 21

where a, b, and m(t) are as defined above.

TABLE 4.1-5. PERFORMANCE MEASURES FOR TIME-BETWEEN-FAILURE MODELS

Model	Hazard Function	Reliability Function	Mean Time to Failure
Jelinski-Moranda De-Eutrophication Model	$Z(X_i) = \phi[N - (i - 1)]$ where N = the total number of initial errors in the program ϕ = a proportionality constant X_i = the length of the ith debugging interval (the time between detection of the (i - 1)st and the ith errors) i = the number of errors discovered	$R(X_i) = \exp[-\phi(N-n)X_i]$	$MTTF = 1/[\phi (N - n)]$ where n = the number of errors found to date
Geometric De-Eutrophication Model	$Z(X_i) = DK^{i-1}$ where X_i = the ith debugging interval D = the initial error detection rate K = a proportionality constant i = the number of errors discovered after i interval	$R(Xi) = \exp[- DK^nX_i]$ where n = the total number of errors discovered	$MTTF = \frac{1}{DK^n}$
Generalized Imperfect Debugging Model	$\lambda(t) = \phi(N - p(i - 1))t^{\alpha - 1}$ where ϕ = a proportionality constant N = the total number of errors p = the probability of perfect programmer debugging behavior α = the parameter that controls the shape of the failure rate	$R(t) = \exp(- \frac{\phi(N - p(i - 1))t^{\alpha}}{\alpha}$	$MTTF = \frac{1}{\alpha} \{ \frac{\alpha}{(N - p(i - 1))} \}^{\frac{1}{\alpha}} \Gamma(\frac{1}{\alpha})$
Bug-Proportional Model	$Z(t) = K\varepsilon_r(\tau) = K[(E_T/I_T) - \varepsilon_c(\tau)]$ where K = an arbitrary constant $\varepsilon_r(\tau)$ = the number of remaining bugs E_T = the total number of errors originally present I_T = the total number of machine instructions $\varepsilon_c(\tau)$ = the number of corrected bugs	$R(t) = \exp\{- [K\varepsilon_r(\tau)]t\}$ $= \exp\{-K[(E_T/I_T) - \varepsilon_c(\tau)]t\}$	$MTTF = \frac{1}{K\varepsilon_r(\tau)} = \frac{1}{\beta(1 - \alpha\tau)}$ where $\beta = \frac{E_T}{I_T} K$ $\alpha = \frac{\rho_0 I_T}{E_T}$ where ρ_0 = a constant rate of error correction

TABLE 4.1-6. PERFORMANCE MEASURES FOR FAULT-COUNT MODELS

Model	Hazard Function	Reliability Function	Mean Time to Failure
Geometric Poisson Model	$Z(t_i) = \lambda K^{i-1}$ where t_i = the ith debugging interval λ = the average number of faults occurring in the first interval K = a proportionality constant, $0 < K < 1$	$R(t) = e^{-\lambda K^i t}$	$MTTF = \frac{1}{\lambda K^i}$
Extended Jelinski-Moranda Model	$Z(t_i) = \phi[N - n_{i-1}]$ where ϕ = a proportionality constant N = the total number of initial errors n_i = the cumulative number of errors found through the ith time interval t_i = the ith debugging interval	$R(t) = e^{-\phi(N - n_i)t}$	$MTTF = \frac{1}{\phi[N - n_i]}$
Modified Geometric De-Eutrophication Model	$Z(t_i) = DK^{M_i - 1}$ where D = the fault detection rate K = a positive constant less than 1 M_{i-1} = the cumulative number of errors detected M_i = the cumulative number of errors found up to the ith time interval	$R(t) = e^{-DK^{M_n}t}$	$MTTF = \frac{1}{DK^{M_n}}$ where n = the total number of time intervals
Shooman Exponential Model	$Z(t) = Ce_r(\tau)$ where C = a proportionality constant	$R(t) = \exp\{-C[(E/I) - e_c(\tau)]t\}$ where E = the total number of errors present at time $\tau = 0$ I = the total number of machine instructions $e_c(\tau)$ = the total number of errors corrected in interval τ	$MTTF = 1/\{[C[E/I) - e_c(\tau)]\}$

Model	Expected Number of Detected Faults	Expected Number of Remaining Faults	Software Reliability
Goel-Okumoto Non-Homogeneous Poisson Process Model	$\hat{m}(t) = \hat{a}(1 - e^{-\hat{b}t})$	$E[\hat{\bar{N}}(t)] = \hat{a}e^{-\hat{b}t}$ where $\bar{N}(t)$ = number of faults remaining in the system at time t	$\hat{R}_{x_k}[S_{k-1}(x\|S) = \exp[-\hat{a}\{e^{-\hat{b}s} - e^{-\hat{b}(s+x)}\}]$ where X_k = the time between failures (k - 1) and k S_k = the time to k failures

In order to obtain the parameter estimates, statistical methods (such as least squares and maximum likelihood estimation) are used. In the case of the Goel-Okumoto model, the method of maximum likelihood estimation is used. This statistic provides an estimate of the likelihood that the sample of error data collected during testing actually represents the true data. This is necessary because testing is based on a selected testing strategy that does not thoroughly test all aspects of the software. Consequently, the error data identified is only a subset of the total error data population and may be subject to testing bias.

The basic formula for the Goel-Okumoto model is:

$$m(t) = a(1 - e^{-bt})$$

where a and b are defined as in table 4.1-4.

The total number of software faults detected by time t, N(t), under the above assumptions is representative of the NHPP with mean value functions m(t) and intensity function $\lambda(t)$ as given above. The distribution of N(t), hence, is given by

$$P\{N(t) = y\} = \frac{\{m(t)\}^y}{y!} e^{-m(t)}, y = 0,1,2,\ldots$$

and

$$E[N(t)] = m(t) = a(1 - e^{-bt}) .$$

Using this equation and the method of maximum likelihood estimation as follows, estimates of a and b can be obtained.

Suppose $y_1, y_2, \ldots, y_n$ are the cumulative number of software faults detected by times $t_1, t_2, \ldots, t_n$, respectively. Then the likelihood function for (a,b) given the data pairs

$\{(y_i, t_i), i = 1,2,\ldots,n\}$ is

$$L(a,b \mid y_1, y_2, \ldots, y_n, t_1, t_2, \ldots, t_n) =$$

$$= \Pr\{N(t_1) = y_1, N(t_2) = y_2, \ldots, N(t_n) = y_n\}$$

$$= \prod_{i=1}^{n} \frac{[m(t_i) - m(t_{i-1})]^{y_i - y_{i-1}}}{(y_i - y_{i-1})!} e^{-\{m(t_i) - m(t_{i-1})\}}$$

$$= \prod_{i=1}^{n} \frac{\{a(e^{-bt_{i-1}} - e^{-bt_i})\}^{y_i - y_{i-1}}}{(y_i - y_{i-1})!} e^{-a(1 - e^{-bt_n})}$$

Maximum likelihood estimation of parameters a and b can then be obtained by solving the following pair of equations simultaneously:

$$a(1 - e^{-bt_n}) = y_n$$

and

$$at_n e^{-bt_n} = \sum_{i=1}^{n} \frac{(y_i - y_{i-1})(t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}})}{e^{-bt_{i-1}} - e^{-bt_i}}$$

The above two equations yield

$$\frac{y_n t_n e^{-bt_n}}{(1 - e^{-bt_n})} = \sum_{i=1}^{n} \frac{(y_i - y_{i-1})(t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}})}{e^{-bt_{i-1}} - e^{-bt_i}}$$

from which we can find $\hat{b}$ numerically and hence

$$a = \frac{y_n}{(1 - e^{-bt_n})}$$

$\hat{a}$ = 142.32 (Expected number of faults)

$\hat{b}$ = 0.1246 (Faults per fault per hour)

4.1.5. Step 5: Computation of Performance Statistics

Tables 4.1-5 and 4.1-6 show the equations used to calculate the performance statistics associated with each model. These performance statistics include: the hazard function, reliability function, and Mean Time to Failure (MTTF). The hazard function estimates the likelihood of software failure at a given instant of time. The reliability function provides an estimate of the probability that the software will not fail in a given time interval. MTTF is the expected value of the time to next failure.

The Goel-Okumoto model uses slightly different performance statistics which are roughly analogous to the performance statistics just described. Computation of these statistics is shown below:

Expected number of software faults detected by time t is given by

$$m(t) = a\,(1 - e^{-bt})$$

Expected number of remaining faults in the software system at time t is given by

$$E[N(t)] = ae^{-bt}$$

where $\hat{N}(t)$ = number of faults remaining in the system at time t.

Let X_k be the time between failures (k-1) and k and let S_k be the time to k failures. Then it can be shown that the conditional reliability function of X_k, given $S_{k-1} = s$, is,

$$R_{X_k \mid S_{k-1}}(x \mid s) = \exp\left[-a\left\{e^{-bs} - e^{-b(s+x)}\right\}\right].$$

The fitted model is obtained by substituting the above estimated values.

$$\hat{m}(t) = 142.32\ (1 - e^{-0.1246t})$$

and

$$\hat{\lambda}(t) = 17.73 \times e^{-0.1246t}$$

4.1.6. Step 6: Application of Predicted Fault Rates

At this stage, the obtained error distribution and the predicted error distribution (based upon the equation used to compute expected number of remaining faults) can be plotted and compared (see figure 4.1-4; Goel, 1983).

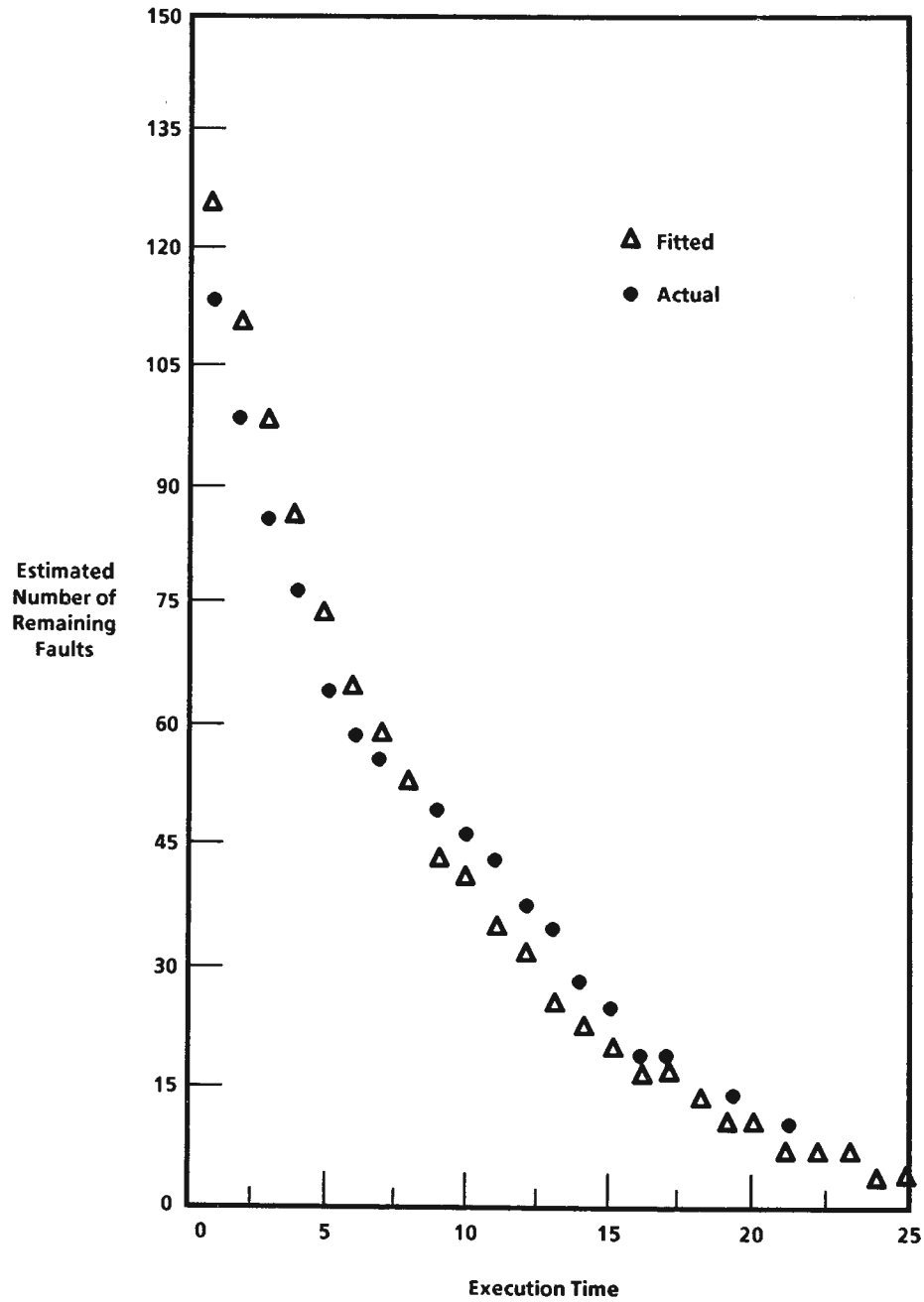


FIGURE 4.1-4. PREDICTED VERSUS ACTUAL NUMBER OF REMAINING FAULTS (from Goel, 1983)

In order to ensure that the predicted model fits the obtained data, goodness-of-fit tests should be applied. If the model fits, results of the reliability modeling process can be appropriately used to make decisions about the software program, for example, to determine whether further testing is required. If the model does not fit, additional testing and data collection may be needed and the process may have to be repeated either using the same model or a more appropriate model based upon the additional data.

4.2. Computation of a Time-Between-Failures Model

In section 4.1, the Goel-Okumoto model, a fault-count model which assumes a poisson distribution, was described in some detail. In Steps 1 and 2, if the data generated from the testing had been exponentially distributed and met the assumptions of one of the time-between-failure models, a different model would have been selected, such as the Jelinski-Moranda De-Eutrophication Model. Steps 3 through 5 would then have been performed as follows:

4.2.1. Step 3: Model Selection

As before, the assumptions of the selected model must be matched with the characteristics of the data. Assume a set of data which matches the assumptions of the Jelinski-Moranda model (see table 4.2-1; Goel, 1983).

4.2.2. Step 4: Statistical Computation to Obtain Parameter Estimates (from Goel, 1983)

The time between the (i-1)st and the ith failures T_i, is distributed exponentially with parameter $\Phi[N - (i-1)]$.

$$\text{pdf of } T_i\text{: } f(t_i) = \phi[N-(i-1)]\exp[-\phi\{N-(i-1)\}t_i]$$

$$\text{cdf of } T_i = F(t_i) = 1 - \exp[-\phi\{N-(i-1)t_i\}]$$

A series of n failures is observed with interfailure times as $t_1, t_2, \ldots, t_n$. Usually, the method of maximum likelihood is used to estimate the parameters N and Φ as shown below.

The likelihood function of N and Φ is

$$L(N, \phi | t_1, t_2, \ldots, t_n) = \prod_{i=1}^{n} \phi[N-(i-1)]\exp[-\phi\{N-(i-1)\}t_i]$$

TABLE 4.2-1. ASSUMPTIONS FOR THE JELINSKI-MORANDA DE-EUTROPHICATION MODEL (from Goel, 1983)

- Initial fault content:

  An unknown fixed constant N.

- Independence of faults:

  Each fault in the program is independent of other faults and each of them is equally likely to cause a failure during testing. Times between occurrences of faults are independent of each other.

- Fault removal process:

  A detected fault is removed with certainty at the end of each testing interval.

  Only one fault is removed during each testing interval.

  The fault removal time is negligible.

  No new faults are introduced during the fault removal process.

- Hazard function:

  The software failure rate of the hazard function during a failure interval is constant and is proportional to the current fault content of the tested program. Thus, during the ith testing interval,

$$z(t_i) = \phi\{N - (i-1)\},$$

  where Φ is a proportionality constant, and N is the initial number of faults in the program.

The Maximum Likelihood Estimates (MLE) of N and Φ are obtained by solving the following pair of equations simultaneously.

$$\sum_{i=1}^{n} \frac{1}{N-(i-1)} - \sum_{i=1}^{n} \phi t_i = 0$$

and

$$\frac{n}{\phi} - \sum_{i=1}^{n} [N-(i-1)]\, t_i = 0$$

The above two equations yield

$$\sum_{i=1}^{n} \frac{1}{N-(i-1)} = \frac{n}{N - \frac{1}{T} \sum_{i=1}^{n} (i-1)t_i},$$

where

$$T = \sum_{i=1}^{n} t_i .$$

The above equations can be solved numerically to find $\hat{N}$ and then $\hat{\Phi}$ can be obtained from the following equation.

$$\hat{\phi} = \frac{n}{NT - \sum_{i=1}^{n} (i-1)t_i}$$

Using the asymptotic properties of the maximum likelihood estimators, it can be shown that

$$\mathrm{Var}(\hat{N}) = \frac{n}{\phi 2} \cdot \frac{1}{\det A}$$

$$\mathrm{Var}(\hat{\phi}) = \Sigma \left(\frac{1}{N-i-1} \right)^2 \cdot \frac{1}{\det A}$$

$$\mathrm{Cov}(\hat{N},\hat{\phi}) = - \frac{T}{\det A}$$

$$\rho_{\hat{N},\hat{\phi}} = - \frac{T\phi}{\sqrt{n} \cdot \sqrt{\Sigma_2}},$$

where

$$\det A = \frac{n}{\phi 2} \sum_{i=1}^{n} \left(\frac{1}{N-i+1} \right)^2 - T^2,$$

and

$$\Sigma_2 = \sum_{i=1}^{n} \frac{1}{(N-i+1)^2}$$

4.2.3. Step 5: Computation of Performance Statistics

Reliability at time t after the nth failure is

$$R_n(t) \equiv P(T_{n+1} > t) = \exp[-\phi \{N-n\} t]$$

MTTF after n failures is

$$MTTF_n = \int_0^\infty R_n(t)\,dt = \frac{1}{\phi[N-n]}$$

Using the variance-covariance of $\hat{N}$, $\hat{\Phi}$, it can be shown that

$$\text{Variance}[R_n(t)] = \frac{e^{-2\phi_1(N-n)t}}{\det A}\{nt^2 - 2(N-n)Tt^2\phi + (N-n)^2t^2\Sigma_2\}$$

$$\text{Variance}(MTTF_n) = \frac{1}{\det A}\left[\frac{1}{\phi 2}v_1^2 - 2Tv_1v_2 + v_2^2\Sigma_2\right],$$

where

$$v_1 = -\frac{1}{(N-n)^2\phi}$$

$$v_2 = -\frac{1}{(N-n)\phi^2}, \text{ and}$$

Σ_2 and det A are as defined before.

All the above quantities can be computed by replacing N and Φ by $\hat{N}$ and $\hat{\Phi}$, respectively.

Worked examples of the Goel-Okumoto and Jelinski-Moranda models are provided in appendices C and D.

5. FAULT-TOLERANT SOFTWARE

5.1. Introduction

Fault-tolerant software is a redundancy technique that is currently being proposed for use in aircraft avionics/electronics applications where system failures are potentially disastrous, and where high reliability is required. The most widely used methods for implementation are the RB and N-version programming, which are roughly analogous to the hardware redundancy techniques of "standby sparing" and multiple modular redundancy. These fault-tolerant features for software are primarily aimed at overcoming the effects of errors in software design and coding; however, they may also compensate for failures due to hardware design deficiencies or malfunctions in input/output channels, depending upon how the versions are implemented. Fault-tolerant software by its very nature tries to incorporate techniques which will ensure that service is maintained by coping with any faults that remain after the application of all fault avoidance measures. In that respect, it does not rely on completion of validation or testing efforts, which can only establish the presence of errors but cannot assure their absence.

5.2. Principles of Fault Tolerance

Anderson and Knight (1982) identify four phases into which fault-tolerance techniques can be divided:

"1. ERROR DETECTION. In order to tolerate a fault its effects must first be detected. Clearly, this can only be achieved by performing checks to determine whether any erroneous situation has arisen.

2. DAMAGE ASSESSMENT. Having detected that the system is in error, it will usually be necessary to identify how much of the state of the system has been corrupted.

3. ERROR RECOVERY. Probably the most important aspect of fault tolerance is the provision of an effective means of transforming an erroneous state of the system into a well defined and error free state. Methods for achieving this transformation can sometimes make good use of the information retained in the erroneous state, but it can be more secure to simply discard the erroneous state and reset the system to some prior state (a recovery point).

4. CONTINUED SERVICE. In order to enable the system to continue to provide the service required by its specification, further action may be needed to ensure that the fault whose effects have been obviated does not immediately recur and thus ruin the whole

approach. Unless the fault was transient and will not recur in any case, it must either be rectified or circumvented (p. 24-3)."

As stated above, two techniques for achieving fault tolerance are the RB and N-version programming. In the case of N-version programming, a number of independently coded programs for a given function are run simultaneously (or nearly so) on N computers. The system is coupled in such a way that the N computers are hardware independent yet able to communicate efficiently so that rapid comparisons of results can be achieved. The key to successful implementation of this approach is the independent design and development of the N-versions, which prevents the occurrence of the same fault in the multiple versions.

The RB, on the other hand, requires only a single version of hardware (even though redundant hardware is often employed) and relies on the use of the acceptance test (often augmented by a "watch-dog" timer) to assure that an acceptable result is furnished within a reasonable, specified period of time. In the case where the acceptance test fails, or time expires, alternate software routines are called until one successfully passes the acceptance test. Once again, it is assumed that the alternates demonstrate independence in the way in which they were designed and coded. The acceptance test is usually structured to:

- Detect deviations from expected program execution.
- Prevent unsafe output.

Both are critical to safe performance (Hecht, 1979).

The advantage of the fault-tolerant software approach over other techniques for software reliability improvement (test, verification, etc.) is that it permits the use of system reliability modeling techniques similar to those used for hardware reliability modeling. Reliability models can be developed to describe the behavior of RB, N-version programming, or any combination of the two approaches.

5.3. Fault-tolerant Software Reliability Models

In the remainder of section 5, two software reliability models will be presented: the Hitt, E. F., J. J. Webb, and M. S. Bridgeman model (Prater, Hitt, and Eldredge, 1986) and the Scott et al., model (1983, 1984, 1987). The Hitt, E. F., J. J. Webb, and M. S. Bridgeman model will demonstrate, through worked examples, how various fault-tolerant software approaches can be decomposed and the reliability of the constituent parts estimated. The Scott et al., model is similar to the Hitt, E. F., J. J. Webb, and M. S. Bridgeman model in its assumptions, and the two models may provide equivalent results. The Scott et al., model is presented, however, to show concurrent work in the area of fault-tolerant software reliability modeling.

In contrast to the fault-intolerant models, neither fault-tolerant model has been verified/validated to the point where it has become accepted for general use. However, the use of fault-tolerant software has started to be accepted for

avionics applications, and the fault/failure data from these efforts are only now becoming available. It is expected that the two fault-tolerant software approaches will be verified and validated using the emerging database.

5.4. A Hierarchical Software Reliability Model

This part of the tutorial will describe the hierarchical software reliability model developed by Hitt, E. F., J. J. Webb, and M. S. Bridgeman (Prater, Hitt, and Eldredge, 1986) to predict the reliability of fault-tolerant software prior to its development (i.e., coding). Fault-tolerant software consists of individual software modules, such as single-version software, NVS, decision algorithms, RBs, and acceptance tests. In the model, these modules are treated as separate "function blocks," similar to piece-parts in hardware reliability modeling. Each function block is subject to specific types of faults.

5.4.1. Software Components

5.4.1.1. Single Version Software

This is a probabilistic model of deterministic or random events. The program execution is usually deterministic, and the development process is probabilistic. Single version software contains characteristic faults that must be accounted for. Examples of these faults are as follows:

- Incorrect specification.
- Misunderstood or unclear specification.
- Algorithmic error (sometimes called a computational or logic error).
- Input data error.
- Program logic error.
- Output data error.

5.4.1.2. N-version Software

This software is a fault-tolerant software technique which implements two or more functionally equivalent versions, usually in parallel. These versions may be produced independently by separate programming teams, or they may be designed differently by examining and forcing differences into the versions. Upon comparison of the alternate versions, the faults should be distinguishable. Some of the faults associated with NVS include:

- Specification error.
- Performance error (due to incomplete, inconsistent, or ambiguous specifications).

- Non-termination error.
- Algorithmic error.
- Input data error.
- Output data error.

5.4.1.3. Decision Algorithm

The decision algorithm determines the specific output. This algorithm may be a majority vote, a median select, a bit-by-bit comparison (with the number of bits that are to be compared or that are significant specified), or an average. Considerations to be made in this software design are as follows:

- The type of decision algorithm used.
- The allowable range of discrepancy of each input from all other inputs to the decision algorithm.
- The data sensitivity of the decision algorithm.

5.4.1.4. Recovery Block

The RB method is a fault-tolerant software technique that provides alternate components which may be switched in (usually serially) to replace a faulty component that has been rejected by the acceptance test. These alternate components are designed independently from the main software component (the primary alternate). They generally provide only partial functionality of the software component. The alternate components perform a reduced and simplified function. Prior to entering an alternate, the conditions existing just before entry to the primary alternate are restored. Some examples of faults that occur in the software for RBs include:

- Specification error.
- Performance error.
- Non-termination error.
- Algorithmic error.
- Input data error.
- Output data error.

5.4.1.4.1. Forward Recovery Block

A forward RB restores the system to a consistent state by compensating for inconsistencies found in the current state. For a single process, the forward RB technique requires a complete understanding of the extent of damage and a scheme for correcting the inconsistencies. Exception values (or ranges) must

be specified for each data abstraction. These exceptions are specified in response to run-time attempts to violate inherent invariant properties. These anticipated faults can be managed by forward RB techniques.

5.4.1.4.2. Backward Recovery Block

Backward RB techniques require the restoration of the system to the last correct state. This is referred to as rollback. The computation resumes from that point. Unanticipated faults, i.e., design faults, are managed by a default exception handler using automatic backward recovery.

5.4.1.5. Acceptance Test

An acceptance test checks the acceptability of data generated by a software component. This is usually implemented in software as a logical expression or algorithm. The faults associated with an acceptance test include:

- Specification error.
- Performance error.
- Algorithmic error.
- Input data error.
- Output data error.

5.4.1.6. Rollback

The rollback restores the software to the condition existing prior to the execution of an incorrect or faulty version. The rollback resets the software to the original input state allowing the next version to run.

A rollback is used in connection with RB and hybrid NVS systems. Faults that are characteristics of rollback are:

- Specification error.
- Input data error.
- Output data error.
- Unrecoverable state.

5.4.1.7. Roll-Forward

The roll-forward is always used in conjunction with a forward RB. The roll-forward transfers the restored state obtained from the forward RB to a forward position in the system. The state for which the forward RB has compensated determines the forward position for this transfer. Faults associated with roll-forward include:

- Specification error.
- Performance error.
- Input data error.
- Output data error.

5.4.2. Inputs to the Software Reliability Model

Software reliability in this model is estimated from the individual probability reliability values (or safety, availability, or accuracy values, if desired) for each function block. Values for each block are either obtained from a software reliability database (which is emerging or assumed to exist), estimated by the lines of code (and the language), derived experimentally by subjecting the function block's software to a number of test cases and counting the failures to determine a reliability value, or obtained from lower (detailed) level models. The reliability values must be real numbers with a range of $0.0 \leq$ probabilistic reliability value (or safety, availability, or accuracy value) ≤ 1.0.

5.4.3. Outputs from the Software Reliability Model

The output of the software reliability model is an overall probability reliability value (or safety, availability, or accuracy values, if desired) for the software as a whole. Reliability values can also be estimated for a specific subset of modules within the software.

5.4.4. Model Representation

Control system notation is used to represent the model. Each software module represents a transformation of input to output. Therefore, the reliability value for an individual module estimates the probability of the desired input/output transformation actually occurring. Similarly, input/output transformations taking place through the interaction of multiple modules can also be represented, allowing similar probability estimations of the likelihood that the transformation will occur. Examples of the application of control system notation to reliability estimation are shown in figures 5.4-1 (Prater, Hitt, and Eldredge) through 5.4-6. These figures are explained in more detail below.

5.4.5. The Modeling Process

The Hitt, E. F., J. J. Webb, and M. S. Bridgeman model was developed to allow software reliability estimation prior to actual software development and coding. This model is intended to be used with a computer software applications package designed to allow alternate versions of the same software architecture to be evaluated. Once a preliminary software architecture has been developed and decisions have been made as to anticipated number of N-versions, etc., these decisions can be entered into the computer. Specifically, the software developer can choose the appropriate modules from a menu system such as that shown in table 5.4-1. With the proposed applications software package, the

selected set of module types can then be shown on the computer display in the form of a block diagram.

TABLE 5.4-1. RELIABILITY MODEL MENU SELECTIONS
(from Prater, Hitt, and Eldredge, 1986)

- Structure Functional Modules
  - Single version software
  - NVS (The number of versions must be specified by the user.)
  - Decision algorithm
  - Recovery block (The number of alternates must be specified by the user.)
  - Acceptance test
  - Rollback
  - Roll-forward
- Transfer Functional Modules
  - Forward path
  - Positive feedback
  - Negative feedback
  - Positive feed-forward
  - Negative feed-forward

By means of this representation, reliability values for each module can be entered and reliability estimates of the preliminary software system obtained. If the reliability estimate is not sufficiently high, the software architecture can then be modified, for example, by adding additional versions for a given N-version module. In this way, progressive iterations can be evaluated until the desired reliability level is attained.

Note that table 5.4-1 (Prater, Hitt, and Eldredge, 1986) lists the decision algorithm and acceptance test independently of the NVS and RBs. This separation is necessary to allow for flexibility in combining individual modules. For example, the decision algorithm and acceptance test can now be used independently, as well as in conjunction with their respective pairs:

- NVS and decision algorithm, and
- Recovery block and acceptance test.

Keeping the decision algorithm and acceptance test as separate entities requires that each NVS, decision algorithm, RB, and acceptance test module have individual reliability values in order to compute the overall software system reliability. This is intended to improve the accuracy of the transfer functions for this portion of the diagram since it will accommodate variations in the implementation of these concepts.

Unfortunately, a software applications package capable of performing iterative reliability estimations is not yet available, in spite of the potential contributions such a tool could make to the process of ensuring high-quality software. Use of the Hitt, E. F., J. J. Webb, and M. S. Bridgeman model is not contingent upon the availability of such an applications package, however. This same concept of selecting individual modules and estimating the reliability of the software as a whole can be used without a software applications package. The steps involved are described below.

5.4.6. Detailed Function Blocks

The Hitt, E. F., J. J. Webb, and M. S. Bridgeman model uses the structure of the software (i.e., the expected reliabilities of the individual modules) to predict overall software reliability. Consequently, the accuracy of these predictions depends upon the extent to which the structure is "filled in." Dissecting the structure into progressively finer modules, and determining reliabilities for these modules, leads to greater predictive accuracy.

Figure 5.4-1 provides an example of a software program which has been decomposed into relatively high-level software modules. This example shows how reliability values are substituted for each function block to permit the determination of the overall system reliability. Because the software structure is represented at a very high level, the reliability values may be less predictive than desired. More precise predictions can be obtained if each of the software design techniques (single version software, N-version software, decision algorithm, RB, and acceptance test) are broken down into more detailed function blocks. (For a more finely structured model, see figure 5.4-6.)

Figure 5.4-2 is an overall representation of the functional constructs associated with each of the function blocks. Errors associated with each of these constructs are shown in tables 5.4-2 and 5.4-3. Each module in figure 5.4-1 has a reliability value associated with it. The occurrence of these errors within a specific software module drops the reliability of that module below 1.0.

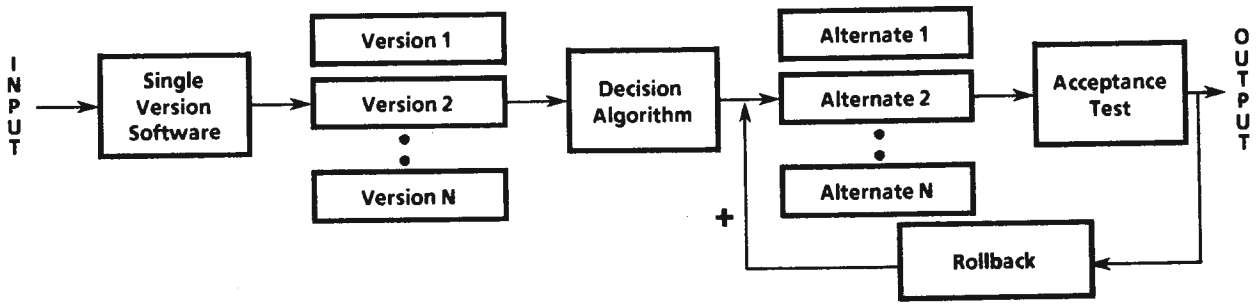


FIGURE 5.4-1. SIMPLE BLOCK DIAGRAM EXAMPLE
(from Prater, Hitt, and Eldredge, 1986)

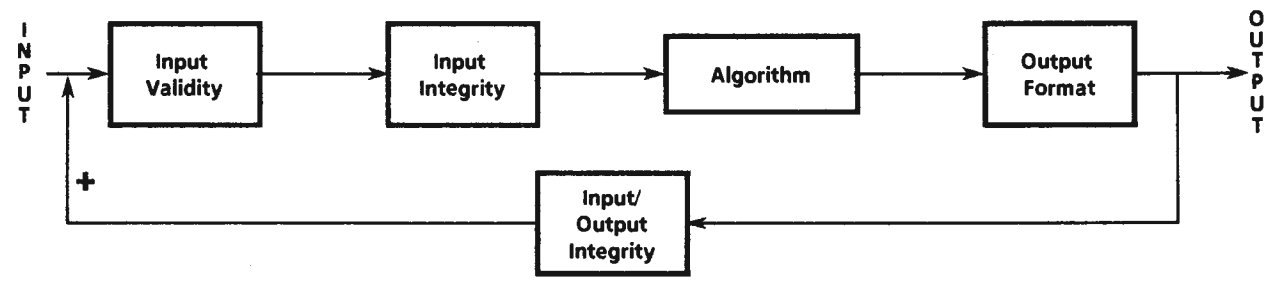


FIGURE 5.4-2. OVERALL REPRESENTATION OF THE FUNCTIONAL CONSTRUCTS
(from Prater, Hitt, and Eldredge, 1986)

TABLE 5.4-2. CORRELATION OF FAULTS WITH THE DETAILED PORTIONS OF THE SINGLE VERSION SOFTWARE FUNCTION BLOCK

	Detailed Function Blocks				
Faults	Input Validity	Input Integrity	Algorithm	Output Format	Input/ Output Integrity
Incorrect Specification	X	X	X	X	X
Misunderstood or Unclear Specification	X	X	X	X	X
Algorithmic Error	X	X	X	X	X
Input Data Error	X				X
Program Logic Error	X	X	X	X	X
Output Data Error				X	X

5.4.7. Function Block States

The four possible states for any function block are as follows:

- The function block fails (an error is detected in the function block) and it is corrupt (contains one or more error).
- The function block passes, yet it is corrupt.
- The function block fails and it is error free.
- The function block passes and it is error free.

A mathematical truth table for these states is given in table 5.4-4.

TABLE 5.4-3. CORRELATION OF FAULTS WITH THE DETAILED PORTIONS OF THE DECISION ALGORITHM FUNCTION BLOCK

	Detailed Function Blocks				
Faults	Input Validity	Input Integrity	Algorithm	Output Format	Input/ Output Integrity
Input Range Error	X	X			
Algorithmic Error	X	X	X	X	X

TABLE 5.4-4. TRUTH TABLE OF FUNCTION BLOCK STATES

State	Error Exists in the Function Block	Error Detected in the Function Block	Error Corrected in the Function Block	Safe	Reliable	Available
a	T	T	T	T	T	T
a	T	T	F	T	T*	F
b	T	F	N/A	F	F	F*
c	F	T	N/A	T	F	F
d	F	F	N/A	T	T	T

Key: N/A = not applicable
* = true or false (the common interpretation is given)

5.4.8. System Reliability

The function blocks will each have an associated transfer function ("reliability"). Block diagram reduction techniques can then be used to determine overall system reliability while signal-flow diagram reduction techniques provide the overall system transfer function ("reliability"). The signal-flow diagram is useful in analyzing multiple-loop feedback systems and in determining the effect of a particular element or parameter in an overall feedback system, whereas the block diagram is useful in the design and analysis of sections of a feedback system. Block diagram reduction techniques become tedious and time consuming as the number of feedback paths increases. To solve complex problems, it is much simpler to use the theorems and properties of signal-flow graphs.

5.4.9. General Equations

The equations used in this analysis follow S. J. Mason's theorems on the properties of signal-flow graphs. The general expression for the (closed loop) system transfer function using the signal-flow diagram reduction technique is given by

$$\text{Reliability} = \frac{\Sigma_K G_K \Delta_K}{\Delta}$$

where

$\Delta = 1 - \Sigma L_1 + \Sigma L_2 - \Sigma L_3 + \ldots + (-1)^n \Sigma L_n$,

L_1 = the gain of each closed loop in the graph,

L_2 = the product of the loop gains of any two non-touching closed loops,

L_3 = the product of the loop gains of any three non-touching closed loops,

L_n = the product of the loop gains of any n non-touching closed loops,

G_K = the gain of the Kth forward path,

Δ_K = the value of Δ for that part of the graph not touching the Kth forward path (Shinners, 1978).

The transfer function for NVS and RBs are dependent upon the number of versions or alternates (n).

5.4.10. N-version Software Calculations

The transfer function for NVS is given by the following expression:

$$1 - \prod_{i=1}^{C_n} (1 - \alpha_i)$$

with

n = the number of versions in the N-version Software;

Cn = (n,z) = the number of z combinations of the n element set;

α_i = the product of reliabilities of the ith combination required for success; and

i = 1, 2,....Cn.

z = [(n/2) + 1] if n is an even number.

z = [(n+1)/2] if n is an odd number.

The examples in this section and appendix E utilize this equation.

5.4.11. Recovery Block Calculations

The transfer function for a RB is dependent upon the number of alternates (n) that are used. This transfer function is calculated with the following equation:

$$G_1 + (1 - G_1)G_2 + (1 - G_1)(1 - G_2)G_3 + \ldots.$$

with G_i = the reliability value for alternate i and

i = 1, 2, 3,...n.

The examples in this section and appendix F demonstrate the determination of the overall software reliability value with this equation and the software reliability model.

The reliability values (transfer functions) for the hybrid NVS and the RB will vary if not all of the N versions or N alternates are used. The above equations will give a higher reliability value than the actual situation in these cases. Section 5.4.16 discusses the accuracy of the reliability values for the NVS and RB and how they can be calculated to reflect the actual situation.

Although the transfer function for most of the function blocks is the reliability value, one exception to this is with rollback or any feedback block. For any feedback path, the transfer function of the equivalent block in the path is (1.0 - reliability value). The second exception is with feed-forward paths. The transfer function of the equivalent block in a feed-forward path (for example, roll-forward) is (1.0 - reliability value).

5.4.12. Examples of Potential Models: Simple, High-Level Model Example

For a sample problem involving a simple, high level model, the example given in figure 5.4-3 will be used. The single-version software will be a commonly used algorithm or process, and therefore, it will have a high reliability which is well documented and stored in the software reliability database. For this example, the reliability will be 0.9991.

The NVS will have three independent versions, running in parallel. This is a common form of NVS, but not one with the highest reliability. Through the user's tests, it is determined that this function block will have a reliability of 0.994.

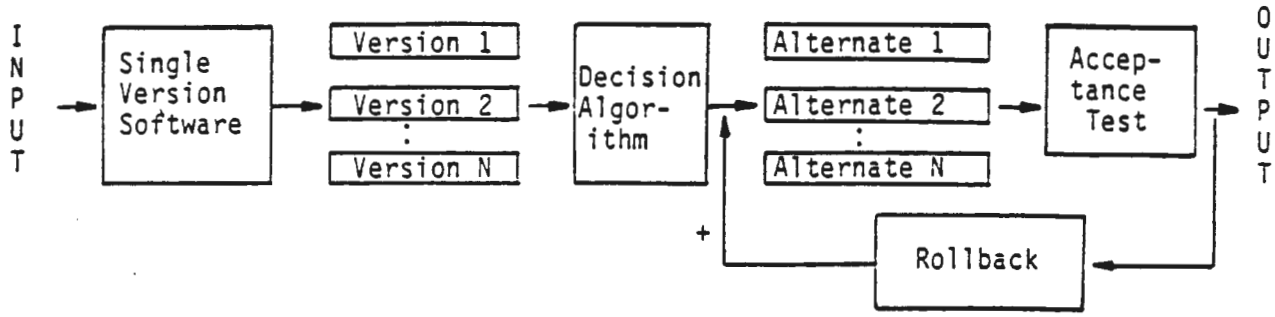


FIGURE 5.4-3. SIMPLE BLOCK DIAGRAM EXAMPLE
(from Prater, Hitt, and Eldredge, 1986)

The decision algorithm will take an average of the outputs from the NVS, excluding any version which does not meet the timing constraints. This type of decision algorithm has a high reliability since it does not have a range check or any other source for determining the validity of the output. This decision algorithm will not eliminate any erroneous outputs and will not detect the occurrence of correlated faults. The decision algorithm is simple (and consequently highly reliable), but it is not always the most desirable since its simplicity detracts from its capability of detecting errors. For this example, it is assumed that the reliability of the decision algorithm is 0.988.

The RB is a backward RB with a primary alternate and two additional, extremely simplified alternates. The RB is of a common form and its reliability can be obtained from the software reliability database. For this example, it will have a reliability value of 0.97.

The acceptance test is an output format check. This is a simplistic algorithm that is commonly used in various models. The reliability, as determined from the software reliability database, will be 0.9997.

Finally, the rollback recovers the input state of the software to its condition upon entry to the RB. This is a retrieval of the data from its memory location. The reliability of this common form of Rollback will be assumed to be available in the software reliability database. For this example, the reliability is 0.9999. This makes the transfer function for the rollback equal to (1.0 - 0.9999).

Hence, for this example,

$L_1 = (0.97) \times (0.9997) \times (+1.0 - 0.9999) = 0.0000969709$

L_2 through $L_n = 0$

$G_1 = (0.9991) \times (0.994) \times (0.988) \times (0.97) \times (0.9997)$

$= 0.951467$

G_2 through $G_K = 0$

$\Delta_1 = 1$

$\Delta = 1 - (+0.0000969709) = 0.99990302$

Therefore,

$$\text{Reliability} = \frac{(0.951467) \times (1)}{(0.99990302)} = 0.951559$$

Reliability = 0.95

5.4.13. Simple, Detailed Level Model Examples

An example of a simple, detailed level model is given in figure 5.4-4. For this example, the structure functional modules would all be single-version software and the transfer functional modules would be either forward path or positive feedback. Hence, an equivalent block diagram for this detailed block diagram is given in figure 5.4-5.

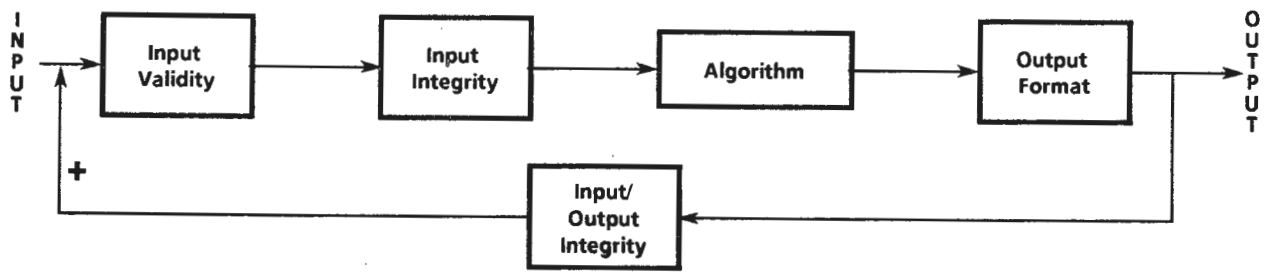


FIGURE 5.4-4. DETAILED DIAGRAM FOR THE SINGLE VERSION SOFTWARE FUNCTION BLOCK OR THE DECISION ALGORITHM FUNCTION BLOCK

In figure 5.4-5, block 1 is a single-version software block which checks the input set. This is a common process, so the reliability value will be assumed to be available in the software reliability database. For this example, the reliability value for block 1 is assumed to be 0.98.

Block 2, a single-version software block, will represent an input integrity check. It will be assumed that this algorithm is common and can consequently be found in the software reliability database. For this example, the reliability value will be 0.97.

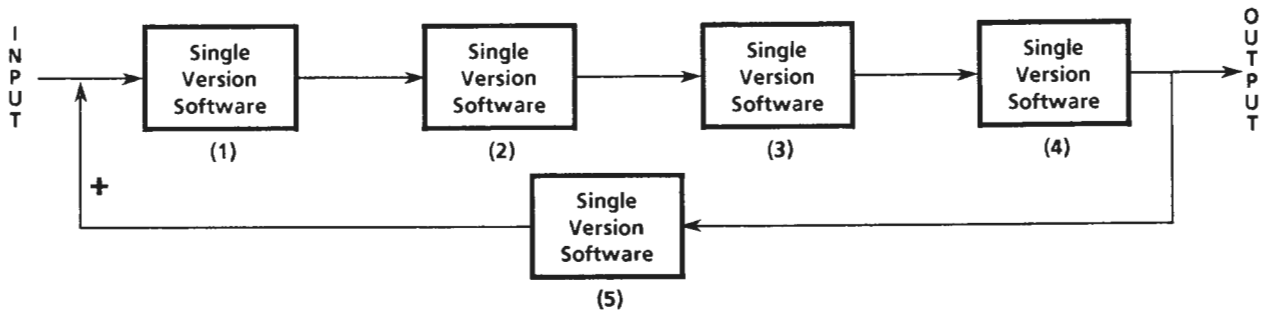


FIGURE 5.4-5. EQUIVALENT DIAGRAM INDICATING THE FUNCTIONAL STRUCTURE TO BE USED IN THE DETAILED DIAGRAM FOR A SINGLE VERSION SOFTWARE FUNCTION BLOCK

Block 3 is a single-version software block which performs an algorithm. For this example, the algorithm will be a simple one. Therefore, the reliability value for this transfer block will be assumed to be 0.992.

Block 4 will represent an output format check, performed by a single-version software block. For this example, the reliability value for this single-version software will be 0.996.

The final single-version software block, block 5, will be used to perform the input/output integrity check in which the output is checked against the input to verify the integrity of the input data. For this example, the reliability value for block 5 will be 0.964. Hence, the transfer function for this block will be (1.0 - 0.964).

Thus, for this example,

$L_1 = (0.98) \times (0.97) \times (0.992) \times (0.996) \times (1.0 - 0.964)$

$= 0.033812$

L_2 through $L_n = 0$

$G_1 = (0.98) \times (0.97) \times (0.992) \times (0.996) = 0.9392232$

G_2 through $G_K = 0$

$\Delta_1 = 1$

$\Delta = 1 - (0.033812)$

$= 0.966188$

Therefore,

$$\text{Reliability} = \frac{(0.9392232) \times (1)}{(0.966188)} = 0.9720916$$

Reliability = 0.97

5.4.14. Complex, High-Level Model Example

The complex, high level model example is shown in figure 5.4-6. Block 1 is a single-version software block. For this example, it will be a simple algorithm with a reliability value of 0.998.

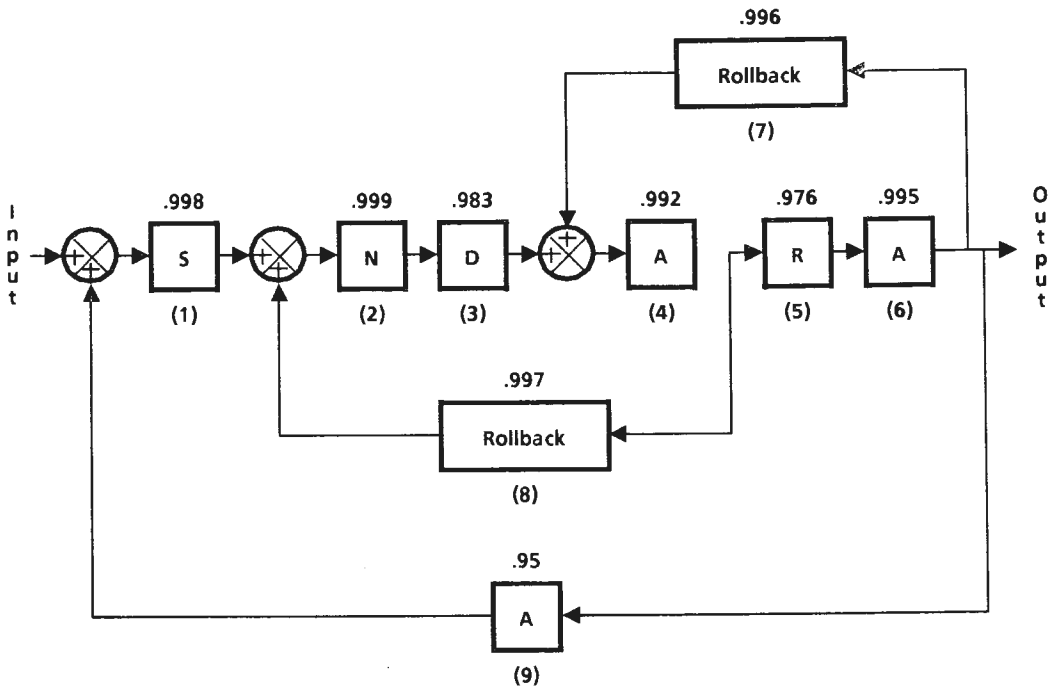
Block 2 is NVS with nine independent versions in which only three versions are run at a time. For this example, it is assumed that the reliability value for the NVS is 0.999.

Block 3 is the decision algorithm for the NVS. In this example, the decision algorithm will be median select with a reliability value of 0.983.

Block 4 is an acceptance test which will check that the range of the output from the decision algorithm is correct. If the decision algorithm failed, then the software will rollback after the acceptance test. Similarly, if the acceptance test fails, then the software will rollback to the NVS. The input to the acceptance test will be stored to accommodate for the rollback from the RB. For this example, the reliability value for the acceptance test will be 0.992.

Block 5 is a RB of the common form with a primary and two additional alternates. For this example, the reliability value of the RB will be 0.976.

Block 6 is the acceptance test for the RB. In this example, it is assumed that the reliability value for this acceptance test is 0.995.



Key:

(1) Single Version Software
(2) N-version Software
(3) Decision Algorithm
(4) Acceptance Test #1
(5) Recovery Block
(6) Acceptance Test #2
(7) Rollback #1--Backward Recovery Block
(8) Rollback #2--N-version Software in Which
Only x Versions are Used at a Time
(9) Acceptance Test #3

FIGURE 5.4-6. COMPLEX, HIGH LEVEL MODEL EXAMPLE

The rollback for the backward RB is given in block 7. This is a retrieval of the data that was stored prior to entry into Acceptance Test #1 (Block 4). For this example, the reliability value for this rollback is 0.996. Thus, the transfer function for this block is (1.0 - 0.996).

The rollback for the NVS is given in block 8. For this example, the reliability value is 0.997. Consequently, the transfer function is (1.0 - 0.997).

Finally, block (9) is an acceptance test which checks the final output against the input. For this example, Acceptance Test #3 will have a reliability value of 0.95, giving a transfer function of (1.0 - 0.95).

Hence, for this example,

Closed Loop 1 = (0.999) x (0.983) x (0.992) x (1.0 - 0.997)

= 0.0029225

Closed Loop 2 = (0.992) x (0.976) x (0.995) x (1.0 - 0.996)

= 0.0038534

Closed Loop 3 = (0.998) x (0.999) x (0.983) x (0.992) x

(0.976) x (0.995) x (1.0 - 0.95)

= 0.0472068

ΣL_1 = (Closed Loop 1) + (Closed Loop 2) + (Closed Loop 3)

= 0.0029225 + 0.0038534 + 0.0472068

= 0.0539827

L_2 through L_n = 0

G_1 = (0.998) x (0.999) x (0.983) x (0.992) x (0.976) x (0.995)

= 0.944135

G_2 through G_K = 0

Δ_1 = 1

Δ = 1 - (0.0539827) = 0.9460173

Therefore,

$$\text{Reliability} = \frac{(0.944135) \times (1)}{(0.9460173)} = 0.9980103$$

Reliability = 0.998

5.4.15. Detailed Worked Examples

Appendices E and F contained detailed worked examples of both N-version and RB systems.

5.4.16. Accuracy Constraints

The accuracy and reliability of the NVS, decision algorithm, RB, and acceptance test are dependent upon the way in which these concepts are implemented. For example, NVS may be implemented in one of the following two ways:

- The x of the N-versions are run at a time, and if these versions fail for some reason, then x or fewer of the remaining (n-x) versions are run.

- A combination of x versions is run, and if these x versions fail for some reason, a different combination of x versions is run, and so on.

  NOTE: This concept is different from the first item above, because this implementation groups different combinations of x versions. The first item, however, uses x versions, and if they fail, the x versions are in essence thrown out, and a completely new group of x versions are used; not a new combination of x versions, but x completely new versions.

Some of the differences which affect the reliability and accuracy of the decision algorithm are:

- Majority vote.

- Median select.

- Average.

- The decision algorithm only considers those values which are in certain range and then it uses one of the methods above.

A few of the items which affect the RB's accuracy and reliability are:

- Backward.

  - How far it rolls back.

  - The number of alternates available.

- Forward.

  - How far it rolls forward.

  - The accuracy of the value(s) assigned prior to the roll.

Some of the variables affecting the accuracy and reliability of the acceptance test are:

- The range of the values accepted.

- The rate of change determination for the variables.

- The format of the data.

Needless to say, all of these implementation characteristics must be considered and will affect the accuracy of the reliability values. With this software reliability model design, the accuracy is improved since these considerations can be taken into account in the assignment of reliability values (or accuracy or safety or availability values) to the function blocks.

5.4.17. Accuracy

The accuracy of the software reliability model will depend upon the accuracy of the individual values that are used as the transfer functions for each of the function blocks. In most cases, the accuracy of a model with detailed function blocks will be better than the high level software reliability model since the accuracy of the individual transfer functions will be improved.

The accuracy of the reliability values that are used for the function blocks' transfer functions will depend upon the method used to obtain such values. If the values are obtained from the software reliability database, then the accuracy of these values are indicated in the technical reports for the research method that determined the values. If the values are obtained through the use of a different software reliability model, then the accuracy is dependent on the type of model used. Regardless, accuracy values can be obtained for any and all of the transfer functions, and therefore, an accuracy for the overall software reliability values can be calculated.

5.4.18. Accuracy of the Hybrid N-version Software

The accuracy of the hybrid NVS reliability value (or safety or availability values) depends upon the number of versions that are actually used. Usually, if the software is run and only y of the N-versions is used, then the reliability value (or safety or availability values) has been overrated by considering the additional (n-y) versions in the calculation of the transfer function for the hybrid NVS. Certainly, if it is known that only y of the N-versions are actually being used, then the calculation of the transfer function for the NVS should only include those y versions.

5.4.19. Accuracy of the Recovery Block

The RB, by definition, consists of n alternates. Of these n alternates, only one is run at a time, and only if that alternate fails will the software rollback and run the next alternate. Hence, if less than the n alternates are actually used, the reliability (or safety or availability) of the RB will generally decrease. Furthermore, this decrease in the RB's transfer function (reliability, safety, or availability value) will cause a decrease in the overall software reliability value (or safety or availability value), calculated with the software reliability model.

5.4.20. Safety

Safety is concerned with the state in which the function block fails (an error is detected), but the function block is error free. Although a function block

is considered to be unsafe when the system is unreliable, safety also covers this extra state. Hence,

safety = [(the probability that an error exists and it is detected) + (the probability that no error exists)]

or

safety = [1 - (the probability that an error exists and it is not detected)]

while

reliability = [(the probability that an error exists and it is detected) + (the probability that no error exists and no error is detected)]

or

reliability = {1 - [(the probability that an error exists, but the error is not detected) + (the probability that no error exists, but an error is detected)]}

Therefore, safety ≥ reliability.

The software reliability model can also be used to determine the safety of the system. Replacing the reliability values with safety values in each function block yields the safety of the overall system.

5.4.21. Availability

As with reliability and safety, availability can be determined through the use of the software reliability model. To do so, the availability values should be used for each function block in the determination of the overall transfer function. By using availability values instead of reliability values, the resultant value will be the availability of the overall system.

The availability values are determined as follows:

availability = [The probability that an error exists, it is detected, and it is corrected) + (the probability that no error exists and no error is detected)]

or

availability = {1 - [(the probability that an error exists, it is detected, but it is not corrected) + (the probability that an error exists and no error is detected) + (the probability that no error exists and an error is detected)]}

Therefore, $0 \leq$ availability ≤ 1.0. By comparison with reliability and safety, availability $\leq$ reliability $\leq$ safety.

5.4.22. Response to Undesired Events

The software reliability model described herein assumes independence between the function blocks. This model neglects the existence of:

- Multiple faults which produce dissimilar outputs but are induced by the same input conditions, or
- Related software design faults causing identical incorrect outputs.

The errors that are manifested by these faults are known as coincident errors and cause a degradation in the reliability (or safety or availability). Therefore, to improve the accuracy of the software reliability model, the coincident errors must be considered. This might be done with an analysis similar to that suggested by Eckhardt and Lee (1985). The analysis makes the assumptions that (1) the input series $X_1, X_2, \ldots,$ is stationary and independent and (2) the versions of software components are designed independently (Eckhardt & Lee, 1985).

When evaluating the probability of coincident errors, the area of concern is the NVS. This analysis is interested in the probability that z or more of the functions fail at the same time, with $z = (n/2)$ if n is even and $z = [(n + 1)/2]$ if n is odd. The following analysis will give a conservative estimate (maximum possible) of the probability of coincident errors for the NVS. This value might be subtracted from the transfer function of the NVS block to produce a conservative value (minimum) of the reliability (or safety or availability) of the NVS and consequently a conservative estimate (minimum) of the overall software reliability value (or safety or availability value).

The following equation gives the maximum probability of coincident errors (E).

$$E = \binom{n}{z}^{*} (1-G_L)^{z} + \binom{n}{z+1}^{*} (1-G_L)^{z+1} +$$

$$\binom{n}{z+2}^{*} (1-G_L)^{z+2} + \ldots + \binom{n}{n}^{*} \binom{n}{1-G_L}^{n}$$

with n = the number of versions in the NVS:

G_i = the reliability (or safety or availability) value for versions i;

i = 1, 2,...n;

G_L = the largest reliability (or safety or availability) value among the group of r versions being evaluated;

z = (n/2) if n is an even number;

z = $[(n + 1)/2]$ if n is an odd number;

$\binom{n}{r}$ = the number of r combinations of an n element set;

$\binom{n}{r}^*$ = the actual r combinations of $(1 - G_i)$ values for the different versions in an n element set of versions.

5.4.23. Assumptions

The assumptions that are frequently made with the various software methods are described below, along with the reasons for such assumptions. These assumptions are grouped logically below the corresponding software type to benefit the reader. In the development of this software reliability model, it is assumed that the software of the function blocks will have complete probabilistic independence. However, paragraph 5.4.21 tries to compensate for the deficiencies resulting from this basic assumption.

Single-version Software

- Errors are not always corrected when detected and new errors may be spawned when correcting errors.
- The time to remove a failure is considered to be negligible and is ignored.
- Inputs which exercise the program are randomly selected.
- The failure rate at any time is proportional to the current number of errors remaining in the program (Prater, Hitt and Eldredge, 1986).

N-version Software

- To benefit from increased reliability, N-version assumes that the probability of a common fault among the versions is extremely low.
- When a fault is determined, the damage incurred is limited to the encapsulation of the individual software version and the overall function that the versions are performing.

Decision Algorithm

- For a majority vote, it is assumed that damage will be limited to the versions in the minority when the decision algorithm is invoked.
- It is possible for a majority vote to yield an incorrect result if a majority of the inputs are incorrect.

Recovery Block

- Faults will manifest themselves within a recovery region.
- The alternate versions of software components are independent; correlated faults are either eliminated or reduced to an acceptably low level.
- The n alternate blocks are independent from the acceptance tests (Hitt, Webb, and Bridgeman, 1984).

Acceptance Test

- The acceptance test will recognize the faults.

5.5. Scott et al.'s Model

5.5.1. Introduction

A second approach to fault-tolerant modeling is that of Scott et al., (1983, 1984, 1987) who developed "data domain" reliability models for the N-version method, RB method, and for an integrated approach called the Consensus RB Method. The models were based on the assumption that "modules with the same functional behavior produce identical correct answers" and that "modules with distinct error behavior produce errors that are distinct from each other as well as distinct from the correct answer." They also assumed that "considerable care would be taken to create the multiple versions using different programmers working independently. The goal of this independent development would be to produce modules with certain independent execution characteristics. For example, it is desirable that the modules have independent error probabilities, that is, the probability that an error is detected in both module A and module B at the same time is the product of the individual module error probabilities. It usually is not desirable, however, that the modules produce distinct correct answers." (Scott et al., 1984, p. 21-1.)

The data domain reliability models for all three fault-tolerant approaches are presented in paragraphs 5.5.3, 5.5.5, and 5.5.7.

5.5.2. Recovery Block (from Scott et al., 1984)

In the RB scheme the problem is to detect a software fault in a program, recover the machine state at the time the faulty program was entered, and execute another program that performs the same function as the faulty one. A number of independently designed programs that perform the same function must be developed. The versions are ranked. When the given task is requested, only the "best" program is executed. If a fault is detected in this program, then the "next best" one is executed, etc., until an acceptable output is obtained. If all versions are deemed faulty, an error is posted.

5.5.3. The Recovery Block Reliability Model

In the RB Reliability Model there are four distinct types of errors that can occur in a RB system. A Type 1 error occurs when any program version produces an incorrect result, but the acceptance test labels the result as correct. In a Type 2 error, the final version produces correct results, but the acceptance test erroneously determines that the results are incorrect. A Type 3 error occurs when the recovery program cannot successfully recover the input state of the previous version in preparation for executing another version. Finally, a Type 4 error occurs when the last version produces incorrect results and the acceptance test judges that the results are incorrect.

These error types are roughly equivalent to the function block states identified by Hitt, E. F., J. J. Webb, and M. S. Bridgeman in the previous section of this report. A Type 1 error [$P(A_I)$ below] is equivalent to E1, a Type 2 error [$P(R_I)$ below] to E2, a Type 3 error [$P(R_C)$ below] to E3, and a Type 4 error [$P(A_C)$ below] to E4.

The reliability model presented is developed using the probabilities of certain RB events. These events and their probabilities are:

$P(C_i)$ = PROBABILITY OF VERSION i EXECUTING CORRECTLY

$P(I_i) = 1 - P(C_i)$

$P(C_R)$ = PROBABILITY OF THE RECOVERY PROGRAM EXECUTING CORRECTLY

$P(I_R) = 1 - P(C_R)$

$P(A_I)$ = PROBABILITY OF ACCEPTING AN INCORRECT RESULT

$P(R_I)$ = PROBABILITY OF REJECTING AN INCORRECT RESULT

$= 1 - P(A_I)$

$P(R_c)$ = PROBABILITY OF REJECTING A CORRECT RESULT

$P(A_c)$ = PROBABILITY OF ACCEPTING A CORRECT RESULT

$= 1 - P(R_c)$

$P(A_B)$ = PROBABILITY OF THE JOINT EVENT A AND B

$P_{RB}(E_K,N)$ = PROBABILITY OF A TYPE K ERROR GIVEN N-VERSIONS

The probability of each of the four types of errors in a 3-version RB with perfect recovery can be found. Assuming version and acceptance test independence, the probabilities are:

$$P_{RB}(E_1,3) = P(I_1)P(A_I) + P(C_1)P(R_C)P(I_2)P(A_I) + P(I_1)P(R_I)P(I_2)P(A_I) + P(C_1)P(R_C)P(C_2)P(R_C)P(I_3)P(A_I) + P(C_1)P(R_C)P(I_2)P(R_I)P(I_3)P(A_I) + P(I_1)P(R_I)P(C_2)P(R_C)P(I_3)P(A_I) + P(I_1)P(R_I)P(I_2)P(R_I)P(I_3)P(A_I)$$

$$P_{RB}(E_2,3) = P(C_1)P(R_C)P(C_2)P(R_C)P(C_3)P(R_C) + P(I_1)P(R_I)P(C_2)P(R_C)P(C_3)P(R_C) + P(C_1)P(R_C)P(I_2)P(R_I)P(C_3)P(R_C) + P(I_1)P(R_I)P(I_2)P(R_I)P(C_3)P(R_C)$$

$$P_{RB}(E_3,3) = 0$$

$$P_{RP}(E_4,3) = P(C_1)P(R_C)P(C_2)P(R_C)P(I_3)P(R_I) + P(I_1)P(R_I)P(C_2)P(R_C)P(I_3)P(R_I) + P(C_1)P(R_C)P(I_2)P(R_I)P(I_3)P(R_I) + P(I_1)P(R_I)P(I_2)P(R_I)P(I_3)P(R_I)$$

If there is dependence among the three program versions, but no dependence between the versions and the acceptance test, the RB model becomes:

$$P_{RB}(E_1,3) = P(I_1)P(A_I) + P(C_1I_2)P(R_C)P(A_I) + P(I_1I_2)P(R_I)P(A_I) + P(C_1C_2C_3)P(R_C)^2P(A_I) + P(C_1I_2I_3)P(R_C)P(R_I)P(A_I) + P(I_1C_2I_3)P(R_I)P(R_C)P(A_I) + P(I_1I_2I_3)P(R_I)^2P(A_I)$$

$$P_{RB}(E_2,3) = P(C_1C_2C_3)P(R_C)^3 + P(C_1I_2C_3)P(R_C)^2P(R_I) + P(I_1C_2C_3)P(R_C)^2P(R_I) + P(I_1I_2C_3)P(R_I)^2P(R_C)$$

$$P_{RB}(E_3,3) = 0$$

$$P_{RB}(E_4,3) = P(I_1I_2I_3)P(R_I)^3 + P(I_1C_2I_3)P(R_I)^2P(R_C) + P(C_1I_2I_3)P(R_I)^2P(R_C) + P(C_1C_2I_3)P(R_C)^2P(R_I)$$

The probability of getting any error is:

$$P_{RB}(E,3) = P_{RB}(E_1,3) + P_{RB}(E_2,3) + P_{RB}(E_3,3) + P_{RB}(E_4,3)$$

And of course, system reliability is:

$$P_{RB} = 1 - P_{RB}(E,n)$$

5.5.4. N-version Programming

Another prevalent method of software fault-tolerance is the N-version Programming method. In this method, n programmers ($n > 1$) are asked to independently develop and debug a program, each working from a common specification. The n programs are placed under the control of a supervisor program that dispatches all versions of the program with the necessary input when the specified task is required. Upon completion, the outputs of the n programs are compared. Since all versions of the program are independently developed, it is assumed that the

probability of a common software error is zero. Therefore, if at least two program versions agree on an output, that output is assumed to be "correct."

5.5.5. The N-version Programming Reliability Model

In the N-version Programming Reliability Model, there are three types of errors that can occur in an N-version Programming system. A Type 1 error occurs when all outputs disagree. A Type 2 error occurs when a particular incorrect output occurs more than once. Finally, a Type 3 error occurs when there is an error in the voting procedure. The probability of a system error is then the sum of the three probabilities of a Type 1, 2, or 3 error.

The probability of a Type 1 error in a 2 of 3 N-version Programming system composed of independent program versions is:

$$P_{NVP}(E_1,3) = P(I_1)P(I_2)P(I_3) + P(C_1)P(I_2)P(I_3) + P(I_1)P(C_2)P(I_3) + P(I_1)P(I_2)P(C_3)$$

$$= P(I_1)P(I_2) + P(I_1)P(I_3) + P(I_2)P(I_3) - 2P(I_1)P(I_2)P(I_3)$$

If the program versions are not independent, then probability of a Type 1 error becomes:

$$P_{NVP}(E_1,3) = P(I_1I_2I_3)P(I_2I_3)P(I_2I_3)P(I_3) + P(C_1I_2I_3)P(I_2I_3)P(I_3) + P(I_1C_2I_3)P(C_2I_3)P(I_3) + P(I_1I_2C_3)P(I_2C_3)P(C_3)$$

$$= P(I_1I_2I_3)P(I_2I_3)P(I_3) + [1 - P(I_1I_2I_3)]P(I_2I_3)P(I_3) + P(I_1C_2I_3)[1 - P(I_2I_3)]P(I_3) + P(I_1I_2C_3)P(I_2C_3)[1 - P(I_3)]$$

If the versions are dependent, the probability of a Type 2 error, the multiple occurrence of an incorrect answer, cannot be assumed to be zero. It is still reasonable that a Type 3 error is zero. Therefore, the probability of an error in a dependent N-version Programming system is:

$$P_{NVP}(E,n) = P_{NVP}(E_1,n) + P_{NVP}(E_2,n)$$

Therefore, the reliability of a 2 of 3 N-version Programming system is:

$$R_{NVP} = 1 - P_{NVP}(E,3)$$

5.5.6. Consensus Recovery Block (from Scott et al., 1984)

The Consensus RB is a software fault-tolerant technique that combines aspects of RBs and N-version Programming. This hybrid technique attempts to overcome some serious problems in the two established fault-tolerant techniques.

The Consensus RB requires the development of n independent versions of a program, an acceptance test, and a voting procedure. The versions are ranked as in a conventional RB system. Upon invocation of the Consensus RB, all versions execute and submit their outputs to a voting procedure. Since the voting procedure assumed that there are no common faults, if two or more of the versions agree on one output, that output is designated as correct. If there

is no agreement, that is, the versions supply incorrect outputs or multiple correct outputs, then a modified RB is entered. The output of the "best" version is examined by the acceptance test. If that output is judged acceptable, it is treated as a correct output and system execution continues. If, on the other hand, the output is not accepted, the next best version's output is subjected to the acceptance test. This process continues until an acceptable output is found, or the n outputs are exhausted. Notice that there is no requirement for input state recovery since all versions execute in a parallel fashion as in an N-version Programming system, not in a serial fashion as in a conventional RB.

5.5.7. The Consensus Recovery Block Reliability Model (from Scott et al., 1984)

The probabilities of error in the Consensus RB Reliability Model are the same as those in the RB model multiplied by P(D), the probability of output disagreement. In addition, a new error, Type 5 Error, can occur if there is agreement on an incorrect output. The Type 5 Error in a Consensus RB is analogous to a Type 2 error in an N-version Programming system. Therefore, in a 3-version independent Consensus RB, the probabilities of error are:

$$P_{CRB}(E_1,3) = P(D)[P(I_1)P(A_1) + P(C_1)P(R_C)P(I_2)P(A_1) + P(I_1)P(R_I)P(I_2)P(A_I) + P(C_1)P(R_C)P(C_2)P(R_C)P(I_3)P(A_I) + P(C_1)P(R_C)P(I_2)P(R_I)P(I_3)P(A_I) + P(I_1)P(R_I)P(C_2)P(R_C)P(I_3)P(A_I) + P(I_1)P(R_I)P(I_2)P(R_I)P(I_3)P(A_I)]$$

$$P_{CRB}(E_2,3) = P(D)\ [P(C_1)P(R_C)P(C_2)P(R_C)P(C_3)P(R_C) + P(I_1)P(R_I)P(C_2)P(R_C)P(C_3)P(R_C) + P(C_1)P(R_C)P(I_2)P(R_I)P(C_3)P(R_C) + P(I_1)P(R_I)P(I_2)P(R_I)P(C_3)P(R_C)]$$

$$P_{CRB}(E_3,3) = 0$$

$$P_{CRB}(E_4,3) = P(D)\ [P(C_1)P(R_C)P(C_2)P(R_C)P(I_3)P(R_I) + P(I_1)P(R_I)P(C_2)P(R_C)P(I_3)P(R_I) + P(C_1)P(R_C)P(I_2)P(R_I)P(I_3)P(R_I) + P(I_1)P(R_I)P(I_2)P(R_I)P(I_2)P(R_I)P(I_3)P(R_I)]$$

$$P_{CRB}(E_5,3) = P(\text{AGREEMENT ON INCORRECT OUTPUT})$$

If there is dependence among the three program versions, but no dependence between the versions and the acceptance test, the Consensus RB model becomes:

$$P_{CRB}(E_1,3) = P(DI_1)P(A_I) + P(DC_1I_2)P(R_C)P(A_I) + P(DI_1I_2)P(R_I)P(A_I) + P(DC_1C_2C_3)P(R_C)^2P(A_I) + P(DC_1I_2I_3)P(R_C)P(R_I)P(A_I) + P(DI_1C_2I_3)P(R_I)P(R_C)P(A_I) + P(DI_1I_2I_3)P(R_I)^2P(A_I)$$

$$P_{CRB}(E_2,3) = P(DC_1C_2C_3)P(R_C)^3 + P(DC_1I_2C_3)P(R_C)^2P(R_I) + P(DI_1C_2C_3)P(R_C)^2P(R_I) + P(DI_1I_2C_3)P(R_I)^2P(R_C)$$

$$P_{CRB}(E_3,3) = 0$$

$$P_{CRB}(E_4,3) = P(DI_1I_2I_3)P(R_I)^3 + P(DI_1C_2I_3)P(R_I)^2P(R_C) + P(DC_1I_2I_3)P(R_I)^2P(R_C) + P(DC_1C_2I_3)P(R_C)^2P(R_I)$$

The probability of getting any error is:

$$P_{CRB}(E,3) = P_{CRB}(E_1,3) + P_{CRB}(E_2,3) + P_{CRB}(E_3,3) + P_{CRB}(E_4,3) + P_{CRB}(E_5,3)$$

And of course, system reliability is:

$$R_{CRB} = 1 - P_{CRB}(E,n)$$

5.5.8. Verification

Scott et al., 1984 and 1987, have presented data from a number of experiments conducted in a university environment. The purpose of these experiments was to validate the ability of the three proposed fault-tolerant models to accurately predict system reliability. The data from these experiments, appear to indicate that the RB method and the Consensus RB method can provide the required system reliability, and that the equations developed to represent each of the models can provide the reliability estimates, which in turn can accurately predict system reliability.

APPENDIX A - ASSUMPTIONS UNIQUE TO FAULT COUNT SOFTWARE RELIABILITY MODELS

This appendix provides a list of the underlying assumptions that are unique to each of the Fault Count Software Reliability Models referenced in section 4 of this report.

- ASSUMPTIONS UNDERLYING THE GOEL-OKUMOTO NON-HOMOGENEOUS POISSON PROCESS MODEL

  - Expected number of software faults to be eventually detected is an unknown fixed quantity.
  - Actual number of faults to be observed is a random variable.
  - Each failure is caused by one fault, and each fault is equally likely to cause a failure during testing.
  - Number of software faults detected during each non-overlapping testing interval is independent of each other.
  - Expected number of software faults detected during $(t, t + \Delta t)$ is proportional to the expected number of software faults undetected by time t.
  - Fault removal time is negligible.
  - No new faults are introduced during the fault removal process.

- ASSUMPTIONS UNDERLYING THE GEOMETRIC POISSON MODEL

  - There is an infinite number of total errors.
  - Each fault in the program is independent of the others and each of them is equally likely to occur.
  - The errors do not have the same likelihood of detection.
  - Data are available only at discrete intervals.
  - During a fixed interval of time, the number of errors detected follows a Poisson distribution.
  - During each of these periods of time, the detection rate is constant.

- Each error discovered is immediately removed or no longer counted.
- No new fault is introduced during the correction process.

- ASSUMPTIONS UNDERLYING THE EXTENDED JELINSKI-MORANDA MODEL
  - There is a fixed number of errors in the program.
  - Each error has an equal chance of being detected.
  - The number of machine language instructions remains relatively constant.
  - The failure rate is proportional to the current error content (number of remaining errors).
  - There is a constant failure rate between consecutive errors.
  - More than one error may occur in a given debugging time period.
  - The program is not being altered except for error correction.
  - No new errors are added during the debugging process.
  - Each error has an equal chance of being detected.

- ASSUMPTIONS UNDERLYING THE MODIFIED GEOMETRIC DE-EUTROPHICATION MODEL
  - The program contains an unknown number of errors.
  - Each fault in the program is independent, and each fault is equally likely to cause a failure during testing.
  - The number of faults detected in any time interval is independent of the number of faults detected in any other time interval.
  - The program is not being altered except for error correction.
  - The error correction time is negligible. Each error discovered is immediately removed.
  - No new errors are added during the debugging process.

- ASSUMPTIONS UNDERLYING THE SHOOMAN EXPONENTIAL MODEL

  - The initial number of errors in a program is a constant.
  - The number of errors remaining in the program decreases as errors are corrected.
  - Each error has an equal chance of being detected.
  - Software errors occur with a probability distribution of

$$f(t) = \lambda \exp(-\lambda t)$$

  - The error detection rate (failure rate) is proportional to the number of remaining errors.
  - The total number of machine language instructions remains constant.
  - Operational software errors occur due to the occasional traversing of a portion of the program in which a software bug is hidden.

APPENDIX B - ASSUMPTIONS UNIQUE TO TIME-BETWEEN-FAILURE SOFTWARE RELIABILITY MODELS

This appendix provides a list of the underlying assumptions that are unique to each of the Time-Between-Failure Software Reliability Models referenced in section 4 of this report.

- ASSUMPTIONS UNDERLYING THE JELINSKI AND MORANDA DE-EUTROPHICATION MODEL

  - Initial fault content is an unknown fixed constant n.
  - Each fault in the program is independent of other faults, and each fault is equally likely to cause a failure during testing.
  - Times between occurrences of faults are independent of each other.
  - The software failure rate of the hazard function during a failure interval is constant and is proportional to the current fault content of the tested program.
  - A detected fault is removed with certainty at the end of each testing interval.
  - Only one fault is removed during each testing interval.
  - The fault removal time is negligible.
  - No new faults are introduced during the fault removal process.

- ASSUMPTIONS UNDERLYING THE GEOMETRIC DE-EUTROPHICATION MODEL

  - There is an infinite number of total errors.
  - Each fault in the program is independent of the others and each of them is equally likely to occur.
  - The errors do not have the same likelihood of detection.
  - The failure rate between successive errors forms a geometric progression and is constant in the interval between errors.
  - Each error discovered is immediately removed. The time to correct the detected faults is negligible.

- No new fault is introduced during the correction time.

- ASSUMPTIONS UNDERLYING THE GENERALIZED IMPERFECT DEBUGGING MODEL

  - All failures are observable and independent.

  - Testing is of uniform intensity and representative of the operational environment.

  - Inputs which exercise the program are randomly selected.

  - The failure rate at any time is proportional to the current number of errors remaining in the program.

  - The failure rate between the (i-1)th failure and the ith failure is $\lambda(t_i) = \phi[N-(i-1)]t^{\alpha-1}$.

  - The time to remove a failure is considered to be negligible and is ignored in the model.

  - Errors are not always corrected when detected, and errors may be spawned when correcting errors.

- ASSUMPTIONS UNDERLYING THE BUG-PROPORTIONAL MODEL

  - The initial number of errors in a program is a constant.

  - The number of errors remaining in the program decreases as errors are corrected.

  - Software errors are caused by the uncovering of residual bugs in a program.

  - The probability that a bug is encountered in the time interval, Δt, after t successful hours of operation is proportional to the fractional number of remaining bugs.

  - The fractional number of remaining bugs is independent of the operating time.

  - The rate of error correction is constant.

APPENDIX C - WORKED EXAMPLE FOR THE GOEL-OKUMOTO ON-HOMOGENEOUS POISSON PROCESS MODEL

Worked example showing computations for the Goel-Okumoto NHPP Model (Fault Count). This example has been adapted from Goel (1983).

Consider the following failure counts.

Interval No.	Cumulative No. of Failures
1	2
2	4
3	5

The parameters to be estimated are a and b, where a is the expected number of software faults to be eventually detected; and b is a proportionality constant.

The MLE of a and b can be obtained by solving the following pair of equations.

$$a(1-e^{-bt_n}) = y_n$$

$$at_n e^{-bt_n} = \sum_{i=1}^{n} \frac{(y_i - y_{i-1})(t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}})}{e^{-bt_{i-1}} - e^{-bt_i}},$$

These equations yield

$$\frac{y_n t_n e^{-bt_n}}{(1-e^{-bt_n})} = \sum_{i=1}^{n} \frac{(y_i - y_{i-1})(t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}})}{e^{-bt_{i-1}} - e^{-bt_i}}$$

which can be used to obtain the MLE of b by the Newton-Raphson method.

Using the Newton-Raphson method to find $\hat{b}$:

$$\text{Let } F \equiv \sum_{i=1}^{n} \frac{(y_i - y_{i-1})(t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}})}{e^{-bt_{i-1}} - e^{-bt_i}} - \frac{y_n t_n e^{-bt_n}}{(1 - e^{-bt_n})}$$

then

$$\frac{dF}{db} = \sum_{i=1}^{n} (y_i - y_{i-1}) \left[\frac{(e^{-bt_{i-1}} - e^{-bt_i})(t_{i-1}^2 e^{-bt_{i-1}} - t_i^2 e^{-bt_i})}{(e^{-bt_{i-1}} - e^{-bt_i})^2} \right.$$

$$\left. - \frac{(t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}})^2}{(e^{-bt_{i-1}} - e^{-bt_i})^2} \right] + \frac{y_n t_n^2 e^{-bt_n}}{(1 - e^{-bt_n})^2}$$

We can then compute the MLE of a as

$$a = \frac{y_n}{(1 - e^{-bt_n})}$$

For the above data set

$$n = 3,\ t_0 = 0,\ t_1 = 1,\ t_2 = 2,\ t_3 = 3$$

$$y_0 = 0,\ y_1 = 2,\ y_2 = 4,\ y_3 = 5$$

The computation of MLE to Estimate a and b is as follows:

<u>Iteration 1.</u> Let an initial value of b = .01; i.e., $b^{(1)} = 0.01$

Then

$$F = \sum_{i=1}^{3} \frac{(y_i - y_{i-1})(t_i e^{-.01t_i} - t_{i-1} e^{-.01t_{i-1}})}{e^{-.01t_{i-1}} - e^{-.01t_i}} - \frac{y_3 t_3 e^{-.01t_3}}{(1 - e^{-.01t_3})}$$

$$= \frac{y_1 (t_1 e^{-.01t_1})}{1 - e^{-.01t_1}} + \frac{(y_2 - y_1)(t_2 e^{-.01t_2} - t_1 e^{-.01t_1})}{e^{-.01t_1} - e^{-.01t_2}}$$

$$+ \frac{(y_3 - y_2)(t_3 e^{-.01t_3} - t_2 e^{-.01t_2})}{e^{-.01t_2} - e^{-.01t_3}} - \frac{y_3 t_3 e^{-.01t_3}}{(1 - e^{-.01t_3})}$$

$$= \frac{2 (e^{-.01})}{1 - e^{-.01}} + \frac{(4 - 2)(2e^{-.02} - e^{-.01})}{e^{-.01} - e^{-.02}} + \frac{(5 - 4)(3e^{-.03} - 2e^{-.02})}{e^{-.02} - e^{-.03}}$$

$$- \frac{5 \cdot 3 \cdot e^{-.03}}{(1 - e^{-.03})}$$

$$= 493.5041667 - 492.5374994$$

$$= .966667222210$$

Now,

$$\frac{dF}{db} = \sum_{i=1}^{3} (y_i - y_{i-1}) \left[\frac{(e^{-.01t_{i-1}} - e^{-.01t_i})(t_{i-1}^2 e^{-.01t_{i-1}} - t_i^2 e^{-.01t_i})}{(e^{-.01t_{i-1}} - e^{-.01t_i})^2} \right.$$

$$\left. - \frac{(t_i e^{-.01t_i} - t_{i-1} e^{-.01t_{i-1}})^2}{(e^{-.01t_{i-1}} - e^{-.01t_i})^2} \right] + \frac{y_3 t_3^2 e^{-.01t_3}}{(1 - e^{-.01t_3})^2}$$

$$= \frac{y_1[(e^{-0.1x0} - e^{-.01})(-e^{-.01}) - e^{-.01x2}]}{(e^{-.01x0} - e^{-.01})^2}$$

$$+ \frac{(y_3 - y_2)[(e^{-.02} - e^{.03})(4e^{-.02} - 9e^{-.03}) - (3e^{-.03} - 2e^{-.02})^2]}{(e^{-.02} - e^{-.03})^2}$$

$$+ \frac{5 \cdot 9 \cdot e^{-.03}}{(1 - e^{-.03})^2}$$

$$+ \frac{(y_2 - y_1)[(e^{-.01} - e^{-.02})(e^{-.01} - 4e^{-.02}) - (2e^{-.02} - e^{-.01})^2]}{(e^{-.01} - e^{-.02})^2}$$

$$= \frac{2[(1 - e^{-.01})(-e^{-.01}) - e^{-.02}]}{(1 - e^{-.01})^2}$$

$$+ \frac{2[(e^{-.01} - e^{-.02})(-e^{-.01} - 4e^{-.02}) - (2e^{-.02} - e^{-.01})^2]}{(e^{-.01} - e^{-.02})^2}$$

$$+ \frac{1[(e^{-.02} - e^{-.03})(4e^{-.02} - 9e^{-.03}) - (3e^{-.03} - 2e^{-.02})^2]}{(e^{-.02} - e^{-.03})^2}$$

$$+ \frac{45 \cdot e^{-.03}}{(1 - e^{-.03})^2}$$

$$= -\left(\frac{2e^{-.01}}{(1 - e^{-.03})^2} + \frac{2e^{-.03}}{(e^{-.01} - e^{-.02})^2} + \frac{e^{-.05}}{(e^{-.02} - e^{-.03})^2}\right) + \frac{45e^{-.03}}{(1 - e^{-.03})^2}$$

$$= -49999.58334 + 49996.25017$$

$$= -3.33316667270$$

and

$$\Delta b^{(1)} = -F \div dF/db = -.966667222210 \div -3.33316667270$$
$$= .2900146669 > .0001$$

Let $\epsilon = 0.0001$. Since $\Delta b^{(1)} > \epsilon$, we go to iteration 2.

Iteration 2.

$$b^{(2)} = b^{(1)} + \Delta b^{(1)} = .3000146669 \simeq .3$$

$$F = \sum_{i=1}^{3} \frac{(y_i - y_{i-1})(t_i e^{-.3t_i} - t_{i-1} e^{-.3t_{i-1}})}{e^{-.3t_{i-1}} e^{-.3t_i}} - \frac{y_3 t_3 e^{-.3t_3}}{(1 - e^{-.3t_3})}$$

$$= \frac{2(e^{-.3})}{1-e^{-.3}} + \frac{2(2e^{-.6} - e^{-.3})}{e^{-.3} - e^{-.6}} + \frac{1(3e^{-.9} - 2e^{-.6})}{(e^{-.6} - e^{-.9})} - \frac{5 \cdot 3e^{-.9}}{(1 - e^{-.9})}$$

$$= 10.2906087 - 10.27600431$$
$$= .0146665525129$$

Now,

$$\frac{dF}{db} = \sum_{i=1}^{3} (y_i - y_{i-1}) \left[\frac{(e^{-.3t_{i-1}} - e^{-.3t_i})(t_{i-1}^2 e^{-.3t_{i-1}} - t_i^2 e^{-.3_i})}{(e^{-.3t_{i-1}} - e^{-.3t_i})^2} \right.$$

$$\left. - \frac{(t_i e^{-.3t_i} - t_{i-1} e^{-.3t_{i-1}})^2}{(e^{-.3t_{i-1}} - e^{-.3t_i})^2} \right] + \frac{y_3 t_3^2 e^{-.3t_3}}{(1 - e^{-.3t_3})^2}$$

$$= \frac{2[(1 - e^{-.3})(-e^{-.3}) - e^{-.6}]}{(1 - e^{-.3})^2}$$

$$+ \frac{2[(e^{-.3} - e^{-.6})(e^{-.3} - 4e^{-.6}) - (2e^{-.6} - e^{-.3})^2}{(e^{-.3} - e^{-.6})^2}$$

$$+ \frac{1[(e^{-.6} - e^{-.9})(4e^{-.6} - 9e^{-.9}) - (3e^{-.9} - 2e^{-.6})^2]}{(e^{-.6} - e^{-.9})^2} + \frac{45e^{-.9}}{(1 - e^{-.9})^2}$$

$$= -\left(\frac{2e^{-.3}}{(1 - e^{-.3})^2} + \frac{2e^{-.9}}{(e^{-.3} - e^{-.6})^2} + \frac{e^{-1.5}}{(e^{-.6} - e^{-.9})^2} \right) + \frac{45e^{-.9}}{(1 - e^{-.9})^2}$$

$$= -55.13532562 + 51.94726586 = -3.18805975584$$

$$\Delta b^{(2)} = -F \div dF/db = -.0146665525129 \div -3.18805975584$$
$$= .004600463491$$

Since $\Delta b^{(2)} > \epsilon$, we continue the computations.

Iteration 3.

$$b^{(3)} = b^{(2)} + \Delta b^{(2)} = .3000146669 + .004600463491$$
$$= .30461513036 \approx .305$$

$$F = \sum_{i=1}^{3} \frac{(y_i - y_{i-1})(t_i e^{-.305t_i} - t_{i-1} e^{-.305t_{i-1}})}{e^{-.305t_{i-1}} - e^{-.305t_i}} - \frac{y_3 t_3 e^{-.305t_3}}{1 - e^{-.305t_3}}$$

$$= \frac{2(e^{-.305})}{1 - e^{-.305}} + \frac{2(2e^{-.610} - e^{-.305})}{e^{-.305} - e^{-.610}} + \frac{1(3e^{-.915} - 2e^{-.610})}{e^{-.610} - e^{-.915}}$$

$$- \frac{5 \cdot 3 \cdot e^{-.915}}{(1 - e^{-.915})}$$

$$= 10.04088223 - 10.04087226 = 9.96719302759 \times 10^{-6}$$

$$\frac{dF}{db} = \sum_{i=1}^{3} (y_i - y_{i-1}) \left[\frac{(e^{-.305t_i - 1} - e^{-.305t_i})(t_{i-1}^2 e^{-.305t_{i-1}} - t_i^2 e^{-.305t_i})}{(e^{-.305t_{i-1}} - e^{-.305t_i})^2} \right.$$

$$\left. - \frac{(t_i e^{-.305t_i} - t_{i-1} e^{-.305t_{i-1}})^2}{(e^{-.305t_{i-1}} - e^{-.305t_i})^2} \right] + \frac{y_3 t_3^2 e^{-.305t_3}}{(1 - e^{-.305t_3})^2}$$

$$= \frac{2[(1 - e^{-.305})(-e^{-.305}) - e^{-.610}]}{(1 - e^{-.305})^2} +$$

$$+ \frac{2[(e^{-.305} - e^{-.610})(e^{-.305} - 4e^{-.610}) - (2e^{-.610} - e^{-.305})^2]}{(e^{-.305} - e^{-.610})^2}$$

$$+ \frac{1[(e^{-.610} - e^{-.915})(4e^{-.610} - 9e^{-.915}) - (3e^{-.915} - 2e^{-.610})^2]}{(e^{-.610} - e^{-.915})^2}$$

$$+ \frac{45 \cdot e^{-.915}}{(1 - e^{-.915})^2}$$

$$= -\left(\frac{2e^{-.305}}{(1 - e^{-.305})^2} + \frac{2e^{-.915}}{(e^{-305} - e^{-.610})^2} + \frac{e^{-1.525}}{(e^{-.610} - e^{-.915})^2} \right)$$

$$+ \frac{45e^{-.915}}{(1 - e^{-.915})^2}$$

$$= -53.470157 + 50.28643996 = -3.18371704092$$

$$\Delta b^{(3)} = -F \div dF/db = -9.96719302759 \times 10^{-6} \div -3.18371704092$$

$$= 3.1306781 \times 10^{-6}$$

Since $\Delta b^{(3)} < \epsilon$ (0.0001), we terminate further iterations.

Computation of $\hat{a}$ and $\hat{b}$

Since we met the criteria for $\Delta b^{(3)} < \epsilon$ (0.0001) we do not have to compute additional iterations, and we can compute $\hat{a}$ and $\hat{b}$ as follows:

$$\hat{b} = b^{(3)} + \Delta b^{(3)} = 3.0461826104 \times 10^{-1}$$

and

$$\hat{a} = \frac{y_3}{(1 - e^{-\hat{b}t_3})} = \frac{5}{(1 - e^{-.30461826104 \times 3})} = 8.346957421$$

Thus, for this data set

$$\hat{a} = 8.34695742$$
$$\hat{b} = 3.0461826104 \times 10^{-1}$$

After estimation of the parameters a and b the next step is to obtain performance measures as follows:

The expected number of faults detected by time t, $E[\hat{N}(t)]$, is obtained using the following formula:

$$E[\hat{N}(t)] = \hat{a}(1 - e^{-bt})$$

Let t=10. Substituting values for $\hat{a}$, $\hat{b}$, and t in the above formula we get

$$E[\hat{N}(10)] = 8.34695742 \times (1 - e^{-3.0461826104 \times 10^{-1} \times 10})$$
$$= 7.950397851 \quad .$$

The expected number of remaining faults by time t, $E[\hat{N}(t)]$, is obtained using the following formula:

$$E[\hat{\overline{N}}(t)] = \hat{a}e^{-\hat{b}t} \quad .$$

Let t=10. Substituting values for $\hat{a}$, $\hat{b}$, and t in the above formula we get

$$E[\hat{\overline{N}}(10)] = 8.3469742 \times (e^{-3.0461826104 \times 10^{-1} \times 10})$$
$$= 8.34695742 \times e^{-3.0461826104}$$
$$= 0.3965595689$$

The reliability at time t after nth testing interval, $\hat{R}_n(t)$, is obtained from the following formula:

$$\hat{R}_n(t) = \exp[-\hat{a}\{\exp[-\hat{b}xn] - \exp[\hat{b}x(n+t)]\}]$$

For the above example n=3. Let t be 1. After substituting values $\hat{a}$, $\hat{b}$, n, and t in the above formula we get

$$\hat{R}_3(1) = \exp[-8.34695742 \times \{\exp[-0.30461826104 \times 3]$$
$$-\exp[-0.30461826104 \times (3+1)]\} = 0.4152494843$$

The MTTF after nth testing interval, $\hat{MTTF}_n$, is obtained using the following formula:

$$\hat{MTTF}_n = \frac{1}{\hat{\lambda}(n)} = \frac{1}{\hat{a}\hat{b}\exp[-\hat{b}xn]}$$

For the above example n=3 and substituting values for $\hat{a}$ and $\hat{b}$ in the above formula we get

$$\hat{MTTF}_3 = \frac{1}{8.34695742 \times 0.30461826104 \times \exp[-0.30461826104 \times 3]}$$

$$= .9808216948$$

APPENDIX D - WORKED EXAMPLE SHOWING COMPUTATIONS FOR THE JELINSKI-MORANDA DE-EUTROPHICATION MODEL

Worked example showing computations for the Jelinski-Moranda De-Eutrophication Model (Time Between Failures). This example has been adapted from Goel (1983).

Consider a set of failure times as follows:

Failure No.	Time Between Failure t_i
1	3
2	30
3	113
4	31
5	115

The parameters to be estimated are N and Φ, where N is the total number of initial errors in the program; and Φ is a proportionality constant.

The maximum likelihood estimate (MLE) of N can be obtained from the following equation using the Newton-Raphson method:

$$\sum_{i=1}^{n} \frac{1}{N-(i-1)} - \frac{n}{N - \frac{1}{\sum_{i=1}^{n} t_i}\left(\sum_{i=1}^{n} (i-1)t_i\right)} = 0$$

Therefore, we can use the Newton-Raphson method to find $\hat{N}$ as follows:

$$F = \sum_{i=1}^{n} \frac{1}{N-(i-1)} - \frac{n}{N - \frac{1}{\sum_{i=1}^{n} t_i}\left(\sum_{i=1}^{n} (i-1)t_i\right)}$$

then

$$\frac{dF}{dN} \equiv \frac{n}{\left\{N - \frac{1}{\sum_{i=1}^{n} t_i}\left(\sum_{i=1}^{n} (i-1)\, t_i\right)\right\}^2} - \sum_{i=1}^{n} \frac{1}{\{N-(i-1)\}^2}$$

We can then find $\hat{\Phi}$ from the following equation by substituting $\hat{N}$ for N

$$\hat{\Phi} = \frac{n}{\hat{N}\left(\sum_{i=1}^{n} t_i\right) - \sum_{i=1}^{n} (i-1)t_i}$$

For the above data set

$$\sum_{i=1}^{n} (i-1)t_i = \sum_{i=1}^{5} (i-1)t_i = t_2 + 2t_3 + 3t_4 + 4t_5$$

$$= 30 + 226 + 93 + 460 = 809$$

The computation of MLE to estimate $\hat{N}$ and $\hat{\Phi}$ is as follows:

Iteration 1. Let an initial value of $N^{(1)} = 5$

$$\text{Then } F^{(1)} = \sum_{i=1}^{5} \frac{1}{5-i+1} - \frac{5}{5 - \frac{1}{292}\cdot(809)}$$

$$= \left(\frac{1}{5} + \frac{1}{4} + \frac{1}{3} + \frac{1}{2} + 1\right) - \frac{5}{2.229452055}$$

$$= 2.283333333 - 2.242703532$$

$$= 4.062980 \times 10^{-2}$$

and

$$\frac{dF^{(1)}}{dN^{(1)}} = \frac{5}{\{5 - \frac{1}{292}(809)\}^2} - \sum_{i=1}^{5} \frac{1}{\{5-i+1\}^2}$$

$$= \frac{5}{\{2.229452055\}^2} - \left(\frac{1}{25} + \frac{1}{16} + \frac{1}{9} + \frac{1}{4} + 1\right)$$

$$\text{or } \frac{dF^{(1)}}{dN^{(1)}} = -4.57667284 \times 10^{-1}$$

Next

$$\Delta N^{(1)} = -F^{(1)} \div \frac{dF^{(1)}}{dN^{(1)}} = (4.062980 \times 10^{-2}) \div (4.57667284 \times 10^{-1})$$

$$= 0.08877584602$$

Let error $\epsilon \leq 10^{-4}$. Since $\Delta N^{(1)} \not< 10^{-4}$, we go through the next iteration.

Iteration 2. The value of N for the second iteration is

$$N^{(2)} = N^{(1)} + \Delta N^{(1)} = 5.08877584602$$

$$F^{(2)} = \sum_{i=1}^{5} \frac{1}{5.08877584602 - i + 1} - \frac{5}{5.08877584602 - \frac{809}{292}}$$

$$\text{or } F^{(2)} = 5.22788908141 \times 10^{-3}$$

and

$$\frac{dF^{(2)}}{dN^{(2)}} = \frac{5}{\{5.08877584602 - \frac{809}{292}\}^2} - \sum_{i=1}^{5} \frac{1}{\{5.08877584602 - i + 1\}^2}$$

$$= .9303743904 - 1.276022534 = -3.45648143667 \times 10^{-1}$$

Now

$$\Delta N^{(2)} = -F^{(2)} \div \frac{dF^{(2)}}{dN^{(2)}} = (5.22788908141 \times 10^{-3}) \div (3.45648143667 \times 10^{-1})$$

$$= 0.0151248869$$

Since this number is $> 10^{-4}$, we go to the next iteration.

Iteration 3. The value of N for the third iteration is

$$N^{(3)} = N^{(2)} + \Delta N^{(2)} = 5.08877584602 + 0.0151248869$$

$$= 5.10390073293$$

$$F^{(3)} = \sum_{i=1}^{5} \frac{1}{5.10390073293 - i + 1} - \frac{5}{5.10390073293 - \frac{809}{292}}$$

$$= 2.142960589 - 2.142839277 = 1.21312176929 \times 10^{-4}$$

and

$$\frac{dF^{(3)}}{dN^{(3)}} = \frac{5}{\{5.10390073293 - \frac{809}{292}\}^2} - \sum_{i=1}^{5} \frac{1}{\{5.10390073293 - i + 1\}^2}$$

$$= .9183520332 - 1.248093532 = -3.29741499203 \times 10^{-1}$$

$$\Delta N^{(3)} = -F^{(3)} \div \frac{dF^{(3)}}{dN^{(3)}} = (1.21312176929 \times 10^{-3}) \div (3.29741499203 \times 10^{-1})$$

$$= 3.679008472 \times 10^{-4}$$

Since $\Delta N^{(3)} > 10^{-4}$, we continue the computations.

Iteration 4. The value of N for the iteration is

$$N^{(4)} = N^{(3)} + \Delta N^{(3)} = 5.10390073293 + 3.679008472 \times 10^{-4}$$
$$= 5.10426863377$$

$$F^{(4)} = \sum_{i=1}^{5} \frac{1}{5.10426863377 - i + 1} - \frac{5}{5.10426863377 - \frac{809}{292}}$$

$$= 2.142501537 - .142501468 = 6.93541917229 \times 10^{-8}$$

$$\frac{dF^{(4)}}{dN^{(4)}} = \frac{5}{\{5.10426863377 - \frac{809}{292}\}^2} - \sum_{i=1}^{5} \frac{1}{\{5.10426863377 - i + 1\}^2}$$

$$= 0.9180625077 - 1.247427058 = -3.29364549966 \times 10^{-1}$$

and

$$\Delta N^{(4)} = -F^{(4)} \div \frac{dF^{(4)}}{dN^{(4)}} = (6.9354191722 \times 10^{-8}) \div (3.29364549966 \times 10^{-1})$$

$$= 2.105696916 \times 10^{-7}$$

Since $\Delta N^{(4)} < 10^{-4}$, we terminate further iterations.

Since we met the criteria for $\Delta N^{(4)} \leq 10^{-4}$, we do not have to compute additional iterations, and we can compute $\hat{N}$ and $\hat{\Phi}$ as follows:

$$\hat{N} = N^{(4)} + \Delta N^{(4)}$$
$$= 5.10426863377 + 2.105696916 \times 10^{-7}$$

or

$$\hat{N} = 5.104268844 \ ;$$

Using the above value of $\hat{N}$, we get

$$\hat{\phi} = \frac{5}{5.104268844 \cdot (292) - 809}$$

$$= \frac{5}{1490.446502 - 809} = 5/681.446502$$

or

$$\hat{\phi} = 7.337333131 \times 10^{-3}$$

Thus, for this data set, we get

$$\hat{N} = 5.104268844$$
$$\hat{\phi} = 7.337333131 \times 10^{-3}$$

After estimating the parameters $\hat{N}$ and $\hat{\Phi}$, we calculate the performance measures using these estimates as follows:

The reliability at time t after nth failure, $\hat{R}_n(t)$, is obtained using the following formula:

$$\hat{R}_n(t) = \exp[-\hat{\phi}(\hat{N} - n)t]$$

Let t=10. Substituting values for $\hat{N}$, $\hat{\Phi}$, n, and t in the above formula we get

$$\hat{R}_5(10) = \exp[-7.337333131 \times 10^{-3} \times (5.104268844 - 5) \times 10]$$

$$= .9923786386$$

The MTTF after nth failure, $\hat{MTTF}_n$, is obtained using the following formula:

$$\hat{MTTF}_n = \frac{1}{\hat{\Phi}(\hat{N} - n)}$$

Substituting values for $\hat{N}$, $\hat{\Phi}$, and n in the above formula we get

$$\hat{MTTF}_5 = \frac{1}{7.33733131 \times 10^{-3} \times (5.104268844 - 5)}$$

$$= 1307.095152$$

APPENDIX E - WORKED EXAMPLES FOR THE N-VERSION SOFTWARE RELIABILITY DESIGN TECHNIQUES

Appendix E contains three worked examples using NVS reliability design techniques.

Example 1: Analysis of a system with NVS whose outputs go into a decision algorithm.

Example 2: Analysis of a system with NVS whose outputs are submitted to an acceptance test, and the outputs from the acceptance test are input to the decision algorithm.

Example 3: Analysis of a hybrid NVS which makes use of acceptance test, decision algorithm, and rollback.

Example 1:

Using the block diagram shown in figure E-1, a system with NVS whose outputs go into a decision algorithm will be analyzed. For this example, NVS will consist of five versions. The reliability values for the five versions and the decision algorithm are given in table E-1.

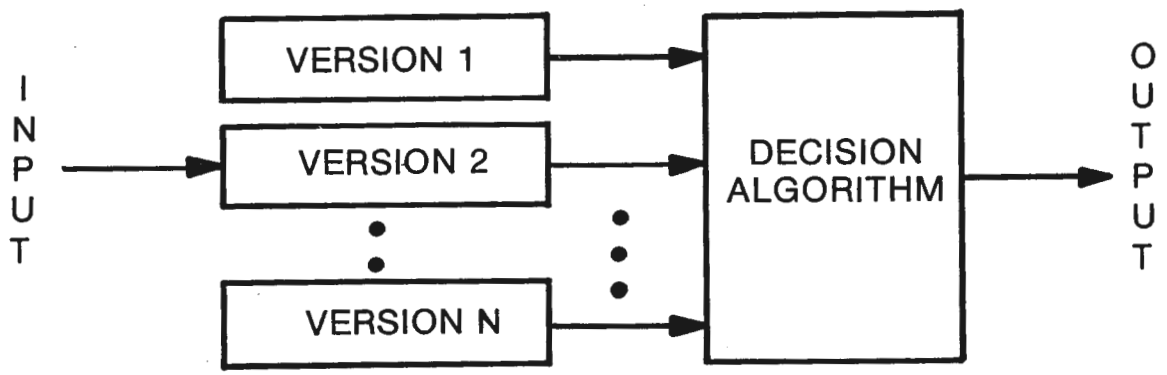


FIGURE E-1. GENERAL FORMAT FOR N-VERSION SOFTWARE

TABLE E-1. RELIABILITY VALUES FOR THE SOFTWARE COMPONENTS IN FIGURE E-1

Software Component	Reliability Value
Version 1	0.77
Version 2	0.82
Version 3	0.65
Version 4	0.91
Version 5	0.89
Decision Algorithm	0.997

To determine the overall software reliability value for this block diagram, the transfer function for the NVS, which is dependent upon the number of versions (in this example n = 5), must first be determined. Hence, the transfer function for the NVS is

$$NVS = [1 - (1-G_1G_2G_3)(1-G_1G_2G_4)(1-G_1G_2G_5)(1-G_1G_3G_4)(1-G_1G_3G_5) \times (1-G_1G_4G_5)(1-G_2G_3G_4)(1-G_2G_3G_5)(1-G_2G_4G_5)(1-G_3G_4G_5)]$$

By substituting in the appropriate reliability values,

$$NVS = 1 - [1-(0.77)(0.82)(0.65)][1-(0.77)(0.82)(0.91)] \times [1-(0.77)(0.82)(0.89)][1-(0.77)(0.65)(0.91)] \times [1-(0.77)(0.65)(0.89)][1-(0.77)(0.91)(0.89)] \times [1-(0.82)(0.65)(0.91)][1-(0.82)(0.65)(0.89)] \times [1-(0.82)(0.91)(0.89)][1-(0.65)(0.91)(0.89)]$$

$$= [1 - (1-0.41041)(1-0.574574)(1-0.561946)(1-0.455455) \times (1-0.445445)(1-0.623623)(1-0.48503)(1-0.47437) \times (1-0.664118)(1-0.526435)]$$

$$= [1 - (0.58959)(0.425426)(0.438054)(0.544545)(0.554555) \times (0.376377)(0.51497)(0.52563)(0.335882)(0.473565)]$$

$$= 1 - 0.0005377 = 0.9994623$$

$$NVS = 0.99946$$

By applying the software reliability model,

L_1 through L_n = 0

$G_1 = (0.99946) \times (0.997) = 0.9964616$

G_2 through $G_K = 0$

$\Delta_1 = 1$

$\Delta = 1$

Therefore,

$$\text{Reliability} = \frac{(0.9964616) \times (1)}{(1)} = 0.9964616$$

Reliability = 0.996

The probability of coincident errors (E) should be determined and subtracted from the transfer function for the NVS to improve the accuracy of the reliability value of the NVS and the accuracy of the overall software reliability value. For this example,

$$E = \binom{5}{3}^* (1-G_L)^3 + \binom{5}{4}^* (1-G_L)^4 + \binom{5}{5}^* (1-G_L)^5$$

The groups for $\binom{5}{3}^*$ would be

$G_1G_2G_3$, $G_1G_2G_4$, $G_1G_2G_5$, $G_1G_3G_4$, $G_1G_3G_5$, $G_1G_4G_5$, $G_2G_3G_4$, $G_2G_3G_5$, $G_2G_4G_5$, and $G_3G_4G_5$,

with respective G_L values being

G_2, G_4, G_5, G_4, G_5, G_4, G_4, G_5, G_4, and G_4.

The G_L values can be grouped as $G_2 + (6 \times G_4) + (3 \times G_5)$.

The groups for $\binom{5}{4}^*$ would be

$G_1G_2G_3G_4$, $G_1G_2G_3G_5$, $G_1G_2G_4G_5$, $G_1G_3G_4G_5$, and $G_2G_3G_4G_5$,

with the respective G_L values being G_4, G_5, G_4, G_4, and G_4, giving

$4 \times G_4 + G_5$.

The group for $\binom{5}{5}^*$ is $G_1G_2G_3G_4G_5$ with $G_L = G_4$.

Therefore,

$$E = (1-G_2)^3 + 6 \times (1-G_4)^3 + 3 \times (1-G_5)^3 + 4 \times (1-G_4)^4 + (1-G_5)^4 + (1-G_4)^5$$

$$= (1-0.82)^3 + 6 \times (1-0.91)^3 + 3 \times (1-0.89)^3 + 4 \times (1-0.91)^4 + (1-0.89)^4 + (1-0.91)^5$$

$$= (0.18)^3 + 6 \times (0.09)^3 + 3 \times (0.11)^3 + 4 \times (0.09)^4 + (0.11)^4 + (0.09)^5$$

$$= 0.005832 + 0.004374 + 0.003993 + 0.0002624 + 0.0001464 + 0.0000059$$

$$E = 0.0146137$$

Subtracting E from the transfer function for the NVS gives

$$NVS = 0.9994623 - 0.0146137 = 0.9848486$$

$$NVS = 0.98485$$

L_1 through $L_n = 0$

$G_1 = (0.98485) \times (0.997) = 0.9818955$

G_2 through $G_K = 0$

$\Delta_1 = 1$

$\Delta = 1$

Hence, the adjusted reliability value is

$$\text{Reliability} = \frac{(0.9818955) \times (1)}{(1)} = 0.9818955$$

$$\text{Reliability} = 0.982$$

Example 2:

This example will evaluate the overall reliability for the system shown in figure E-2. In this example, the NVS will consist of four versions. Each of the outputs from the NVS are submitted to an acceptance test (the identical acceptance test is used for all four versions), and then the outputs from the acceptance test are input to the decision algorithm. The reliability values for each of the software components are given in table E-2.

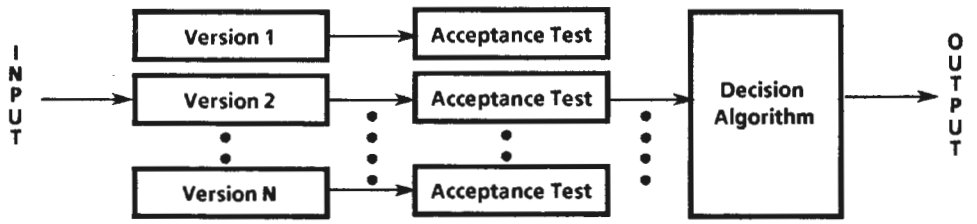


FIGURE E-2. N-VERSION SOFTWARE WITH ACCEPTANCE TESTS

TABLE E-2. RELIABILITY VALUES FOR THE SOFTWARE COMPONENTS IN FIGURE E-2

Software Component	Reliability Value
Version 1	0.86
Version 2	0.79
Version 3	0.94
Version 4	0.68
Acceptance Test	0.98
Decision Algorithm	0.93

First, the transfer function for the NVS must be determined. In this example,

$$NVS = [1 - (1-G_1G_2G_3)(1-G_1G_3G_4)(1-G_1G_2G_4)(1-G_2G_3G_4)].$$

By substituting in the appropriate reliability values,

$$NVS = \{1 - [1 - (0.86)(0.79)(0.94)][1 - (0.86)(0.94)(0.68)] \times [1 - (0.86)(0.79)(0.68)][1 - (0.79)(0.94)(0.68)]\}$$

$$= [1 - (1-0.638636)(1-0.549712)(1-0.461992)(1-0.504968)]$$

$$= [1 - (0.361364)(0.450288)(0.538008)(0.495032)]$$

$$= 1 - 0.0433368 = 0.9566632$$

$$NVS = 0.95666$$

By applying the software reliability model,

L_1 through $L_n = 0$

$G_1 = (0.95666) \times (0.98) \times (0.93) = 0.8718999$

G_2 through $G_K = 0$

$\Delta_1 = 1$

$\Delta = 1$

Therefore,

$$\text{Reliability} = \frac{(0.8718999) \times (1)}{(1)} = 0.8718999$$

$$\text{Reliability} = 0.872$$

The probability of coincident errors (E) for this example is

$$E = \binom{4}{2}^* (1-G_L)^2 + \binom{4}{3}^* (1-G_L)^3 + \binom{4}{4}^* (1-G_L)^4$$

The groups for $\binom{4}{2}^*$ are

G_1G_2, G_1G_3, G_1G_4, G_2G_3, G_2G_4, and G_3G_4.

Their respective G_L values are G_1, G_3, G_1, G_3, G_2, and G_3. These values can be grouped as $(2 \times G_1) + (2 \times G_2) + (2 \times G_3)$.

The groups for $\binom{4}{3}^*$ are $G_1G_2G_3$, $G_1G_2G_4$, $G_1G_3G_4$, and $G_2G_3G_4$, with the G_L values G_3, G_1, G_3, and G_3, respectively, giving $G_1 + (3 \times G_3)$.

The group for $\binom{4}{4}^*$ 4 is $G_1G_2G_3G_4$ with $G_L = G_3$.

Hence,

$$E = 2 \times (1-G_1)^2 + 2 \times (1-G_2)^2 + 2 \times (1-G_3)^2 + (1-G_1)^3 + 3 \times (1-G_3)^3 + (1-G_3)^4$$

$$= 2 \times (1-0.86)^2 + 2 \times (1-0.79)^2 + 2 \times (1-0.94)^2 + (1-0.86)^3 + 3 \times (1-0.94)^3 + (1-0.94)^4$$

$$= 2 \times (0.14)^2 + 2 \times (0.21)^2 + 2 \times (0.06)^2 + (0.14)^3 + 3 \times (0.06)^3 + (0.06)^4$$

$$= 0.0392 + 0.0882 + 0.0072 + 0.002744 + 0.000648 + 0.00001296$$

$$E = 0.1380049$$

Re-evaluating the reliability value for the NVS and the overall software reliability value gives

$$NVS = 0.9566632 - 0.00001296 = 0.9566503$$

$$NVS = 0.95665$$

L_1 through $L_n = 0$

$$G_1 = (0.95665) \times (0.98) \times (0.93) = 0.8718908$$

G_2 through $G_K = 0$

$\Delta_1 = 1$

$\Delta = 1$

Therefore, the adjusted reliability value is

$$\text{Reliability} = \frac{0.8718908 \times 1}{1} = 0.8718908$$

$$\text{Reliability} = 0.872$$

Example 3:

The hybrid NVS format, shown in figure E-3, will be evaluated in this example. The NVS here consists of three versions. The outputs of these versions are fed into a decision algorithm. If the decision algorithm fails, then the software will rollback and run through the three versions again. However, this time the outputs of the versions are input to an acceptance test prior to entry to the decision algorithm. For this example, it is assumed that the reliability values for the individual software components are as given in table E-3.

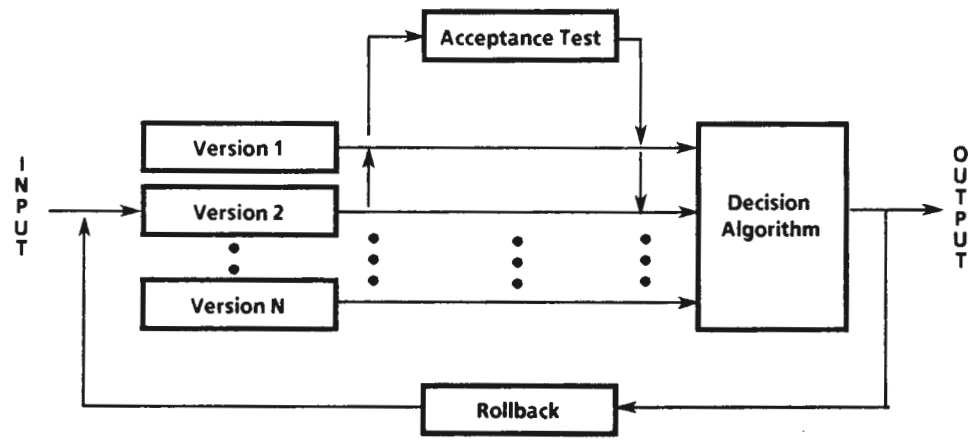


FIGURE E-3. N-VERSION SOFTWARE IN WHICH THE OUTPUTS ARE SUBJECTED TO AN ACCEPTANCE TEST IF THE DECISION ALGORITHM FAILS (Scott)

TABLE E-3. RELIABILITY VALUES FOR THE SOFTWARE COMPONENTS IN FIGURE E-3

Software Component	Reliability Value
Version 1	0.67
Version 2	0.78
Version 3	0.89
Acceptance Test	0.86
Decision Algorithm	0.88
Rollback	0.98

The transfer function for the NVS, based on the data in table E-3, is

$$NVS = 1-(1-G_1G_2)(1-G_1G_3)(1-G_2G_3)$$

$$= 1-[1-(0.67)(0.78)][1-(0.67)(0.89)][1-(0.78)(0.89)]$$

$$= 1-(1-0.5226)(1-0.5963)(1-0.6942)$$

$$= 1-(0.4774)(0.4037)(0.3058)$$

$$NVS = 0.94106428$$

The variables of the software reliability model are

$$\text{Closed Loop \#1} = (0.94106428) \times (0.88) \times (1-0.98)$$

$$= 0.01656273$$

$$\text{Closed Loop \#2} = (0.94106428) \times (1-0.86) \times (0.88) \times (1-0.98)$$

$$= 0.0023187824$$

[Remember that the transfer function of the equivalent block in a feedback or feed-forward path is (1.0 - reliability value).]

ΣL_1 = Closed Loop #1 + Closed Loop #2 = 0.018881512

ΣL_2 through $\Sigma L_n = 0$

$G_1 = (0.94106428) \times (0.88) = 0.82813657$

$G_2 = (0.94106428) \times (1-0.86) \times (0.88) = 0.11593912$

G_3 through $G_K = 0$

$\Delta_1 = 1$

$\Delta_2 = 1$

Δ_3 through $\Delta_K = 0$

$\Delta = 1 - \Sigma L_1 = 1 - 0.018881512 = 0.981118488$

Therefore,

$$\text{Reliability} = \frac{(0.82813657 \times 1) + (0.11593912 \times 1)}{(0.981118408)} = 0.9622444$$

Reliability = 0.962

To improve the accuracy of the software reliability model, the probability of coincident errors (E) might be considered. For this example,

$$E = \binom{3}{2}^* \ (1\text{-}GL)^2 + \binom{3}{3}^* \ (1\text{-}GL)^3$$

The groups for $\binom{3}{2}^*$ are G_1G_2, G_1G_3, and G_2G_3 with respective G_L values of G_2, G_3, and G_3, or $G_2 + (2 \times G_3)$. The group for $\binom{3}{3}^*$ is $G_1G_2G_3$ with $G_L = G_3$.

Thus,

$$E = (1\text{-}G_2)^2 + 2 \times (1\text{-}G_3)^2 + (1\text{-}G_3)^3$$

$$= (1\text{-}0.78)^2 + 2 \times (1\text{-}0.89)^2 + (1\text{-}0.89)^3$$

$$= (0.22)^2 + 2 \times (0.11)^2 + (0.11)^3$$

$$= 0.0484 + 0.0242 + 0.001331$$

$$E = 0.073931$$

By subtracting the probability of coincident errors from the NVS transfer function, a conservative value of the reliability value for the NVS and the overall software reliability value can be determined.

$$NVS = 0.94106428 - 0.073931 = 0.86713328$$

$$\text{Closed Loop \#1} = (0.86713328) \times (0.88) \times (1\text{-}0.98)$$

$$= 0.015261546$$

$$\text{Closed Loop \#2} = (0.86713328) \times (1\text{-}0.86) \times (0.88) \times (1\text{-}0.98)$$

$$= 0.0021366164$$

$$\Sigma L_1 = 0.015261546 + 0.0021366164 = 0.017398162$$

ΣL_2 through $\Sigma L_n = 0$

$G_1 = (0.86713328) \times (0.88) = 0.76307729$

$G_2 = (0.86713328) \times (1\text{-}0.86) \times (0.88) = 0.10683082$

G_3 through $G_K = 0$

$\Delta_1 = 1$

$\Delta_2 = 1$

Δ_3 through $\Delta_K = 0$

$\Delta = 1 - \Sigma L_1 = 1 - 0.017398162 = 0.98260184$

Therefore,

$$\text{Reliability} = \frac{(0.76307729 \times 1) + (0.10683082 \times 1)}{(0.98260184)} = 0.88531089$$

Reliability = 0.885

APPENDIX F - WORKED EXAMPLES FOR THE RECOVERY BLOCK SOFTWARE RELIABILITY DESIGN TECHNIQUES

Appendix F contains three worked examples using RB software reliability design techniques.

Example 1: Analysis of a system which uses the general format of the backward RB.

Example 2: Analysis of a system which uses the general format of a forward RB.

Example 3: Analysis of a system which uses a variation of the forward RB format.

Example 1:

Figure F-1 shows the general format of a backward RB. For this example, the number of alternates will be four. The reliability value for each of the software components is listed in table F-1.

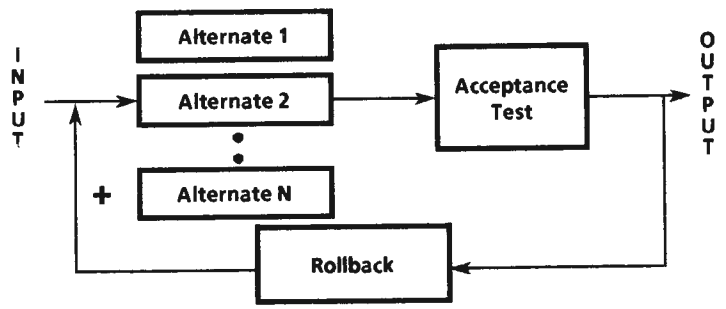


FIGURE F-1. GENERAL FORMAT OF A BACKWARD RECOVERY BLOCK

TABLE F-1. RELIABILITY VALUES FOR THE SOFTWARE COMPONENTS IN FIGURE F-1

Software Component	Reliability Value
Alternate 1	0.86
Alternate 2	0.75
Alternate 3	0.79
Alternate 4	0.84
Acceptance Test	0.91
Rollback	0.93

The transfer function for the RB is

$$RB = G_1 + (1 - G_1)G_2 + (1 - G_1)(1 - G_2)G_3 + (1 - G_1)(1 - G_2)(1 - G_3)\,G_4$$

$$= 0.86 + (1 - 0.86)(0.75) + (1 - 0.86)(1 - 0.75)(0.79) + (1 - 0.86)(1 - 0.75)(1 - 0.79)(0.84)$$

$$= 0.86 + (0.14)(0.75) + (0.14)(0.25)(0.79) + (0.14)(0.25)(0.21)(0.84)$$

$$= 0.86 + 0.105 + 0.02765 + 0.006174$$

$$RB = 0.998824$$

The variables of the software reliability model will be

$$L_1 = (0.998824) \times (0.91) \times (1.0 - 0.93) = 0.0636251$$

[Recall that the transfer function for rollback is (1.0 - reliability value).]

L_2 through $L_n = 0$

$G_1 = (0.998824) \times (0.91) = 0.9089298$

G_2 through $G_K = 0$

$\Delta_1 = 1$

Δ_2 through $\Delta_K = 0$

$\Delta = 1 - 0.0636251 = 0.9363749$

Therefore,

$$\text{Reliability} = \frac{(0.9089298) \times (1)}{(0.9363749)} = 0.9706901$$

Reliability = 0.97

If only two of the four possible alternatives are actually used, the reliability of the RB will decrease. The overall software reliability will also decrease. The following calculations show this:

$$RB = G_1 + (1 - G_1)G_2 = 0.86 + (1 - 0.86)(0.75)$$
$$= 0.86 + 0.105 = 0.965$$

$L_1 = (0.965) \times (0.91) \times (1 - 0.93) = 0.0614705$

L_2 through $L_n = 0$

$G_1 = (0.965) \times (0.91) = 0.9089298$

G_2 through $G_K = 0$

$\Delta_1 = 1$

Δ_2 through $\Delta_K = 0$

$\Delta = 1 - 0.0614705 = 0.9385295$

Hence,

$$\text{Reliability} = \frac{(0.9089298) \times (1)}{(0.9385295)} = 0.9684616$$

Reliability = 0.968

when only two of the alternates are used.

Example 2:

The general format of a forward RB is shown in figure F-2. This example will evaluate the overall software reliability of this figure (six alternatives will be used: one primary and five alternates), with the reliability values assigned as shown in table F-2.

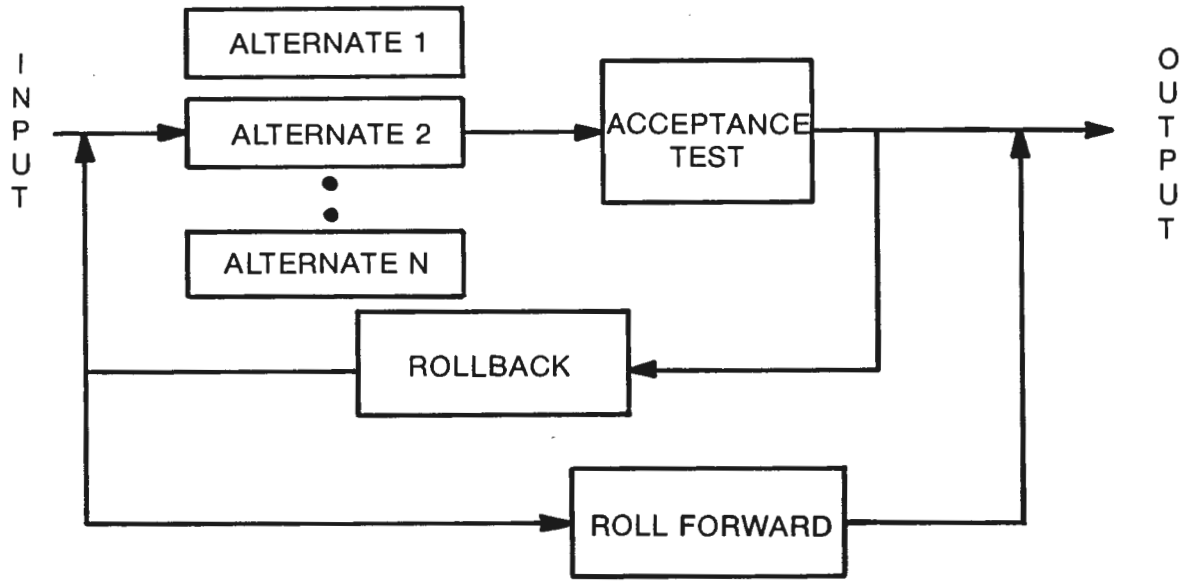


FIGURE F-2. GENERAL FORMAT OF A FORWARD RECOVERY BLOCK

TABLE F-2. RELIABILITY VALUES FOR THE SOFTWARE COMPONENTS IN FIGURE F-2

Software Component	Reliability Value
Alternate 1	0.81
Alternate 2	0.72
Alternate 3	0.73
Alternate 4	0.74
Alternate 5	0.85
Alternate 6	0.86
Acceptance Test	0.97
Rollback	0.98
Roll-Forward	0.89

The transfer function for the RB is

$$RB = G_1 + (1 - G_1)G_2 + (1 - G_1)(1 - G_2)G_3 + (1 - G_1)(1 - G_2)(1 - G_3)G_4 + (1 - G_1)(1 - G_2)(1 - G_3)(1 - G_4)G_5 + (1 - G_1)(1 - G_2)(1 - G_3)(1 - G_4)(1 - G_5)G_6$$

$$= 0.81 + (1 - 0.81)(0.72) + (1 - 0.81)(1 - 0.72)(0.73) + (1 - 0.81)(1 - 0.72)(1 - 0.73)(0.74) + (1 - 0.81)(1 - 0.72)(1 - 0.73)(1 - 0.74)(0.85) + (1 - 0.81)(1 - 0.72)(1 - 0.73)(1 - 0.74)(1 - 0.85)(0.86)$$

$$= 0.81 + (0.19)(0.72) + (0.19)(0.28)(0.73) + (0.19)(0.28)(0.27)(0.74) + (0.19)(0.28)(0.27)(0.26)(0.85) + (0.19)(0.28)(0.27)(0.26)(0.15)(0.86)$$

$$= 0.81 + 0.1368 + 0.038836 + 0.01062936 + 0.003174444 + 0.00048176856$$

$$= 0.99992157256$$

$$RB = 0.9999216$$

With the software reliability model,

$$L_1 = (0.9999216) \times (0.97) \times (1 - 0.98) = 0.019398479$$

[Note that the transfer function for rollback is (1.0 - reliability value).]

L_2 through $L_n = 0$

$$G_1 = (0.9999216) \times (0.97) = 0.96992395$$

$$G_2 = (0.9999216) \times (0.97) \times (1 - 0.98) \times (1 - 0.89) = 0.0021338327$$

[Remember that the transfer function for rollback and roll-forward is (1.0 - reliability value).]

G_3 through $G_K = 0$

$\Delta_1 = 1$

$\Delta_2 = 1$

Δ_3 through $\Delta_K = 0$

$\Delta = 1 - 0.019398479 = 0.98060152$

Therefore,

$$\text{Reliability} = \frac{(0.96992395 \times 1) + (0.0021338327 \times 1)}{(0.98060152)}$$

$$= (0.97205778)/(0.98060152) = 0.99128725$$

$$\text{Reliability} = 0.991$$

Discussion of the Results

This result is as expected. With just the n alternates, acceptance test, and rollback, the overall reliability would be

$$\text{Reliability} = \frac{(0.9999216)(0.97)}{1 - (0.9999216)(0.97)(1 - 0.98)}$$

$$\text{Reliability} = 0.9891112 = 0.989$$

The roll-forward should increase this reliability value, as it does.

Alternative Calculations (Fewer than n alternatives)

To evaluate the effect on accuracy when fewer than the n alternates (in this example n = 6) are actually used, the reliability of this example will be evaluated with n = 3, n = 4, and n = 5.

For n = 3,

$$RB = G_1 + (1 - G_1)G_2 + (1 - G_1)(1 - G_2)G_3$$

$$= 0.81 + (1 - 0.81)(0.72) + (1 - 0.81)(1 - 0.72)(0.73)$$

$$= 0.81 + 0.1368 + 0.038836$$

$$RB = 0.985636$$

Using the software reliability model,

$$L_1 = (0.985636)(0.97)(1 - 0.98) = 0.019121338$$

L_2 through $L_n = 0$

$$G_1 = (0.985636)(0.97) = 0.95606692$$

$$G_2 = (0.985636)(0.97)(1 - 0.98)(1 - 0.89) = 0.0021033472$$

G_3 through $G_K = 0$

$\Delta_1 = 1$

$\Delta_2 = 1$

Δ_3 through $\Delta_K = 0$

$\Delta = 1 - 0.019121338 = 0.98087866$

gives

$$\text{Reliability} = \frac{(0.95606692 \times 1) + (0.0021033472 \times 1)}{(0.98087866)} = 0.97684893$$

Reliability = 0.977

For n = 4,

RB = 0.81 + 0.1368 + 0.038836 + 0.01062936

RB = 0.99626536

Using the software reliability model,

$L_1 = (0.99626536)(0.97)(1 - 0.98) = 0.019327547$

L_2 through $L_n = 0$

$G_1 = (0.99626536)(0.97) = 0.9663774$

$G_2 = (0.99626536)(0.97)(1 - 0.98)(1 - 0.89) = 0.0021260303$

G_3 through $G_K = 0$

$\Delta_1 = 1$

$\Delta_2 = 1$

Δ_3 through $\Delta_K = 0$

$\Delta = 1 - 0.019327547 = 0.98067245$

gives

$$\text{Reliability} = \frac{(0.9663774 \times 1) + (0.0021260303 \times 1)}{(0.98067245)} = 0.98759115$$

Reliability = 0.988

For n = 5,

$RB = 0.99626536 + 0.003174444$

$RB = 0.999439804$

Using the software reliability model,

$L_1 = (0.999439804)(0.97)(1 - 0.98) = 0.019389132$

L_2 through $L_n = 0$

$G_1 = (0.999439804)(0.97) = 0.96945661$

$G_2 = (0.999439804)(0.97)(1 - 0.98)(1 - 0.89) = 0.0021328045$

G_3 through $G_K = 0$

$\Delta_1 = 1$

$\Delta_2 = 1$

Δ_3 through $\Delta_K = 0$

$\Delta = 1 - 0.019389132 = 0.98061087$

gives

$$\text{Reliability} = \frac{(0.96945661 \times 1) + (0.0021328045 \times 1)}{(0.98061087)} = 0.99080017$$

$\text{Reliability} = 0.991$

Discussion of the Results

Table F-3 compares the reliability values that are obtained by using fewer than n alternates in this example.

This is as expected. Actually using fewer than the n alternates, the reliability values for the RB and the overall software will generally decrease. However, as was seen in the case with n = 5, by not using the sixth alternate (which has a reliability value of 0.86 in this example), an extremely slight increase in reliability was found.

TABLE F-3. ACCURACY EFFECTS WHEN FEWER THAN n ALTERNATES ARE USED

Number of Alternates Used	Recovery Block Reliability Value	Overall Software Reliability Value
n = 3	0.98564	0.977
n = 4	0.99627	0.988
n = 5	0.99944	0.991
n = 6	0.99992	0.991

Example 3:

Figure F-3 shows a possible variation of a forward RB. For this example, the number of alternates will be three. The reliability value for each of the software components is listed in table F-4.

A variation of the forward RB format might be:

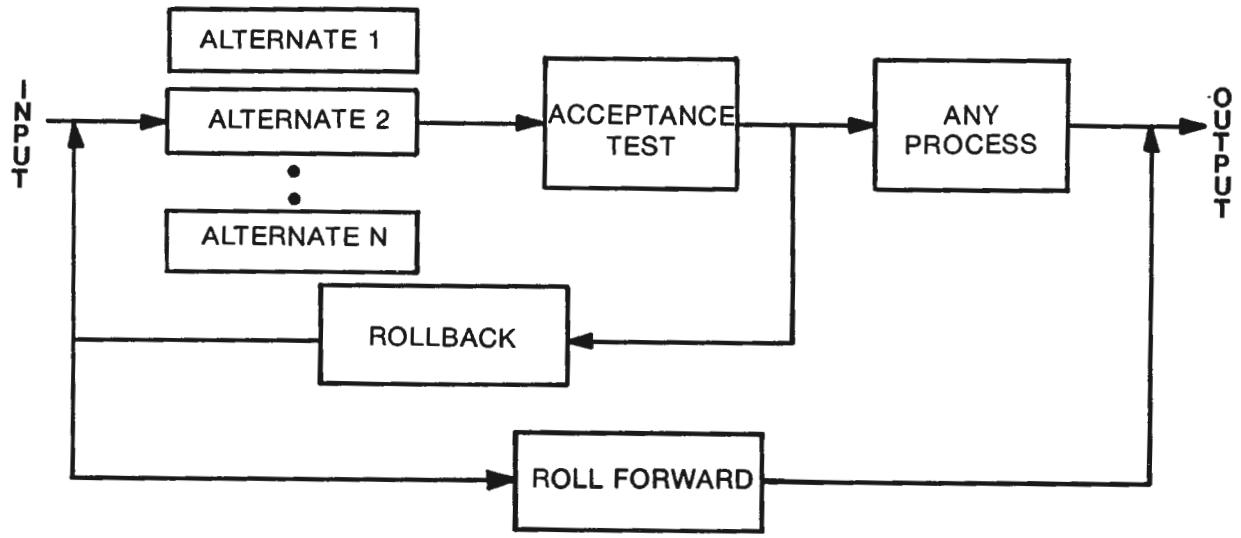


FIGURE F-3. ALTERNATE FORMAT FOR A FORWARD RECOVERY BLOCK

TABLE F-4. RELIABILITY VALUES FOR THE SOFTWARE COMPONENTS IN FIGURE F-3

Software Component	Reliability Value
Alternate 1	0.80
Alternate 2	0.70
Alternate 3	0.90
Acceptance Test	0.98
Any Process	0.97
Rollback	0.95
Roll-Forward	0.96

The transfer function for the RB is

$$RB = G_1 + (1 - G_1)G_2 + (1 - G_1)(1 - G_2)G_3$$

$$= 0.80 + (1 - 0.80) \times (0.70) + (1 - 0.80) \times (1 - 0.70) \times (0.90)$$

$$= 0.80 + (0.20 \times 0.70) + (0.20 \times 0.30 \times 0.90)$$

$$= 0.80 + 0.14 + 0.054$$

$$RB = 0.994$$

The variables of the software reliability model are

$L_1 = (0.994) \times (0.98) \times (1 - 0.95) = 0.048706$

L_2 through $L_n = 0$

$G_1 = (0.994) \times (0.98) \times (0.97) = 0.9448964$

$G_2 = (0.994) \times (0.98) \times (1 - 0.95) \times (1 - 0.96) = 0.0019482$

G_3 through $G_K = 0$

$\Delta_1 = 1$

$\Delta_2 = 1$

Δ_3 through $\Delta_K = 0$

$\Delta = 1 - L_1 = 1 - 0.048706 = 0.951294$

Therefore,

$$\text{Reliability} = \frac{(0.9448964 \times 1) + (0.0019482 \times 1)}{(0.951294)} = 0.99532279$$

Reliability = 0.995

To demonstrate the effect on accuracy if fewer than the n alternates (in this example n = 3) are actually used, the reliability of the RB and overall software reliability value will be re-calculated for n = 1 and n = 2.

For n = 1, RB = 0.80

With the software reliability model,

$L_1 = (0.80) \times (0.98) \times (1 - 0.95) = 0.0392$

L_2 through L_n = 0

$G_1 = (0.80) \times (0.98) \times (0.97) = 0.76048$

$G_2 = (0.80) \times (0.98) \times (1 - 0.95) \times (1 - 0.96) = 0.001568$

G_3 through G_K = 0

$\Delta_1 = 1$

$\Delta_2 = 1$

Δ_3 through Δ_K = 0

$\Delta = 1 - L_1 = 1 - 0.0392 = 0.9608$

$$\text{Reliability} = \frac{(0.76048 \times 1) + (0.001568 \times 1)}{(0.9608)} = 0.7931391$$

Reliability = 0.793

For n = 2, RB = 0.80 + 0.14 = 0.94

With the software reliability model,

$L_1 = (0.94) \times (0.98) \times (1 - 0.95) = 0.04606$

L_2 through L_n = 0

$G_1 = (0.94) \times (0.98) \times (0.97) = 0.893564$

$G_2 = (0.94) \times (0.98) \times (1 - 0.95) \times (1 - 0.96) = 0.0018424$

G_3 through G_K = 0

$\Delta_1 = 1$

$\Delta_2 = 1$

Δ_3 through Δ_K = 0

$\Delta = 1 - L_1 = 1 - 0.04606 = 0.95394$

$$\text{Reliability} = \frac{(0.893564 \times 1) + (0.0018424 \times 1)}{(0.95394)} = 0.93864017$$

Reliability = 0.939

Discussion of the Results

Table F-5 compares the RB reliability values and the overall software reliability values that are obtained by using n or fewer than n alternates in this example.

TABLE F-5. COMPARISON OF RELIABILITY VALUES WHEN FEWER THAN n ALTERNATES ARE USED

Number of Alternates Used	Recovery Block Reliability Value	Overall Software Reliability Value
n = 1	0.80	0.793
n = 2	0.94	0.939
n = 3	0.994	0.995

BIBLIOGRAPHY

Anderson, T., and J. Knight, "Software Fault Tolerance for Real-Time Avionics Systems," AGARD Conference Proceedings No. 330, Software for Avionics, September 1982.

Eckhardt, D. E. Jr., and L. D. Lee, A Theoretical Basis for the Analysis of Redundant Software Subject to Coincident Errors, NASA Langley Research Center, NASA Technical Memorandum 86369, January 1985.

Gephart, L. S., et al., Software Reliability: Determination and Prediction, Technical Report AFFDL-TR-78-77, Air Force Flight Dynamics Laboratory, Air Force Systems Command, Wright-Patterson Air Force Base, OH, June 1978.

Goel, A. L., A Guidebook for Software Reliability Assessment, Technical Report RADC-TR-83-176 (AD A139240), Rome Air Development Center, Air Force Systems Command, Griffiss Air Force Base, NY, August 1983.

Hecht, H., "Fault-Tolerant Software," IEEE Transactions on Reliability, Vol. R-28, No. 3, pp. 227-232, August 1979.

Hitt, E. F., J. J. Webb, and M. S. Bridgman, Comparative Analysis of Fault-Tolerance Software Design Techniques, Prepared Under Contract Number NAS1-17412, February 15, 1984.

Krauson, S. S., and D. R. Baker, "Software Reliability Enhancement Techniques and Assessment Method for Embedded Computer Systems," IEEE 1982 National Aerospace and Electronics Conference, NAECON 1982, May 1982.

Larsen, W., et al., "An Overview of the Digital Avionics Assessment Activities Being Conducted by the Federal Aviation Administration at NASA-Ames Research Center," Proceedings of the AIAA/IEEE 6th Digital Avionics Systems Conference, December 1984.

Prater, S. A., E. F. Hitt, and D. Eldredge, Software Dependability Assessment Methods, DOT/FAA/CT-86/27, FAA Technical Center, Atlantic City Airport, NJ, November 1986.

Ramamoorthy, C. V., and F. B. Bastani, "Software Reliability - Status and Perspectives," IEEE Transactions on Software Engineering, Vol. SE-8, No. 4, 354-371, July 1982.

Scott, R. K., J. W. Gault, and D. F. McAllister, "Modeling Fault Tolerant Software Reliability," Proceedings of the Third Symposium on Reliability in Distributed Software and Database Systems, 1983.

Scott, R. K., J. W. Gault, and D. F. McAllister, "Fault Tolerant Software Reliability Modeling," IEEE Transactions on Software Engineering, 1987, Vol. SE-13, No. 5, pp. 582-592, May 1987.

_____, et al., "Investigating Version Dependence in Fault-Tolerant Software," AGARD Conference Proceedings No. 361, Design for Tactical Avionics Maintainability, May 1984.

Shinners, S. M., Modern Control System Theory and Application, Addison-Wesley Publishing Company, Inc., U.S.A., 1978.

Shooman, M. L., "Software Reliability: A Historical Perspective," IEEE Transactions on Reliability, Vol. R-33, No. 1, pp. 48-54, April 1984.

GLOSSARY

ERROR. A state of the system which (in the absence of any corrective action by the system) could lead to a failure that would not be attributed to any event subsequent to the error. (More accurately known as an erroneous state.)

FAILURE. The situation when the external behavior of a system does not conform to that prescribed by the system specification.

FAULT. The adjusted cause of error.

HAZARD FUNCTION. The conditional probability that a fault is exposed in the interval t to Δt given that the fault did not occur prior to time t.

VOTING PROCEDURE. An algorithm included in fault tolerant software which uses the consensus RB method. It compares outputs of the n independent versions and determines which outputs are correct by identifying agreements among two or more versions.

ACRONYMS AND ABBREVIATIONS

AFFDL	Air Force Flight Dynamics Laboratory
AGARD	Advisory Group for Aerospace Research and Development
cdf	Cumulative Density Function
DOD	Department of Defense
DOT	Department of Transportation
FAA	Federal Aviation Administration
MLE	Maximum Likelihood Estimates
MTTF	Mean Time to Failure
NHPP	Non-Homogeneous Poisson Process
NVS	N-version Software
pdf	Probability Density Function
RADC	Rome Air Development Center
RB	Recovery Block

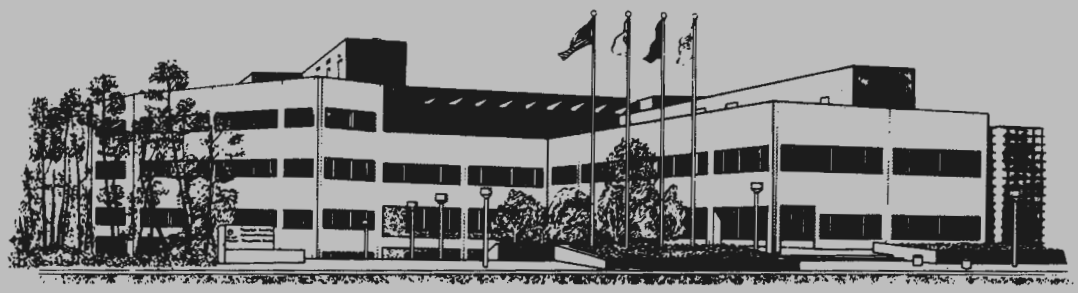
COPY — 1

U.S. Department of Transportation
Federal Aviation Administration

DOT/FAA/CT-88/10

HANDBOOK-VOLUME II
DIGITAL SYSTEMS VALIDATION

CHAPTER 9
FAULT TOLERANT SOFTWARE



PREPARED BY:

COMPUTER RESOURCE MANAGEMENT, INC.
950 HERNDON PARKWAY, SUITE 360
HERNDON, VIRGINIA 22070

PREPARED FOR:

FEDERAL AVIATION ADMINISTRATION
TECHNICAL CENTER
ATLANTIC CITY INTERNATIONAL AIRPORT, NEW JERSEY 08405

FAA WJH Technical Center
00021057

NOTICE

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The United States Government assumes no liability for the contents or use thereof.

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the objective of this report.

TABLE OF CONTENTS

Section		Page
1.	INTRODUCTION	9-1
	1.1. Motivation for Fault Tolerant Software	9-1
2.	PARTIAL SOFTWARE FAULT TOLERANCE TECHNIQUES	9-5
	2.1. Robustness	9-5
	2.2. Rollback	9-6
	2.3. Issues in Partial Fault Tolerance	9-6
3.	N-VERSION SOFTWARE	9-9
	3.1. Description	9-9
	3.2. Concerns	9-11
	3.2.1. The Software Specification	9-11
	3.2.2. Independence of Coding Faults	9-11
	3.2.3. Run-time Environment	9-12
	3.2.4. Recovery of Failed Versions	9-13
	3.2.5. Maintenance and Upgrades	9-13
4.	RECOVERY BLOCKS	9-15
	4.1. Description	9-15
	4.2. Critical Issues	9-21
	4.2.1. Acceptance Tests	9-21
	4.2.2. Run-time Environment	9-22
	4.2.3. Independence of the Alternate Routine	9-22
5.	FAULT AVOIDANCE VERSUS FAULT TOLERANCE	9-25
6.	INTEGRATION OF FAULT TOLERANCE TECHNIQUES	9-29
	6.1. Coverage	9-29
	6.2. Retry Requirements	9-29
	6.3. Cost and Complexity	9-29
7.	CONCLUSION	9-33
BIBLIOGRAPHY		9-35
GLOSSARY		9-37
ACRONYMS AND ABBREVIATIONS		9-38

LIST OF ILLUSTRATIONS

Figure		Page
3.1-1	A 4-VERSION SOFTWARE IMPLEMENTATION FOR FLIGHT CONTROL	9-10
4.1-1	BASIC RECOVERY BLOCK	9-16
4.1-2	RECOVERY BLOCK FOR REAL-TIME APPLICATIONS WITH A SINGLE WATCHDOG TIMER	9-18
4.1-3	A REAL-TIME RECOVERY BLOCK WITH TWO WATCHDOG TIMERS	9-20
4.1-4	DISTRIBUTED RECOVERY BLOCK	9-21

LIST OF TABLES

Table		Page
6-1	SOME RELEVANT CHARACTERISTICS OF FAULT TOLERANCE TECHNIQUES	9-30

1. INTRODUCTION

As digital systems for flight control, engine control, power distribution, and other flight critical functions have increased in sophistication and complexity, the need for fault tolerant software to overcome software reliability problems has become more apparent. This chapter describes fault tolerant software techniques and concerns in certification and validation. Section 1 explains why traditional software development methods are inadequate and where the use of fault tolerant software is required. Sections 2 through 4 discuss the three major categories of software fault tolerance: partial fault tolerance, N-version software, and recovery blocks. Section 5 describes the tradeoffs and considerations in fault tolerant versus non-fault tolerant software, and section 6 discusses the combination and integration of several fault tolerance techniques.

1.1. Motivation for Fault Tolerant Software

At present, the most common approach for development of critical real-time software can be termed fault avoidance, i.e., attempting to prevent any software faults in the final delivered product through disciplined software development practices, testing, and Independent Verification and Validation (IV&V).

However, fault avoidance methods have their limitations. Most test methodologies used in verification do not have well-defined termination criteria; i.e., it is usually difficult to decide when enough testing has been performed. Investigations of software test strategies (Dijkstra, 1972, Howden, 1978, and Miller, 1981) have shown that fault densities of approximately 1 per 1000 lines of source code remain even after extensive testing. The limitations of testing were capsulized in the famous saying of E.W. Dijkstra, "Program testing can be used to show the presence of bugs but never their absence." IV&V limitations include:

- The program specification on which verification occurs is generally incomplete because it cannot include enough end-to-end test cases, exception conditions, and comprehensive run time attributes, such as response time or memory utilization.

- Hardware development is frequently conducted in parallel with the software development. IV&V activities are not carried out on the processors on which the operational code will be executed.

- Changes in the system requirements, environments, or the program itself result in the need to perform extensive retesting. The amount of regression testing necessary to ensure verification after changes is uncertain and is often dictated by cost or schedule rather than technical considerations.

Such limitations prompted the developers of Bell Laboratories' No. 4 ESS (Electronic Switching System) to conclude that it

> "...is not technically or economically feasible to detect and fix all software problems in a system as large as the No. 4 ESS. Consequently, a strong emphasis has been placed on making it sufficiently tolerant of software errors to provide successful operation and fault recovery in an environment containing software problems." (Davis and Giloth, 1981)

The No. 4 ESS program referred to in this quotation occupies slightly over two million words of computer memory. Although current flight control and management software is typically only half this size, increasing use of higher order languages, the advent of the "glass cockpit," fly-by-wire control systems, full authority digital engine controllers, and other trends point in the direction of increasing software size and complexity. As these applications become more sophisticated, the need for fault tolerance to achieve higher software reliability is correspondingly greater. However, fault tolerance provisions use resources; therefore, their practical use will require tradeoffs between the need for fault tolerance and the cost incurred by its implementation. Resource utilizations will usually consist of the following components:

- Software development: Staff resources required for the additional code plus corresponding increases in documentation and test.
- Performance penalty: Increase in execution time due to tests and alternate versions that have been incorporated in the program.
- Storage expansion: Additional memory required for storage of tests, alternate versions, alternate routines (for recovery blocks), and recovery cache provisions.
- Higher maintenance costs: Additional costs due to both the inherent increase in the amount of code and the need for increased testing after changes to ensure that the fault tolerance provisions are still intact.

Thus, it is impractical to apply fault tolerance provisions throughout the entire code. Instead, they should be confined to critical modules and components. Another technique to reduce the cost of fault tolerance provisions is to generate a highly "robust" kernel that can then be utilized to direct the error detection and exception handling in other segments of the program. This kernel would include fault tolerance provisions. The "nucleus" of this kernel would contain a small amount of very highly verified software that could be used for fault detection and recovery of other portions of the kernel which, in turn, would be responsible for recovery of the balance of the system.

The discussion of software fault tolerance techniques in the following sections proceeds in the order of increasing effectiveness and computing resource requirements. Only the most effective and therefore costly fault tolerance techniques should be utilized where a failure can cause complete termination of further program execution. Where failure can only cause faulty displays which

will be readily recognized by the operator who has continued ability to call for the display of other screens, only the less costly measures will be considered.

2. PARTIAL SOFTWARE FAULT TOLERANCE TECHNIQUES

Partial software fault tolerance techniques have the capability to minimize rather than totally mask a fault. They detect and recover from faults without resorting to an alternate, redundant version of the function in order to produce an acceptable result. Thus, partial software fault tolerance techniques are nonredundant fault tolerance provisions. The two general classes are:

- Robustness: Substitution of an alternate value and continuation of execution if a software fault is detected.
- Rollback: Retrying the calculation in the event that a failure is detected, under the assumption that some external condition may have changed thereby resolving the anomaly.

2.1. Robustness

The Data Analysis Center for Software in Rome, New York, defined Robustness as the ability of the code to perform despite some violation of the assumptions in its specifications (DACS, March 1979). An example is the proper handling of out-of-range inputs without degrading the performance of functions not dependent on that input. Under such conditions, a new input may be requested (particularly if it can be supplied by a human operator), the last acceptable value can be used, or a predefined default can be assigned. In all cases, a program flag is set to notify the operator of a program exception state and to facilitate handling of the exception condition by other program elements.

Most of the techniques described as "self-checking" software are included in this definition. Self-checking features include (Yau and Cheung, 1975):

- Functionality: Range or reasonableness checks on the output.
- Control sequences: Setting an upper limit on loop iterations.
- Input data: Parity and type checks.

A distinctive feature of robustness is the protection it provides against predefined causes of software problems. An advantage of this specificity is that errors are usually detected before they can contaminate related programs or data sets. It can therefore be referred to as a technique of forward recovery. Robustness cannot usually be depended on to provide complete protection against faulty algorithms or implementations because of the possibility of faults for which no checks were incorporated.

2.2. Rollback

Rollback consists of the following three elements: an error detection mechanism, a protective action, and a recovery cache.

The recovery cache is the specific feature which distinguishes rollback from robustness. Error detection mechanisms can be input or output assertion checks, accounting tests, range checks, reasonableness tests, hardware alarms (e.g., divide by zero, overflow), and timeout interrupts. When the detection mechanism senses an exception condition, it usually sets a flag to notify the operator and to facilitate handling of the exception condition by using other programs. The usual protective actions are rollback to the beginning of the affected module or restart of the entire program. The recovery cache contains the data necessary for reexecution of the program or segment. During execution the program writes to a temporary memory section. The newly written values are accepted as permanent, replacing those in the recovery cache after the program has passed all checks. This methodology also prevents failures in one section of the code from contaminating other data. For these reasons, rollback has also been referred to as fault containment (Hecht and Hecht, 1986).

Rollback is effective when the failure is triggered by a temporary and unusual data value or computer control state. As the name implies, it is primarily aimed at limiting the propagation of faulty data. Rollback is a broader technique than robustness, but it does not always assure that the program will continue to run after an error has been detected. If the failure-inducing condition is short lived, the program can be expected to resume normal operation after a rollback or restart or in the next iteration. However, if the condition persists, the restoration of service will not be immediate or automatic. Because the technique depends on a recovery cache, it is a backward recovery technique.

2.3. Issues in Partial Fault Tolerance

Three issues to consider in the course of certification of partial fault tolerance provisions for flight critical systems are verification, performance, and coverage.

One concern of verification of robustness provisions relates to the substituted value. First, is it available, particularly when the source of that value is a real-time input such as a sensor or an operator? Second, does the substitute value allow a resumption of execution without any further downstream processing anomalies under all circumstances?

Another concern is the effectiveness of the fault detection provisions. These provisions, typically assertions in the code, must work for all the anticipated conditions and must not spuriously trigger the substitution when it is not called for. Because robustness can be relied on only for anticipated fault conditions, additional system-level analyses such as Failure Modes and Effects Analyses (FMEAs) and fault tree analyses should be conducted to ensure that major problems have not been overlooked.

The verification issues also apply to rollback. An additional concern is the reliability of the recovery cache and ensuring that data are not corrupted. The recovery cache mechanism typically involves portions of the operating system, memory management scheme (that may include hardware as well as software), and application software procedures that access the cache.

For code to be robust, the substitute value must be available within the time frame required. If this value is contained in the processor memory, then it will be available as required. However, if input from a sensor, operator, or external computer is required, then this input must occur within the required time frame, or a secondary source for substitute values must exist.

For rollback and retry, the recovery action is to restart the calculation. If the calculation is time-critical, this strategy can be problematic. In considering the use of rollback and retry, the designer and verifier must be aware of performance and response time requirements. Reasonable assurance is needed that these requirements can be met even if several retries are necessary before a result is produced. In some cases, an upper limit on the number of retries may be necessary, and a default or other value used when this upper limit is reached. A secondary concern is that nested rollback and retry provisions may cause a propagation of retries (Hitt and Prater, 1987) which could in turn result in a major system processing anomaly.

Coverage is the probability that when a fault occurs, it will be detected and recovery from the fault will be successful. Partial fault tolerance provisions provide error detection and recovery capabilities primarily for anticipated faults. An overriding question is the adequacy of fault detection mechanisms and recovery provisions. In the case of a sensor-reading, conversion, and display task which may be repeating several times per second, the argument for adequacy is fairly clear; even if the provisions fail, new data and a new display will be generated immediately. However, in other cases, e.g., in the scheduling or dispatching portions of an operating system, they may not be.

3. N-VERSION SOFTWARE

N-version software, also referred to as multiversion software, relies on voting as the fault tolerance mechanism. Section 3.1 describes the technique. Section 3.2 lists issues of concern to those involved with the validation and certification of this software in flight critical systems.

3.1. Description

N-version software uses several independently coded versions of a specified function running in parallel (at least conceptually) (Avizienis and Kelly, 1984). Their output is voted upon, and the majority answer is used as the result. The hardware analogy is passive redundancy, i.e., having n components in parallel with a voter circuit that processes all outputs. The basic strength of the approach is that the code need not be absolutely correct. If enough versions of a critical function have been written, high reliability can be achieved through logical redundancy just as hardware reliability is achieved with physical redundancy.

Figure 3.1-1 shows how a quadruple-version program might be implemented for a quadruple (hardware) redundant flight control system based on work done at Lockheed Georgia (Mulcare and Barton, 1987). Each task would run on a different processor. Input data are placed in one or more buffers. (There may be a single buffer if the four processors can access a global memory). The operating system initiates the execution of the four tasks. As the results are completed, they are passed onto the voting task which assembles the output and determines the majority value. In the quad-redundant configuration shown here, the voter must also have an algorithm for breaking 2-2 ties. For diagnostic purposes the voting task also tracks any disagreements; if a consistent pattern emerges, it notifies the operating system. The operating system then takes an appropriate action, e.g., performing diagnostics and configuring out a processor if versions running on the processor are consistently outvoted. In flight control, where mission lengths typically do not exceed more than 20 hours, a more prudent approach may be to simply retire the processor.

There have been many research projects on N-version programming in addition to the work cited above (Avizienis and Kelly, 1984, and Knight and Leveson, 1986). Two-version programs with a comparator rather than a voter have been certified for flight control and are being used on passenger aircraft. The Airbus A310 slat and flap control systems (Martin, 1982, and Hills, 1985), the Boeing 737-300, 757, and 767 (Yount, 1984), and the Airbus A320 flight control system are all operational examples of 2-version software. For these applications, the primary benefits have been as follows (Yount, 1984):

- More efficient testing: A side-by-side comparison of the two versions aids in the identification of anomalous result.

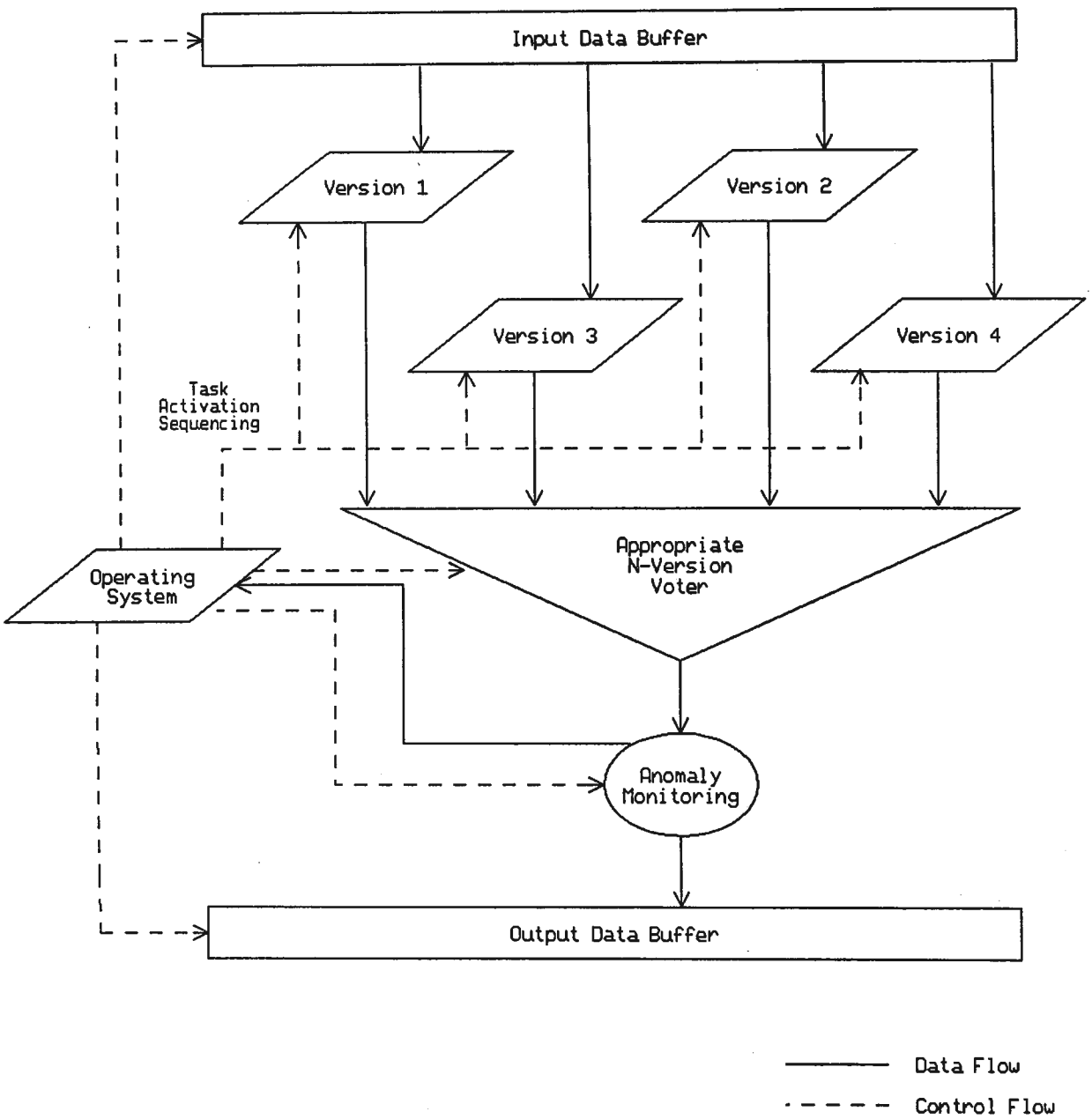


FIGURE 3.1-1. A 4-VERSION SOFTWARE IMPLEMENTATION FOR FLIGHT CONTROL

- Easier certification: The FAA gave tangible credit to Boeing for the use of diverse software.

- Acceptable system reliabilities: With data gathered from 15 airlines, the design Mean Time Between Failures (MTBF) of more than 10,000 hours for the slat and flap control systems was exceeded by approximately 30 percent.

N-version software is a fully fault tolerant technique that provides forward recovery. If a minority of versions are faulty, then the voting process causes their outputs to be disregarded, i.e., masked, and execution proceeds without any delay. This behavior is in contrast to rollback and retry which was discussed earlier or the recovery block which will be described next.

3.2. Concerns

Certification of N-version programs for flight critical systems entails a number of significant issues, including software specification, independence of coding faults, the execution environment, recovery of failed versions, and maintenance of multiple versions.

3.2.1. The Software Specification

In conventional software development environments, many misunderstandings that a programmer may have about the application are cleared up in the course of interaction with other members of the development team. However, because the normative N-version programming framework calls for totally separate development efforts, this important but informal supplement to the normal documentation is lost. Therefore, the software specification in N-version programming is of critical importance. A flawed specification will lead to an unreliable program. Therefore, the specification must be complete, consistent, and correct.

Avizienis and Kelly (1984) suggested that a formal specification language be used. Although such a language can reduce ambiguity, it is more difficult for end-users and system-level developers to verify that the specification is a complete and correct statement of what was wanted. Many specification languages are as complicated--or even more complicated--than programming languages. Hence, even if the specifications are complete and correct, they may not be understood by all development team members and outside specialists unless they have specific training in the particular specification language being used.

3.2.2. Independence of Coding Faults

The fundamental assumption of N-version programming is that coding defects are random and uncorrelated. If this assumption is correct, the likelihood of the same fault existing in all or most modules is very low. Unfortunately, this assumption has not been established by experience. A major problem is that a difficult requirement will result in a greater likelihood that all programmers will make the same type of mistake even if the developments are totally isolated from each other. These mistakes, which have been termed correlated errors or faults, are often subtle and difficult to find.

One example of the correlated fault is a proposed recent airworthiness directive issued by the FAA for the 737-300 flight control system noted earlier. There have been reports of at least 18 uncommanded flight control changes. These uncommanded changes apparently occur once every several thousand times that an aircrew resets the target altitude in the autopilot's altitude select window (Aviation Week, 1988). A second example of a correlated error was that the hardware implementations of floating point arithmetic on the VAX 11/780 computer did not allow for adequate precision for some cases in an experiment performed at the University of Virginia and the University of California, Irvine, in 1985 (Knight and Leveson, 1986).

The problem of independence among versions is compounded by the fact that all versions must produce output in exactly the same format, at the same rate, and with the same precision in order for the voting algorithm to be kept simple and reliable.

One way to relieve some of these drawbacks is the notion of directed N-version design. Instead of relying totally on independence to produce the required degree of diversity, specifications for the different versions could be written to include some algorithmic constraints. These constraints might include the order of operations, specific numerical techniques that may or may not be used, or specified differences in techniques, e.g., table look-up versus direct calculation.

3.2.3. Run-time Environment

As figure 3.1-1 shows, N-version programming involves more than the n programs and a voter. The run-time environment for execution of this software in a flight control system must include provisions for:

- Inputting identical data to all versions.
- Synchronizing all versions; i.e., in order to perform a valid vote, each version of the software must complete within a certain time interval, and all results must be from the current execution frame or time slice.
- Collecting results for input to the voter.
- Routing the results of the voter to the appropriate destination, e.g., an actuator motor or display.
- Data recording and diagnostics so that consistently disagreeing versions--and possibly their associated processors if they run in a multiprocessing environment--are configured out of the operational system for the duration of the mission.

The run-time environment is critical to the correct functioning of the technique and must be protected. If the software and hardware components of the environment are incorrectly designed or become corrupted during a flight an unrecoverable system failure will probably occur.

3.2.4. Recovery of Failed Versions

If the application requires awareness of past events; e.g., for integration or averaging, filtering, or target tracking, a failed version must have its data structures updated before participating in subsequent votes. The nature of the updating mechanism, techniques for performing the update, and synchronization techniques are application dependent. However, their correctness is critical for realizing the potential reliability benefits through restoration of a failed resource.

3.2.5. Maintenance and Upgrades

Questions of maintenance affect errors in a single version and across all versions. In a single version, there is a question of whether a known bug should be altered if it has not been found in other versions. Economic considerations, configuration management problems, and the concern that fixing a known bug can introduce additional unknown problems must all be addressed before a decision is made.

Management of a version-wide modification presents other problems. If the same maintenance coder modifies all versions of the software, independence is lost. On the other hand, it is impractical to have four or five different coders maintaining different versions. The practices of the developer in handling this problem can have a significant impact on the reliability of the software once the development is concluded.

4. RECOVERY BLOCKS

Recovery blocks are a backward recovery technique which incorporate explicit error detection and recovery provisions. In this regard, they are related to the rollback technique described in section 2. However, they include redundant logic in the form of an alternate routine which is used if an error is detected. They are therefore fully fault tolerant. Section 4.1 describes the basic recovery block and its variations. Section 4.2 lists concerns related to their validation in flight-critical software.

4.1. Description

Recovery blocks were first described by B. Randell of the University of Newcastle in 1975 (Randell, 1975). The three elements of the basic recovery block are:

- The primary routine which implements a given function or program specification.
- The acceptance test which determines that the requirements of the function or specification have been met after each execution of the primary routine.
- One or more alternate routines which perform the same function as the primary routine using an independent algorithm. The alternate routines are invoked if the acceptance test condition is not met.

The basic structure of a recovery block with only a single alternate routine is shown in figure 4.1-1. Randell stated it formally as:

```
Ensure AT

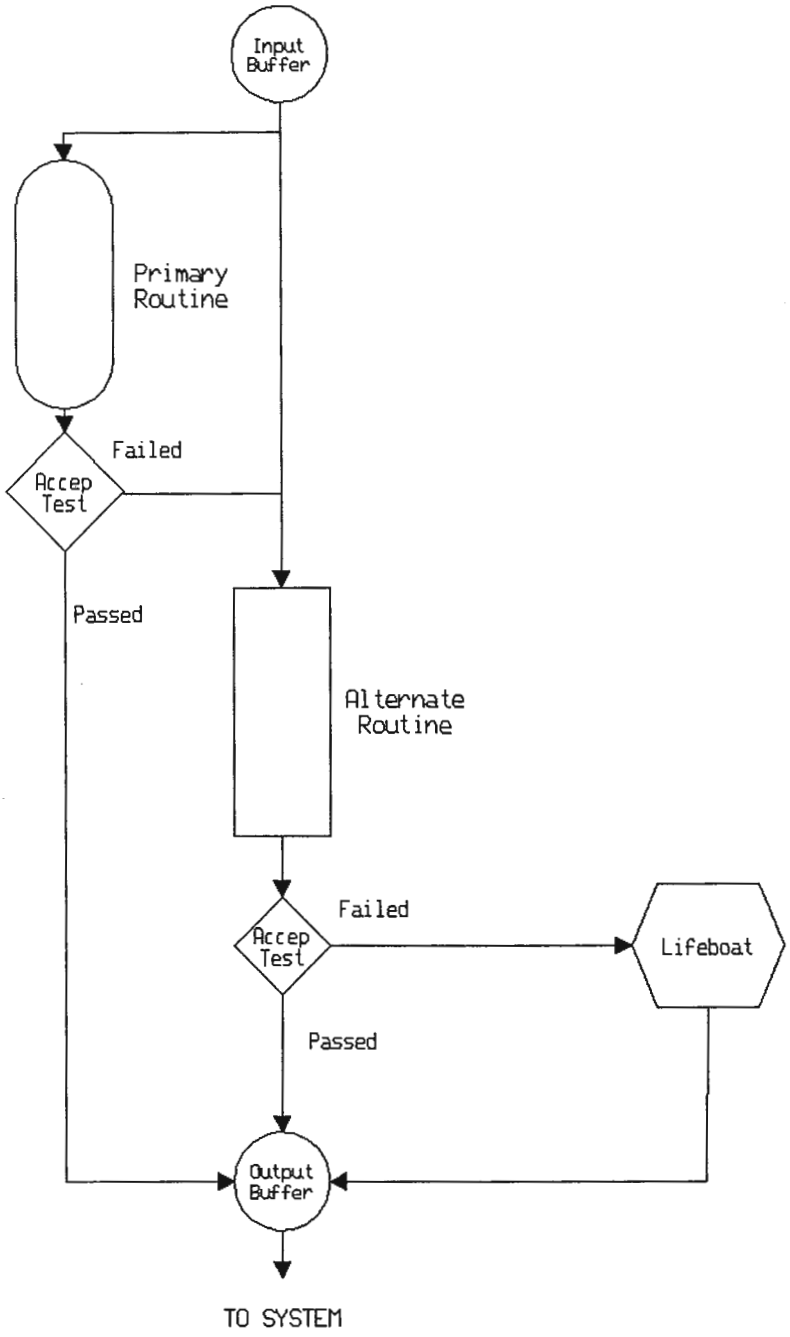
        By P
        Else by Q

Else Abort
```

Where AT is the acceptance test, P is the primary routine for a given function, Q is the alternate, and the Abort condition invokes a higher level recovery procedure. When more than one alternate is provided, additional 'Else by...' statements (i.e., Else by R, Else by S, etc.) follow 'Else by Q'. Until the acceptance test is passed, the input data are protected in their original state by a recovery cache in a manner similar to the fault containment structure described in section 2.

Important features of the recovery block are:

- Software outside the recovery block is unaffected by which routine (primary or alternate) furnished the satisfactory result.



FIGUR E 4.1-1. BASIC RECOVERY BLOCK

- The alternate routine is executed only upon a failure of the primary routine. Thus, the alternate can use inefficient but highly failure-resistant code; this avoids a run-time penalty under normal conditions.

- The recovery block allows for a subsequent retry of the primary routine. Software failures result from the inability of the program to process a specific set of input data when in a certain computer state. Therefore, it is undesirable to make a permanent switch to the backup module after only an isolated failure in the primary routine. Returning to the primary may be desirable because of greater efficiency, versatility, or confidence. A simple diagnostic procedure can be used to switch the roles of the primary and backup modules when repeated failures of the primary occur.

- Failures of individual routines are visible from the state of a flag set by the acceptance test AT. Therefore, online monitoring and testability are readily incorporated into the recovery block. The technique also permits capture of the data associated with each failure. Hence, complete diagnostic information for later improvements of software can be made available.

In order to implement recovery blocks effectively, the primary routine, alternate routine, and acceptance test must be deliberately independent. This approach contrasts with N-version programs where independence is usually assumed to be implicit by virtue of the independent development of the versions. Deliberate independence can be achieved in a variety of ways. In mathematical and logical operations, the basic commutative, associative, and distributive properties can be used to change the order of calculation. In systems which control physical processes, it may be possible to derive a result from several different physical inputs.

In a demonstration of recovery blocks for use in flight navigation (Hitt and Prater, 1987), independence was achieved by removing some of the functionality in the primary routine. However, because no alternative to the fundamental form of the navigation equations was available, an approximation of the sine and cosine functions was used in the alternate version; the primary relied on library trigonometric routines in the Ada compiler that was being used.

Acceptance tests can often be designed to check for a plausible result rather than a correct result. For example, data processed from sensors and sent out to actuators can be checked for continuity; i.e., the difference between the value of a previous iteration and the current value should be less than some stated limit.

An additional consideration for real-time applications is to ensure that computations are completed in the required time. One possible approach is the inclusion of a "watchdog timer" which forces entry into a recovery condition if the acceptance criteria are not met within a given time. The structure of such a recovery block is shown in figure 4.1-2.

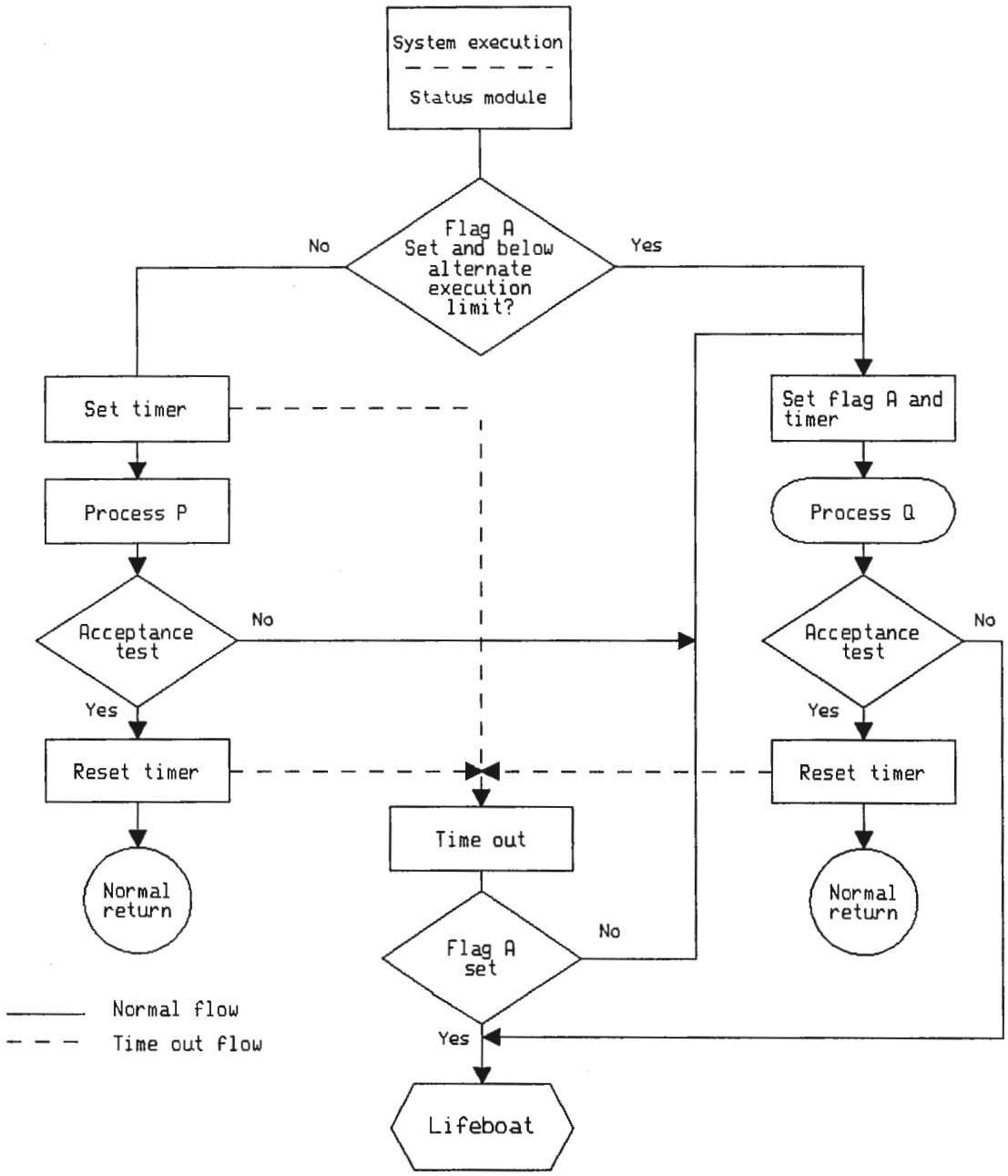


FIGURE 4.1-2. RECOVERY BLOCK FOR REAL-TIME APPLICATIONS WITH A SINGLE WATCHDOG TIMER

Before entering the recovery block, calls to the primary and alternate routines are prepared, and the allocated execution time (including an allowance for the acceptance test) is loaded into the watchdog timer. The primary call is then executed and the result of the processing is subjected to the acceptance test. If passed, a normal return results. If not passed or if the watchdog timer interrupts the primary process, the alternate routine is executed and the acceptance test is run on its results. Once again, a normal return will take place if the results pass the acceptance test. However, if the alternate routine also does not produce an acceptable result or if an alternate routine time-out occurs, then an abort-return results and recovery must take place on a higher level.

Figure 4.1-3 (Hitt and Prater, 1987) shows the flow chart for a recovery block implemented with two watchdog timers: The first is for the "minor frame," i.e., the time allotted for execution of the total recovery block. The second is for each version of the software. The function of this second watchdog timer is similar to the single watchdog timer example described in the previous paragraph.

An additional variation on recovery blocks is nesting. In a nested recovery block, a failure of the lower level primary and alternate would cause a return to the higher level recovery block with a failure indication. The higher level recovery block would then invoke its alternate higher level routine. The advantage of such a structure is that it provides additional software redundancy, and that a single higher level alternate routine may substitute for a number of secondary or tertiary alternates on a lower level. The primary cost is a performance penalty because rollback to an earlier stage in execution must occur to effect a recovery.

When both the primary and alternate routines are implemented on a single processor, the performance penalty entailed by rolling back and restarting the calculation with an alternate routine using data in the recovery cache may be unacceptable. However, because most future digital flight control designs will have several processors, it is possible to use a Distributed Recovery Block (DRB) (Kim 1984) as shown in figure 4.1-4. In the DRB, each processor contains its own copy of the primary and alternate. The processor in control of the system executes the primary routine and the backup processor(s) execute(s) the alternate routine(s) in parallel. All processors run the same acceptance test. Thus, if the primary processor fails, the backup processor's results are already available with no time lost for the retry. The use of different processors with different versions of the software unifies both hardware and software fault tolerance into a single framework of distributed systems fault tolerance. This integration provides protection against a broader scope of problems (i.e., both hardware and software failures) and has significant performance advantages. Additional research should be pursued on issues such as identifying the actions to be taken following a failure and integrating hardware status checks into the DRB (Hecht, 1988).

There is little data available on the effectiveness of recovery blocks in operational systems. However, early results indicate that increases in software MTBF on the order of two to three times could be achieved in an academic setting (Anderson, et al., 1985). The authors suggest that significantly higher

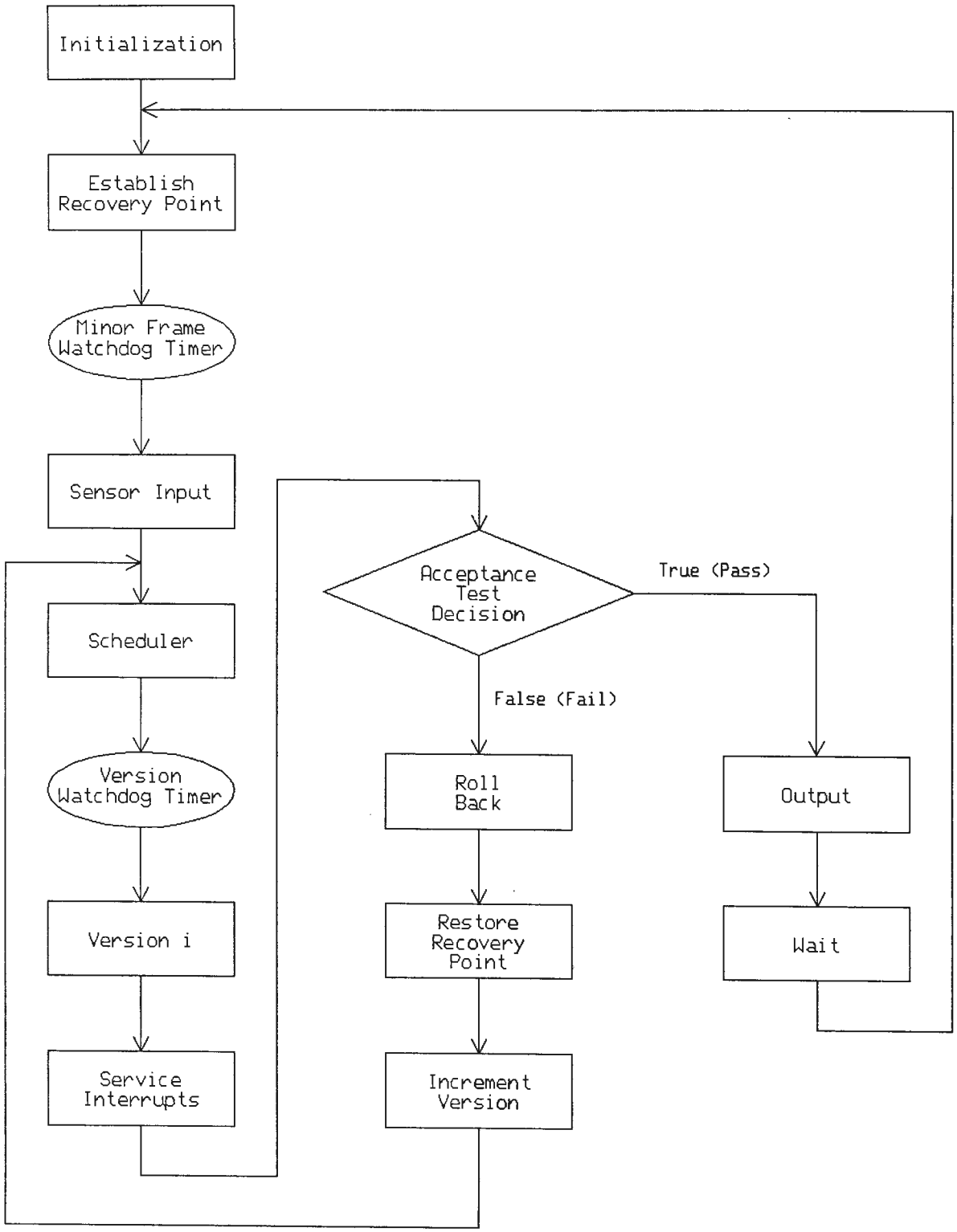


FIGURE 4.1-3. A REAL-TIME RECOVERY BLOCK WITH TWO WATCHDOG TIMERS

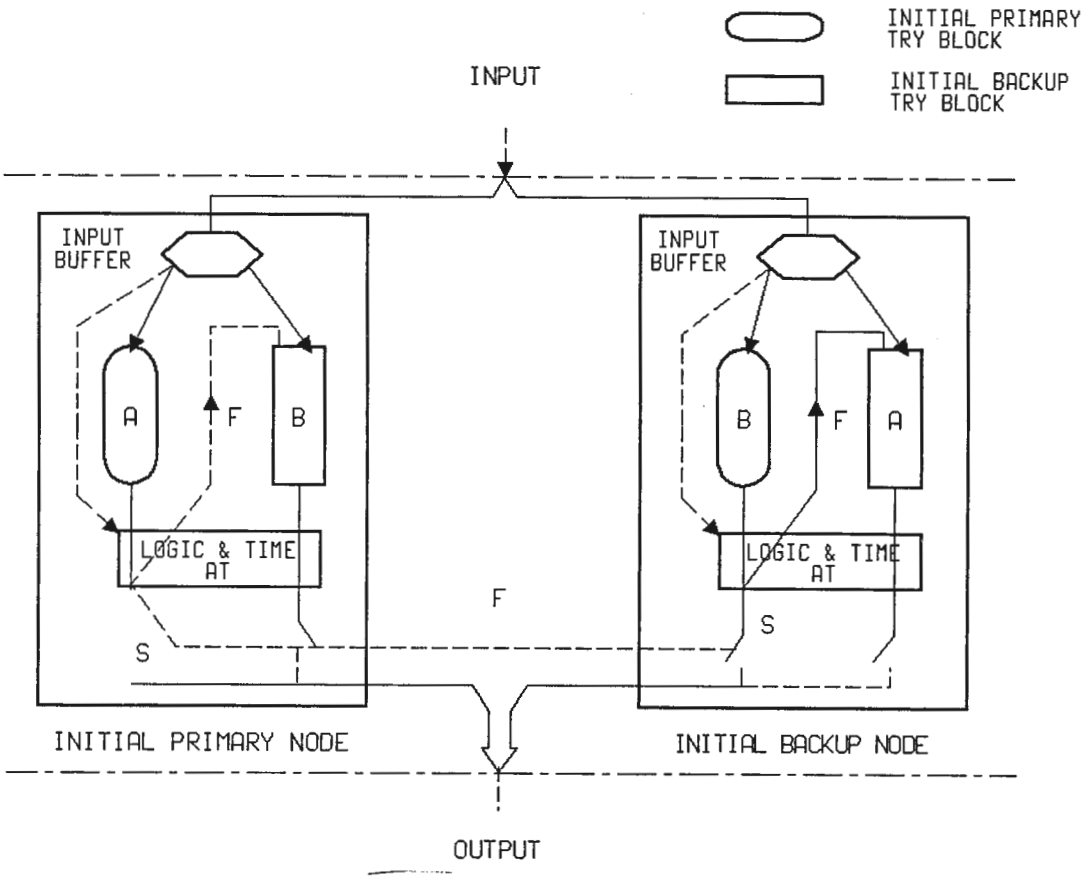


FIGURE 4.1-4. DISTRIBUTED RECOVERY BLOCK

reliability increases can be expected in a conventional software production setting; system-unique factors were cited as underestimating the true effects of software fault tolerance in this particular experiment.

4.2. Critical Issues

Critical issues for recovery blocks used in flight critical software include the correctness of the acceptance test, reliability of the underlying operating environment, and independence of the alternate routine.

4.2.1. Acceptance Tests

The acceptance test is a single point of failure in the recovery block. The two possible failures are: First, that the acceptance test accepts an incorrect result. Second, that the acceptance test rejects a correct result. The consequences of both failures can be serious. For example, failure to detect erroneous navigation data may result in an aircraft flying into the flight path

of another aircraft, an unauthorized airspace, or a military conflict area. The acceptance test's spurious rejection of correct results by both the primary and the alternate routine will result in a cessation of computation and a total failure of the recovery block.

The most effective means of minimizing this risk is fault avoidance in the acceptance test. Specifically, the acceptance tests should be very simple and amenable to thorough verification. The need to create simple, verifiable acceptance tests should be considered when designing the construction and modularization of this system. Further considerations in the use of fault avoidance versus fault tolerance are listed in section 5.

A second method of reducing the risk of defective acceptance tests is through nesting. Mitigation of the non-detection risk occurs because a faulty result may cause an acceptance test failure in subsequent calculations. This, in turn, can cause a rollback to an alternate routine which resumes the calculation using data unaffected by the fault. Mitigation of the spurious rejection risk occurs because the alternate path will use a different set of acceptance tests than was used on the primary path. However, there is a significant performance penalty in nesting. Extensive rollback can cause response time requirements to be missed. This performance penalty may not be acceptable in unstable aircraft because an unrecoverable situation can occur if the control system does not react in time.

4.2.2. Run-time Environment

As is the case for all other software fault tolerance techniques, there is a dependence on the underlying operating system and run-time provisions. Concerns for recovery blocks are: maintenance of the Recovery Cache, invocation of the Alternate Routine, and maintenance of the Watchdog Timer.

Where the mechanisms for these functions are sufficiently simple, they can be subject to exhaustive verification and considered to be part of the protected, unalterable, "hard core" of the operating system. In other cases, these mechanisms can in and of themselves be made fault tolerant and rely on simpler mechanisms contained within an operating system hard core.

4.2.3. Independence of the Alternate Routine

The type of degree of independence required in the alternate routine is determined largely by the nature of the fault classes which are anticipated. In the recovery block implementation for the navigation system cited above, independence was achieved by avoiding use of library procedures in the alternate routine. This measure ensures that precision or other subtle problems in the libraries will not affect the result. Because the alternate used the same equations as the primary, however, the developers needed to ensure that these equations would not be a source of error.

In other cases, it may be acceptable to use the same underlying library routines in both versions while using different equations or algorithms. This measure is appropriate if the developers are concerned about the appropriateness of the algorithms. A third measure is to use different languages and different

libraries for the primary and alternate. The availability of a reliable linking and calling mechanism alleviates the concerns of defects in libraries and compilers.

Definition of the degree of independence should be based on system-level reliability analyses performed as a part of the design process. During the validation of flight control systems, the analyses, the independence requirements derived from these analyses, and conformance with these requirements must be performed.

5. FAULT AVOIDANCE VERSUS FAULT TOLERANCE

Fault tolerant software provisions have costs in terms of development time, processing overhead, and recovery time. In addition, these provisions rely on underlying systems software or firmware in order to function properly. Thus, fault tolerance is not a substitute for fault avoidance, i.e., traditional software development practices designed to eliminate coding defects. These practices include:

- Software Development Process Controls including formal requirements generation, design methodologies, structured programming reviews, inspections, and configuration management.

- Software Testing at the unit, Computer Software Component (CSC), and Computer Software Configuration Item (CSCI) level.

- IV&V of deliverables, procedures, practices, and standards.

Ideally, stringent controls during the software development process can ensure that the entire project team works from a single design, uses well-defined interfaces, follows consistent variable naming and typing conventions, and develops adequate documentation. However, while such control can prevent faults due to a lack of communication or coordination among various team members, it cannot guarantee that subtle (or not so subtle) flaws in the specification or detail design will not result in faulty code. These design flaws are most likely to occur when timing, synchronization, exception handling, input/output, or interprocessor communication are involved.

Software testing can occur at the unit, component, and configuration item level. Strategies used in testing include branch testing, path testing, testing for singularities of critical variables, independent testing, and use of automated test tools. An important test methodology that has not received much attention in the literature is stress testing, which is deliberately exposing the program to faulty data sets or higher workloads than called for in the specification. Although still subject to the general limitations of testing, stress tests will frequently expose performance deficiencies which are not addressed by other strategies.

Code verification compares the detailed design with the actual code and devises tests and inspections that indicate compliance. Validation is done over a larger scope of system development, e.g., comparison of the delivered code with the A-specification or a lower level requirements document. For large real-time systems in critical applications, it is current practice to have a separate organization perform the IV&V. Usually, such IV&V activities are performed using standard procedures and sampling methods that are intended to provide a reasonable but by no means complete demonstration that the code fulfills its requirements.

Examples of where fault avoidance measures are either necessary or more efficient than real-time fault tolerance provisions include:

- Critical operating system software: Software that manages the invocation of fault-tolerant tasks; i.e, queues, buffers, interrupts, and timers must function flawlessly in order for the fault tolerance provisions to work effectively. Modules which perform these functions are generally large and may be amenable to the inclusion of fault tolerance provisions. However, there is still the need for total verification of these fault detection, isolation, and recovery provisions. Normal verification techniques described above are unsuitable for these critical code sections. More intensive techniques such as the Enhanced Condition Table (Tai, Hecht, and Hecht, 1987) together with extensive testing on the target system are generally needed.

- Constants: Faults in constants embedded in the code (e.g., size of memory or number of mailboxes for interprocessor communication) cannot easily be detected by fault tolerance provisions but can be readily found during pre-operational verification activities. System-level reliability analyses such as FMEAs and fault-tree analyses can be used to identify critical constants which must be verified. Utility programs or test tools can be built to perform this verification automatically or semi-automatically.

- Simple sections of code: While verification of entire real-time software systems is impractical, it may be possible to totally verify certain small sections of code--particularly if their input domains and operational environments are well-defined, and there are no timing or communication dependencies. Even when fault tolerance provisions can be defined, run time penalties and the need for additional exception handling frequently make them less desirable alternatives to fault avoidance.

- Static data: Many initiating events in software failures are related to input data. While much of the data in a flight control system is dynamic, there are significant portions which are static, e.g., geographical positions and aircraft parameters. As was the case for constants embedded in the code, verification and automated tools can be an effective and efficient means of detecting and correcting faults in such data.

On the other hand, fault avoidance provisions cannot be relied upon to provide complete coverage when the state of the system cannot be determined in advance. Examples include:

- Interprocessor communication: Requirements for interprocessor timing, sequencing, and message processing (whether on the same bus or from another network node) are generally not totally predictable during the design phase. Thus, it is generally impossible to develop a complete set of test cases that demonstrate correct functioning of the communication and message processing software. Under these circumstances, fault containment is a better approach. Data from interprocessor communication can be checked for integrity using error coding techniques, for reasonableness using range

checking or wraparound tests, and for security using various encoding schemes. Failure to pass these tests can cause a retry, substitution of default values, or use of previous values.

- Input/output: As was the case previously, the timing, content, and format of incoming messages cannot be predicted with sufficient certainty that an exhaustive set of test vectors can be produced. Under such conditions, online assertion checking such as that contained in fault containment or recovery blocks will constitute a more reliable approach.

- Critical response times: When it is difficult or impossible to ensure that all response time requirements will be met, a recovery block (which includes a watchdog timer as part of the acceptance test) is generally necessary.

6. INTEGRATION OF FAULT TOLERANCE TECHNIQUES

The previous sections have described critical issues and limitations associated with specific techniques. However, many of these limitations can be overcome by judicious combination of different fault tolerance techniques and the use of fault avoidance. Table 6-1 lists considerations that affect the choice of methods. The following subsections discuss the specific considerations in greater detail.

6.1. Coverage

Most robustness provisions address specific fault types, such as data out of range or improper arguments passed to a program. Fault containment and recovery blocks can use reasonableness or other types of acceptance tests which provide much broader scope and do not require prior knowledge of the expected faults. N-version programming provides the broadest coverage because its only limits are the common assumptions used in the specification or the design of all versions.

6.2. Retry Requirements

Retry of the same program always entails a possibility that the failure will be repeated in subsequent trials. Retry also impacts timing and throughput requirements. As long as no errors are encountered, the additional execution time is expected to be small; it is limited only to the assertion check at the end of the routine. However, when errors are encountered, the execution time may increase considerably--perhaps beyond the acceptable limits for time-critical applications. Because of the rarity of exception cases, average execution time is hardly affected by this consideration.

Failures covered by rollback can be resolved by assignment of default values or similar forms of forward recovery. Thus, retry is not always necessary using fault containment or robustness techniques.

By way of contrast, recovery blocks always use a new program if there is a failure. Similarly, use of N-version code does not require a retry because the final output is determined by a majority vote. Thus, for critical timing requirements or where fully fault tolerant software is necessary, the preferred techniques are N-version software or recovery blocks.

6.3. Cost and Complexity

Cost factors are related to the introduction of additional code including the development and maintenance expense of additional code and resources utilized during execution. Additional code introduces complexity which in turn increases the likelihood of added faults. Because robustness and rollback require

relatively little additional code, fault tolerant development costs represent only a minor fraction of the total cost for the delivered code. So long as retry is acceptable, these two techniques also impose the lowest overhead.

TABLE 6-1. SOME RELEVANT CHARACTERISTICS OF FAULT TOLERANCE TECHNIQUES

Techniques	Coverage	Retry[1]	Cost and Complexity
Robustness	Narrow (specific to anticipated faults)	No (default or alternate is used)	Generally low
Rollback	Can be broad (determined by assertion)	Usually (default may be used)	Low to medium
Recovery Block	Broad (determined by acceptance test)	No (handled in alternate routine)	High
N-version software	Very broad (limited only by assumptions in the specifications common design assumptions	No (N-version run once)	High to very high

[1]Whether a retry of the same code which induced the fault initially is necessary.

When the recovery block is used, at least three independent sections of code (the primary routine, the alternate routine, and the acceptance test) have to be designed, coded, tested, and maintained. A considerable increase in cost and complexity is therefore incurred. In most cases, the acceptance test is small relative to the code being tested and hence, run-time overhead is low. However, the alternate routine may be significantly less efficient than the primary routine when an error condition occurs, and repeated execution of the entire recovery block (i.e., primary, acceptance test, and alternate routine) may result in a major throughput penalty. The likelihood of such an occurrence should be assessed as part of the design process.

In N-version software cost can be even higher because (1) multiple versions must be developed separately and (2) special development of specifications may be necessary to ensure that programmers can work independently and produce

compatible results. During run-time, throughput is limited to the slowest of the routines. If the technique is practiced using true independence, it may not be possible to know the extent of this run time penalty until after completion of the coding when it may be too late. This run-time risk must be assessed as part of the software design because it may ultimately be the major cost driver in the development.

A major cost common to all fault tolerant techniques is verification during both development and maintenance. The verification of recovery provisions is generally difficult, and the major function of fault tolerance is to provide for recovery. For fault containment and recovery blocks, the goal of verification is to ensure that the acceptance test correctly identifies all error conditions and does not spuriously identify an acceptable result as a failure. For N-version programming, verification must be performed on all versions of the routine as well as on the voter for all possible results. During maintenance, thorough testing is necessary in order to ensure that coding changes do not defeat fault tolerance measures that are already in place.

Because of the high cost associated with fault tolerant software development, maintenance, and verification, its use should be restricted to functions critical to the continued safe operation of the system.

7. CONCLUSION

In order to achieve the very high reliability required in flight-critical systems, traditional software development methods are insufficient. Fault tolerance provisions can provide the additional reliability by detecting and recovering from faults occurring within the software before they affect aircraft operation and safety. However, in order to be effective, decisions on which fault tolerant techniques to use, where they are to be applied, and how they are to be verified must be made early in the design process and in the framework of the appropriate system-level analyses such as FMEA and Fault Tree Analysis.

The proper choice of fault avoidance versus fault tolerance, and judicious application of the appropriate fault tolerance techniques will result in a maintainable, cost effective, reliable, and high performance system. Those involved in the validation and certification of flight-critical digital systems should ensure that these analyses have been performed, and that they take into account the strengths and weaknesses of the techniques that have been described in this chapter.

BIBLIOGRAPHY

Anderson, T., et al., Software Fault Tolerance: An Evaluation, University of Newcastle upon Tyne, Technical Report No. 202, September, 1985.

Avizienis, A. and J. P. J. Kelly, "Fault Tolerance by Design Diversity: Concepts and Experiments," Computer, Vol. 17, No. 8, pp. 67-80, August 1984.

Chen, L. and A. Avizienis, "N-Version Programming: A Fault Tolerance Approach to Reliability of Software Operation," Digest of Papers, FTCS 8, pp 3-9, June 1978.

Data and Analysis Center for Software (DACS), "Quantitative Software Models," DACS SRR-1, March 1979.

Davis, E. A. and P. K. Giloth, "No. 4 ESS: Performance Objectives and Service Experience," Bell System Technical Journal, Vol. 60, No. 6, pp. 1203-1224, August, 1981.

Dijkstra, E. W., "Notes on Structured Programming," Structured Programming, Academic Press, New York, 1972.

Harvey, P. R., Fault Tree Analysis of Software, Thesis for the Master of Science in Information and Computer Science, University of California at Irvine, 1982.

Hecht, H. and M. Hecht, "Use of Fault Trees in the Design of Recovery Blocks," Proceedings 1982 Fault Tolerant Computing Symposium (FTCS-12), Santa Monica, CA, June, 1982.

Hecht, H. and M. Hecht, "Fault Tolerant Software," Fault Tolerant Computing, D. K. Pradhan, ed., Prentice Hall, Englewood Cliffs, NJ, 1986.

Hecht, M., "Extended Distributed Recovery Blocks," Proceedings of the Annual National Joint Conference on Software Quality and Reliability, National Security Industrial Association, Arlington, VA, March, 1988.

Hills, A. D., "Digital Fly-by-Wire Experience," Proceedings AGARD Lecture Series No. 143, October, 1985

Hitt, E. F. and S. A. Prater, Navigation Recovery Block Design Description, Federal Aviation Administration Technical Center, DOT/FAA/CT-87-15, Atlantic City, 1987.

Howden, W. E., "An Evaluation of the Effectiveness of Symbolic Testing," Software-Practice and Experience, Vol. 8, pp. 381-397, John Wiley & Sons, 1978.

Kelley, J. P. J., "A Specification Oriented Multi-Version Software Experiment," Proceedings of the 13th International Conference on Fault Tolerant Computing (FTCS-13), Milan, Italy, 1983.

Kim, K. H., "Distributed Execution of Recovery Blocks: An Approach to Uniform Treatment of Hardware and Software Faults," Proceedings of the 4th Annual International Conference on Distributed Computing Systems, pp. 526-532, IEEE Cat. No. 84CH2149-3, May, 1984.

Knight, J. and N. Leveson, "An Experimental Evaluation of the Assumption of Independence in Multiversion Programming," IEEE Transactions Software Engineering, Vol. SE-12, No. 1, pp. 96-109, January, 1986.

Martin, D. J., "Dissimilar Software in High Integrity Applications in Flight Control," Proceedings AGARD CPP-330, September, 1982.

Miller, E. F., Jr. et al., "Application of Structural Quality Standards to Software," Software Eng. Stand. Appl. Workshop, IEEE Cat. 81CH1663-7, pp. 51-57, July 1981.

Mulcare, D. B. and L. A. Barton, N-Version Software Demonstration for Digital Flight Controls, Federal Aviation Administration Technical Center, DOT/FAA/CT-86/33, April, 1987.

Randell, B., "System Structure for Software Fault Tolerance," IEEE Transactions on Software Engineering, Vol. SE-1, No. 2, pp. 220-232, June 1975.

Tai, A., M. Hecht, and H. Hecht, "Enhanced Condition Table Method for Verification of Critical Software," Proceedings of the 11th Annual International Computer Software and Applications Conference (COMPSAC87), IEEE Computer Society Press, pp. 317-323, October, 1987.

Yau, S. S. and R. C. Cheung, "Design of Self-Checking Software," Proceedings 1975 International Conference on Reliable Software, IEEE Cat. 75CH0940-7CSR, pp. 450-475, April 1975.

Yount, L. J., "Architectural Solutions to Safety Problems of Digital Flight Critical Systems for Commercial Transports," Proceedings AIAA/IEEE Digital Avionics Systems Conference, Long Beach, CA, December 1984.

GLOSSARY

A-SPECIFICATION. The highest level specification typically produced by the contracting organization to define a system (see MIL-STD-1521).

BACKWARD RECOVERY. Restoration of the system to some previous known correct state and restarting the computation from that point.

COVERAGE. The probability that when a fault occurs, it will be detected and recovery from the fault will be successful.

FAULT AVOIDANCE. The attempt to prevent any software faults in the final delivered product through disciplined software development practices, testing, and IV&V.

FAULT CONTAINMENT. The capacity of a system to prohibit errors and/or failures from propagating from the source throughout the system.

FAULT TOLERANCE. The capability to endure errors and/or failures without causing total system failure.

FAULT TOLERANCE. Software which continues to operate satisfactorily in the presence of faults.

FORWARD RECOVERY. Restoration of the system to a consistent state by compensating for inconsistencies found in the current state so that the system may continue processing.

GLASS COCKPIT. Advanced state-of-the-art electronic displays utilizing flat panel and/or cathode ray tube display technology for cockpit instrumentation.

RECOVERY CACHE. The location used to preserve input values until the outputs resulting from them have been accepted.

ROBUSTNESS. The ability of the code to perform despite some violation of the assumptions in its specifications usually via substitution of an alternate value and continuation of execution if a software fault is detected.

ROLLBACK. Retrying the calculation in the event that a failure is detected, under the assumption that some external condition may have changed thereby resolving the anomaly.

ACRONYMS AND ABBREVIATIONS

AIAA	American Institute for Aeronautics and Astronautics
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
DRB	Distributed Recovery Block
ESS	Electronic Switching System
FMEAs	Failure Modes and Effects Analyses
IEEE	The Institute for Electrical and Electronics Engineers, Incorporated
IV&V	Independent Verification and Validation
MTBF	Mean Time Between Failures

U.S. Department of Transportation
Federal Aviation Administration

DOT/FAA/CT-88/10

HANDBOOK-VOLUME II
DIGITAL SYSTEMS VALIDATION

CHAPTER 10
LATENT FAULTS



PREPARED BY:

COMPUTER RESOURCE MANAGEMENT, INC.
950 HERNDON PARKWAY, SUITE 360
HERNDON, VIRGINIA 22070

PREPARED FOR:

FEDERAL AVIATION ADMINISTRATION
TECHNICAL CENTER
ATLANTIC CITY INTERNATIONAL AIRPORT, NEW JERSEY 08405

NOTICE

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The United States Government assumes no liability for the contents or use thereof.

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the objective of this report.

COPY 1

ERRATA

Report No. DOT/FAA/CT-88/10

DIGITAL SYSTEMS VALIDATION HANDBOOK - VOLUME II

February 1989

U.S. DEPARTMENT OF TRANSPORTATION
FEDERAL AVIATION ADMINISTRATION
TECHNICAL CENTER
ATLANTIC CITY INTERNATIONAL AIRPORT, NJ 08405

Replace the following pages in Chapter 10:

10-13, 10-14, 10-15, 10-16, 10-17, 10-18, 10-19, 10-20

Released October 1989

(Henceforth, we will assume that there is only a single auxiliary program. If more than one auxiliary program were involved, the complexity of the model would increase.)

The parameters of the model are not independent. In fact,

$$p_1 + p_2 + p_3 = 1$$

$$p_2 = (1 - p_1)\left[\frac{as}{as + ds}\right]$$

$$p_3 = (1 - p_1)\left[\frac{ds}{as + ds}\right]$$

From the results of the FIIS and previous experiments it was expected to obtain estimates of p_1, p_3, e_α, and e_β. The parameters p_2, as, and ds would be obtained from typical FCS scenarios (e.g., duration of and time between initiation of auxiliary programs).

3.2. Parameter Estimates

- p_1: From the Bendix study, $.85 < p_1$; from FIIS, $.96 < p_1 < .98$
- p_2: It is estimated that $p_2 = 0$, approximately
  (With $p_2 = 0$, $1 - p_1$ = proportion of latent faults)
- p_3: From the Bendix study, $p_3 < .15$; from FIIS, $.02 < p_3 < .04$
- e_α: From the FIIS study, $e_\alpha > 7200$/hour
- e_β: From the FIIS study, $e_\beta > 900$/hour
- as: From FCS scenarios, .01/hour < as < 10/hour
- 1/ds: From FCS scenarios, .01 hours < 1/ds < 1 hour
- f: The current range is .001/hour < f < .0001/hour

3.3. Potential Criticality of Latent Faults

In order to assess the effects of latent faults, we present two examples of conventional FCSs which could be vulnerable to the accumulation of latent faults. The examples noted are Triplex FCS (section 3.4) and Quadruplex FCS (section 3.5). The scenario is one in which latent faults, when activated, lead to a rapid and unexpected loss of components. Such a condition can arise in a Quadruplex FCS, where for example, in time, three out of the four lanes develop latent faults. Eventually, due to a single source of excitation, such as the callup of an outer-loop program, the faults become active. Even assuming, as we do, that the time interval between the successive activation of the faults (referred to, hereafter, as "avalanching latent faults") is sufficient to allow the comparators to detect and isolate each fault in turn, the result is a loss

of the three lanes in quick succession. It is our contention that the probability of such an event may not be insignificant.

3.4. Example 1 - Triplex FCS

The conventional reliability analysis, which assumes that there are no faults at the start of each flight, would conclude that the probability of loss of the FCS function in a flight of one hour is $3f^2$.

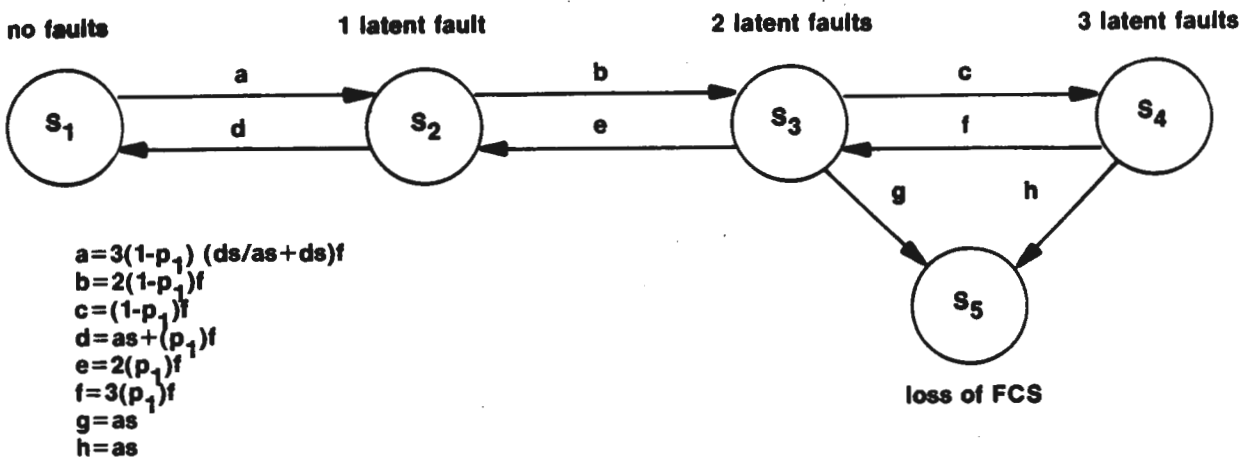


FIGURE 3.4-1. MARKOV MODEL/TRIPLEX FCS WITH LATENT FAULTS

A Markov reliability model of the FCS, representing the effects of avalanching latent faults, is shown in figure 3.4-1. The following assumptions are implicit:

- Loss of control is due, exclusively, to avalanching latent faults.
- Faults are exponentially distributed.
- The time between the successive activation of multiple latent faults is small relative to mission time.
- α-faults (i.e., faults active at their occurrence) are detected immediately as they occur, at which time the entire lane is replaced. Thus, existing latent faults are effectively repaired.

- Excitation of latent faults is assumed to be correlated across lanes (i.e., a single excitation activates latent faults in different lanes).

Referring to figure 3.4-1:

- State S_1 represents the condition that all lanes are fault-free.
- State S_2 has a latent fault in one lane.
- State S_3 has a latent fault in two lanes.
- State S_4 has a latent fault in three lanes.
- State S_5 represents loss of the FCS function.

The transition rates are:

$$a = 3(1-p_1)f\left[\frac{ds}{as + ds}\right]$$

$$b = 2(1-p_1)f$$

$$c = (1-p_1)f$$

$$d = as + (p_1)f$$

$$e = 2(p_1)f$$

$$f = 3(p_1)f$$

$$g = as$$

$$h = as$$

It is noted that the backward transitions (d, e, and f) are the result of repairs due to the detection of α-faults.

Let x_1, x_2, x_3, x_4, and x_5 denote the occupancy probabilities of states S_1, S_2, S_3, S_4, and S_5, respectively. From the figure we obtain the following differential equations:

$$sX_1 - x_1(0) = -aX_1 + dX_2$$

$$sX_2 = aX_1 - (b+d)X_2 + eX_3$$

$$sX_3 = bX_2 - (c+e+g)X_3 + fX_4$$

$$sX_4 = cX_3 - (f+h)X_4$$

$$sX_5 = gX_3 + hX_4$$

with initial conditions, $x_1(0) = 1$, $x_2(0) = x_3(0) = x_4(0) = x_5(0) = 0$. X_1, X_2, X_3, X_4, and X_5 denote the Laplace transforms of x_1, x_2, x_3, x_4, and x_5, respectively, and s denotes the Laplace operator.

Figure 3.4-2 shows the mean failure rate (MFR) (see appendix A), for a range of values of $1-p_1$ and f. The parameters p_2 and ds were fixed at the values $p_2 = 0$ and ds = 10/hour. Sensitivity studies indicate that the MFR is insensitive to values of ds between 1 and 10 per hour.

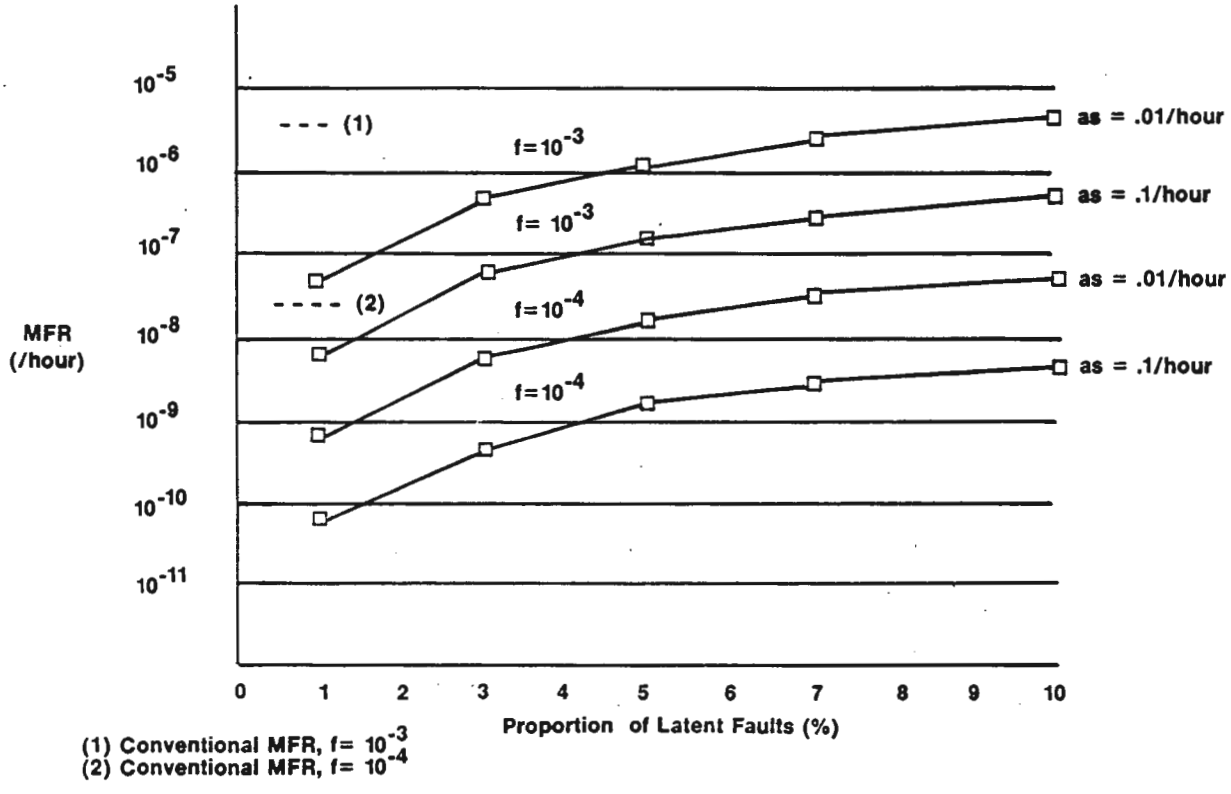


FIGURE 3.4-2. MEAN FAILURE RATE/TRIPLEX FCS

These results are summarized in tables 3.4-1 and 3.4-2 for lane failure rates (10^{-3}/hour and 10^{-4}/hour, respectively). It is noted that the corresponding MFRs, based on conventional reliability analysis, are $3x10^{-6}$/hour and $3x10^{-8}$/hour, respectively.

TABLE 3.4-1. MFR, TRIPLEX FCS, $f = 10^{-3}$/HOUR

as(/hour)	$1-p_1$(%)	MFR(/hour)
.1	10	5.76×10^{-7}
.1	7	2.82×10^{-7}
.1	5	1.44×10^{-7}
.1	3	5.19×10^{-8}
.1	1	5.77×10^{-9}
.01	10	4.47×10^{-6}
.01	7	2.20×10^{-6}
.01	5	1.13×10^{-6}
.01	3	4.07×10^{-7}
.01	1	4.53×10^{-8}

TABLE 3.4-2. MFR, TRIPLEX FCS, $f = 10^{-4}$/HOUR

as(/hour)	$1-p_1$(%)	MFR(/hour)
.1	10	5.92×10^{-9}
.1	7	2.90×10^{-9}
.1	5	1.48×10^{-9}
.1	3	5.33×10^{-10}
.1	1	5.92×10^{-11}
.01	10	5.81×10^{-8}
.01	7	2.85×10^{-8}
.01	5	1.45×10^{-8}
.01	3	5.23×10^{-9}
.01	1	5.82×10^{-10}

3.4.1. Mean Failure Rate Approximation

An approximate value of MFR can be obtained as follows:

On the average, loss of control will occur if two latent faults occur in a time interval of length 1/as hours and no α-faults occur in either of the lanes containing the latent faults. If q denotes this probability, then

$$q \approx 6[(1-p_1)f/as]^2[1 - 2(p_1)f/as]$$
$$\approx 6(1-p_1)^2f^2(1/as)^2 \qquad (1)$$

Thus, the Mean Time to Failure (MTTF) is:

$$MTTF \approx (1/as)[\ q + 2(1-q)q + 3(1-q)^2q + \ldots\ ] \quad (2)$$
$$= (1/as)/q$$

$$MFR \approx q(as) \quad (3)$$
$$\approx 6(1-p_1)^2f^2/as$$

3.4.2. Summary of Example 1

- The relative effect of latent faults is conveniently described by the ratio

$$R = \frac{\text{probability of loss of control with latent faults}}{\text{probability of loss of control by conventional analysis}}$$

  For values of $R \ll 1$, the effects of latent faults are insignificant. From the tables, the maximum value of R is .97 (for $f = 10^{-4}$, $as = .01$, $p_1 = .9$).

- Relative to the conventional MFR, a Triplex FCS does not appear to be especially vulnerable to latent faults; in the worst case, survivability is reduced by a factor of two.

- MFR is directly proportional to $(1-p_1)^2$.

- MFR is directly proportional to 1/as, the "time on risk."

- Survivability is adversely affected by the proportion of latent faults, $1 - p_1$, although for the range of parameters selected, this effect is not significant.

3.5. Example 2 - Quadruplex FCS

The conventional reliability analysis would indicate that probability of loss of the FCS function in a flight of one hour = $4f^3$.

A Markov reliability model of the FCS is shown in figure 3.5-1. The assumptions are the same as for the triplex system.

Referring to figure 3.5-1:

- State S_1 represents the condition that all lanes are fault-free.

- State S_2 has a latent fault in one lane.

- State S_3 has a latent fault in two lanes.

- State S_4 has a latent fault in three lanes.

- State S_5 has a latent fault in four lanes.

- State S_6 represents loss of the FCS function.

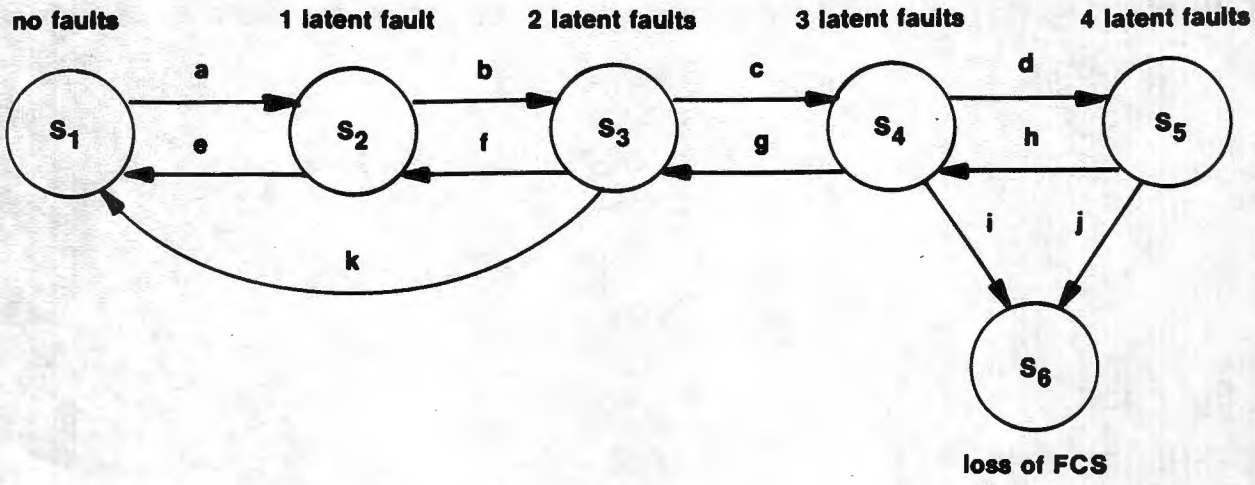


FIGURE 3.5-1. MARKOV MODEL/QUADRUPLEX FCS WITH LATENT FAULTS

The transition rates are:

$$a = 4(1-p_1)f\left[\frac{ds}{as + ds}\right]$$

$$b = 3(1-p_1)f$$

$$c = 2(1-p_1)f$$

$$d = (1-p_1)f$$

$$e = as + (p_1)f$$

$$f = 2(p_1)f$$

$$g = 3(p_1)f$$

$$h = 4(p_1)f$$

$$i = as$$

$$j = as$$

$$k = as$$

Again, it is noted that the backward transitions (e, f, g, h, and k) are the result of repairs due to the detection of α-faults.

From the figure we obtain the following differential equations:

$$sX_1 - x_1(0) = -aX_1 + eX_2 + kX_3$$

$$sX_2 = aX_1 - (b + e)X_2 + fX_3$$

$$sX_3 = bX_2 - (c + f + k)X_3 + gX_4$$

$$sX_4 = cX_3 - (d + g + i)X_4 + hX_5$$

$$sX_5 = dX_4 - (h + j)X_5$$

$$sX_6 = iX_4 + jX_5$$

with initial conditions, $x_1(0) = 1$, $x_2(0) = x_3(0) = x_4(0) = x_5(0) = x_6(0) = 0$. X_1, X_2, X_3, X_4, X_5, and X_6 denote the Laplace transforms of x_1, x_2, x_3, x_4, x_5, and x_6, respectively.

Figure 3.5-2 shows the MFR for a range of $1-p_1$ and f with p_2 and ds again fixed at $p_2 = 0$, ds = 10/hour. These results are summarized in tables 3.5-1 and 3.5-2 for lane failure rates, 10^{-3}/hour and 10^{-4}/hour, respectively.

It is noted that the corresponding MFRs, based on conventional reliability analysis, are 4×10^{-9}/hour and 4×10^{-12}/hour, respectively.

TABLE 3.5-1. MFR, QUADRUPLEX FCS, F = 10^{-3}/HOUR

as(/hour)	$1-p_1$(%)	MFR(/hour)
.1	10	2.23×10^{-9}
.1	5	2.80×10^{-10}
.1	3	6.04×10^{-11}
.1	1	2.24×10^{-12}
.01	10	1.37×10^{-7}
.01	5	1.73×10^{-8}
.01	3	3.75×10^{-9}
.01	1	1.39×10^{-10}

TABLE OF CONTENTS

Section		Page
1.	INTRODUCTION AND HISTORY	10-1
	1.1. Bendix Study	10-1
	1.2. Charles Stark Draper Laboratories Study	10-2
	1.3. Results	10-3
	1.4. Conclusions	10-4
2.	OVERVIEW OF THE FIIS EXPERIMENTS	10-5
	2.1. Objectives	10-5
	2.2. Test Ground Rules and Procedures	10-5
	2.2.1. Open-loop Scenario	10-5
	2.2.2. Closed-loop Scenario	10-6
	2.2.3. Self-Test	10-6
	2.2.4. Test Conditions	10-6
	2.3. Results	10-7
	2.4. Summary of the FIIS Experiments	10-8
	2.5. Conclusions of the FIIS Experiments	10-9
3.	FUNCTION CRITICALITY	10-11
	3.1. Single-Fault Model	10-11
	3.2. Parameter Estimates	10-13
	3.3. Potential Criticality of Latent Faults	10-13
	3.4. Example 1 - Triplex FCS	10-14
	3.4.1. Mean Failure Rate Approximation	10-17
	3.4.2. Summary of Example 1	10-18
	3.5. Example 2 - Quadruplex FCS	10-18
	3.5.1. Mean Failure Rate Approximation	10-22
	3.5.2. Summary of Example 2	10-22
	3.6. Self-Test	10-22
4.	UNRESOLVED ISSUES	10-23
5.	SUPPLEMENTAL WORKED EXAMPLE	10-25
	5.1. FCS Configuration	10-25
	5.2. Unsafe Condition	10-26
	5.3. Repair of Latent Faults	10-26

TABLE OF CONTENTS (Continued)

Section		Page
5.4.	Markov Reliability Model	10-26
	5.4.1. Model Parameters	10-28
5.5.	Effects of Latent Faults on Survivability	10-28
5.6.	The Conventional Reliability Analysis	10-29
5.7.	Numerical Example	10-30
5.8.	Summary of Example	10-30
6.	CONCLUSIONS	10-31
APPENDIX		
	A - MEAN FAILURE RATE	10-33
BIBLIOGRAPHY		10-35
GLOSSARY		10-36
ACRONYMS AND ABBREVIATIONS		10-37

LIST OF ILLUSTRATIONS

Figure		Page
3.1-1	SINGLE-FAULT MODEL	10-12
3.4-1	MARKOV MODEL/TRIPLEX FCS WITH LATENT FAULTS	10-14
3.4-2	MEAN FAILURE RATE/TRIPLEX FCS	10-16
3.5-1	MARKOV MODEL/QUADRUPLEX FCS WITH LATENT FAULTS	10-19
3.5-2	MEAN FAILURE RATE, QUADRUPLEX FCS	10-21
5.1-1	DUAL, FAIL SAFE FLIGHT CONTROL SYSTEM	10-26
5.4-1	MARKOV MODEL/DUAL, FAIL SAFE, FLIGHT CONTROL SYSTEM	10-27

LIST OF TABLES

Table		Page
2.3-1	OPEN-LOOP, UNWEIGHTED FAULTS	10-7
2.3-2	CLOSED-LOOP, UNWEIGHTED FAULTS	10-8
2.3-3	SUMMARY OF BASELINE/AUXILIARY FAULTS	10-8
3.4-1	MFR, TRIPLEX FCS, $f = 10^{-3}$/HOUR	10-17
3.4-2	MFR, TRIPLEX FCS, $f = 10^{-4}$/HOUR	10-17
3.5-1	MFR, QUADRUPLEX FCS, $F = 10^{-3}$/HOUR	10-20
3.5-2	MFR, QUADRUPLEX FCS, $F = 10^{-4}$/HOUR	10-21

1. INTRODUCTION AND HISTORY

The reliability of a flight control system (FCS) is achieved by a combination of (1) reliable components, (2) redundancy, and (3) the ability of the FCS to detect, isolate, and recover from faults. All highly reliable FCS designs incorporate some form of comparison-monitoring to detect faults. When the fault, or rather, the error produced by the fault, reaches a comparator it is immediately detected, at which time the isolation and recovery mechanisms are invoked.

Experience has shown that comparison-monitoring is an effective and reliable means of detecting faults. Moreover, because a comparator only detects a fault when the fault produces an error, it retains the faulty component for as long as the component functions correctly. In the interim, the fault may remain latent for long periods of time before it finally produces a malfunction. To detect these latent faults, some form of self-test is employed, either in a preflight or maintenance test or, during flight, in background.

With respect to latent faults, the critical issues are:

- What proportion of faults are latent?
- What mechanism activates a latent fault?
- What are the effects, if any, of latent faults on aircraft survivability?
- If latent faults are a contributing factor to reduced survivability, what level of coverage is required to detect and eliminate them?

Prior to the Fault Insertion and Instrumentation System (FIIS) experiments (the results and implications of which comprise the subject matter of this tutorial), most of the published information on data latency in digital FCSs was obtained from two important studies conducted independently by Bendix (McGough and Swern, 1983) and by the Charles Stark Draper Laboratories (CSDL) (Lala and Smith, 1983).

1.1. Bendix Study

The purpose of this study was to estimate detection coverage of several candidate self-test programs and to estimate fault latency in a conventional digital FCS. Fault injection experiments were conducted on a software-simulated version of the Bendix BDX-930 bit-sliced flight control computer (FCC), which featured the 2901A arithmetic logic unit (ALU) as the principal processing element. The simulated components consisted of the central processing unit (CPU), micromemory, program memory (ROM), and random access memory (RAM). The input/output (I/O) circuitry was not simulated. Sensor inputs were randomly generated and deposited in RAM at the start of each computational frame. Each

device of the CPU was represented by a gate-equivalent circuit and stuck-at faults were injected at the gate nodes, something not possible in actual hardware fault testing. The micromemory was represented functionally and faults were simulated by reversing the logic states of single bits.

The ROM and RAM memories were also represented functionally but no faults were injected into these devices. Faults were randomly selected and injected, one at a time. Fault selection was weighted in proportion to the failure rate of the device (i.e., the average number of faults injected into a particular device was proportional to the device failure rate). The simulation technique was "parallel mode," which allowed for the simultaneous simulation of up to 32 computers, one of which was always the non-failed version. This made it possible to compare the responses of the failed and non-failed computers at any time in the compute cycle and at any device.

Although all devices were simulated at the gate-level, faults were injected at both the pin-level and gate-level in order to determine differences in detection coverage between the two fault types.

Two types of experiments were conducted, depending upon the method of detection:

- To determine the effectiveness of comparison-monitoring, the computed outputs of a failed and non-failed computer were compared at the end of each frame. Any discrepancy was defined as a "detected failure". These comparisons were performed by the simulator executive and did not involve the detection mechanisms that would normally be resident in the FCCs. Each computer executed the same flight control program, which consisted of the inner loops of a high performance aircraft. In order to reduce the simulation time, a fault run was terminated after detection or after eight repetitions, whichever occurred first. To avoid any ambiguity in the comparison process, each computer received identical sensor inputs at the start of each frame. The simulation was conducted "open-loop", with sensor values selected independently and at random.

- To determine the effectiveness of self-test, each fault was injected and a candidate self-test program was executed. A fault was defined as "detected" if the mechanism of the self-test routine so indicated.

1.2. Charles Stark Draper Laboratories Study

The purpose of this study was to assess the fault detection, identification, and reconfiguration capabilities of the CSDL-designed fault-tolerant multiprocessor (FTMP). This system featured the Rockwell/Collins CAPS-6 computer as the principal processing element. The CAPS-6 is similar to the BDX-930 in that it, too, is a bit-sliced processor utilizing the 2901A ALU. FTMP is a triple modular redundant (TMR), bit-synchronized multiprocessor, designed for ultra high reliability (10^{-10} failures/hour) and fault-tolerance. The design is based on independent processor-cache memory modules and common memory modules, which communicate via redundant serial buses. All information processing and transmission is performed in triplicate. Data transmitted over the bus network is monitored by Bus Guardian Units (BGUs), which compare the transmitted data, bit by bit. Faults are detected when they produce errors at the BGUs. Faults are

also detected by self-test programs, which are executed continuously in background.

The fault injection experiments were conducted on real hardware. Stuck-at faults were injected, one at a time, on device pins of one of the three processing elements. (The fault injection hardware was identical to that used in the FIIS experiments.) Faults were systematically selected (unweighted) and injected on pins of eight circuit boards. FTMP executed a flight control software program consisting of inner-loops and autopilot modes, which were patterned after the L-1011 Tristar FCS. The fault runs were terminated after 15 seconds of real time. Unlike the Bendix experiments, the simulation was closed-loop (i.e., the sensor values were obtained from the motion parameters of a simulated aircraft).

1.3. Results

Both studies indicated that most faults are were detected within a few computational frames of their occurrence (e.g., 0 to 500 ms). Because fault runs were terminated after eight repetitions, the Bendix study gave no information about detection beyond 800 ms (assuming 100 ms per frame). The CSDL results indicated that relatively few faults (2 to 4 percent) were detected in later repetitions and these were detected by background self-test programs. From these results it was conjectured that most faults are activated by the baseline software program and are independent of the input sequence.

CSDL injected 21,055 faults (each pin fault was injected five times, in different locations of the program) and 3,637 (17.3 percent) were undetected. CSDL estimated that about 3000 of the undetected faults were "don't care" faults (i.e., faults on unused pins or on signals that were always low or always high under normal circumstances). Of the remainder, a few were analyzed and found to be "don't cares". In the Bendix study, 3000 faults were injected during the self-test evaluation phase and each undetected fault was analyzed. It was found that 466 faults (15.5 percent) were "don't cares" and, of these, 245 occurred in the bits of the micromemory and 47 in the control Programmable Read-Only Memory (PROM). When memory bit-faults were excluded, 7.2 percent of gate-level faults were "don't cares". Both Bendix and CSDL concluded that the identification of "don't care" faults was a non-trivial task, but essential to obtaining reasonably accurate estimates of fault coverage.

The Bendix study found that 86.5 percent of gate-level faults (excluding bit-faults in the micromemory and control PROMS) were detected during the eight repetitions of the FCS program, based on 148 injected "care" faults. Pin-level faults were not injected while executing the FCS program. The equivalent coverage in the CSDL study is not clear because of the large number of undetected, unanalyzed faults. Most of these faults (but, possibly, not all) were "don't cares." CSDL did conclude, however, that between 2 percent and 4 percent of all detected faults were detected by the background self-test programs and not by the comparators. Since self-test was not executed during the FCS experiments, Bendix would have concluded that these faults were undetected.

- The Bendix study showed that 88 percent of 2901A gate-level faults were detected during the FCS experiments, based on 52 injected "care" faults. The CSDL study gave no results for this device.

- In the Bendix study 97.4 percent of gate-level faults and 97.6 percent of pin-level faults were detected by self-test, based on 2234 and 376 injected "care" faults, respectively. The gate-level coverage excluded bit-faults in the memory elements. When these faults were included, coverage was 92 percent. The self-test program consisted of 346 Assembly Language instructions and required two to three ms to complete. The CSDL study gave no results for self-test coverage.

- The Bendix study concluded that gate-level faults were more difficult to detect than pin-level faults, especially when faults were injected into single bits of the memories. When executing software, other than the FCS, the ratio of gate-level to pin-level undetected faults was a factor of two. Unfortunately, "don't care" faults were not identified in these runs. As a result, detection coverage estimates tended to be pessimistic.

1.4. Conclusions

Prior to the FIIS experiments it was concluded, based upon the Bendix and CSDL studies, that:

- Most detected faults are detected within a few computational frames of their occurrence.

- The proportion of faults not detected by comparison-monitoring while executing the baseline program can range from 2 percent to 13.5 percent.

- Detection, isolation, and recovery could take up to several seconds, during which time the system (i.e., FTMP) is potentially vulnerable to second faults. The occurrence of a second fault, before the first fault is isolated, could confuse the majority vote and result in a system breakdown.

- "Don't care" faults constitute a statistically significant proportion of injected faults (between 6 percent and 17.3 percent). These faults are difficult to identify. Unidentified "don't care" faults could result in uncertain and pessimistic detection coverage estimates.

- Self-test coverage of 95 percent is easily achieved for pin-level faults and for gate-level faults if memory bit-faults are excluded.

- There is a significant difference in detection coverage between pin-level and gate-level faults, particularly if memory bit-faults are included in the latter.

2. OVERVIEW OF THE FIIS EXPERIMENTS

2.1. Objectives

The Bendix and CSDL studies were intended to provide an initial database for future FCS survivability assessment. The objectives of the FIIS experiments were (Benson, Mulcare, and Larsen, 1987):

- Corroborate and augment the results of the Bendix and CSDL studies.
- Provide a database of detection coverage and fault latency which future experimenters could use as goals or as a basis of comparison.
- Evaluate the FIIS fault injection methodology. (It was hoped that this methodology would be the first step in establishing guidelines for future fault injection experiments and fault detection coverage estimation.)
- Provide a database for the construction of single and multiple-fault models which could be used by reliability programs to assess FCS survivability.
- Identify unresolved issues associated with FCS survivability assessment and recommend studies to resolve them.

2.2. Test Ground Rules and Procedures

The target hardware was a dual FCC, which implemented a complete FCS for a commercial aircraft (Lockheed L-1011 Tristar). The FCS was dual/dual, but only a single pair was used in the experiments. Faults were detected by comparison monitoring of computed variables which were periodically exchanged between FCCs. Either FCC could request and effect system disengagement if it observed a discrepancy in any of these monitored variables. System disengagement constituted "detection." In addition, the system featured hardware comparators which were located in the secondary actuators. These comparators, which were not faulted, effectively measured the difference between the surface commands generated by the two FCCs. Again, comparator exceedances constituted "detection" and resulted in system disengagement. The experiments were conducted in open-loop and closed-loop scenarios.

2.2.1. Open-loop Scenario

The simulated airframe was disconnected for all runs. As a consequence, sensor inputs were invariant. In this configuration the operational program consisted of the inner loops, mode logic servicing, executive functions, synchronization, control panel and display servicing, voting and monitoring, and intercomputer communications. The open-loop program executed approximately 11,000 Assembly instructions every 50 ms.

2.2.2. Closed-loop Scenario

The simulated airframe was connected, including the sensor feedback signals.

In addition to the open-loop programs, the operational program consisted of the autopilot modes: cruise altitude hold, cruise climb, cruise turn, localizer capture, and glideslope capture/track. The land modes consisted of approximately 1200 additional Assembly Language instructions. In the autopilot modes the aircraft was perturbed from equilibrium flight by initial conditions and control wheel steering commands. The localizer mode was executed every 100 ms, the glideslope modes every 200 ms, and the other modes every 50 ms.

The open-loop experiments were conducted first. All faults were first injected in the open-loop scenario and only undetected faults were injected in the closed-loop scenarios. This saved time since most faults were detected in the open-loop experiments. Each undetected fault was subsequently injected during an autopilot mode, the purpose being to determine the proportion of faults detected by programs other than the baseline program. (It was assumed that the baseline program consisted of the inner-loop programs and cruise altitude hold.) The same fault was successively injected while executing each of the autopilot modes.

2.2.3. Self-Test

Each FCC contained a self-test program which was normally executed in the background. During the open and closed-loop experiments the self-test program was disabled. In order to determine self-test coverage and to identify "care faults", all faults which were undetected by both the open and closed-loop programs, and a larger sample of detected faults were injected separately, executing only the self-test program.

2.2.4. Test Conditions

Faults consisted of permanent S-a-0, S-a-1 and pin inversions, injected on input and output pins of almost every device on the data path and control cards of the CAPS-6 computer. Devices were only excluded when they would not function with the FIIS multiplexers.

- No faults were injected in the analog interface hardware.
- Faults were injected in a single FCC of the dual pair.
- Only single faults were injected.
- Faults were detected by comparison monitoring, exclusively. Self-test was not executed in the open and closed-loop scenarios.
- No faults were injected in the hardware associated with failure detection (i.e., hardware comparators and disengage mechanisms).
- Detection was recorded at 50 ms intervals. (This was the same as the minor frame interval.)

- An undetected fault run was terminated after 15 seconds of real time (i.e., 300 open-loop iterations).
- Each fault was individually identified.
- Coverage was tabulated for unweighted and weighted faults. In the latter, faults were weighted in proportion to the failure rate of the device; in the former, all faults were weighted equally.
- All undetected faults were analyzed and "don't care" faults were eliminated from the tabulations. "Don't care" faults consisted of unused pins or signals that were always high or low under all operating conditions.

2.3. Results

Detailed test results can be found in Benson, Mulcare, and Larsen (1987). In this section only the results for unweighted S-a-0 and S-a-1 faults are given.

Table 2.3-1: Out of 1670 injected "care" faults, 1597 (95.63 percent) were detected and 73 (4.37 percent) were undetected. Self-test detected 1115 faults out of 1115 injected.

TABLE 2.3-1. OPEN-LOOP, UNWEIGHTED FAULTS

Device	# Faults Injected	# Faults Detected	# Faults Undetected	Don't Cares	Percent Detected	Percent Undetected
Data Path	1016	990	26	60	97.44	2.56
Control	654	607	47	63	92.81	7.19
S-a-0/1	1670	1597	73	123	95.63	4.37
2901A	283	283	0	13	100.	0.0
Micromemory	146	146	0	9	100.	0.0
Input Pins	1184	1133	51	69	95.69	4.31
Output Pins	486	464	22	54	95.47	4.53
Self-Test	1115	1115	0	0	100.	0.0

[1] Excludes self-test
[2] Detected by self-test
[3] "Don't cares" = 6.86 percent

Table 2.3-2: Each of the 73 undetected open-loop faults was successively injected while executing each of the six autopilot modes. Thus, the same fault could have been detected while executing several modes, as indeed the results show. Of the 73 injected "care" faults, 33 (45.21 percent) were detected and 40 (54.71 percent) were undetected.

TABLE 2.3-2. CLOSED-LOOP, UNWEIGHTED FAULTS

Mode	# Faults Injected	# Faults Detected	# Faults Undetected	Don't Cares	Percent Detected	Percent Undetected
Alt.Hold	73	11	62	0	15.07	84.93
Climb	73	10	63	0	13.7	86.3
Turn	73	10	63	0	13.7	86.3
Loc/Capt	73	13	60	0	17.81	82.12
GS/Capt	73	9	64	0	12.33	87.67
GS/Track	73	12	61	0	16.44	83.56
Self-Test	73	73	0	0	100.	0.0
Totals	73	33(1)	40(1)	0	45.21	54.79

1 Detected by self-test

Table 2.3-3: These results indicate that most faults are detected while executing the baseline program. However, there is a significant number of faults not detected by either the baseline or auxiliary programs (e.g., 2.4 percent).

TABLE 2.3-3. SUMMARY OF BASELINE/AUXILIARY FAULTS

Total faults injected	1670
Detected by baseline	1597 (95.63%)
Undetected by baseline/detected by auxiliary	33 (1.98%)
Undetected by baseline/undetected by auxiliary	40 (2.4%)

2.4. Summary of the FIIS Experiments

- Most detected faults were detected within a few computational frames of their occurrence (e.g., within 10 frames = 500 ms). This result corroborates the results of the Bendix and CSDL studies.

- Faults were activated, i.e., produced errors, primarily by the operating software and not by time-varying sensor inputs, as evidenced by the high coverage in the open-loop scenario.

- Most faults were activated (and detected) by the baseline software program (95.63 percent). This coverage is considerably better than the Bendix study indicated, which is not surprising considering that the Bendix FCS program was 2200 words whereas the open-loop program was 11,000 words.

- A small number of open-loop detected faults were detected between 1 and 6.5 seconds (0.3 percent).

- A small number of faults were not activated by the baseline program but were activated by an auxiliary program (1.98 percent).

- A small number of faults were not activated by either the baseline or auxiliary programs (2.4 percent).

- Detection statistics for weighted and unweighted faults were similar (e.g., 96.5 percent for weighted versus 95.6 percent for unweighted).

- Invert faults are less latent than stuck-ats. For example, 1.71 percent are undetected when invert faults are included versus 4.37 percent when excluded. (One-third of all faults were inverts.)

- Micromemory and 2901A faults were 100 percent detected. This is not surprising since pin-level faults on these devices tend to produce massive errors.

- Self-test coverage was 100 percent. This was impressive, even at the pin-level. The self-test designer was obviously well-acquainted with the hardware.

- The percentage of all faults classified as "don't cares" is 6.11. This is consistent with Bendix's results, which estimated 7 percent at the pin-level. The relatively large number of these faults, together with the difficulty in identifying them, precludes accurate estimates of detection coverage.

2.5. Conclusions of the FIIS Experiments

- FIIS test results generally corroborate the results of the Bendix and CSDL studies. This was surprising considering the dissimilarity of input sensor selection, computer architecture, and operational software between the three studies.

- The validity of the FIIS approach hinges on the validity of pin-level fault models. Until this issue is resolved the results of the FIIS and previous experiments must be considered tentative. However, the results can be used as a relative measure of fault detection capability.

- The FIIS methodology was thoroughly tested during the study. FIIS performance was impressive, especially in the following areas:

  - Fault Generation Capability: Although only stuck-at and invert faults were injected, FIIS provides the capability of simulating a wide variety of fault types.

  - Ease of Use: Injecting faults and recording data was tedious but relatively simple after the initial setup.

- Fidelity: Since faults are injected into the target hardware there is no question about the fidelity of the experiments nor is there any need to validate the simulation.

- Speed: All of the FIIS experiments were run in real time. As a consequence, it was possible and practicable to determine detection coverage over long periods of time.

3. FUNCTION CRITICALITY

3.1. Single-Fault Model

The FIIS, Bendix, and CSDL studies established that, although most faults are detected within a few computational frames of their occurrence, there is a small number, perhaps 5 percent, which are not detected for much longer periods of time. These latent faults occur in devices which are infrequently exercised by the software program. As a consequence, they could accumulate and, if eventually activated by a common source of excitation, could result in a near-simultaneous loss of redundancy.

In planning the FIIS experiments, it was agreed by all of the participants that the principal and fundamental objective was to obtain a database for the reliability assessment of FCSs. As a consequence, it was necessary to anticipate the kind of data that might be required. Recognizing that the key element of a reliability assessment program was a single-fault model, it was decided to generate a strawman model in order to identify parameters which could be obtained from the FIIS experiments.

The results of the Bendix and CSDL studies seemed to indicate that most faults are activated by the baseline software (i.e., the executive and inner-loops). Only a small proportion (not activated by the baseline) are activated by auxiliary programs. These observations suggested the following definitions:

- Baseline Program: A set of continuously executed software modules.
- Auxiliary Programs: Software executed occasionally.
- Active Fault: A fault that can produce an error (for some input) while executing the current program.
- α-Fault: A fault activated by the baseline program.
- β-Fault: A fault not activated by the baseline program.
- Benign Fault: A fault that cannot produce an error while executing the current program, regardless of input, but may produce an error for some other program.
- Latent Fault: A fault which has not yet produced a malfunction. (In the context of the single-fault model, benign and latent faults are equivalent.)

Typically the baseline program is the operating system kernel while an application program, executed on demand, would be considered an auxiliary program.

The structure of the strawman single-fault model is shown in figure 3.1-1.

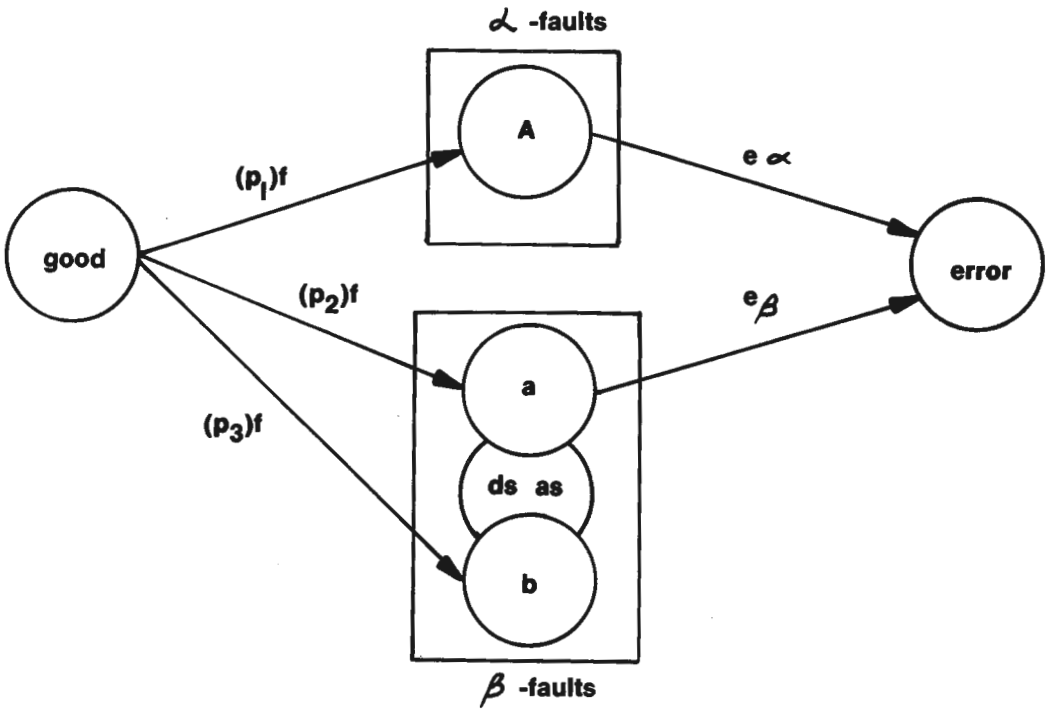


FIGURE 3.1-1. SINGLE-FAULT MODEL

The model is Markovian and appears to have sufficient degrees-of-freedom to model a wide variety of fault and error dynamics, particularly as observed in the Bendix and CSDL studies. The model is intended to model permanent faults only. The following parameters describe the fault and error dynamics:

f = failure rate of a lane (failures/hour)

p_1 = proportion of α-faults

p_2 = proportion of β-faults which are active at their occurrence

p_3 = proportion of β-faults which are benign at their occurrence

e_α = rate at which α-faults produce errors (errors/hour)

e_β = rate at which β-faults produce errors (errors/hour)

as = rate at which auxiliary programs are brought online (programs/hour)

1/ds = average duration of an auxiliary program (hours/program)

q = proportion of auxiliary programs that activate a benign fault

(Henceforth, we will assume that there is only a single auxiliary program. If more than one auxiliary program were involved, the complexity of the model would increase.)

The parameters of the model are not independent. In fact,

$$p_1 + p_2 + p_3 = 1$$

$$p_2 = (1 - p_1)\left[\frac{as}{as + ds}\right]$$

$$p_3 = (1 - p_1)\left[\frac{ds}{as + ds}\right]$$

From the results of the FIIS and previous experiments it was expected to obtain estimates of p_1, p_3, e_α, and e_β. The parameters p_2, as, and ds would be obtained from typical FCS scenarios (e.g., duration of and time between initiation of auxiliary programs).

3.2. Parameter Estimates

- p_1: From the Bendix study, $.85 < p_1$; from FIIS, $.96 < p_1 < .98$
- p_2: It is estimated that $p_2 = 0$, approximately
  (With $p_2 = 0$, $1 - p_1$ = proportion of latent faults)
- p_3: From the Bendix study, $p_3 < .15$; from FIIS, $.02 < p_3 < .04$
- e_α: From the FIIS study, $e_\alpha > 7200$/hour
- e_β: From the FIIS study, $e_\beta > 900$/hour
- as: From FCS scenarios, .01/hour < as < 10/hour
- 1/ds: From FCS scenarios, .01 hours < 1/ds < 1 hour
- f: The current range is .001/hour < f < .0001/hour

3.3. Potential Criticality of Latent Faults

In order to assess the effects of latent faults, we present two examples of conventional FCSs which could be vulnerable to the accumulation of latent faults. The examples noted are Triplex FCS (section 3.4) and Quadruplex FCS (section 3.5). The scenario is one in which latent faults, when activated, lead to a rapid and unexpected loss of components. Such a condition can arise in a Quadruplex FCS, where for example, in time, three out of the four lanes develop latent faults. Eventually, due to a single source of excitation, such as the callup of an outer-loop program, the faults become active. Even assuming, as we do, that the time interval between the successive activation of the faults (referred to, hereafter, as "avalanching latent faults") is sufficient to allow the comparators to detect and isolate each fault in turn, the result is a loss

of the three lanes in quick succession. It is our contention that the probability of such an event may not be insignificant.

3.4. Example 1 - Triplex FCS

The conventional reliability analysis, which assumes that there are no faults at the start of each flight, would conclude that the probability of loss of the FCS function in a flight of one hour is $6f^2$.

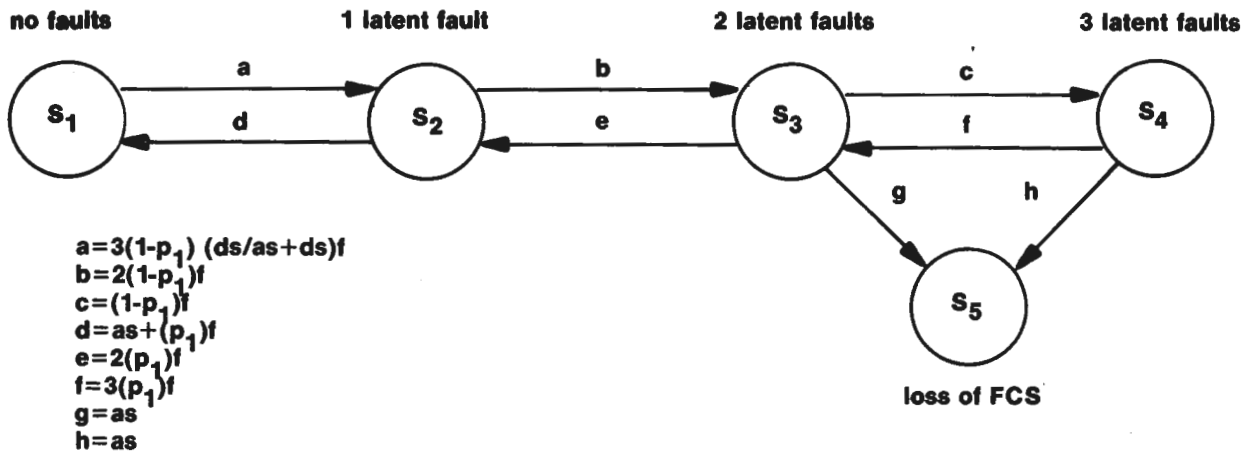


FIGURE 3.4-1. MARKOV MODEL/TRIPLEX FCS WITH LATENT FAULTS

A Markov reliability model of the FCS, representing the effects of avalanching latent faults, is shown in figure 3.4-1. The following assumptions are implicit:

- Loss of control is due, exclusively, to avalanching latent faults.
- Faults are exponentially distributed.
- The time between the successive activation of multiple latent faults is small relative to mission time.
- α-faults (i.e., faults active at their occurrence) are detected immediately as they occur, at which time the entire lane is replaced. Thus, existing latent faults are effectively repaired.

- Excitation of latent faults is assumed to be correlated across lanes (i.e., a single excitation activates latent faults in different lanes).

Referring to figure 3.4-1:

- State S_1 represents the condition that all lanes are fault-free.
- State S_2 has a latent fault in one lane.
- State S_3 has a latent fault in two lanes.
- State S_4 has a latent fault in three lanes.
- State S_5 represents loss of the FCS function.

The transition rates are:

$$a = 3(1-p_1)f[\frac{ds}{as + ds}]$$

$$b = 2(1-p_1)f$$

$$c = (1-p_1)f$$

$$d = as + (p_1)f$$

$$e = 2(p_1)f$$

$$f = 3(p_1)f$$

$$g = as$$

$$h = as$$

It is noted that the backward transitions (d, e, and f) are the result of repairs due to the detection of α-faults.

Let x_1, x_2, x_3, x_4, and x_5 denote the occupancy probabilities of states S_1, S_2, S_3, S_4, and S_5, respectively. From the figure we obtain the following differential equations:

$$sX_1 - x_1(0) = -aX_1 + dX_2$$

$$sX_2 = aX_1 - (b+d)X_2 + eX_3$$

$$sX_3 = bX_2 - (c+e+g)X_3 + fX_4$$

$$sX_4 = cX_3 - (f+h)X_4$$

$$sX_5 = gX_3 + hX_4$$

with initial conditions, $x_1(0) = 1$, $x_2(0) = x_3(0) = x_4(0) = x_5(0) = 0$. X_1, X_2, X_3, X_4, and X_5 denote the Laplace transforms of x_1, x_2, x_3, x_4, and x_5, respectively, and s denotes the Laplace operator.

Figure 3.4-2 shows the mean failure rate (MFR) (see appendix A), for a range of values of $1-p_1$ and f. The parameters p_2 and ds were fixed at the values $p_2 = 0$ and ds = 10/hour. Sensitivity studies indicate that the MFR is insensitive to values of ds between 1 and 10 per hour.

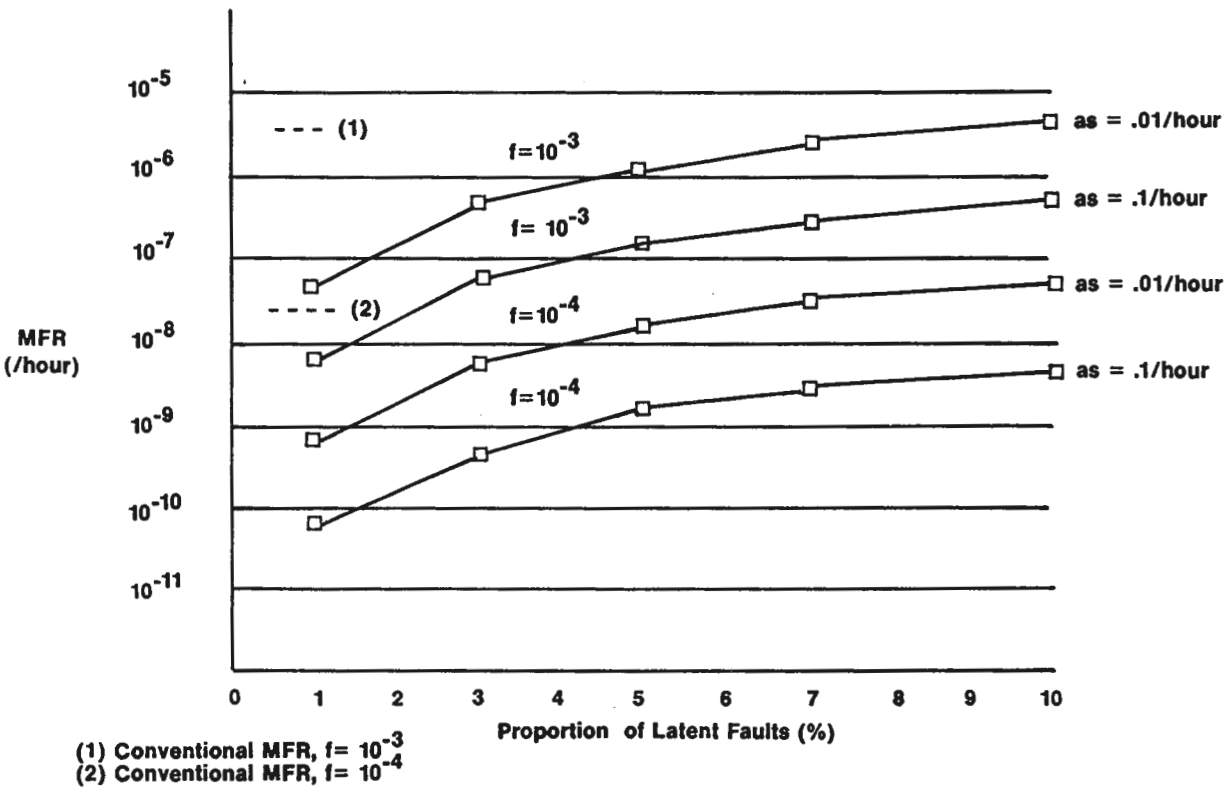


FIGURE 3.4-2. MEAN FAILURE RATE/TRIPLEX FCS

These results are summarized in tables 3.4-1 and 3.4-2 for lane failure rates (10^{-3}/hour and 10^{-4}/hour, respectively). It is noted that the corresponding MFRs, based on conventional reliability analysis, are $6x10^{-6}$/hour and $6x10^{-8}$/hour, respectively.

TABLE 3.4-1. MFR, TRIPLEX FCS, $f = 10^{-3}$/HOUR

as(/hour)	$1-p_1$(%)	MFR(/hour)
.1	10	5.76×10^{-7}
.1	7	2.82×10^{-7}
.1	5	1.44×10^{-7}
.1	3	5.19×10^{-8}
.1	1	5.77×10^{-9}
.01	10	4.47×10^{-6}
.01	7	2.20×10^{-6}
.01	5	1.13×10^{-6}
.01	3	4.07×10^{-7}
.01	1	4.53×10^{-8}

TABLE 3.4-2. MFR, TRIPLEX FCS, $f = 10^{-4}$/HOUR

as(/hour)	$1-p_1$(%)	MFR(/hour)
.1	10	5.92×10^{-9}
.1	7	2.90×10^{-9}
.1	5	1.48×10^{-9}
.1	3	5.33×10^{-10}
.1	1	5.92×10^{-11}
.01	10	5.81×10^{-8}
.01	7	2.85×10^{-8}
.01	5	1.45×10^{-8}
.01	3	5.23×10^{-9}
.01	1	5.82×10^{-10}

3.4.1. Mean Failure Rate Approximation

An approximate value of MFR can be obtained as follows:

On the average, loss of control will occur if two latent faults occur in a time interval of length 1/as hours and no α-faults occur in either of the lanes containing the latent faults. If q denotes this probability, then

$$q \approx 6[(1-p_1)f/as]^2[1 - 2(p_1)f/as]$$
$$\approx 6(1-p_1)^2f^2(1/as)^2 \qquad (1)$$

Thus, the Mean Time to Failure (MTTF) is:

$$MTTF \approx (1/as)[\ q + 2(1-q)q + 3(1-q)^2q + \ldots\ ]$$
$$= (1/as)/q \qquad (2)$$

$$MFR \approx q(as)$$
$$\approx 6(1-p_1)^2f^2/as \qquad (3)$$

3.4.2. Summary of Example 1

- The relative effect of latent faults is conveniently described by the ratio

$$R = \frac{\text{probability of loss of control with latent faults}}{\text{probability of loss of control by conventional analysis}}$$

  For values of $R \ll 1$, the effects of latent faults are insignificant. From the tables, the maximum value of R is .97 (for $f = 10^{-4}$, $as = .01$, $p_1 = .9$).

- Relative to the conventional MFR, a Triplex FCS does not appear to be especially vulnerable to latent faults; in the worst case, survivability is reduced by a factor of two.

- MFR is directly proportional to $(1-p_1)^2$.

- MFR is directly proportional to 1/as, the "time on risk."

- Survivability is adversely affected by the proportion of latent faults, $1 - p_1$, although for the range of parameters selected, this effect is not significant.

3.5. Example 2 - Quadruplex FCS

The conventional reliability analysis would indicate that probability of loss of the FCS function in a flight of one hour = $24f^3$.

A Markov reliability model of the FCS is shown in figure 3.5-1. The assumptions are the same as for the triplex system.

Referring to figure 3.5-1:

- State S_1 represents the condition that all lanes are fault-free.

- State S_2 has a latent fault in one lane.

- State S_3 has a latent fault in two lanes.

- State S_4 has a latent fault in three lanes.

- State S_5 has a latent fault in four lanes.

- State S_6 represents loss of the FCS function.

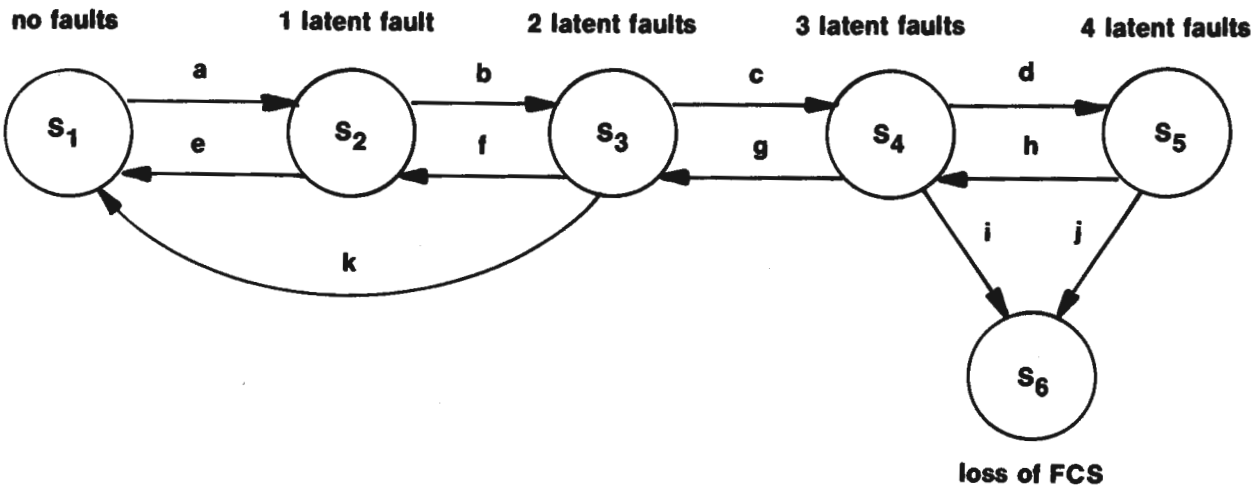


FIGURE 3.5-1. MARKOV MODEL/QUADRUPLEX FCS WITH LATENT FAULTS

The transition rates are:

$$a = 4(1-p_1)f\left[\frac{ds}{as + ds}\right]$$

$$b = 3(1-p_1)f$$

$$c = 2(1-p_1)f$$

$$d = (1-p_1)f$$

$$e = as + (p_1)f$$

$$f = 2(p_1)f$$

$$g = 3(p_1)f$$

$$h = 4(p_1)f$$

$$i = as$$

$$j = as$$

$$k = as$$

Again, it is noted that the backward transitions (e, f, g, h, and k) are the result of repairs due to the detection of α-faults.

From the figure we obtain the following differential equations:

$$sX_1 - x_1(0) = -aX_1 + eX_2 + kX_3$$

$$sX_2 = aX_1 - (b + e)X_2 + fX_3$$

$$sX_3 = bX_2 - (c + f + k)X_3 + gX_4$$

$$sX_4 = cX_3 - (d + g + i)X_4 + hX_5$$

$$sX_5 = dX_4 - (h + j)X_5$$

$$sX_6 = iX_4 + jX_5$$

with initial conditions, $x_1(0) = 1$, $x_2(0) = x_3(0) = x_4(0) = x_5(0) = x_6(0) = 0$. X_1, X_2, X_3, X_4, X_5, and X_6 denote the Laplace transforms of x_1, x_2, x_3, x_4, x_5, and x_6, respectively.

Figure 3.5-2 shows the MFR for a range of $1-p_1$ and f with p_2 and ds again fixed at $p_2 = 0$, ds = 10/hour. These results are summarized in tables 3.5-1 and 3.5-2 for lane failure rates, 10^{-3}/hour and 10^{-4}/hour, respectively.

It is noted that the corresponding MFRs, based on conventional reliability analysis, are 2.4×10^{-8}/hour and 2.4×10^{-11}/hour, respectively.

TABLE 3.5-1. MFR, QUADRUPLEX FCS, F = 10^{-3}/HOUR

as(/hour)	$1-p_1$(%)	MFR(/hour)
.1	10	2.23×10^{-9}
.1	5	2.80×10^{-10}
.1	3	6.04×10^{-11}
.1	1	2.24×10^{-12}
.01	10	1.37×10^{-7}
.01	5	1.73×10^{-8}
.01	3	3.75×10^{-9}
.01	1	1.39×10^{-10}

TABLE 3.5-2. MFR, QUADRUPLEX FCS, F = 10^{-4}/HOUR

as(/hour)	$1-p_1$(%)	MFR(/hour)
.1	10	2.36×10^{-12}
.1	5	2.95×10^{-13}
.1	3	6.38×10^{-14}
.1	1	2.37×10^{-15}
.01	10	2.25×10^{-10}
.01	5	2.82×10^{-11}
.01	3	6.1×10^{-12}
.01	1	2.26×10^{-13}

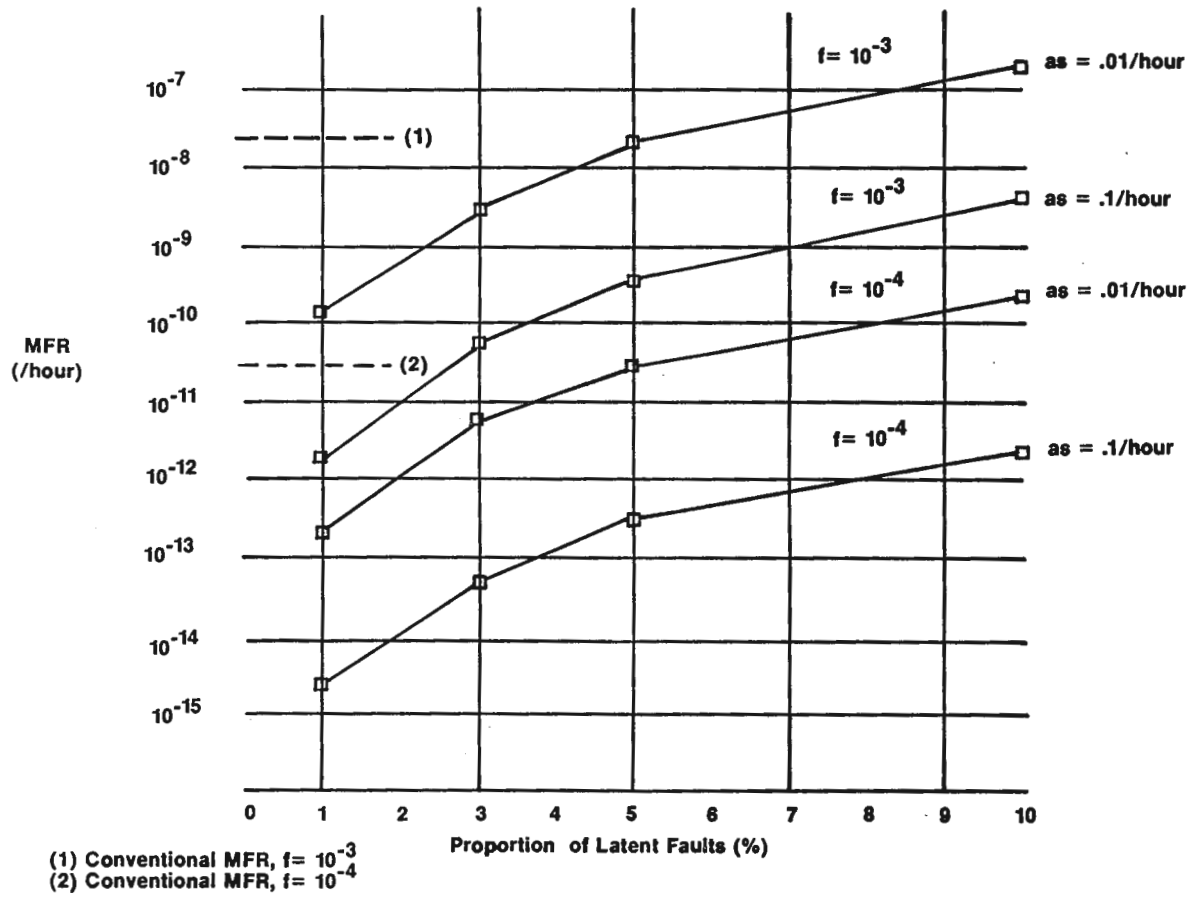


FIGURE 3.5-2. MEAN FAILURE RATE, QUADRUPLEX FCS

3.5.1. Mean Failure Rate Approximation

An approximate value of MFR for the Quadruplex FCS can be obtained as follows:

On the average, loss of control will occur if three latent faults occur in a time interval of length 1/as hours and no α-faults occur in any of the lanes containing the latent faults. If q denotes this probability, then

$$q \approx 24[(1-p_1)f/as]^3[1 - 3(p_1)f/as] \approx 24(1-p_1)^3f^3/as^3 \quad (4)$$

$$MTTF \approx (1/as)[\, q + 2(1-q)q + 3(1-q)^2q + \ldots] = (1/as)/q \quad (5)$$

$$MFR \approx q(as) \approx 24(1-p_1)^3f^3/as^2 \quad (6)$$

3.5.2. Summary of Example 2

- For the range of parameters selected, the maximum value of R is 10 ($f = 10^{-4}$, $as = .01$, $p_1 = .9$), indicating that latent faults could have a significant effect on survivability.
- Relative to the conventional MFR, a Quadruplex FCS is more vulnerable to latent faults than a Triplex FCS.
- MFR is directly proportional to $(1-p_1)^3$.
- MFR is directly proportional to $(1/as)^2$. Thus, for example, increasing the "time on risk", 1/as, by a factor of 10 increases the MFR by a factor of 100.

3.6. Self-Test

From the preceding examples, it is apparent how self-test can reduce the degrading effects of latent faults: by reducing the proportion of latent faults, $1-p_1$. If it is assumed that self-test coverage of α- and β-faults is the same, then if 1-u denotes self-test coverage, the proportion of latent faults can be reduced to $u(1-p_1)$. In the Triplex FCS a self-test coverage of only 90 percent reduces the MFR by a factor of 100 and, in the Quadruplex FCS, by a factor of 1000.

4. UNRESOLVED ISSUES

Although the Bendix, CSDL, and FIIS studies have contributed significantly to our understanding of fault, error, and detection dynamics, a number of important issues connected with FCS survivability assessment remain unresolved.

- Failure Modes of Digital Devices

  Previous fault injection experiments have employed stuck-at faults either on device pins or on internal gate nodes. It remains to be determined to what extent stuck-at faults represent failure modes of real devices.

- Pin-Level Versus Gate-Level Faults

  Until actual failure mode data becomes available, experimenters will, no doubt, continue to inject pin-level and gate-level faults. The Bendix study indicated a significant difference between pin-level and gate-level detection coverage. At issue is the validity of either fault type and the extent to which one is more or less latent than the other.

- Single-Fault Model Incorporating Intermittent/Transient Faults

  The single-fault model of figure 3.1-1 appears to be adequate for permanent faults. The model should be extended to include intermittent and transient faults. Estimates of occurrence, duration, and reoccurrence rates are required.

- Effects of Periodic Maintenance

  Periodic maintenance can have a significant effect on reducing latent faults to acceptable levels. Unfortunately, incorporating maintenance effects in reliability models leads to computational difficulties. McGough, Reibman, and Trivedi (to be published) offers one approach to this problem.

- Multiple-Fault Models

  In order to perform a reliability assessment of an FCS it is essential to model multiple fault effects. The multiple-fault models of figures 3.4-1 and 3.5-1 are too rudimentary to be of general use. Required is a more comprehensive multiple-fault model which includes the correlation of latent faults across different lanes.

5. SUPPLEMENTAL WORKED EXAMPLE

In the two examples of section 3, the assessment of latent faults was based on several assumptions e.g., the structure of the single-fault model and the correlation of excitation across different lanes.

In this section another perspective of the effects and potential criticality of latent faults on aircraft survivability is presented.

In general, the effects and criticality of latent faults are dependent upon the detailed FCS design, in particular, the levels of redundancy and the fault detection, isolation, and recovery mechanisms employed. As a consequence, the strawman FCS selected for this example should not be construed as being typical or even representative of FCSs. It was selected for the following reasons:

- It is a realistic FCS; similar configurations have been used in the past.
- The associated fault and reliability models are relatively simple and easily understood; the state occupancy probabilities can be obtained analytically.

5.1. FCS Configuration

The FCS is required to increase the fatigue life of the aircraft. This is achieved by improving the damping of the short period and dutch roll modes and controlling the phase of one or two bending modes. The aircraft, however, is controllable without the FCS. Thus, loss of the FCS function does not, in any way, impair the safety of the vehicle. However, an undetected fault could result in an unsafe condition. Thus, it is a firm requirement that such faults be precluded, at least to a probability of 10^{-7}/hour, the generally accepted goal for military aircraft.

The FCS configuration is shown in figure 5.1-1. Referring to the figure:

- The FCS consists of two independent lanes, each containing a processor (P) and dedicated disengage logic (Q and associated relay).
- Each processor drives a different actuator.
- Each processor receives information from an independent set of sensors.
- Each processor computes the commands for its own and the other actuator and compares both commands with those of the other processor. This data is exchanged via the intercomputer data links. Any observed difference, exceeding a comparator threshold, will cause that processor to send a discrete to both Q_1 and Q_2, opening the relays and disengaging both actuators.

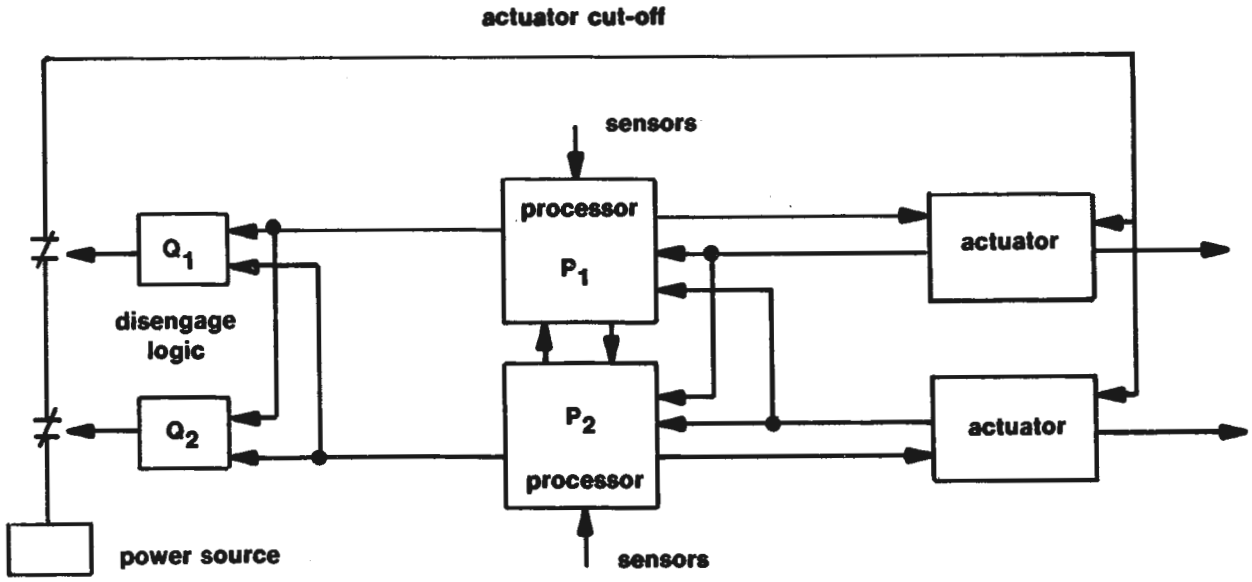


FIGURE 5.1-1. DUAL, FAIL SAFE FLIGHT CONTROL SYSTEM

5.2. Unsafe Condition

An unsafe condition can arise only if a processor fails and it and the non-failed processor are unable to disengage the actuators. Such an event could only occur if both disengage elements, Q_1 and Q_2, have failed previously. In practice, the disengage logic would be tested in preflight test. The issue is, what level of failure detection coverage is required?

5.3. Repair of Latent Faults

In generating the reliability model it must be determined when and how latent faults are repaired. In this example it is assumed that a detected fault will result in the replacement of the entire lane, both processor and disengage logic, by non-faulted components.

5.4. Markov Reliability Model

The reliability model of the FCS is shown in figure 5.4-1. The following assumptions are implicit:

- The model only includes the effects of latent faults.
- Faults are exponentially distributed.
- Mission duration is small relative to the mean time between failures.

- All faults in the disengage logic, Q_1 and Q_2, are latent (i.e., they produce no effects until a fault in P_1 or P_2 occurs).
- Any fault in the disengage logic prevents disengagement via the associated relay.
- All faults in the processors are detected immediately as they occur at which time the entire lane is replaced.
- A fault, undetected by preflight test, will remain undetected no matter how often the test is executed.
- All faults in P_1 and P_2 are detected by preflight test.

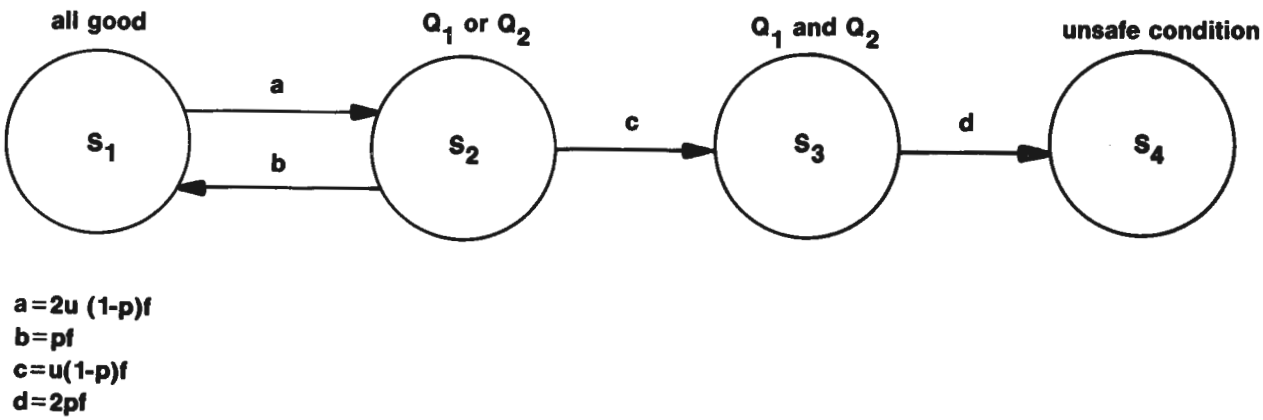


FIGURE 5.4-1. MARKOV MODEL/DUAL, FAIL SAFE, FLIGHT CONTROL SYSTEM

Referring to figure 5.4-1:

- State S_1 represents the condition that all lanes are fault-free.
- State S_2 represents the condition of a latent fault in Q_1 or Q_2 but not both.
- State S_3 represents the condition of latent faults in both Q_1 and Q_2.
- State S_4 represents the unsafe condition.
- A transition from S_1 to S_2 occurs when a fault occurs in Q_1 or Q_2.

- A transition from S_2 to S_1 occurs if Q_1 (Q_2) fails and P_1 (P_2) fails subsequently. Thus, the latent fault is repaired.

- A transition from S_2 to S_3 occurs when a second logic element fails.

- A transition from S_3 to S_4 occurs when both Q_1 and Q_2 have failed and a processor fails subsequently.

5.4.1. Model Parameters

- f = failure rate of a lane (failures/hour)

- pf = failure rate of a processor (failures/hour)

- (1-p)f = failure rate of the disengage logic (failures/hour)

- u = proportion of faults in the disengage logic not detected by pre-flight test

- a = 2u(1-p)f

- b = pf

- c = u(1-p)f

- d = 2pf

- 1-p = proportion of faults in the disengage logic

- p = proportion of faults in the processor, P_1 or P_2

Typical parameter values are

$$10^{-4}/\text{hour} < f < 10^{-3}/\text{hour}$$

$$.9 < p < 1$$

$$.9 < u < 1$$

5.5. Effects of Latent Faults on Survivability

Let x_1, x_2, x_3, and x_4 denote the occupancy probabilities of states S_1, S_2, S_3, and S_4, respectively. From the figure we obtain the following differential equations:

$$sX_1 - x_1(0) = -aX_1 + bX_2$$

$$sX_2 = aX_1 - (b+c)X_2$$

$$sX_3 = cX_2 - dX_3$$

$$sX_4 = dX_3$$

with initial conditions, $x_1(0) = 1$, $x_2(0) = x_3(0) = x_4(0) = 0$. X_1, X_2, X_3, and X_4 denote the Laplace transforms of x_1, x_2, x_3, and x_4, respectively, and s denotes the Laplace operator.

We could solve for $x_4(t)$, the probability of an unsafe condition, analytically, as a function of time. Instead, we compute the MFR. As shown in Appendix A:

$$MTTF = -\lim_{s \to 0} sX_4(s)/ds \tag{7}$$

We can solve directly to obtain

$$X_4(s) = \frac{acd}{s(s + d)(s^2 + (a + b + c)s + ac)} \tag{8}$$

$$MTTF = \frac{ac + d(a + b + c)}{acd} \tag{9}$$

$$MFR = \frac{acd}{ac + d(a + b + c)} = \frac{4u^2p(1-p)^2f}{2u^2(1-p)^2 + 2p[3u(1-p) + p]} \tag{10}$$

Setting $v = u\left(\frac{1-p}{p}\right)$,

$$MFR = \frac{2v^2pf}{v^2 + 3v + 1} \quad (\approx 2v^2pf) \tag{11}$$

5.6. The Conventional Reliability Analysis

The conventional reliability analysis assumes that no faults exist at the start of a flight. In this case, an unsafe condition can occur if and only if Q_1 and Q_2 both fail and P_1 or P_2 fails subsequently. Assuming a one-hour flight, this probability is given by

$$q = (1-p)^2f^2(2pf)/6 = p(1-p)^2f^3/3 \tag{12}$$

This event is order-dependent (i.e., Q_1 and Q_2 must both fail before P_1 or P_2). It can be shown that the order of occurrence is equiprobable if the flight time is small relative to 1/f. There are 12 possible orderings of Q_1, Q_2, P_1, and P_2, taken three at a time, but only four result in an unsafe condition, for example, (Q_1,Q_2,P_1), (Q_1,Q_2,P_2), (Q_2,Q_1,P_1), (Q_2,Q_1,P_2).

Employing Bernoulli trials, the mean time to first failure (i.e., an unsafe condition) is

$$MTTF = 1/q$$

and, thus,

$$q = MFR \approx p(1-p)^2 f^3/3 \tag{13}$$

Comparing (11) and (13) it can be seen that the MFR, due exclusively to latent fault conditions, is proportional to f whereas the conventional MFR is proportional to f cubed.

5.7. Numerical Example

To assess the relative effects of latent fault accumulation on system survivability we give several numerical examples, assuming p = .95 and f = 10^3/hour.

Case 1. No preflight test (i.e., u = 1).

MFR(latent) = 4.53×10^{-6}/hour.

Case 2. Preflight test with u = 0.1 (90 percent fault detection coverage).

MFR(latent) = 4×10^{-8}/hour.

In contrast, the conventional MFR is

MFR(conventional) = 7.91×10^{-13}/hour.

5.8. Summary of Example

From this simple example, it can be seen that the accumulation of latent faults can have a profound effect on system survivability. In fact, they are often the dominant cause of system loss. As a consequence, it is incumbent on the system designer to assess the vulnerability of the FCS to such faults and, if significant, establish self-test coverage requirements accordingly.

6. CONCLUSIONS

In the Introduction, four questions were raised in connection with latent faults. As the examples given in this tutorial illustrate, the answers depend upon the details of the FCS design, the levels of redundancy, the fault detection, isolation, and recovery mechanisms employed, and the survivability goals of the FCS. As a consequence, we venture to give some answers but only with respect to the FCSs presented in this tutorial.

- What proportion of faults are latent?

  In the FCSs used in the Bendix, CSDL, and FIIS experiments the proportion of latent faults ranged between 2 percent and 15 percent. A less pessimistic estimate, based on the FIIS experiments, indicates a range between 2 percent and 4 percent. In the FCS of section 5, the proportion is a function of the relative failure rate of the disengage logic.

- What mechanisms activate latent faults?

  Latent faults are activated when the faulty components are exercised. In the systems used in the Bendix, CSDL, and FIIS experiments latent faults appear to be activated primarily by software programs. In the FCS of section 5, latent faults are activated when faults occur in other components.

- What are the effects of latent faults on aircraft survivability?

  In the two examples of section 3 these effects are given in figures 3.4-2 and 3.5-2. In the FCS of section 5 these effects are given by equation (10). Obviously, the effects of latent faults are highly dependent on the FCS configuration. In any case, their significance can only be evaluated relative to the survivability goal of the FCS. (In the examples given, this goal was assumed to be the survivability as determined by conventional analysis.)

- If latent faults are a contributing factor to reduced survivability, what level of coverage is required to detect and eliminate them?

  In the examples given in sections 3 and 5, a modest coverage could reduce these effects to insignificance (e.g., a 90 percent coverage will improve survivability by several orders of magnitude).

APPENDIX A - MEAN FAILURE RATE

Occupancy Probability

A homogeneous Markov model is described by a set of linear differential equations of the form

$$dx/dt = Ax \tag{1}$$

where

$A = (a_{ij}) = nxn$, is a constant matrix of transition rates, and

$x = col(x_1, x_2, \ldots, x_n)$, is a vector of occupancy probabilities of states S1, S2,..., Sn, respectively.

In most FCS reliability models there is exactly one absorbing state, the loss of control state, which we designate as S_n. The associated occupancy probability, $x_n(t)$, is then a cumulative distribution function and can be expressed in the form

$$dx_n(t)/dt = a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn-1}x_{n-1} \tag{2}$$

In many cases, $x_n(t)$ is not a convenient measure of survivability. For example:

- For models with large numbers of states or models which contain cyclic paths (Marie, Reibman, and Trivedi, 1987), the computation of $x_n(t)$ requires the aid of a reliability analysis program such as ARIES, CARE III and HARP.

- In the conventional reliability model the FCS is assumed to be completely repaired at the start of each flight. Thus, the probability of loss of control during the first flight is the same for all flights. When latent faults are present, however, the FCS may only be partially repaired at the start of a flight. Consequently, the probability of loss of control is different for each flight, tending to increase with time. In these scenarios the occupancy probability of loss of control is not a practicable measure of survivability because it requires a recursive solution of the Markov model over a large number of flights.

We propose an alternate measure of survivability, called MFR. The MFR is defined as the reciprocal of the MTTF (McGough, Reibman, and Trivedi [to be published]).

The advantages of the MFR criteria are as follows:

- It is relatively easy to compute, even for models with cyclic paths, requiring only solutions of linear, algebraic equations.

- It is independent of time.
- It characterizes survivability over the life of the airplane.
- It provides a relative measure of survivability even if rejected as an absolute measure.

The disadvantage of MFR is that it gives no indication of how survivability degrades with time.

Computation of Mean Failure Rate

Taking the Laplace transform of both sides of equation (2) gives, for $x_n(0) = 0$,

$$sX_n(s) = a_{n1}X_1(s) + a_{n2}X_2(s) + \ldots + a_{nn-1}X_{n-1}(s) \tag{3}$$

where

$X_k(s)$ is the Laplace transform of $x_k(t)$ and s is the Laplace operator.

Using the moment generating property of Laplace transforms:

$$\begin{aligned} MTTF &= -\lim_{s\to 0} sdX_n(s)/ds \\ &= -[anlX_1'(s) + an2X_2'(s) + \ldots, + ann\text{-}1X_{n-1}'(s)] \end{aligned} \tag{4}$$

where the prime denotes the derivative with respect to s.

If we let $X(s) = \text{col}\,[X_1(s), X_2(s), \ldots, X_n(s)]$, equation (1) can be expressed as

$$sX(s) - x(0) = AX(s) \tag{5}$$

Differentiating both sides of equation (5), we obtain

$$X(s) + sX'(s) = AX'(s) \tag{6}$$

Setting $s = 0$ gives

$$X(0) = AX'(0) \tag{7}$$

The quantities $X_1(0)$, $X_2(0)$, ..., $X_{n-1}(0)$ are obtained by setting $s = 0$ in equation (5) and solving the resultant linear system by using the method of Gaussian elimination or an iterative method like Gauss-Seidel. Substituting these values into the first n-1 equations in (7) we can solve for $X_1'(0)$, $X_2'(0)$, ..., $X_{n-1}'(0)$. A unique solution exists because S_n is the only absorbing state. The resulting values $X_1'(0)$, $X_2'(0)$, ..., $X_{n-1}'(0)$ are substituted into equation (4) to obtain the MTTF. Then

$$MFR = 1/MTTF.$$

BIBLIOGRAPHY

Benson, W., Mulcare, D., and Larsen, W., Hardware Fault Insertion and Instrumentation System: Experimentation and Results, DOT/FAA/CT-86/34, FAA Technical Center, March 1987.

Lala, J. and Smith, T. B., Development and Evaluation of a Fault-Tolerant Multiprocessor (FTMP) Computer - Volume III, FTMP Test and Evaluation, NASA CR 166073, NASA Langley Research Center, May 1983.

Marie, R. A., Reibman, A. L., and Trivedi, K. S., Transient Solution of Acyclic Markov Chains, Performance Evaluation, Vol.7, No.3, 1987, pp. 175-194.

McGough, J. G., Reibman, A. L., and Trivedi, K. S., "Markov Reliability Models for Digital Flight Control Systems," AIAA Journal of Guidance and Control (to be published).

McGough, J. G. and Swern, F., Measurement of Fault Latency in a Digital Avionic Processor, Part II, NASA CR 3651, NASA Langley Research Center, January 1983.

GLOSSARY

AVALANCHING LATENT FAULTS. The successive activation of latent faults.

MEAN FAILURE RATE. A measure of survivability defined as the reciprocal of the mean time to system failure.

BASELINE PROGRAM. A set of continuously executed software modules.

AUXILIARY PROGRAMS. Software executed occasionally.

ACTIVE FAULT. A fault that can produce an error (for some input) while executing the current program.

α-FAULT. A fault activated by the baseline program.

β-FAULT. A fault not activated by the baseline program.

BENIGN FAULT. A fault that cannot produce an error while executing the current program, regardless of input, but may produce an error for some other program.

LATENT FAULT. A fault which has not yet produced a malfunction. (In the context of the single-fault model, benign and latent faults are equivalent.)

ACRONYMS AND ABBREVIATIONS

ALU	Arithmetic Logic Unit
BGU	Bus Guardian Unit
CPU	Central Processing Unit
CSDL	Charles Stark Draper Laboratories
FCC	Flight Control Computers
FCS	Flight Control System
FIIS	Fault Insertion and Instrumentation System
FTMP	Fault-Tolerant Multiprocessor
GS	Glideslope
I/O	Input/Output
MFR	Mean Failure Rate
ms	Millisecond
MTTF	Mean Time to Failure
P	Processor
PROM	Programmable Read-Only Memory
RAM	Random Access Memory
ROM	Program Memory
S-a-0	Stuck at Zero
S-a-1	Stuck at One
TMR	Triple Modular Redundant