UNITED STATES
NUCLEAR REGULATORY COMMISSION
WASHINGTON, D.C. 20555-0001

December 31, 2003

MEMORANDUM TO: Ledyard (Tad) Marsh, Director
Division of Licensing Project Management
Office of Nuclear Reactor Regulation

FROM: Michael E. Mayfield, Director
Division of Engineering Technology
Office of Nuclear Regulatory Research

SUBJECT: TRANSMITTAL OF DIGITAL SYSTEMS RELIABILITY AND SAFETY ASSESSMENT TECHNICAL REPORTS

References: (1) Letter, Richard J. Barrett from Michael Mayfield, "Transmittal of Digital Systems Technical Report, "A Tutorial on Architectural Analysis Using Reliability and Safety," October 8, 2002.
(2) NUREG-GR-0020, "Embedded Digital System Reliability and Safety Analyses," February 2001.

This memorandum transmits letter reports that expand on, and provide digital system reliability assessment examples of, the material presented in the report entitled "A Tutorial on Architectural Analysis Using Reliability and Safety" (reference 1). This is the fourth transmittal in a series of reports that provide the technical bases (reference 2), process and methods (attachment 2) for a new reliability and safety analysis method that accommodates both custom designed and Commercial Off The Shelf (COTS)-based digital systems.

An overview of the assessment technique is given in the paper "Reliability and Safety Assessment of Digital Systems Using Fault Injection" (attachment 1). The reports demonstrate the application of methods and tools to a PWR digital feedwater control system (attachments 2,3,4,5). However, the assessment process, methods and tools can be easily used to assess many types of digital systems, such as a digital safety-related Reactor Protection System or Engineered Safeguards Actuation System.

The research is being conducted to meet an ACRS request and an NRR User Need Request number 2002-17, which stated in part, "Investigate and develop methods and models for quantitative assessment of highly reliable safety-critical software-based critical systems..." The research provides a greater understanding and clarifies the main quantitative determinants of the reliability and safety of digital systems used in nuclear power plants. It will reduce the uncertainties associated with review of these systems, and provide an alternate review method more focused on safety. Safety in this case means that given the occurrence of a fault, the digital system, including hardware and software, recognizes the fault occurrence and can continue safe operation or it can shut down in a safe manner.

Some general conclusions that can be drawn form this research are:
1.) the same digital system architecture can result in different safety properties depending on the value of *fault coverage (the conditional probability that given the occurrence of a fault in the system, it is detected and the system recovers);*

2.) the methodology is not dependent on any specific knowledge of the hardware or software *development* methods or attributes, but will assess the results of the development methods in an objective manner;

3.) the methodology can accommodate proprietary special purpose systems or COTS component systems;

4.) the methodology tests the ability, accuracy, and efficiency of internal software and hardware diagnostics to perform their error detection and management capabilities;

5) the methodology is versatile enough to be used to simulate the correct functional performance, testability, and timing in addition to providing metrics for reliability and safety;

6) The methodology accounts for the combined operation of hardware and software in a digital system, not just one or the other.

It is requested that you review the attachments and provide comments by April 2, 2004.

If you have any questions about this research, please contact Steven Arndt (415-6502) or John Calvert (415-6323), members of my staff in the Engineering Research Application Branch, Digital I&C Team.

Attachments:
1.) *"Reliability and Safety Assessment of Digital Systems Using Fault Injection,"* J.A. Calvert, R.A. Shaffer, B.W. Johnson, paper presented at Nuclear Safety Research Conference October 21, 2003.
●Technical Reports from the University of Virginia Center for Safety-Critical Systems Department of Electrical and Computer Engineering Charlottesville, VA
2.) *"A Numerical Safety Evaluation Process for Safety-Critical Systems,"* Technical Report UVA-CSCS-NSE-001, Revision 2, August 01, 2003.
3.) *"Analytical Safety Model,"* Technical Report UVA-CSCS-NSE-002, Revision 01, August 1, 2003.
4.) *"Statistical Model,"* Technical Report UVA-CSCS-NSE-003, Revision 00, August 01, 2003.
5.) *"Generic Processor Fault Model,"* Technical Report UVA-CSCS-NSE-004, Revision 00, August 01, 2003.

Distribution:
ERAB r/f, DET r/f, R. Barrett, J. Calvo, A. Marinos, M. Chiramal, C. Antonescu, D. Tifft, R. Shaffer, S. Arndt, T. Govan
DOCUMENT NAME: G:\jNRRxmtl2ProcessPkg.wpd
OAR in ADAMS? (Y or N) Y ADAMS ACCESSION NO:_____TEMPLATE NO. RES:006
Publicly Available? (Y or N) Y DATE OF RELEASE TO PUBLIC_____SENSITIVE?_____
To receive a copy of this document, indicate in the box: "C" = Copy without attachment/enclosure "E" = Copy with attachment/enclosure "N" = No copy * See Previous Concurrence

OFFICE	ERAB/DET	ERAB/DET	ERAB/DET	DET
NAME	J. Calvert*	A. Hsia	M. Evans	M. Mayfield
DATE	12/30/03	12/31/03	12/31/03	12/31/03

OFFICIAL RECORD COPY

Reliability and Safety Assessment of Digital Systems Using Fault Injection Techniques

By
John A. Calvert, Roman Shaffer
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
and
Barry W. Johnson
University of Virginia, Center for Safety-Critical Systems

Presented at the Nuclear Regulatory Commission's
Nuclear Safety Research Conference
October 20-22, 2003

Abstract

This paper presents an overview of methods to assess reliability and safety aspects of digital systems. A methodology is built using Markov models, simulation, and fault injection to determine fault coverage. Fault coverage is an important parameter used in the calculation of such metrics as mean-time-to-unsafe-failure (MTTUF). The methods are extensible to digital system-in-the-loop using nuclear process models for computer simulation of normal performance and faulted consequences. A case study using the methodology for a PWR digital feedwater control system is presented. The results from applying the methodology leads to realistic regulatory decisions concerning advanced digital systems.

Introduction

Fault injection involves inserting faults into a system and monitoring the system to determine its behavior in response to the fault for the purpose of providing parameter estimations for reliability and safety assessments. Computer simulation can be used to perform the fault injection on a model of the design, as opposed to performing the fault injection on the actual system. With today's computing power the fault injection and recording of the results can be very efficient for complex digital systems. This makes reliability and safety assessment more obtainable than ever before and can include the combined effect of hardware, software, and operators/maintainers. These developments, when implemented according to a valid technical basis and coupled with modeling/simulation methods available over the world-wide-web, can result in more realistic and accurate digital system reliability and safety assessments compared to present methods.

The report NUREG/CR-0020 "Embedded Digital System Reliability and Safety Analyses" provided the technical basis for a method to analyze the behavior of digital systems under the influence of internal/external and normal/faulted conditions of the inputs, outputs, hardware, and software. The uniqueness of the analysis method is that it can simulate software design faults and diagnostic faults while most methods are hardware oriented and only account for random hardware faults. The method is presently being used for safety-critical digital system safety evaluation in Europe and has been reviewed by safety assessors from TUV Germany.

Research being conducted will develop the selected methods in the areas of safety assessment, detailed methods, and tools to properly characterize and analyze digital systems for performance, reliability, failure modes, and subsystem and system safety. The goal of the research is to promote more efficient staff review of digital systems by providing a sharp focus on risk significant areas.

The research is being performed under a Cooperative Agreement between NRC and the University of Virginia, Center for Safety-Critical Systems (CSCS) and involves developing assessment methods for evaluating digital systems for reliability and safety including the interrelated dependencies of hardware and software. This includes areas such as the efficacy of software-hardware diagnostics and other fault management mechanisms at both the channel and system levels. The methodology will enable the

quantification of digital system reliability and safety for advanced complex processor-based safety-critical systems. This makes the methodology suitable for the modeling and analysis of failures of hardware or software, and finding the resultant effect on a computer-based system. The methodology also has the ability to generate probabilistic analyses of computer-based system failures.

The reliability and safety of digital systems can be depicted as a state-oriented model (analytical or system model) where the end states are failed-safe and failed-unsafe. The main parameters involved in the state transitions are failure rate and fault coverage. The fault coverage is a conditional probability that if a fault occurs, it is detected and the system transitions to the fail-safe state; an undetected fault causes a transition to the failed-unsafe state.

The modeling and influence of fault coverage on reliability and safety has been known for thirty years, but until recently has been very difficult to estimate for digital systems with very high certainty. The coverage estimate can be made from experimentally testing a system under various fault conditions to determine if the system recognizes the fault and transitions to either the failed-safe or failed-unsafe state. The data thus collected experimentally can be used in the coverage parameter estimation within known confidence limits. The resulting values are used to solve the equations resulting from the analytical safety model of the system.

Methodology Overview

The process (Figure 1) used to assess the reliability and safety of a system involves evaluation of the normal and abnormal system functional and fault processing capabilities using fault injection. The general approach is to inject randomly selected faults into the system on a simulation model or a working prototype to determine if the fault processing capabilities mitigate the faults. There are three major steps described below.

A.) The assessment process begins with the development of a high-level ***analytical model***. The purpose of this model is to provide a mathematical framework for calculating the estimate of the numerical safety specification. The input to the model is the numerical safety specification along with the required confidence in the estimate. Analytical models include Markov models, Petri nets, and Fault Trees. For the purposes of a safety assessment, the analytical model is used to model, at a high level, the faulty behavior of the system under analysis. The model uses various critical parameters such as failure rates, fault detection latencies, and fault coverage to describe the faulty behavior of the system under analysis. Of all the parameters that typically appear in an analytical model used as part of a safety assessment, the fault coverage is by far the most difficult to estimate.

The analytical safety model is developed from the system architecture and inter-component dependencies to derive an expression for the probability of unsafe failure or other metric. The expression is a function of critical model parameters such as the fault coverage values and failure rates for the various components of the system.

B.) A ***statistical model*** of the digital system is then developed and is used to estimate the critical parameters that are required by the analytical safety model. The statistical model for coverage estimation is used to estimate the fault coverage parameter.

The statistical models are derived based on the fact that the fault coverage is a binomial random variable which can be estimated using a series of Bernoulli trials. At the beginning of the estimation process the statistical model is used to estimate the minimum number of trails required to demonstrate the desired level of fault coverage. This information is crucial at the beginning of a safety assessment because it helps to determine the amount of resources required to estimate fault coverage. One of the most important uses of the statistical model is to determine how many fault injection experiments are required in order to estimate the fault coverage at the desired confidence level.

There are important details needed to complete the use of the statistical model and these are described below.

--A high-level *generic processor fault model* is defined to specify the types of faults (and their associated probabilities) that will be injected into the system under analysis. This process builds upon recent research in generic processor fault modeling, undertaken with the goal of characterizing low-level internal processor faults at the higher register-transfer level, in order to demonstrate that faults injected on the actual hardware are representative of the low-level processor faults of concern. The fault model is used to generate the fault space, F. Generally speaking, completely proving the sufficiency of the fault model used to generate F is very difficult. It is more traditional to assume that the fault model is sufficient, justifying this assumption to the

greatest extent possible with experimental data, historical data, or results published in literature.

--One or more *operational profiles* are then defined which will be used to drive the inputs to the system under analysis during the fault injection process. The operational profiles must be representative of the different system configurations and workloads that would be experienced in actual field operation. The operational profile is represented mathematically as a probability density function. The input sequences used for fault injection are selected randomly using the input probability density function. After a number of input sequences are generated then the fault coverage estimation moves to the next stage.

--A *fault-free execution trace* is created for each selected operational profile that will be used to generate the list of faults to inject into the system under analysis. This trace will also be used during the analysis of the fault injection experimental results. The fault free trace records all relevant system activity for a given operation profile. Typical information stored in trace is read/write activity associated with processor registers, address bus, data bus, and memory locations in the system under test. The purpose of the fault free execution trace is to determine the system activity such as memory locations used, instructions executed, and processor register usage. The system activity can then be analyzed to accelerate the experimental data collection. Specifically, the system activity trace can be used to ensure that only faults which will produce an error are selected during the list generation process. The set of experiments will not be an exhaustive set due to the size the fault space, which is assumed to be infinite, but it is assumed that the set of experiments is a representative sample of the fault space to the extent of the fault model and to the confidence interval established by the number of samples. It is typically assumed that using a random selection process results in a sample which is representative of the overall population.

--Using the fault-free execution trace and the fault categories from the generic processor fault model, a *fault list construction algorithm* is applied to generate a list of possible faults that can be injected into the system under analysis, and which are likely to have an effect on the system. From this complete set of responsive faults, a *fault list selection algorithm* is then applied to randomly select a list of faults to be injected into the system, using the fault categories and associated occurrence probabilities from the generic processor fault model.

--A *fault equivalence algorithm* is applied to each fault list to identify those sub-sets of faults which will have the same effect on the system, known as fault equivalence classes. The set of faults to be injected into the system is then reduced using the fault equivalence information, since only one fault from each class needs to be injected. This reduces the amount of time required to perform the evaluation of the system under analysis by reducing the total number of fault injection experiments which must be performed. The results from applying the fault equivalence algorithm is a reduced fault list.

C.) Each fault from each reduced fault list is then injected into the system under analysis, using one of four fault injection techniques: (a) hardware-based, (b) software-based, (c) simulation-based, or (d) a hybrid approach. The system is monitored during ***fault injection experimentation***, and each fault is classified as covered, uncovered, or non-responsive by comparing the actual execution trace to the fault-free execution trace. Once all of the results of the fault injection experiments have been collected, they are fed back into the statistical model to calculate the estimates of the critical parameters. The statistically valid ***coverage parameter estimates*** are then in turn fed back into the analytical model to calculate the estimate of the reliability or safety.

Fault Injection

Fault Injection is defined as a dependability validation technique that is based on the realization of controlled experiments where the observation of the system behavior in present of faults, is explicitly induced by the deliberate introduction (injection) of faults into the system (Figure 2).

Artificial faults are injected into the system and the resulting behavior is observed. This technique can speed up the occurrence and the propagation of faults into the system for observing its effects on the system performance. Fault injection can be performed on either model simulations or working prototypes or systems in the field. In this manner the weaknesses of interactions can be discovered, but this

is a haphazard way of debugging the design errors in a system. It is better used to test the resilience of a fault tolerant system against known faults, and thereby measure the effectiveness of the fault tolerant mechanisms+.

Fault injection techniques can be used in both electronic hardware systems and software systems to measure the fault tolerance of the system. For hardware, faults can be injected into simulations of the system, as well as into implementation, both on a pin or external level and, recently, on an internal level for some chips. For software, faults can be injected into simulations of software systems, such as distributed systems, or into running software systems, at levels from the CPU registers to memory to disk to networks.

There are two main categories of fault injection techniques: execution-based and simulation-based. In the former, the system itself is deployed, and some mechanism used to cause faults in the system, and its execution is then observed to determine the effects of the fault. These techniques are more useful for analyzing final designs, but are typically more difficult to modify afterwards. In the latter, a model of the system is developed and faults are introduced into that model. The model is then simulated to find the effects of the fault on the operation of the system. These methods are often slower to test, but easier to change.

For each fault injection experiment that is performed there are three possible outcomes, or events, that can occur as a results of injecting the fault into the system under analysis. First, the fault could be covered. A fault being covered means that the presence and activation of the fault has been correctly mitigated by the system. Here, correct, and likewise incorrect, mitigation are defined by the designers of the system as a consequence of the designers defining the valid and invalid inputs, outputs and state for the system. The second possible outcome for a given fault injection experiment is that the fault is uncovered. An uncovered fault is a fault that is present and active as with a covered fault and thus producing an error. However, there is no additional mechanism added to the system to respond to this fault, or the mechanism is somehow insufficient and cannot identify the incorrect system behavior. The final possible outcome for a given fault injection experiment is that the fault causes no response from the system.

Application to a PWR Digital Feedwater Control System (DFWCS)

An overview of the DFWCS is shown in Figure 3. The function of the DFWCS is to control the water level in its associated steam generator from approximately 1% reactor output power up to 100% reactor output power.

The DFWCS contains a Main and Backup control system which read various inputs and provide control outputs. The monitored inputs include reactor power level, steam and water flow, water temperature, and water level. The controlled outputs set the operating positions of the Main Flow Valve (MFV), Bypass Flow Valve (BFV), and the Feedwater Pump (FWP) as shown in the overview figure. The Main and Backup Controllers operate as a redundant pair, each reading inputs and delivering outputs. In the event that the Main Controller fails, the Backup Controller can take over the operation of the system. The outputs of both the Main and Backup Controllers are passed to a set of four Proportion, Integral, and Derivative (PID) Controllers. Three of these controllers are dedicated to the MFV, BFV, and FWP respectively, with the fourth controller acting as a spare for either the MFV or BFV PID controllers. The PID associated with each of the controlled devices performs a comparison of the desired setting from the Main and Backup Controller and deliver the final output to the controlled device.

The DFWCS operates in one of two primary modes: (1) Low Power Control Mode, or (2) High Power Control Mode. The first mode is used when the reactor power is less than approximately 15% and in this mode the MFV is normally closed, and the BFV is manipulated to control the water level in the associated steam generator. The second mode is used when the reactor power is between approximately 15% to 100% and in this mode the BFV is normally closed, and the MFV is manipulated to control the water level in the associated steam generator. In order to increase the fault tolerance of the system, several Override Modes exist which under certain conditions allow the MFV or BFV to be used if the normally controlled valve has failed. For example, in Low Power Control Override Mode, the BFV can be bypassed and the MFV can be manipulated to control the water level. The Main and Backup Controllers are implemented with Industrial PCs with 486DX4 processors running at 33 MHz. The PID Controllers are implemented with digital Process Control Stations.

From a reliability perspective, the above detailed operation can be represented with the Dynamic Fault Tree shown in Figure 4. This dynamic fault tree can be converted to a Markov chain and solved using standard numerical

techniques to serve as the Analytical Safety Model for the DFWCS.

However, from a system safety perspective, the safety-related operation of the system can be viewed simply as two controllers operating in a redundant fashion, with their outputs being voted upon and delivered by a Comparator/Switch function. This model is the classic Reconfigurable Duplication with Comparison (RDWC) model discussed next (see Figure 5) and will be used to represent the safety-related operation of the DFWCS system. The Markov model is shown in Figure 6.

The Reconfigurable Duplication with Comparison (RDWC) architecture is a modification of the Duplicate with Comparison (DWC) architecture to incorporate active redundancy. As in DWC, two modules operate in synchrony while performing the same computation on the same set of inputs. The outputs of the two modules are either constantly or periodically compared by a Comparator and any discrepancy in the outputs indicates the presence of a fault. Each module also possesses self-diagnostic routines which attempt to detect any faults within the module itself. The different feature in RDWC is that if a module detects an internal fault, it will remove itself from the system via a switch mechanism and system operation will continue in Simplex mode using the other module. There may also be a concept of repair such that a failed module may recover and return the system to operation in full RDWC mode.

The resulting equation for the Mean Time to Unsafe Failure(MTTUF) is shown in Figure 7. The importance of this equation is that it shows that for given values of failure rate and comparator coverage, MTTUF can be increased significantly with increasing values of module coverage. This result shows that by increasing coverage, with the same architecture (in this case, commercial-off-the-shelf (COTS)), digital system safety can be increased.

It is planned that both software-based and simulation-based fault injection techniques will be applied to the DFWCS. The software-based approach will use software interrupts in the operating system to modify register and memory contents during execution of the DWFCS software in the main controller. The simulation-based approach will use a commercial simulation tool (SIMICS) to seamlessly replace the main controller hardware with a simulation model which executes the unmodified DFWCS software. The SIMICS tool allows complete access to the programmer's model of the processor, such that memory contents can be corrupted as desired during the execution of the software.

The results of the fault injection experiments will provide the information needed to determine the coverage and hence estimate the point on the plot of the MTTUF equation (Figure 7) for the DFWCS.

Conclusions

This paper presented an overview of a valid safety assessment methodology for digital systems and the application to a PWR digital feedwater control system. The details of the analytical safety modeling approach were illustrated with Markov model representations of the systems under study. It was shown how Mean Time To Unsafe Failure (MTTUF) safety metric may be derived from the Markov model and the MTTUF can be estimated from fault injection experiments on the digital system.

Some general conclusion that can be drawn form this research at this time are:

1.) the same digital system architecture can result in different safety properties depending on the value of *fault coverage;*

2.) the methodology is not dependent on any knowledge of the hardware or software *development* methods or attributes, but will assess the results of the development methods in an objective manner;

3.) the methodology can accommodate proprietary special purpose systems or COTS component systems;

4.) the methodology tests the ability, accuracy, and efficiency of internal software and hardware diagnostics to perform their error detection and management capabilities.

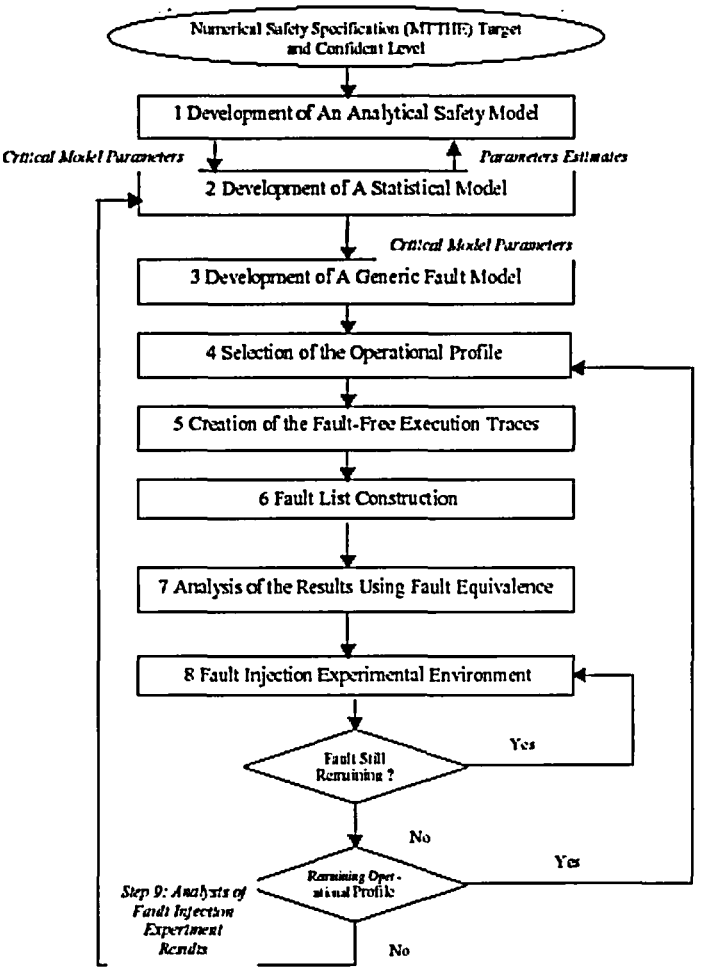


Figure 1: Methodology Overview

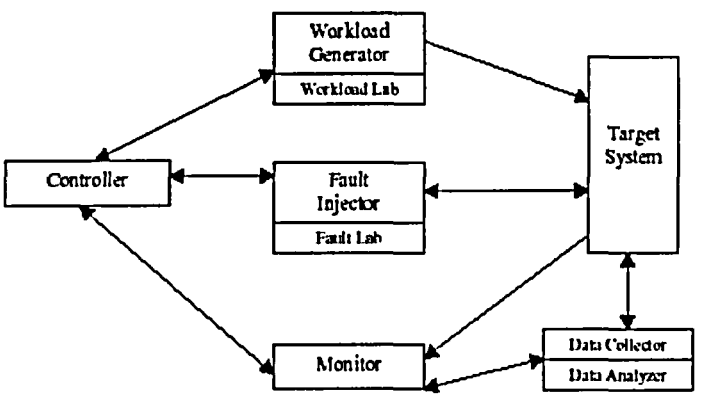


Figure 2: Fault Injection Environment

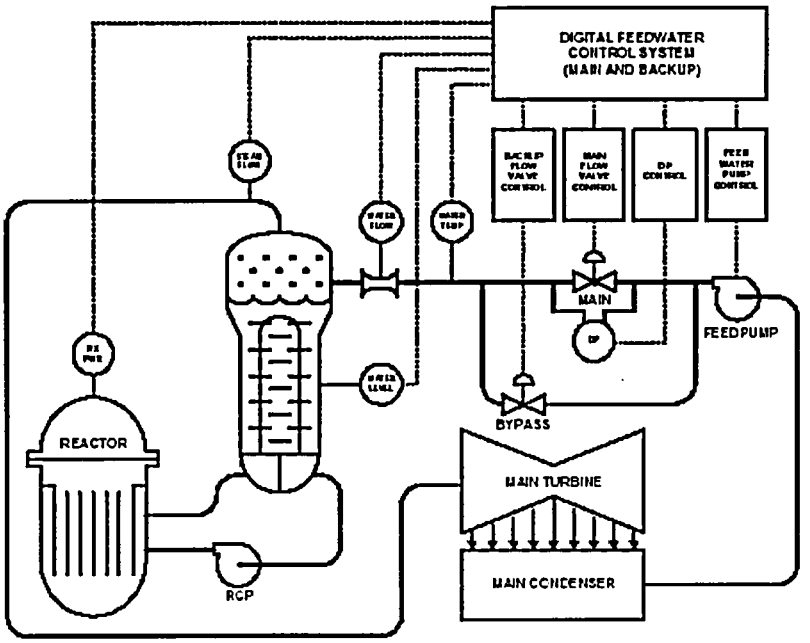


Figure 3: DFWCS Overview

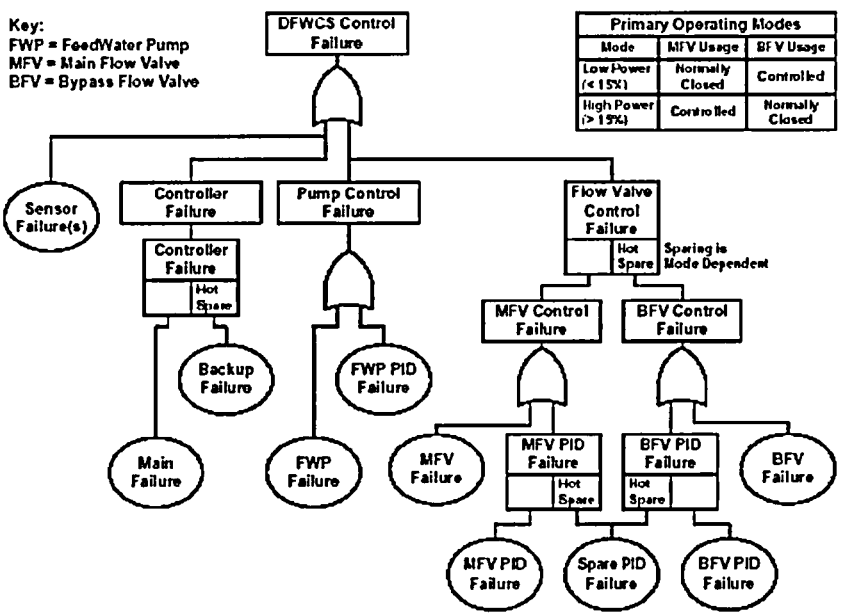


Figure 4: DFWCS Dynamic Fault Tree

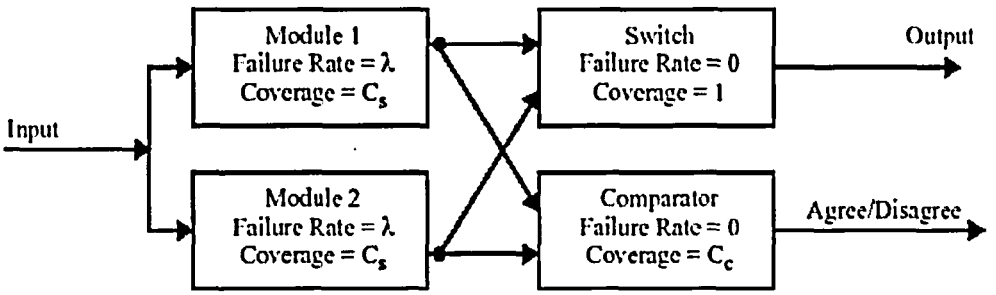
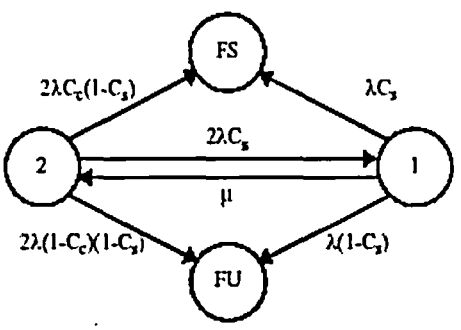


Figure 5: DFWCS Standard Architecture



States:
1 Single Unit Operational
2 Both Units Operational
FS Failed-Safe
FU Failed-Unsafe

Figure 6: Markov Model for DFWCS

$$\frac{1+2C_s}{2\lambda(1-C_s)[1-C_c+C_s]}$$

Figure 7: MTTUF Equation for DFWCS

A Numerical Safety Evaluation Process for Safety-Critical Systems

Technical Report UVA-CSCS-NSE-001
Revision 02
August 1, 2003

Author(s)	Contact Info
Eric Cutright	edc2u@virginia.edu
Todd DeLong	tad2x@virginia.edu
Barry Johnson	bwj@virginia.edu

University of Virginia Center for Safety-Critical Systems
Department of Electrical and Computer Engineering
Thornton Hall
351 McCormick Road
Charlottesville, VA 22904-4743

Prepared for:
Project Manager: John Calvert
Division of Engineering Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555

Table of Contents

List of Figures

Glossary of Acronyms and Abbreviations

Acronyms and Abbreviations

ALU	Arithmetic and Logic Unit
C	Fault coverage
CSCS	Center for Safety-Critical Systems
E	Event Space
F	Fault Space
FS	Failed-Safe state
FTA	Fault Tree Analysis
FU	Failed-Unsafe state
IC	Integrated Circuit
I/O	Input/Output
MTBHE	Mean Time Between Hazardous Event
MTTF	Mean Time To Failure
MTTFD	Mean Time To Fail Dangerous
MTTHE	Mean Time To Hazardous Event
MTTR	Mean Time To Repair
MTTUF	Mean Time To Unsafe Failure
O	Operational state
PFD(t)	Probability of Failure on Demand
PFD_{ss}	Steady-State Probability of Failure on Demand
Q(t)	Unreliability
R(t)	Reliability
S_{ss}	Steady-State Safety
S(t)	Safety
UVA	University of Virginia
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

1. Introduction

The University of Virginia (UVA) Center for Safety-Critical Systems (CSCS) has developed a methodology to quantify system safety for complex processor-based safety-critical systems. The approach is built upon a campaign of extensive fault injection which is used to derive a numerical expression of system safety known as the Mean Time To Hazardous Event (MTTHE) [41] (also known alternatively as the Mean Time To Unsafe Failure (MTTUF) [41] or Mean Time Between Hazardous Events (MTBHE) [11][14]). A separate risk assessment methodology is employed at the overall system level to derive MTTHE allocations for individual processor-based systems. The UVA safety quantification methodology is then applied to the individual processor-based systems in order to demonstrate compliance with their MTTHE allocations.

It is important to note that the described methodology is a strictly quantitative approach to demonstration of system safety, and must be accompanied by appropriate qualitative safety methods such as design reviews, hazard analyses, and fault tree analyses in order to present the complete safety case for a given system. These qualitative safety analyses would typically be performed by the system supplier, and are outside the scope of the numerical safety evaluation methodology presented in this document.

This overview document consists of thirteen sections. Following this introduction, Section 2 presents important background and terminology information, including the development of the MTTHE and fault coverage concepts. Next Section 3 presents an overview of the numerical safety evaluation process developed at the University of Virginia Center for Safety-Critical Systems. Section 4 through Section 11 provide further details on each major step in the process, with an overall summary presented in Section 12. Note that separate documents in this series will provide complete details on each of the process steps described in these sections. Section 13 then provides a list of the numerous references to industry literature that are made throughout the document to provide background material and to illustrate the state-of-the-art in fault injection. Copies of all papers referenced in the document can be provided upon request.

2. Background and Terminology

This section provides background and terminology information which is useful to the understanding of the numerical safety evaluation process. The concept of Mean Time To Hazardous Event (MTTHE) is developed along with several related safety and reliability metrics and their dependent factors.

2.1. Modeling of System Safety

A discussion on the modeling of system safety naturally begins with the definition of safety:

> **Definition 1: Safety,** $S(t)$, is the probability that a system will either perform its functions correctly or will discontinue its functions in a manner that does not disrupt the operation of other systems or compromise the safety of any people associated with the system [27].

There are two important things to note about this definition. First, we define safety as a probability, which implies that safety is a quantitative measure that can take on values between zero and one, inclusive.

$$0.0 \le S(t) \le 1.0 \tag{2.1}$$

Second, there is an inherent notion of system states corresponding to whether the system is safe or unsafe. The first state is associated with the system functioning correctly, and we will refer to this state as the Operational State. The second state is associated with the system discontinuing "... its functions in a manner that does not disrupt the operation of other systems or compromise the safety of any people associated with the system." In other words, the system ceases to perform its function, but it does so in a safe manner. We will refer to this state as the Failed-Safe State because the system has failed to perform its function but in a safe manner. The third and final state is essentially all the scenarios that are not included in the first two states. That is, the third state is associated with the system not performing its function correctly, and in addition possibly jeopardizing the safety of the system. We will refer to this third state as the Failed-Unsafe State because like the Failed-Safe State the system has failed to perform its function but has done so in an unsafe manner. The relationship between these three states is shown in **Figure 2.1.**

Referring to **Figure 2.1,** we will assume that the system begins in the Operational state, where it is performing its intended function correctly. According to **Figure 2.1,** the system can transition to either the Failed-Safe state or the Failed-Unsafe state (indicated by the arrows in **Figure 2.1**). Associated with each state transition is a parameter that indicates the rate at which the transition occurs, and as such is referred to as the transition rate. The state transition rate depends on two parameters: (1) the failure rate function, $\lambda(t)$, and (2) the fault coverage, $C(t)$, defined below.

> **Definition 2: Failure rate function,** $\lambda(t)$, represents the rate at which the system fails. For safety studies, this rate considers only those failures that can potentially result in an unsafe system state. The failure rate function may be time-varying or may be a constant. The typical approach is to assume that the failure rate function follows an exponential distribution, resulting in a constant failure rate, λ. One assumption that we will make regarding the failure rate function is that the value of the failure rate function is always

greater than zero. As such, the system will eventually transition from the Operational state to either the Failed-Safe state or the Failed-Unsafe state (that is, the system will eventually fail as $t \rightarrow \infty$).

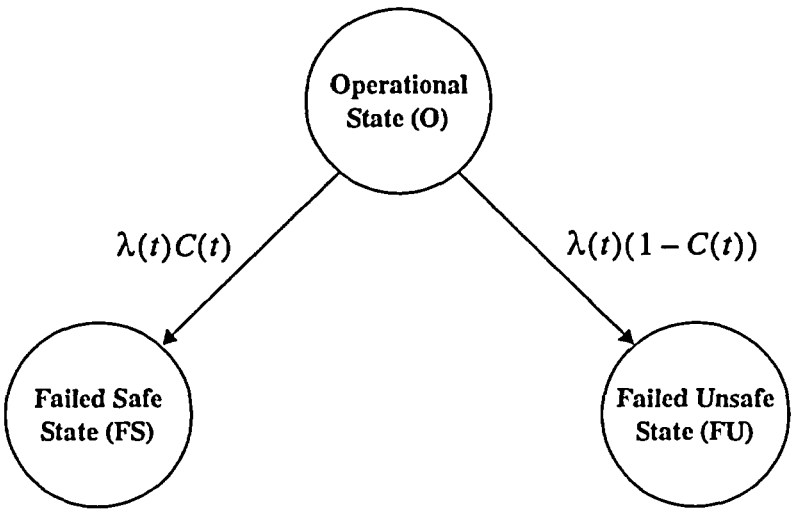


Figure 2.1. Three State System Model used to Calculate Safety

Definition 3: Fault coverage, $C(t)$, is the conditional probability that a system correctly handles a fault, given that a fault has occurred. The fault coverage function may be time-varying or may be a constant. The typical approach is to assume that the fault coverage is a constant, C. And, as with the safety, because fault coverage is a probability, it will take on values between zero and one, inclusive.

$$0.0 \leq C(t) \leq 1.0 \tag{2.2}$$

Referring to **Figure 2.1**, the transition rate from the Operational state to the Failed-Safe state is $\lambda(t)C(t)$, representing the case where a potentially unsafe fault has occurred and the system has correctly handled the fault and transitioned to a Failed-Safe state. The transition rate from the Operational state to the Failed-Unsafe state is $\lambda(t)(1-C(t))$, representing the case where a potentially unsafe fault has occurred and the system has NOT correctly handled the fault and has transitioned to a Failed-Unsafe state. For safety studies, it is assumed that a potentially unsafe fault which remains undetected by the system will place the system into a Failed-Unsafe state.

From the model of **Figure 2.1**, there are two general classes of safety and reliability metrics which can be computed:

1. The probability that a system is operating in a given state at time t, and
2. The expected time to some event of interest

Common "state probability"-based metrics from the first class include:

- Safety, $S(t)$
- Steady-State Safety, S_{ss}
- Probability of Failure on Demand, $PFD(t)$
- Steady-State Probability of Failure on Demand, PFD_{ss}

Common "expected time to event"-based metrics from the second class include:

- Mean Time To Hazardous Event, *MTTHE*
- Mean Time to Unsafe Failure, *MTTUF*
- Mean Time To Fail Dangerously, *MTTFD*
- Mean Time To Failure, *MTTF*
- Mean Time To Repair, *MTTR*

Each of these metrics will be discussed in the following sections.

2.2. Derivation of Safety (S(t)) Metric

To derive an expression for safety, *S(t)*, refer back to Definition 1. From this definition, we can calculate an expression for safety by determining the probability of being in either the Operational or Failed-Safe states for the simple three state model of **Figure 2.1**. Assuming a constant fault coverage C and constant failure rate λ, the safety expression for this model is given by:

$$S(t) = p_{OP}(t) + p_{FS}(t) = e^{-\lambda t} + (C - C \cdot e^{-\lambda t}) = C + (1 - C) \cdot e^{-\lambda t} \qquad (2.3)$$

where $p_{OP}(t)$ and $p_{FS}(t)$ indicate the probability of being in the Operational and Failed-Safe states, respectively, at time t.

2.3. Derivation of Steady-State Safety (S_{SS}) Metric

Another useful and common metric for quantifying system safety is the steady-state safety, S_{ss}. The steady-state safety may be calculated from the expression for system safety, $S(t)$. Again, from Definition 1, we conclude that the safety for a system which can be represented by **Figure 2.1** is the probability of either residing in the Operational State or the Failed-Safe State.

> **Definition 4:** The **Steady-State Safety, S_{SS}**, is a probability representing the evaluation of safety as a function of time, in the limiting case as time approaches infinity:

$$S_{ss} = \lim_{t \to \infty} S(t) \tag{2.4}$$

Note that S_{ss} represents the probability that the system will operate safely after it is placed into service. S_{ss} represents the minimum safety for a safety-critical system and as such provides a lower bound for the system safety under most conditions. It is also a useful metric in that the fault coverage represents a conservative lower bound on steady-state safety, such that this bound can be expressed independently of the system failure rate (which may be difficult to estimate, especially for design faults), in contrast to MTTHE which requires estimation of the system failure rate.

As an S_{ss} example, consider the simple model of **Figure 2.1**, with a constant fault coverage C and constant failure rate λ. In this case, the S_{ss} expression for the system can be derived from the safety expression of **Equation (2.3)** as:

$$S_{ss} = \lim_{t \to \infty} S(t) = \lim_{t \to \infty} (C + (1 - C) \cdot e^{-\lambda t}) = C \tag{2.5}$$

2.4. Derivation of Probability of Failure On-Demand (PFD(t)) Metric

A useful metric for quantifying system safety is the Probability of Failure on Demand, $PFD(t)$. This metric represents the probability that the system has failed in an dangerous manner at a given instant of time. In other words, this metric represents the probability that the system has failed dangerously at the time it is needed to respond to a demand [20]. This metric is particularly useful for quantifying the safety of protection systems (e.g. nuclear reactor protection systems), which perform safety-critical interventions in the event of a system failure to bring the system to a safe state.

The formulation of this metric requires a slight modification to the three-state safety model as illustrated in **Figure 2.2**. Comparing this to **Figure 2.1**, the three states are Operational, Failed-Safe and Failed-Dangerous instead of Operational, Failed-Safe, and Failed-Unsafe. The definition for the Operational state is the same as stated in **Section 2.1**. In this case, the Failed-Safe state refers to the situation where the system shuts down even when there is no potentially dangerous situation. A failure of this type would be considered a safe failure, a false alarm or a false trip. The Failed-Dangerous state covers the case when the system would be unable to respond to a demand, a potentially dangerous condition [20]. Referring to **Figure 2.1**, the failed-unsafe state of this model is replaced with the failed-dangerous state. The remaining states remain unchanged.

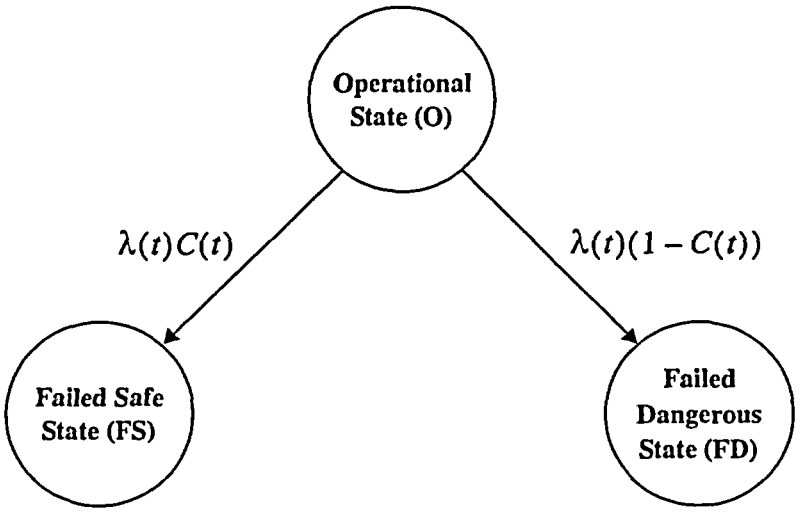


Figure 2.2. Modified Three State System Model used to Calculate MTTFD and PFD

Referring to **Figure 2.2**, we can calculate $PFD(t)$ as the following:

$$PFD(t) = p_{FD}(t) = 1 - p_O(t) - p_{FS}(t) \tag{2.6}$$

As an example of the $PFD(t)$ metric, consider the simple model of **Figure 2.2**, with a constant fault coverage C and constant failure rate λ. In this case, the $PFD(t)$ expression for the system is:

$$PFD(t) = p_{FD}(t) = 1 - p_O(t) - p_{FS}(t) = 1 - e^{-\lambda t} - (C - C \cdot e^{-\lambda t}) = (1 - C) + (1 - C) \cdot e^{-\lambda t} \tag{2.7}$$

2.5. Derivation of Steady-State Probability of Failure On-Demand (PFD_{SS}) Metric

Another useful metric for quantifying system safety is the steady-state Probability of Failure on Demand, PFD_{ss}, which can be calculated from the expression for $PFD(t)$. From **Figure 2.2** we conclude that PFD(t) is the likelihood of residing in the failed-dangerous state.

> **Definition 5:** : The **Steady-State Probability of Failure On-Demand, PFD_{SS}**, is a probability representing the evaluation of PFD as a function of time, in the limiting case as time approaches infinity:
>
> $$PFD_{ss} = \lim_{t \to \infty} PFD(t) \tag{2.8}$$

As an PFD_{ss} example, consider the simple model of **Figure 2.2**, with a constant fault coverage C and constant failure rate λ. In this case, the PFD_{ss} expression for the system is:

$$PFD_{ss} = \lim_{t \to \infty} PFD(t) = \lim_{t \to \infty} [(1 - C) + (1 - C) \cdot e^{-\lambda t}] = (1 - C) \tag{2.9}$$

Note that $PFD(t)$ is related to $S(t)$ (sometimes referred to as Safety Availability, *SA(t)* [20]) by the following relationship:

$$PFD(t) = 1 - S(t) = 1 - SA(t) \tag{2.10}$$

In addition, PFD_{ss} is related to S_{ss} by:

$$PFD_{ss} = 1 - S_{ss} \tag{2.11}$$

2.6. Derivation of Mean Time To Hazardous Event (MTTHE) Metric

For safety studies, a useful mean time metric is the Mean Time to Hazardous Event (MTTHE), also known as the Mean Time To Unsafe Failure (MTTUF) [41]. Here the "event" is an unsafe system failure (as opposed to a system failure for the MTTF). The MTTHE is the primary safety metric that will be used in the UVA numerical safety evaluation process and is defined below:

> **Definition 6:** The **Mean Time To Hazardous Event (MTTHE)** is the expected time that a system will operate before the first *unsafe* failure occurs. Mathematically, the MTTHE is defined as the expected value of a random variable, X, which represents the time to the first unsafe failure. It can be defined as a continuous-time or discrete-time function, but will be defined as a discrete-time function here since that is how it is used later in this document. Given this, then
>
> $$E[X] = MTTHE = \sum X_i \times P_i \tag{2.12}$$
>
> where E is the statistical expectation operator for a probability mass function, and P_i is the likelihood of occurrence of X_i [41].

As an MTTHE example, consider the simple model of **Figure 2.1**, with a constant fault coverage C and constant failure rate λ. In this case, the MTTHE expression for the system is:

$$MTTHE = \frac{1}{\lambda(1-C)} \tag{2.13}$$

Note that MTTHE is equivalent to the metrics Mean Time Between Hazardous Events (MTBHE) and Mean Time To Unsafe Failure (MTTUF), which have been documented in literature [41][14]. As such, MTTHE will be used throughout the remainder of this report, knowing that the results apply equally to MTBHE or MTTUF.

2.7. Derivation of Mean Time to Fail Dangerous (MTTFD) Metric

As with the Probability of Failure on Demand metric, the Mean Time To Fail Dangerous (MTTFD) metric is particularly useful for quantifying the safety of protection systems (e.g. nuclear reactor protection systems), which perform safety-critical interventions in the event of a system failure to bring the system to a safe state. From the model of **Figure 2.2**, the MTTFD metric is defined as:

> **Definition 7:** : The **Mean Time To Fail Dangerous (MTTFD)** is the expected time that a system will operate before the first dangerous failure occurs [20].

Mathematically, the MTTFD can be defined the same manner as MTTHE in **Equation (2.12)**.

As an MTTFD example, consider the simple model of **Figure 2.2**, with a constant fault coverage C and constant failure rate λ. In this case, the MTTFD expression for the system is:

$$MTTFD = \frac{1}{\lambda(1-C)} \tag{2.14}$$

Note that the MTTFD is equivalent to the MTTHE metric presented earlier. This metric is also equivalent to the MTBHE and MTTUF. As such, MTTHE will be used throughout the remainder of this report.

2.8. Other Important Metrics

Other important reliability metrics commonly used in safety-critical applications are shown below. In contrast to safety studies, reliability studies are typically concerned with all types of failures, not just those that can affect the safety of the system. Thus, whereas the failure rate function for safety studies is limited to consideration of only those failures that can potentially result in an unsafe system state, the failure rate function for reliability studies will also reflect other types of failures in the system, especially those that interrupt system service.

> **Definition 8:** The **reliability,** $R(t)$, of a system is a function of time, defined as the conditional probability that the system will perform correctly throughout the interval $[t_0,t]$, given that the system was performing correctly at time t_0 [27].
>
> **Definition 9:** The **unreliability,** $Q(t)$, of a system is a function of time, defined as the conditional probability that the system will perform *incorrectly* during the interval $[t_0,t]$, given that the system was performing *correctly* at time t_0 [27].

Note that the unreliability and reliability of a system are directly related. Specifically, the probability that the system is failed or operational must always equal 1.0. This relationship is expressed as:

$$R(t) + Q(t) = 1.0 \tag{2.15}$$

> **Definition 10:** The **Mean Time To Failure (MTTF)** [27] is the expected time that a system will operate before the *first* failure occurs.

Note that based on this definition, it can be shown that the MTTF for a system can be computed if the reliability of the system, $R(t)$, is known, and $R(t)$ approaches zero as $t \rightarrow \infty$ (from [27]):

$$MTTF = \int_0^\infty R(t)dt \tag{2.16}$$

3. Numerical Safety Evaluation Using Fault Injection

One quantitative method to demonstrate the MTTHE compliance of a system is the evaluation of the system fault processing capabilities using fault injection. The general approach is to inject randomly selected faults into the system (on a working prototype, for example) to determine if the fault processing capabilities mitigate the faults. The safety quantification approach described in this document builds upon the University of Virginia's extensive research experience in this area. The methodology is outlined in **Figure 3.1** and consists of the following steps:

1. An analytical safety model is developed from the system architecture and inter-component dependencies to derive an expression for the MTTHE. The MTTHE expression is a function of *Critical Model Parameters* such as the fault coverage values and failure rates for the various components of the system. The analytical safety model is discussed in Section 4.
2. A statistical model is then developed based on those found in published literature and is used to estimate the *Critical Model Parameters* that are required by the analytical safety model. This statistical model is also used to calculate the number of fault injection experiments required to meet the numerical safety target for a given confidence level. The statistical model is discussed in Section 5.
3. A high-level generic processor fault model is defined to specify the types of faults (and their associated probabilities) that will be injected into the system under analysis. This process builds upon recent research at UVA in generic processor fault modeling, undertaken with the goal of characterizing low-level internal processor faults at the higher register-transfer level, in order to demonstrate that faults injected on the actual hardware are representative of the low-level processor faults of concern. The generic processor fault model is discussed in Section 6.
4. One or more operational profiles are then defined which will be used to drive the inputs to the system under analysis during the fault injection process. The operational profiles must be representative of the different system configurations and workloads that would be experienced in actual field operation. The selection of the operational profiles is discussed in Section 7.
5. A fault-free execution trace is created for each selected operational profile that will be used to generate the list of faults to inject into the system under analysis. This trace will also be used during the analysis of the fault injection experimental results. The creation of this fault-free execution trace is discussed in Section 8.
6. Using the fault-free execution trace and the fault categories from the generic processor fault model, a fault list construction algorithm is applied to generate a list of possible faults that can be injected into the system under analysis, and which are likely to have an effect on the system. From this complete set of responsive faults, a fault list selection algorithm is then applied to randomly select a list of faults to be injected into the system, using the fault categories and associated occurrence probabilities from the generic processor fault model. The fault list construction and fault list selection algorithms are discussed in Section 9.
7. A fault equivalence algorithm is applied to each fault list to identify those sub-sets of faults which will have the same effect on the system, known as fault equivalence classes. The set of faults to be injected into the system is then reduced using the fault equivalence information, since only one fault from each class needs to be injected. This reduces the amount of time

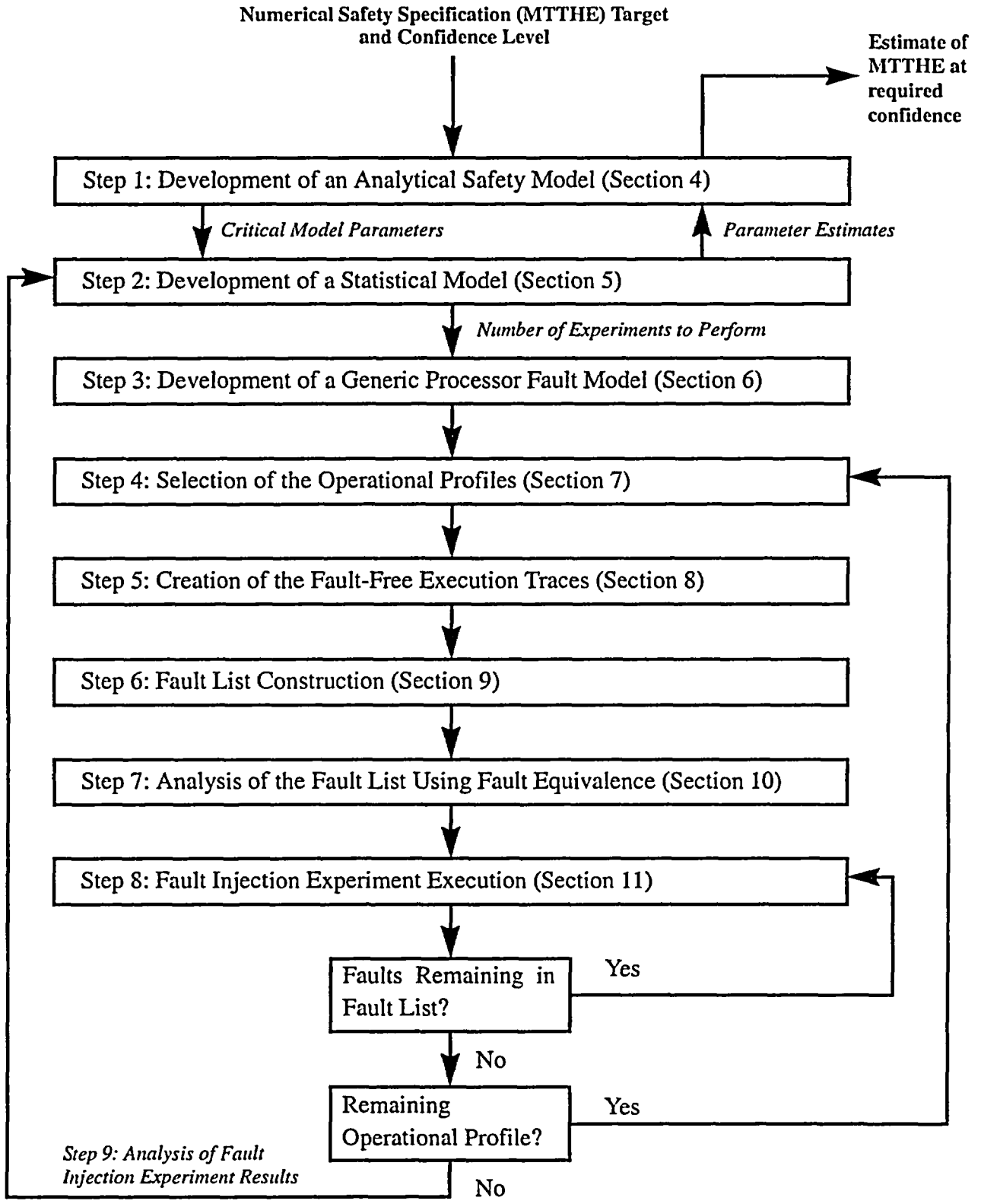


Figure 3.1. Process for quantitative safety evaluation using fault injection

required to perform the evaluation of the system under analysis by reducing the total number of fault injection experiments which must be performed. The fault equivalence approach is discussed in Section 10.

8. Each fault from each reduced fault list is then injected into the system under analysis, using one of four fault injection techniques: (1) hardware-based, (2) software-based, (3) simulation-based, or (4) a hybrid approach. A summary and comparison of the four fault injection techniques and the execution of the fault injection experiments, including the experimental environment that is needed to support this effort, is discussed in Section 11.
9. The system is monitored during each fault injection experiment, and each fault is classified as covered, uncovered, or non-responsive by comparing the actual execution trace to the fault-free execution trace. Once all of the results of the fault injection experiments have been collected, they are fed back into the statistical model to calculate the estimates of the *Critical Model Parameters*. The *Parameter Estimates* are then in turn fed back into the analytical model to calculate the estimate of the MTTHE, as shown in **Figure 3.1.**

4. Development of an Analytical Safety Model

The evaluation process outlined in **Figure 3.1** begins with the development of a high-level analytical model. The purpose of this model is to provide a mathematical framework for calculating the estimate of the MTTHE. The input to the model is the MTTHE allocation along with the confidence level required in the estimate. For the purposes of a safety evaluation, the analytical model is used to represent, at a high level, the faulty behavior of the system under analysis. Suitable analytical models include Markov models [9][31], Petri nets [9][37][38], and fault trees [36]. The model uses various critical parameters such as failure rates, fault detection latencies, and fault coverages to describe the faulty behavior of the system under analysis. This is shown in **Figure 3.1** as the arrow labeled *Critical Model Parameters*, shown as an input to the box that represents the statistical model (see Section 5). Once the faulty behavior has been sufficiently described, an analytical model is converted to a corresponding mathematical representation that can be numerically solved to provide a quantitative estimate of dependability measures, such as safety. The solution of the model to generate the quantitative estimate is highly dependent upon the parameters used to describe the faulty behavior. The estimates to these parameters are shown in **Figure 3.1** as the arrow labeled *Parameter Estimates*. As such, estimating numerical values for these parameters becomes the crux of the evaluation process. Of all the parameters that typically appear in an analytical model used as part of a safety evaluation, the fault coverage is the most difficult to estimate.

5. Development of a Statistical Model

Once the analytical model has been developed, the next step in the evaluation process outlined in **Figure 3.1** is to develop a statistical model to estimate the parameters that are required by the analytical model. The estimates of the parameters are generated after all of the fault injection experiments have been performed and the results have been collected.

Common statistical models from the literature that can be applied in this process include:

- Single-sided confidence interval
- Double-sided confidence interval
- Bayesian approach
- Statistics of the extremes

One of the most important uses of the statistical model is to determine how many fault injection experiments must be performed in order to estimate the parameters at the desired confidence level. This information is crucial at the beginning of a safety evaluation because it helps to determine which of the four fault injection techniques will be used to generate the results needed by the statistical model to estimate the parameters in the analytical model. See Section 11 for a review of the four fault injection techniques, listing the advantages and disadvantages of each technique.

6. Development of a Generic Processor Fault Model

The next step in the evaluation process is the development of a generic processor fault model that accurately represents the types of faults that can occur within the system under analysis. The fault model is used to generate the fault space, *F*. *F* is usually a multi-dimensional space whose dimensions can include fault characteristics such as location, time, and value, as shown in **Figure 6.1** [27]. Here, time represents the time of occurrence and duration, location is where the fault occurs within the system under analysis, and value represents the form of the corruption. Note that value can be something as simple as a mask, or something more complex that is state dependent.

In general, completely proving the sufficiency of the fault model used to generate *F* is usually very difficult, if not impossible. The fault modeling of the applied processor is obviously the most problematic. It is more traditional to assume that the fault model is sufficient, justifying this assumption to the greatest extent possible with experimental data, historical data, or results published in literature.

To this end, UVA has developed a behavioral-level generic processor fault model that represents the faulty behavior of a general-purpose, implementation-independent microprocessor like the one shown in **Figure 6.2**. The processor shown performs a basic fetch-execute instruction cycle typical of a von Neumann architecture. As such, it contains a control unit, which operates as a synchronous finite state machine, and a datapath, consisting mainly of combinational logic and some storage elements, which performs the information processing within the processor. The datapath will typically contain some sort of register file that contains general-purpose and special-purpose registers, a program counter, an Arithmetic and Logic Unit (ALU), and some sort of fetch and decode logic block. And finally, the processor contains an interface that allows for communication with entities external to the processor. These are labeled as the **Data Bus**, **Address Bus**, and **Control Bus** in **Figure 6.2**. In addition, the processor contains internal signals that allow for communication between the datapath and control unit. These are labeled as **Control Signals** and **Status Information** in **Figure 6.2**.

The basis for this fault model is the behavioral, processor-level fault models that have been published in the literature. As early as 1980 [42], researchers have attempted to develop a processor-level fault model that accurately represents the faulty behavior of a microprocessor like the one shown in

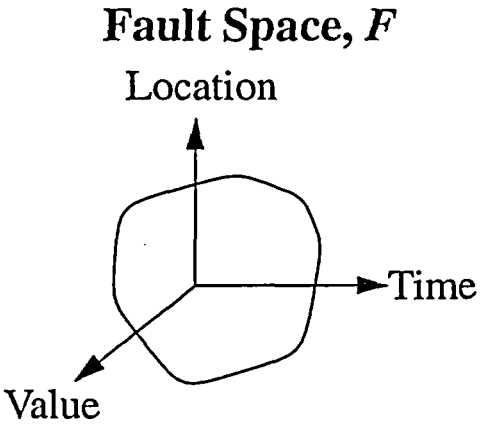


Figure 6.1. The system fault space, *F*, characterized by location, value, and time

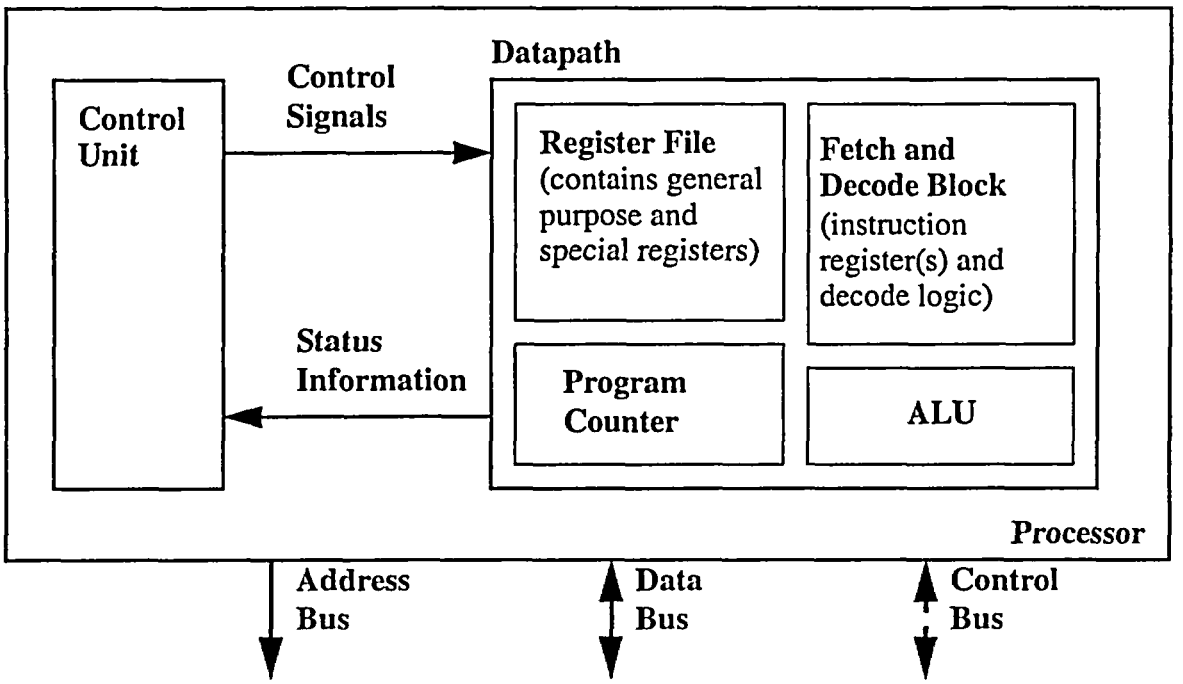


Figure 6.2. Generic, implementation-independent architecture for a general-purpose processor

Figure 6.2. Most of the research focused on using the fault models for developing test scenarios for detecting the various faulty behaviors defined by the fault model. However, some researchers used the models that were developed to perform a dependability evaluation of the system under analysis, similar to our approach [1][2][3][4][5][6][7][10][12][13][16][17][18][22][23] [28][29][30][33][39][43][44][45].

The effectiveness of the generic processor fault model has been demonstrated by developing a gate-level model of an actual 32-bit RISC processor (with a structure similar to the one shown in **Figure 6.2**) and performing simulation-based fault injection experiments to demonstrate that all faulty behaviors that are produced by gate-level faults (for instance, stuck-at faults) can be represented by the behavioral-level fault model. Any unusual faulty behavior discovered as a result of these fault injection experiments was then used to augment the behavioral-level generic processor fault model.

Once the generic processor fault model has been developed, an analysis may be performed to ensure that the experimental environment that will be used to perform the fault injection experiments (see Section 11) fully supports the fault model.

7. Selection of the Operational Profiles

Operational profiles to be used in the fault injection experiments must be selected to be representative of the system under various modes of operation and configuration. System configurations may invoke different hardware and software modules, and it is important that the experiments include sufficient combinations of these to ensure a thorough evaluation of their behavior in the presence of faults.

Similarly, each configuration will generally have many modes of operation, and not all of these provide appropriate times during which to perform fault injection experiments. For example, a typical fault injection scenario would include a startup sequence used to bring the system to a well defined state prior to the period during which faults may be injected. From a statistical point of view, since the startup time is negligible compared to the operational time, the system startup is not a representative operational profile.

Most real-time hardware/software systems operate on an event-triggered basis; that is, when an event occurs, it is immediately serviced by the system. Of course, the event itself can generate other events to service. If there is no event from the outside (simulated) environment, only a reduced set of software and hardware resources may be used (cyclic idle tasks, diagnostics, and so forth). This portion of the operational profile will be referred to as a system ***light*** workload. A transient fault that occurs during a system light workload has a high probability of <u>not</u> being activated as an error. If activated as an error it may not produce a failure until an input/output activity starts. Also a permanent fault that occurs during a system light workload has a high probability of <u>not</u> being activated as an error by the idle tasks, but it has a high probability of being activated as an error and detected by the system diagnostics. This assumes that the observation interval is longer than the time in which diagnostics are completed. Therefore, a system light workload, followed by some input/output, has to be in the selected operational profiles to properly exercise the system.

If there are many events from the outside (simulated) environment, a majority of the software and hardware resources are used. This portion of the operational profile will be referred to as a system ***heavy*** workload. In general, when testing the functionality of a given system, the operational profiles should be selected to exercise as much of the system as possible (assuming that exhaustive testing is infeasible due to time and resource limitations). This becomes especially important in a safety evaluation effort using fault injection since it has been shown that certain operational profiles can mask faults within the system [15]. A transient fault that occurs during a system heavy workload has a high probability of being activated as an error, and it could quickly produce (if not detected) a failure due to the high input/output activity. Also, a permanent fault that occurs during a system heavy workload has a high probability of being activated as an error before being detected by some diagnostic. It also has a certain probability of being activated as an error and detected by the system diagnostics, assuming that the observation interval is longer than the time in which diagnostics are completed.

Faults injected during the system heavy workload can be used to determine if the system is able to mitigate faults in a safe manner before an incorrect output changes the environment state (that is, an incorrect output has to be stable for a given period, T, to affect the environment). Therefore, a system heavy workload, followed by a short "no activity" interval (to allow the outputs to stabilize), has to be

included in the selected operational profiles to properly exercise the system. A system heavy workload can be defined, compressing in time and mixing various operational conditions of the environment controlled by the selected system configuration. It is not necessary to create complex operational scenarios, but to define simple operational conditions and execute them serially or in parallel. Conceptually this is similar to defining a set of simple numerical expressions and then building a workload as a sequence of them instead of defining a complex numerical expression [1].

In summary, the operational profile shown in **Figure 7.1** is considered to be representative of the types of operational profiles that need to be used for the fault injection experiments in order to properly exercise complex hardware/software systems. As such, operational profiles like the one shown in **Figure 7.1** are used during the fault injection experiments, the results of which are then used to statistically estimate the fault coverage for the various system components. As shown in **Figure 7.1**, the operational profiles will consist of

- a system start-up (**Phase 1**),
- a system light workload (**Phase 2**),
- a system heavy workload (**Phase 3**), and
- a short system light workload (**Phase 4**).

Faults are injected only during **Phase 2** and **Phase 3**, but not during **Phase 1** and **Phase 4** due to the justifications presented above. It is worth noting that the window shown in **Figure 7.1** can be seen as the operational profile to be applied for a given fault injection experiment. Sliding the window left and right, we have a set of operational profiles with many stress conditions.

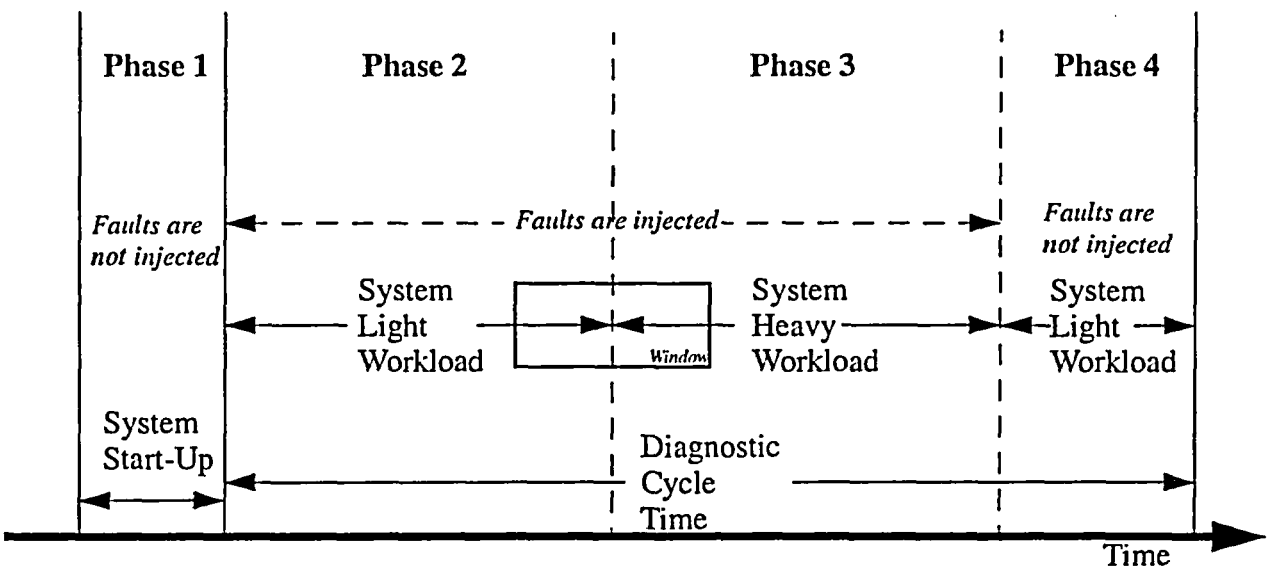


Figure 7.1. Typical operational profiles applied during fault injection

8. Creation of the Fault-Free Execution Traces

For each operational profile that is selected, a fault-free execution trace must be created to support the fault list generation and analysis efforts in the safety evaluation process (see Section 9 and Section 10). The trace will contain information, such as the sequence of instructions that are executed during a given period of observation, as well as the state information that is visible (that is, the information that is accessible by a programmer using the programmer's model view of the processor).

The experimental environment that will be used to perform the fault injection experiments is constructed in such a manner as to support the creation of the fault-free execution traces as well. In general, an execution trace requires information that is typically accessible using tools such as logic analyzers, bus analyzers, in-circuit emulators, or software debuggers.

9. Fault List Construction

Once the fault space, *F*, has been defined, the next step is to randomly sample the fault space to generate a set of fault injection experiments. The fault injection experiments are shown in **Figure 9.1** as the **Set of Experiments**. The set of experiments will not be an exhaustive set due to the size of the fault space (which is assumed to be infinite), but it is assumed that the set of experiments is a representative sample of the fault space to the extent of the fault model and to the confidence interval established by the number of samples.

For each fault injection experiment that is performed, there are three possible outcomes, or events, that can occur as a result of injecting the fault into the system under analysis. First, the fault could be covered. A fault being covered means that the presence and activation of the fault (which produces an error [32]) has been correctly mitigated by the system. The second possible outcome for a given fault injection experiment is that the fault is uncovered. An uncovered fault is a fault that is present and active as with a covered fault (and thus producing an error). However, there is no additional mechanism added to the system to respond to this fault, or the mechanism is somehow insufficient and cannot identify the incorrect system behavior. As such, any discrepancies that are produced as a result of the fault are not identified, and thus the fault is uncovered.

The final possible outcome for a given fault injection experiment is that the fault causes no response from the system. A no-response fault is one that is present, but it is not active. For instance, a fault could be present in a given portion of memory that is never accessed by the system. Or, if it is active, it is not producing an error due to the fact that its effect is being masked by the system. For instance, a stuck-at-1 fault could be present on a given signal line whose value is always a logic one. These no-response faults are typically referred to as latent or dormant faults [27][32]. During a system evaluation using fault injection, it is desirable to reduce or eliminate no-response faults. No-response faults provide no additional information with regard to computing the system coverage. In essence, the results from fault injection experiments for no-response faults can be discarded without affecting the coverage estimate. Note that in order to discard the fault, it must be shown that the fault is truly a no-response fault. For instance, a fault that occurs within a portion of memory that is never accessed by the system can be considered a no-response fault (assuming a single fault occurrence so that the rest of the system is fault-free). In this case, the fault is discarded since it will always be a no-response fault. However, if the

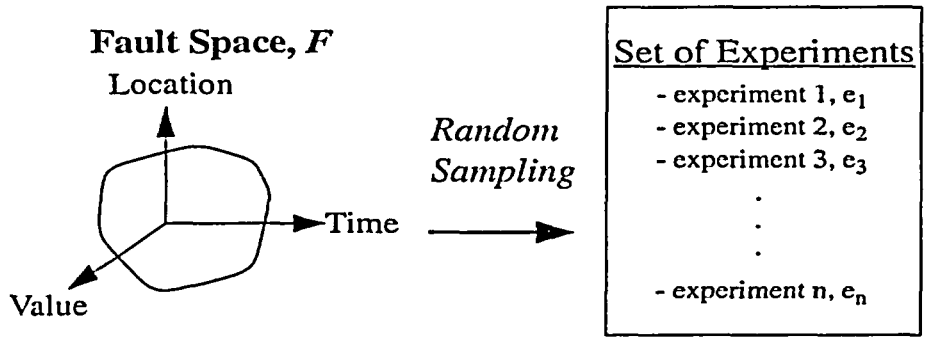


Figure 9.1. The generation of the set of experiments using the fault space, *F*

portion of memory where the fault occurs is used by the system, but not during the time interval for which the experiment is being conducted, then the fault cannot be considered a no-response fault. The reason for this is that the fault could eventually become active if the portion of memory where it occurs is accessed by the system. In this case, the fault is returned to the fault space as a potential candidate for selection when a parameter such as the time of observation, input operational profile, and so forth changes that could activate the fault.

In addition, no-response faults require the maximum amount of time to perform the fault injection experiment. This is due to the fact that the experiment is not terminated prematurely by a fault mitigation mechanism (such as a hardware watchdog timer or a software diagnostic) in response to an active fault. In general, reducing, or ideally eliminating, the number of fault injection experiments involving no-response faults will help to minimize the time and effort needed to perform the fault injection-based evaluation, which in general is a very resource-intensive process.

To this end, several researchers have addressed the problem of no-response faults by constraining the fault space from which the set of experiments is generated by performing some sort of algorithmic processing of the fault space [7][23][39][43] (using the fault-free execution trace that was mentioned in Section 8). This is shown graphically in **Figure 9.2** where a portion of the fault space has been identified as containing only covered and uncovered faults (shown as the darkened portion of the fault space in **Figure 9.2**) which always produce some effect in response to the presence and activation of the fault. The faults that are used to generate the set of experiments are randomly selected from this portion of the fault space to ensure that every fault that is injected will produce an error. Thus, no unnecessary fault injection experiments are performed, and the efficiency of the evaluation procedure is improved dramatically, even when taking into account the overhead associated with the extra processing required to identify the portion of the fault space that does not contain any no-response faults. Note that even though the procedure outlined in **Figure 9.2** ensures that the faults that are discarded are definitely no-response faults, the procedure does not ensure that the faults in the **Set of Experiments** will not be masked by some complex behavior of the system after the error is produced by the injected fault. These types of faults must be further analyzed to determine if they are covered or uncovered.

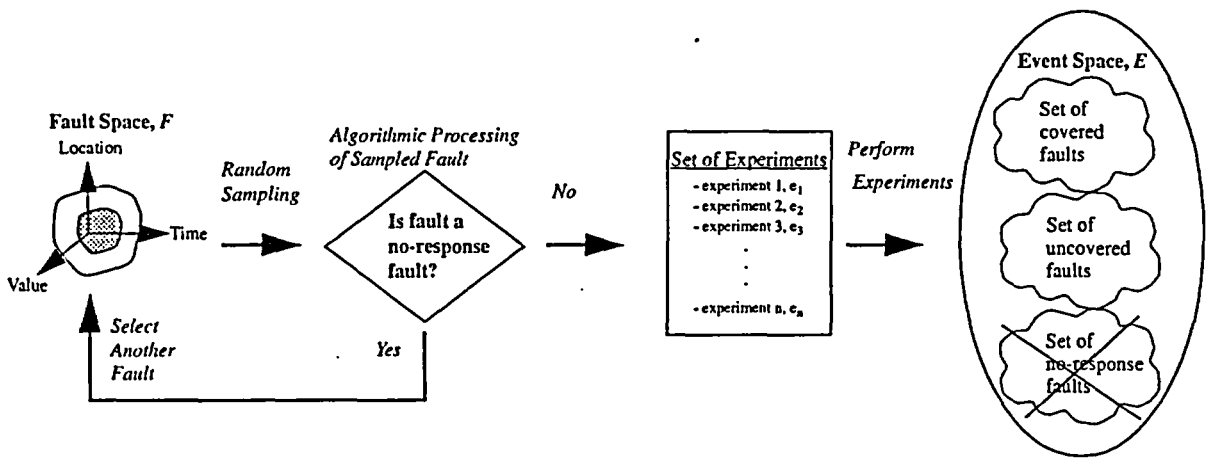


Figure 9.2. Constraining the fault space to eliminate no-response faults

10. Analysis of the Fault List Using Fault Equivalence

Another issue related to the safety evaluation of safety-critical systems has to do with the large number of fault experiments needed to estimate the coverage. Even if the no-response faults can be eliminated using some sort of algorithmic processing to constrain the fault space (as discussed in Section 9), the number of fault injection experiments needed to estimate the coverage with a reasonable confidence interval can still be extremely large. For instance, a rough rule of thumb is that a coverage estimate of 0.9_i, with a reasonable confidence interval (for example, 90%), requires performing approximately 10^i fault injection experiments. One can see how this could quickly become a huge liability, given that there will more than likely be limited time and resources to devote to performing the evaluation.

To overcome this problem, several researchers have proposed approaches that try to exploit some aspect of the system in order to reduce the total number of fault injection experiments that are actually performed while still maintaining a coverage estimate with a sufficient confidence interval [7][40][43]. Essentially, the idea is to identify sets of faults that have an identical effect on the system under analysis even though each fault in the set is distinct. These sets are referred to as fault equivalence classes. As a simple example, consider two faults, f_1 and f_2, that are only distinguishable by their times of occurrence, t_1 and t_2. If the faults do not become active until a time, $t_3 > t_1, t_2$, then these faults can be considered to be equivalent since they will have the same impact (that is, produce an error) at time, t_3. So, if it is possible to identify f_1 and f_2 as being equivalent, then only one fault injection experiment needs to be performed, using either fault, to determine the impact of both faults. The procedure developed by the University of Virginia to identify the fault equivalence classes is shown in **Figure 10.1.** Note that fault equivalence may be algorithmically determined by examining any of the three fault space axes of location, time, and value.

The approach for this effort differs from previous approaches, such as the earlier UVA approach presented in [40]. The current UVA approach has been developed to overcome the criticisms that have been expressed about the earlier approach in [40]. The fundamental difference is as follows. The technique presented in [40] begins by randomly selecting k responsive faults from the set of all faults, F. The faults are selected assuming that all faults have an equal probability of occurrence (that is, the faults in F occur according to a uniform distribution). Each of the k faults is then expanded using an algorithm

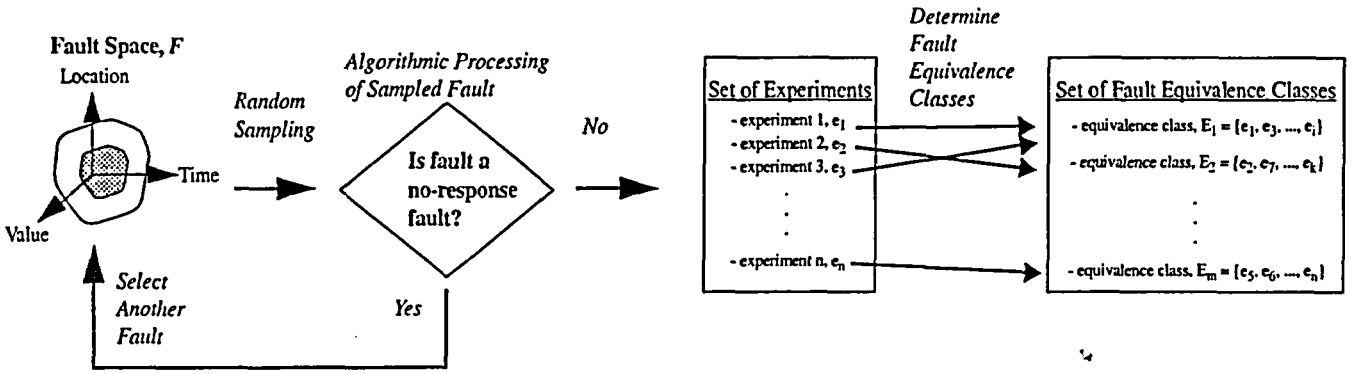


Figure 10.1. Determination of fault equivalence classes after randomly selecting n faults

to determine other faults that produce an equivalent effect on the system. The result is a total of z faults. The criticism of this approach is that the $(z-k)$ faults that are found algorithmically are not chosen randomly from the set of all faults, F, and thus are not a random unbiased representative sample.

In contrast, the current UVA approach is dramatically different. Here, a set of n responsive faults are randomly chosen, where n is at least as large as the total number of faults needed to quantify the coverage to the confidence level desired. Then, subsets of the n faults are algorithmically determined, where all faults in a particular subset produce an equivalent effect on the system. The result is m subsets that collectively contain a total of n faults. All of the n faults were randomly chosen from the total fault set, F, assuming that all faults have the same probability of occurrence (that is, the faults in F occur according to a uniform distribution). However, only m fault experiments need to be performed because of the equivalence of faults within a given fault equivalence class.

Thus, the UVA approach ensures that all n faults used in the estimation of coverage are indeed chosen randomly from the set of all faults, F. However, only m experiments must be performed to determine the results for the n faults because of the fault equivalence techniques shown in **Figure 10.1.** One important thing to note about the procedure outlined in **Figure 10.1** is that the determination of the fault equivalence classes from the sample fault space does not bias the coverage estimate in any way. If the assumption is made that all the faults within F are equally likely to occur, then there are no faults within the fault equivalence classes that violate this assumption. This is due to the fact that all of the faults that appear in the fault equivalence classes are faults from the sample fault space that are randomly selected from the fault space, F.

The use of fault equivalence provides a means to obtain results for a large number of faults (that is, whether a fault is covered or uncovered) while actually performing a relatively small number of fault injection experiments. Thus, the total time to perform the safety evaluation is reduced, and yet the required confidence level for the coverage estimate is maintained.

11. Fault Injection Experiment Execution

This section provides a summary of the four techniques that can be used in a fault injection-based quantitative safety analysis: (1) hardware-based, (2) software-based, (3) simulation-based, and (4) a hybrid approach, which is a combination of the other three.

11.1. Hardware-based fault injection

Hardware-based fault injection involves augmenting the system under analysis with specially designed test hardware to allow for the injection of faults into the system. Typically, these faults are injected at the Integrated Circuit (IC) pin level, though processors can sometimes be subjected to internal faults depending upon the test facilities built into the processor itself. See [2][3][4][22][34][35] for examples of hardware-based fault injection efforts.

A typical hardware-based fault injection environment is illustrated in **Figure 11.1.**

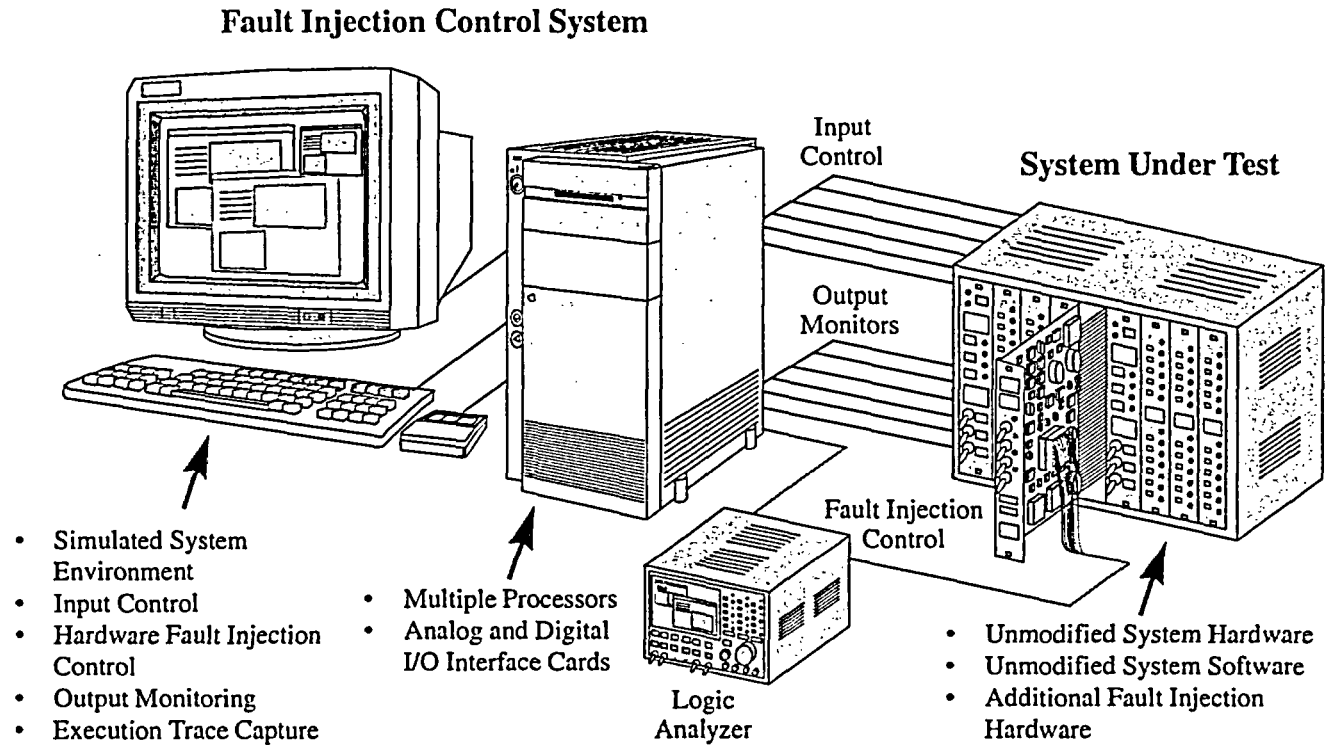


Figure 11.1. Typical hardware-based fault injection environment

11.1.1. Assumptions

1. Faults that are injected into the system are representative of the actual faults that occur within the system.
2. The additional hardware required to inject the faults does not affect the functional behavior of the system in response to the injected fault. Essentially, the assumption states that the hardware that is used to inject the fault is independent of the rest of the system, and that any faults present in the fault injection hardware will not affect the system under analysis.

11.1.2. Advantages

1. Experiments can be run in near real-time, allowing for the possibility of running a large number of fault injection experiments.
2. Running the fault injection experiments on the real hardware that is executing the real software has the advantage of including any design faults that might be present in the actual hardware and software design.
3. Fault injection experiments are performed using the same software that will run in the field.
4. No model development or validation required.
5. Able to model permanent faults at the pin level.

11.1.3. Disadvantages

1. A recent paper indicates that the setup time for each experiment might, in fact, offset the time gained by being able to performed the experiments in near real-time [19].
2. Requires special-purpose hardware in order to perform the fault injection experiments. This hardware is used to inject faults into the processor by applying the rail voltages (representing logic one and zero) to the Input/Output (I/O) pins of the processor. Also, if the processor contains appropriate special-purpose hardware known as scan chains [8], then the external hardware could also be used to inject stuck-at-1 and stuck-at-0 faults into the internal registers of the processor. In general, this hardware can be very difficult and costly to build.
3. Limited observability and controllability. At best, one would be able to corrupt the I/O pins of the processor and the internal processor registers.
4. It may be very difficult (if not impossible) to model permanent faults in the internal registers of the processor, even if the processor contains the appropriate additional hardware necessary to access them.

11.2. Software-based fault injection

Traditionally, software-based fault injection involves the modification of the software executing on the system under analysis in order to provide the capability to modify the system state (both processor registers and memory) according to the programmer's model view of the system. See [6][16][24][28] [29][33][43] for examples of software-based fault injection efforts.

A typical software-based fault injection environment is illustrated in **Figure 11.2.**

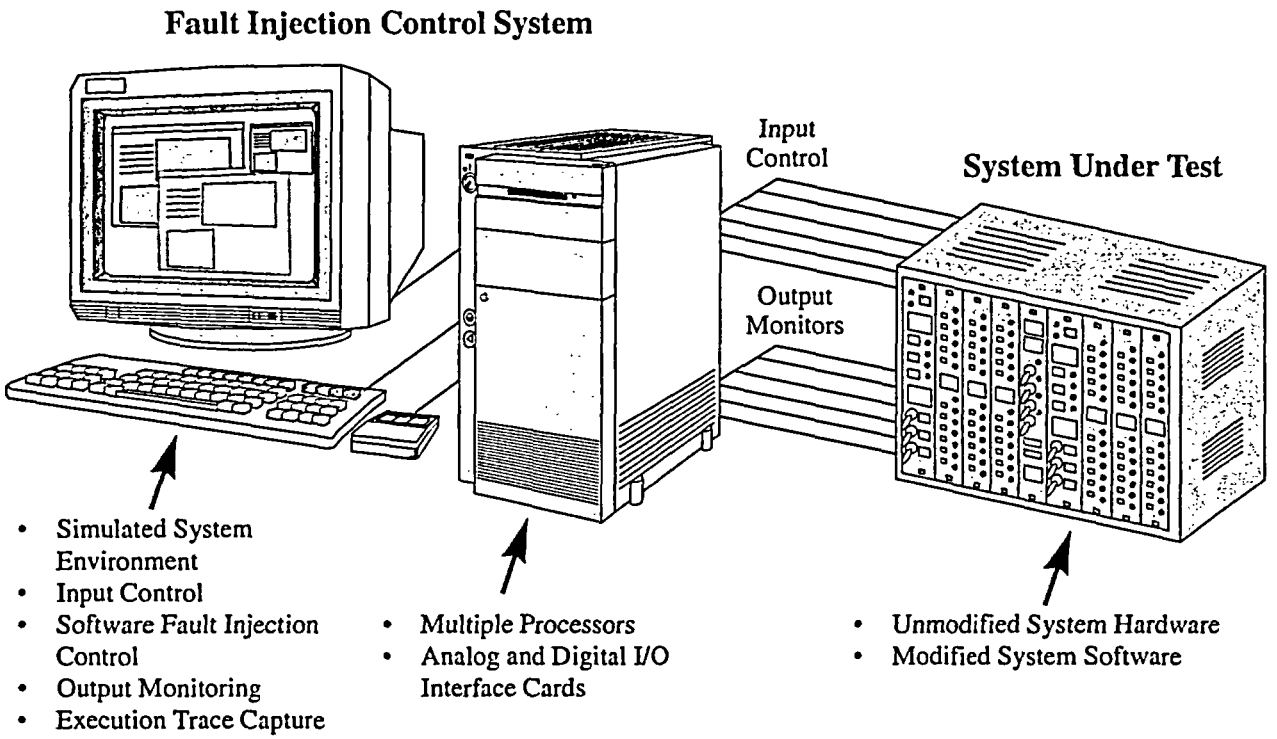


Figure 11.2. Typical software-based fault injection environment

11.2.1. Assumptions

1. Faults that are injected into the system are representative of the actual faults that occur within the system.
2. The additional software required to inject the faults does not affect the functional behavior of the system in response to the injected fault. Essentially, the assumption states that the software that is used to inject the fault is independent of the rest of the system, and that any faults present in the fault injection software will not affect the system under analysis.

11.2.2. Advantages

1. Experiments can be run in near real-time, allowing for the possibility of running a large number of fault injection experiments.
2. Running the fault injection experiments on the real hardware that is executing the real software has the advantage of including any design faults that might be present in the actual hardware and software design.
3. Does not require any special-purpose hardware.
4. No model development or validation required.

11.2.3. Disadvantages

1. Does require a modification of the source code to support the fault injection, which means that the code that is executing during the fault experiment is not the same code that will run in the field.
2. Limited observability and controllability. At best, one would be able to corrupt the internal processor registers (as well as locations within the memory map) that are visible to the programmer, traditionally referred to as the programmer's model of the processor.
3. Very difficult to model permanent faults.
4. Execution of the fault injection software could affect the scheduling of the system tasks in such a way as to cause hard, real-time deadlines to be missed, which violates assumption two.

11.3. Simulation-based fault injection

Simulation-based fault injection involves the construction of a simulation model of the system under analysis, including a detailed simulation model of the processor in use. The simulation models are developed using a hardware description language such as the Very high speed integrated circuit Hardware Description Language (VHDL) [25]. See [12][13][21][26][39][45] for examples of simulation-based fault injection efforts.

11.3.1. Assumptions

1. Model is an accurate representation of the actual system under analysis.

11.3.2. Advantages

1. Does not require any special-purpose hardware.
2. Fault injection experiments are performed using the same software that will run in the field.
3. Maximum amount of observability and controllability. This allows for support of a wide variety of fault models. Essentially, given sufficient detail in the model, any signal value can be corrupted in any desired way, with the results of the corruption easily observable regardless of the location of the corrupted signal within the model. This flexibility allows any potential failure mode to be accurately modeled.
4. Able to model both transient and permanent faults.
5. Allows modeling of timing-related faults since the amount of simulation time required to inject the fault is effectively zero.

11.3.3. Disadvantages

1. Extensive time required to perform model development.
2. Extensive time required to perform model validation.
3. Extensive time required to perform the fault experiments.
4. Model may not include any of the design faults that may be present in the real hardware.

11.4. Hybrid fault injection

A hybrid approach combines two or more of the other fault injection techniques to more fully exercise the system under analysis. For instance, performing hardware-based or software-based fault injection experiments can provide significant benefit in terms of time to perform the fault injection experiments, can reduce the initial amount of setup time before beginning the experiments, and so forth. However, given the significant gain in controllability and observability with a simulation-based approach, it might be useful to combine a simulation-based approach with one of the others in order to more fully exercise the system under analysis. For instance, most researchers and practitioners might choose to model a portion of the system under analysis, such as the Arithmetic and Logic Unit (ALU) within the microprocessor, at a very detailed level, and perform simulation-based fault injection experiments due to the fact that the internal nodes of an ALU are not accessible using a hardware-based or software-based approach. See [1][19][23][30][33][44] for examples of some hybrid efforts.

12. Summary

This document has presented an overview of a numerical safety evaluation process developed by the University of Virginia Center for Safety-Critical Systems. This approach can be applied to complex processor-based safety-critical systems to quantitatively demonstrate compliance to their numerical safety specifications, in this case the Mean Time To Hazardous Event (MTTHE). The methodology consists of the following steps:

1. Development of an Analytical Safety Model (Section 4)
2. Development of a Statistical Model (Section 5)
3. Development of a Generic Processor Fault Model (Section 6)
4. Selection of the Operational Profiles (Section 7)
5. Creation of the Fault-Free Execution Traces (Section 8)
6. Fault List Construction (Section 9)
7. Analysis of the Fault List Using Fault Equivalence (Section 10)
8. Fault Injection Experiment Execution (Section 11)

Each of these steps will be described in further detail in separate documents within this series.

13. References

[1] Amendola, A.M., L. Impagliazzo, P. Marmo, and F. Poli, "Experimental Evaluation of Computer-Based Railway Control Systems", Proceedings of the 27th International Symposium on Fault-Tolerant Computing, 1997, pp. 380-384.

[2] Arlat, J., Y. Crouzet, and J.-C. Laprie, "Fault Injection for Dependability Validation of Fault-Tolerant Computing Systems", Proceedings of the 19th International Symposium on Fault-Tolerant Computing, 1989, pp. 348-355.

[3] Arlat, J., M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell, "Fault Injection for Dependability Validation: A Methodology and Some Applications", *IEEE Transactions on Software Engineering*, Vol. 16, No. 2, February 1990, pp. 166-182.

[4] Arlat, J., Alain Costes, Yves Crouzet, J.-C. Laprie, and D. Powell, "Fault Injection and Dependability Evaluation of Fault-Tolerant Systems", *IEEE Transactions on Computers*, Vol. 42, No. 8, August 1993, pp. 913-923.

[5] Avresky, D.R., J. Arlat, J.-C. Laprie, and Y. Crouzet, "Fault Injection for the Formal Testing of Fault Tolerance", Proceedings of the 22nd International Symposium on Fault-Tolerant Computing, June 1992, pp. 345-354.

[6] Barton, James H., Edward W. Czeck, Zary Z. Segall, and Daniel P. Siewiorek, "Fault Injection Experiments Using FIAT", *IEEE Transactions on Computers*, Vol. 39, No. 4, April 1990, pp. 575-582.

[7] Benso, A., M. Rebaudengo, L. Impagliazzo, and P. Marmo, "Fault-List Collapsing for Fault Injection Experiments", *Proceedings Annual Reliability and Maintainability Symposium*, 19-22 January, 1998, pp. 383-388.

[8] Carreira, Joao Viegas, Diamantino Costa, and Joao Gabriel Silva, "Fault Injection Spot-Checks Computer System Dependability", *IEEE Spectrum*, August 1999, pp. 50-55.

[9] Cassandras, C.G., *Discrete Event Systems: Modeling and Performance Analysis*, Aksen Associates Inc., and Richard D. Irwin Inc., Boston, Massachusetts, 1993.

[10] Chillarege, Ram and Nicholas S. Bowen, "Understanding Large System Failure - A Fault Injection Experiment", Proceedings of the 19th International Symposium on Fault-Tolerant Computing, 1989, pp. 356-363.

[11] Choi, Charles Y., Barry W. Johnson, and Joseph A. Profeta III, "Safety Issues in the Comparative Analysis of Dependable Architectures," *IEEE Transactions on Reliability*, Vol. 46, No. 3, September 1997, pp 316-322.

[12] Choi, G., R. Iyer, and V. Carreno, "FOCUS: An Experimental Environment for Validation of Fault-Tolerant Systems Case Study of a Jet Engine Controller", Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors, 1989, pp. 561-564.

[13] Choi, Gwan S., and Ravishankar K. Iyer, "FOCUS: An Experimental Environment for Fault Sensitivity Analysis", *IEEE Transactions on Computers*, Vol. 41, No. 12, December 1992, pp. 1515-1526.

[14] Choi, H., W. Wang, and K.S. Trivedi, "Conditional MTTF and its Computation in Markov Reliability Models," *Proc. 1993 Annual Reliability and Maintainability Symposium*, January 25-28, 1993, pp. 55-63.

[15] Choi, Jong Gyun and Poong Hyun Seong, "Dependability Estimation of Digital System By Operational Profile-Based Fault Injection", Proceedings of the Probabilistic Safety Assessment, Washington D.C., August 22-26, 1999, pp. 499-506.

[16] Clark, Jeffrey A., and Dhiraj K. Pradhan, "Fault Injection - A Method for Validating Computer-System Dependability", *IEEE Computer*, June 1995, pp. 47-56.

[17] Corno, F., M. Damiani, L. Impagliazzo, P. Prinetto, M. Rebaudengo, G. Sartore, and M. Sonza Reorda, "On-Line Testing of an Off-the-shelf Microprocessor Board for Safety-Critical Applications", Proceedings of the 2nd European Dependable Computing Conference - EDDC-2, October 2-4, 1996, Taormina, Italy, pp. 190-201.

[18] Echtle, K., and T. Chen, "Evaluation of Deterministic Fault Injection for Fault-Tolerant Protocol Testing", Proceedings of the 21st International Symposium on Fault-Tolerant Computing, June 1991, pp. 418-425.

[19] Folkesson, P., S. Svensson, and J. Karlsson, "A Comparison of Simulation-Based and Scan Chain Implemented Fault Injection", *Proceedings of FTCS-28*, IEEE Computer Society Press, Munich, June 1998, pp. 284-293.

[20] Goble, W.M., *Control Systems Safety Evaluation and Reliability*, Instrument Society of America, 1998.

[21] Goswami, K.K., and R.K. Iyer, "A Simulation-Based Study of a Triple Modular Redundant System using DEPEND", Proceedings of the 5th International Conference on Fault-Tolerant Computing Systems - GI/ITG/GMA, 1991, pp. 300-311.

[22] Gunneflo, U., J. Karlsson, and J. Torin, "Evaluation of Error Detection Schemes Using Fault Injection by Heavy-ion Radiation", Proceedings of the 19th International Symposium on Fault-Tolerant Computing, 1989, pp. 340-347.

[23] Guthoff, J., and V. Sieh, "Combining Software-Implemented and Simulation-Based Fault Injection into a Single Fault Injection Method", Proceedings of the 25th International Symposium on Fault-Tolerant Computing, June 1995, pp. 196-206.

[24] Han, S., K.G. Shin, and H.A. Rosenberg, "DOCTOR: An Integrated Software Fault Injection Environment for Distributed, Real-Time Systems", Proceedings of the International Computer Performance and Dependability Symposium, April 1995, pp. 204-213.

[25] *IEEE Std 1076-1993, IEEE Standard VHDL Language Reference Manual*, IEEE, New York, NY, 1994.

[26] Jenn, E., J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, "Fault Injection into VHDL Models: The MEFISTO Tool", Digest of Papers of the 24th International Symposium on Fault-Tolerant Computing, June 1994, pp. 66-75.

[27] Johnson, Barry W., *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Wesley Publishing Company, 1989.

[28] Kanawati, G.A., N.A. Kanawati, and J.A. Abraham, "FERRARI: A Tool for the Validation of System Dependability Properties", Proceedings of the 22nd International Symposium on Fault-Tolerant Computing, July 1992, pp. 336-344.

[29] Kanawati, Ghani A., Nasser A. Kanawati, and Jacob A. Abraham, "FERRARI: A Flexible Software-Based Fault and Error Injection System", *IEEE Transactions on Computers*, Vol. 44, No. 2, February 1995, pp. 248-260.

[30] Kao, Wei-lun and Ravishankar K. Iyer, "DEFINE: A Distributed Fault Injection and Monitoring Environment", *Fault-Tolerant Parallel and Distributed Systems*, 1995, pp. 252-259.

[31] Karlin, S., and H.M. Taylor, *A First Course in Stochastic Processes*, Second Edition, Academic Press, New York, 1975.

[32] Laprie, J.-C., "Dependable Computing and Fault Tolerance: Concepts and Terminology", *Proceedings of the 15th International Symposium on Fault-Tolerant Computing*, June 1985, pp. 2-11.

[33] Lovric, Tomislav, "Processor Fault Simulation with PROFI", Proceedings of the European Simulation Symposium ESS 95, October 1995, pp. 353-357.

[34] Madeira, H., F. Moreira, M. Rela, and J.G. Silva, "RIFLE: A General-Purpose Pin-Level Fault Injector", European Dependable Computing Conference EDCC-1, 4-6 Oct., 1994, pp. 199-216.

[35] Miremadi, G., J. Karlsson, U. Gunneflo, and J. Torin, "Two Software Techniques for On-line Error Detection", Proceedings of the 22nd International Symposium on Fault-Tolerant Computing, 1992, pp. 328-335.

[36] Modarres, M., What Every Engineer Should Know about Reliability and Risk Analysis, Marcel Dekker, New York, 1993.

[37] Murata, T., "Petri Nets: Properties, Analysis and Applications," Proceedings of the IEEE, Vol. 77, pp. 541-580, 1989.

[38] Peterson, J.L., *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[39] Smith, D. Todd, Barry W. Johnson, and Joseph A. Profeta III, "System Dependability Evaluation via a Fault List Generation Algorithm", *IEEE Transactions on Computers*, Vol. 45, No. 8, August 1996, pp. 974-979.

[40] Smith, D.T., B. W. Johnson, N. Andrianos, and J. Profeta, "A Variance-Reduction Technique via Fault-Expansion for Fault-Coverage Estimation", *IEEE Transactions on Reliability,* Sept. 1997, Vol. 46, No. 3, pp. 366-374.

[41] Smith, D. Todd, Allan White, Todd A. DeLong, Barry W. Johnson, and Ted C. Giras, "A Tutorial on Architectural Analysis Using Reliability and Safety", Technical Report No. 990609, Center for Safety Critical Systems, University of Virginia, 1999.

[42] Thatte, Satish M., and Jacob A. Abraham, "Test Generation for Microprocessors", *IEEE Transactions on Computers*, Vol. C-29, No. 6, June 1980, pp. 429-441.

[43] Tsai, T. K., Mei-Chen Hsueh, Hong Zhao, Zbigniew Kalbarczyk, and R.K. Iyer, "Stress-Based and Path-Based Fault Injection", *IEEE Transactions on Computers*, Vol. 48, No. 11, November 1999, pp. 1183-1201.

[44] Young, L.T., C. Alonso, R.K. Iyer, and K.K. Goswami, "A Hybrid Monitor Assisted Fault Injection Environment", Proceedings of the 3rd IFIP International Working Conference on Dependable Computing for Critical Applications, 1993, pp. 281-302.

[45] Yount, Charles R., and Daniel P. Siewiorek, "A Methodology for the Rapid Injection of Transient Hardware Errors", *IEEE Transactions on Computers*, Vol. 45, No. 8, August 1996, pp. 881-891.

Analytical Safety Model

Technical Report UVA-CSCS-NSE-002
Revision 01
August 1, 2003

Author(s)	Contact Info
Eric Cutright	edc2u@virginia.edu
Todd DeLong	tad2x@virginia.edu
Barry Johnson	bwj@virginia.edu

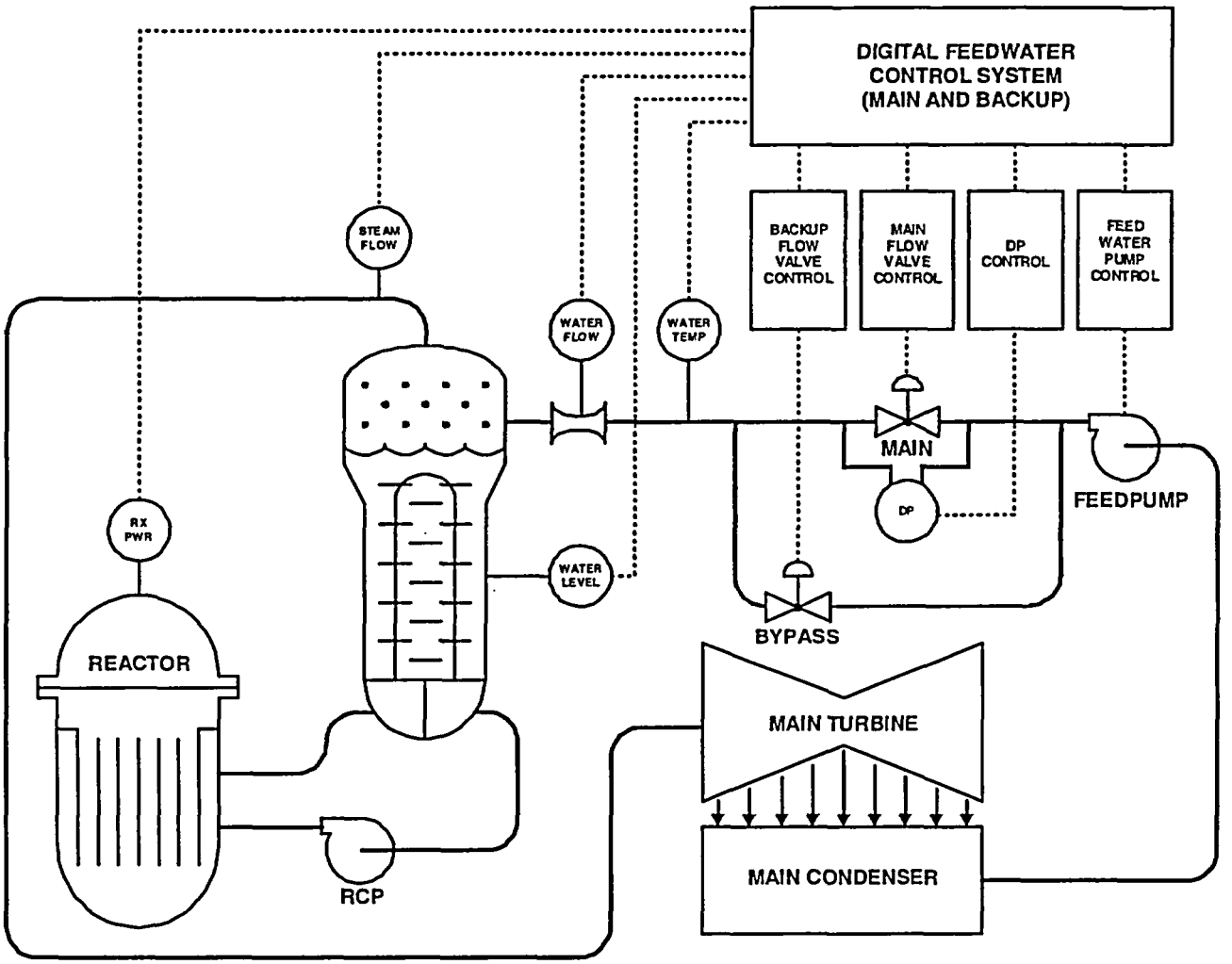
University of Virginia Center for Safety-Critical Systems
Department of Electrical and Computer Engineering
Thornton Hall
351 McCormick Road
Charlottesville, VA 22904-4743

Prepared for:
Project Manager: John Calvert
Division of Engineering Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555

Foreword

This generic document presents Analytical Safety Models for a variety of system architectures commonly used in safety-critical applications. For the Calvert Cliffs Digital Feedwater Control System (DFWCS) safety quantification effort under the Nuclear Regulatory Commission project, the Reconfigurable Duplication with Comparison (RDWC) model discussed in Section 5.3 of the document will be used to represent the DFWCS system, as described below.

An overview of the DFWCS is illustrated in the figure below. The function of the DFWCS is to control the water level in its associated steam generator from approximately 1% reactor output power up to 100% reactor output power.



Overview of Calvert Cliffs Digital Feedwater Control System (DFWCS)

Each DFWCS contains a Main and Backup Controller which read various inputs and provide control outputs. The monitored inputs include reactor power level, steam and water flow, water temperature, and water level. The controlled outputs set the operating positions of the Main Flow Valve (MFV), Bypass

Flow Valve (BFV), and the Feedwater Pump (FWP) as shown in the overview figure. The Main and Backup Controllers operate as a redundant pair, each reading inputs and delivering outputs. In the event that the Main Controller fails, the Backup Controller can take over the operation of the system.

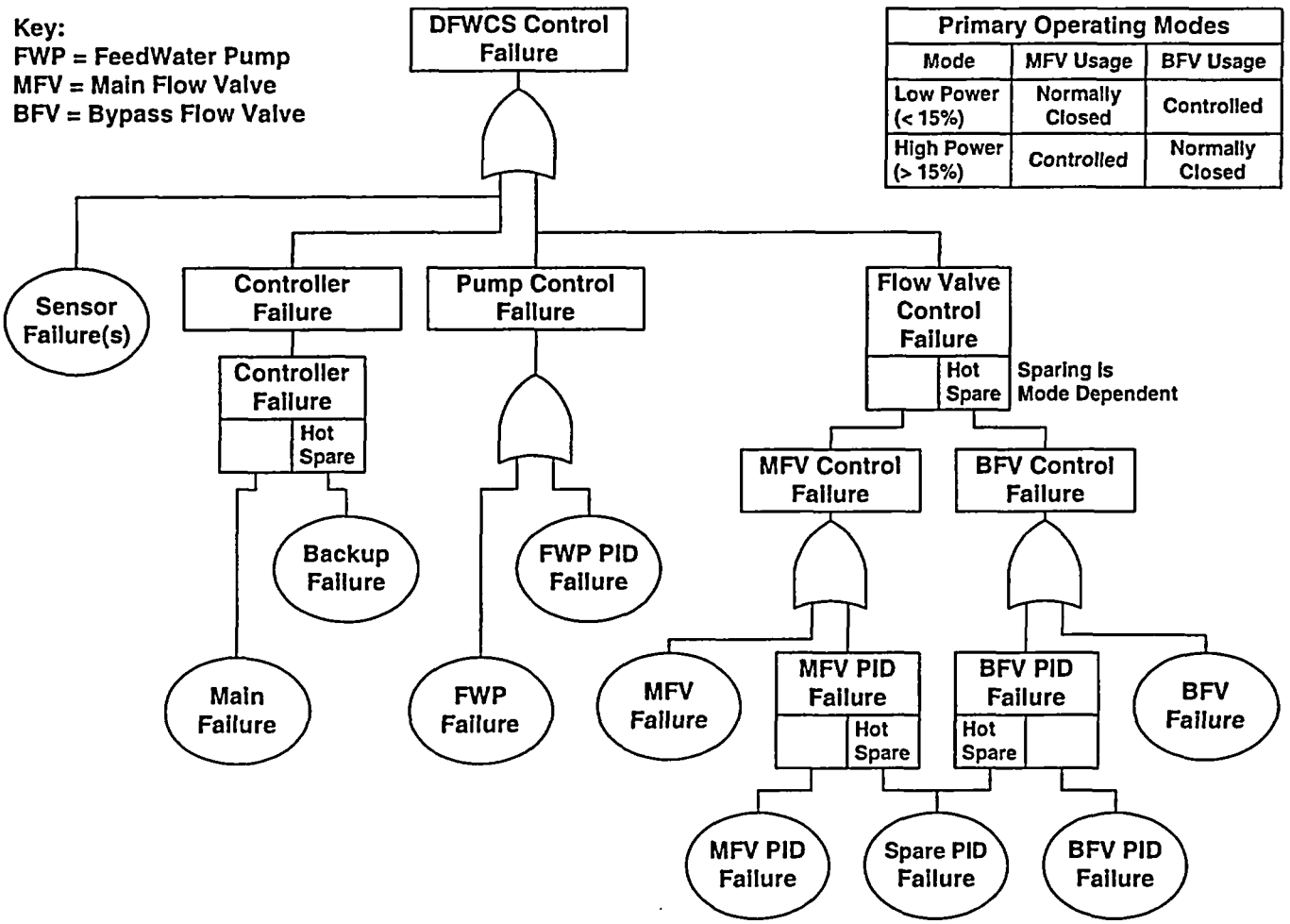
The outputs of both the Main and Backup Controllers are passed to a set of four Proportion, Integral, and Derivative (PID) Controllers. Three of these controllers are dedicated to the MFV, BFV, and FWP respectively, with the fourth controller acting as a spare for either the MFV or BFV PID controllers. The PID associated with each of the controlled devices performs a comparison of the desired setting from the Main and Backup Controller and deliver the final output to the controlled device.

The DFWCS operates in one of two primary modes: (1) Low Power Control Mode, or (2) High Power Control Mode. The first mode is used when the reactor power is less than approximately 15% and in this mode the MFV is normally closed, and the BFV is manipulated to control the water level in the associated steam generator. The second mode is used when the reactor power is between approximately 15% to 100% and in this mode the BFV is normally closed, and the MFV is manipulated to control the water level in the associated steam generator. In order to increase the fault tolerance of the system, several Override Modes exist which under certain conditions allow the MFV or BFV to be used if the normally controlled valve has failed. For example, in Low Power Control Override Mode, the BFV can be bypassed and the MFV can be manipulated to control the water level.

The Main and Backup Controllers are implemented with Azonix uMAC7000 Industrial PCs with 486DX4 processors running at 33 MHz. The PID Controllers are implemented with Bailey Fischer and Porter 53MC5000 Series Process Control Stations.

From a reliability perspective, the above detailed operation can be represented with the Dynamic Fault Tree shown on the next page. This dynamic fault tree can be converted to a Markov chain and solved using standard numerical techniques to serve as the Analytical Safety Model for the DFWCS.

However, from a system safety perspective, the safety-related operation of the system can be viewed simply as two controllers operating in a redundant fashion, with their outputs being voted upon and delivered by a Comparator/Switch function. This model is the classic Reconfigurable Duplication with Comparison (RDWC) model discussed in Section 5.3 of this document and will be used to represent the safety-related operation of the DFWCS system.



Detailed Dynamic Fault Tree Model of the DFWCS Operation (Reliability View)

Table of Contents

List of Figures

List of Tables

Glossary of Acronyms and Abbreviations

Acronyms and Abbreviations

C	Fault coverage
CSCS	Center for Safety-Critical Systems
E	Event Space
F	Fault Space
FS	Failed-Safe state
FTA	Fault Tree Analysis
FU	Failed-Unsafe state
IC	Integrated Circuit
I/O	Input/Output
LRU	Line Replaceable Unit
MTBHE	Mean Time Between Hazardous Event
MTTF	Mean Time To Failure
MTTFD	Mean Time To Fail Dangerous
MTTHE	Mean Time To Hazardous Event
MTTR	Mean Time To Repair
MTTUF	Mean Time To Unsafe Failure
O	Operational state
PFD(t)	Probability of Failure on Demand
PFD_{ss}	Steady-State Probability of Failure on Demand
Q(t)	Unreliability
R(t)	Reliability
S_{ss}	Steady-State Safety
S(t)	Safety
UVA	University of Virginia

1. Introduction

The University of Virginia (UVA) Center for Safety-Critical Systems (CSCS) has developed a methodology to quantify system safety for complex processor-based safety-critical systems [1]. The approach is built upon a campaign of extensive fault injection which is used to derive a numerical expression of system safety known as the Mean Time To Hazardous Event (MTTHE) [2] (also known alternatively as the Mean Time To Unsafe Failure (MTTUF) [2] or Mean Time Between Hazardous Events (MTBHE) [3][4]). A separate risk assessment methodology is employed at the overall system level to derive MTTHE allocations for individual processor-based systems. The UVA safety quantification methodology is then applied to the individual processor-based systems in order to demonstrate compliance with their MTTHE allocations.

This document describes the first step in the MTTHE compliance process - the development of an analytical safety model. The analytical safety model results in a mathematical expression for the MTTHE based on the critical safety-related parameters in the system, which may include failure rates, fault detection latencies, repair rates, and fault coverages. The estimation of these critical parameters is the focus of the subsequent steps in the safety quantification methodology. Once parameter estimates are obtained, the values are fed back into the analytical safety model in order to derive a numerical estimate for the system MTTHE at the desired confidence level.

This document consists of seven sections. Following this introduction, Section 2 presents important background and terminology information, including the development of the MTTHE and fault coverage concepts. Section 3 then provides an overview of the UVA numerical safety evaluation process to demonstrate where the analytical safety modeling fits into the overall methodology. Next Section 4 describes the analytical safety modeling process in detail, including the mathematical derivation of the MTTHE expression from a Markov model as an illustrative example. Section 5 then presents the MTTHE expressions for a variety of common safety-critical architectures. Next Section 6 provides a short summary of the document. Finally, Section 7 presents a list of important references.

2. Background and Terminology

This section provides background and terminology information which is useful to the understanding of the numerical safety evaluation process. The concept of Mean Time To Hazardous Event (MTTHE) is developed along with several related safety and reliability metrics and their dependent factors.

2.1. Modeling of System Safety

A discussion on the modeling of system safety naturally begins with the definition of safety:

> **Definition 1: Safety,** $S(t)$, is the probability that a system will either perform its functions correctly or will discontinue its functions in a manner that does not disrupt the operation of other systems or compromise the safety of any people associated with the system [5].

There are two important things to note about this definition. First, we define safety as a probability, which implies that safety is a quantitative measure that can take on values between zero and one, inclusive.

$$0.0 \leq S(t) \leq 1.0 \tag{2.1}$$

Second, there is an inherent notion of system states corresponding to whether the system is safe or unsafe. The first state is associated with the system functioning correctly, and we will refer to this state as the Operational State. The second state is associated with the system discontinuing "... its functions in a manner that does not disrupt the operation of other systems or compromise the safety of any people associated with the system." In other words, the system ceases to perform its function, but it does so in a safe manner. We will refer to this state as the Failed-Safe State because the system has failed to perform its function but in a safe manner. The third and final state is essentially all the scenarios that are not included in the first two states. That is, the third state is associated with the system not performing its function correctly, and in addition possibly jeopardizing the safety of the system. We will refer to this third state as the Failed-Unsafe State because like the Failed-Safe State the system has failed to perform its function but has done so in an unsafe manner. The relationship between these three states is shown in **Figure 2.1.**

Referring to **Figure 2.1**, we will assume that the system begins in the Operational state, where it is performing its intended function correctly. According to **Figure 2.1**, the system can transition to either the Failed-Safe state or the Failed-Unsafe state (indicated by the arrows in **Figure 2.1**). Associated with each state transition is a parameter that indicates the rate at which the transition occurs, and as such is referred to as the transition rate. The state transition rate depends on two parameters: (1) the failure rate function, $\lambda(t)$, and (2) the fault coverage, $C(t)$, defined below.

> **Definition 2: Failure rate function,** $\lambda(t)$, represents the rate at which the system fails. For safety studies, this rate considers only those failures that can potentially result in an unsafe system state. The failure rate function may be time-varying or may be a constant. The typical approach is to assume that the failure rate function follows an exponential distribution, resulting in a constant failure rate, λ. One assumption that we will make regarding the failure rate function is that the value of the failure rate function is always

greater than zero. As such, the system will eventually transition from the Operational state to either the Failed-Safe state or the Failed-Unsafe state (that is, the system will eventually fail as $t \rightarrow \infty$).

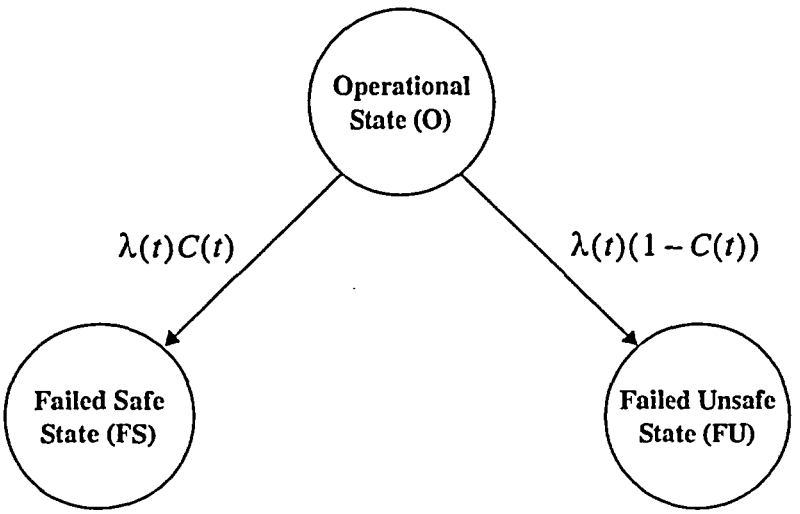


Figure 2.1. Three State System Model used to Calculate Safety

Definition 3: Fault coverage, $C(t)$, is the conditional probability that a system correctly handles a fault, given that a fault has occurred. The fault coverage function may be time-varying or may be a constant. The typical approach is to assume that the fault coverage is a constant, C. And, as with the safety, because fault coverage is a probability, it will take on values between zero and one, inclusive.

$$0.0 \leq C(t) \leq 1.0 \tag{2.2}$$

Referring to **Figure 2.1**, the transition rate from the Operational state to the Failed-Safe state is $\lambda(t)C(t)$, representing the case where a potentially unsafe fault has occurred and the system has correctly handled the fault and transitioned to a Failed-Safe state. The transition rate from the Operational state to the Failed-Unsafe state is $\lambda(t)(1-C(t))$, representing the case where a potentially unsafe fault has occurred and the system has NOT correctly handled the fault and has transitioned to a Failed-Unsafe state. For safety studies, it is assumed that a potentially unsafe fault which remains undetected by the system will place the system into a Failed-Unsafe state.

From the model of **Figure 2.1**, there are two general classes of safety and reliability metrics which can be computed:

1. The probability that a system is operating in a given state at time t, and
2. The expected time to some event of interest

Common "state probability"-based metrics from the first class include:

- Safety, $S(t)$
- Steady-State Safety, S_{ss}
- Probability of Failure on Demand, $PFD(t)$
- Steady-State Probability of Failure on Demand, PFD_{ss}

Common "expected time to event"-based metrics from the second class include:

- Mean Time To Hazardous Event, *MTTHE*
- Mean Time to Unsafe Failure, *MTTUF*
- Mean Time To Fail Dangerously, *MTTFD*
- Mean Time To Failure, *MTTF*
- Mean Time To Repair, *MTTR*

Each of these metrics will be discussed in the following sections.

2.2. Derivation of Safety (S(t)) Metric

To derive an expression for safety, *S(t)*, refer back to Definition 1. From this definition, we can calculate an expression for safety by determining the probability of being in either the Operational or Failed-Safe states for the simple three state model of **Figure 2.1**. Assuming a constant fault coverage C and constant failure rate λ, the safety expression for this model is given by:

$$S(t) = p_{OP}(t) + p_{FS}(t) = e^{-\lambda t} + (C - C \cdot e^{-\lambda t}) = C + (1 - C) \cdot e^{-\lambda t} \qquad (2.3)$$

where $p_{OP}(t)$ and $p_{FS}(t)$ indicate the probability of being in the Operational and Failed-Safe states, respectively, at time t.

2.3. Derivation of Steady-State Safety (S_{SS}) Metric

Another useful and common metric for quantifying system safety is the steady-state safety, S_{ss}. The steady-state safety may be calculated from the expression for system safety, $S(t)$. Again, from Definition 1, we conclude that the safety for a system which can be represented by **Figure 2.1** is the probability of either residing in the Operational State or the Failed-Safe State.

> **Definition 4:** The **Steady-State Safety, S_{SS},** is a probability representing the evaluation of safety as a function of time, in the limiting case as time approaches infinity:
>
> $$S_{ss} = \lim_{t \to \infty} S(t) \tag{2.4}$$

Note that S_{ss} represents the probability that the system will operate safely after it is placed into service. S_{ss} represents the minimum safety for a safety-critical system and as such provides a lower bound for the system safety under most conditions. It is also a useful metric in that the fault coverage represents a conservative lower bound on steady-state safety, such that this bound can be expressed independently of the system failure rate (which may be difficult to estimate, especially for design faults), in contrast to MTTHE which requires estimation of the system failure rate.

As an S_{ss} example, consider the simple model of **Figure 2.1**, with a constant fault coverage C and constant failure rate λ. In this case, the S_{ss} expression for the system can be derived from the safety expression of **Equation (2.3)** as:

$$S_{ss} = \lim_{t \to \infty} S(t) = \lim_{t \to \infty} (C + (1 - C) \cdot e^{-\lambda t}) = C \tag{2.5}$$

2.4. Derivation of Probability of Failure On-Demand (PFD(t)) Metric

A useful metric for quantifying system safety is the Probability of Failure on Demand, $PFD(t)$. This metric represents the probability that the system has failed in an dangerous manner at a given instant of time. In other words, this metric represents the probability that the system has failed dangerously at the time it is needed to respond to a demand [6]. This metric is particularly useful for quantifying the safety of protection systems (e.g. nuclear reactor protection systems), which perform safety-critical interventions in the event of a system failure to bring the system to a safe state.

The formulation of this metric requires a slight modification to the three-state safety model as illustrated in **Figure 2.2**. Comparing this to **Figure 2.1**, the three states are Operational, Failed-Safe and Failed-Dangerous instead of Operational, Failed-Safe, and Failed-Unsafe. The definition for the Operational state is the same as stated in **Section 2.1**. In this case, the Failed-Safe state refers to the situation where the system shuts down even when there is no potentially dangerous situation. A failure of this type would be considered a safe failure, a false alarm or a false trip. The Failed-Dangerous state covers the case when the system would be unable to respond to a demand, a potentially dangerous condition [6]. Referring to **Figure 2.1**, the failed-unsafe state of this model is replaced with the failed-dangerous state. The remaining states remain unchanged.

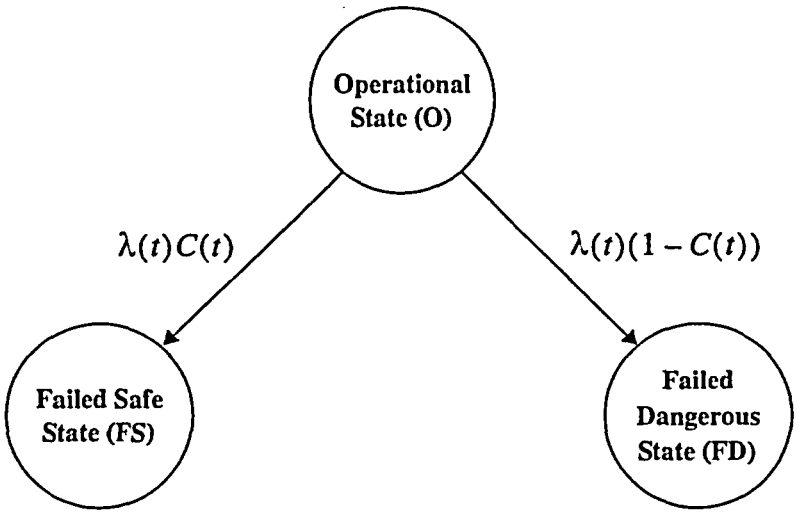


Figure 2.2. Modified Three State System Model used to Calculate MTTFD and PFD

Referring to **Figure 2.2**, we can calculate $PFD(t)$ as the following:

$$PFD(t) = p_{FD}(t) = 1 - p_O(t) - p_{FS}(t) \tag{2.6}$$

As an example of the $PFD(t)$ metric, consider the simple model of **Figure 2.2**, with a constant fault coverage C and constant failure rate λ. In this case, the $PFD(t)$ expression for the system is:

$$PFD(t) = p_{FD}(t) = 1 - p_O(t) - p_{FS}(t) = 1 - e^{-\lambda t} - (C - C \cdot e^{-\lambda t}) = (1 - C) + (1 - C) \cdot e^{-\lambda t} \tag{2.7}$$

2.5. Derivation of Steady-State Probability of Failure On-Demand (PFD_{SS}) Metric

Another useful metric for quantifying system safety is the steady-state Probability of Failure on Demand, PFD_{ss}, which can be calculated from the expression for $PFD(t)$. From **Figure 2.2** we conclude that PFD(t) is the likelihood of residing in the failed-dangerous state.

> **Definition 5:** : The **Steady-State Probability of Failure On-Demand, PFD_{SS},** is a probability representing the evaluation of PFD as a function of time, in the limiting case as time approaches infinity:

$$PFD_{ss} = \lim_{t \to \infty} PFD(t) \tag{2.8}$$

As an PFD_{ss} example, consider the simple model of **Figure 2.2**, with a constant fault coverage C and constant failure rate λ. In this case, the PFD_{ss} expression for the system is:

$$PFD_{ss} = \lim_{t \to \infty} PFD(t) = \lim_{t \to \infty} [(1 - C) + (1 - C) \cdot e^{-\lambda t}] = (1 - C) \tag{2.9}$$

Note that $PFD(t)$ is related to $S(t)$ (sometimes referred to as Safety Availability, *SA(t)* [6]) by the following relationship:

$$PFD(t) = 1 - S(t) = 1 - SA(t) \tag{2.10}$$

In addition, PFD_{ss} is related to S_{ss} by:

$$PFD_{ss} = 1 - S_{ss} \tag{2.11}$$

2.6. Derivation of Mean Time To Hazardous Event (MTTHE) Metric

For safety studies, a useful mean time metric is the Mean Time to Hazardous Event (MTTHE), also known as the Mean Time To Unsafe Failure (MTTUF) [2]. Here the "event" is an unsafe system failure (as opposed to a system failure for the MTTF). The MTTHE is the primary safety metric that will be used in the UVA numerical safety evaluation process and is defined below:

> **Definition 6:** The **Mean Time To Hazardous Event (MTTHE)** is the expected time that a system will operate before the first *unsafe* failure occurs. Mathematically, the MTTHE is defined as the expected value of a random variable, X, which represents the time to the first unsafe failure. It can be defined as a continuous-time or discrete-time function, but will be defined as a discrete-time function here since that is how it is used later in this document. Given this, then
>
> $$E[X] = MTTHE = \sum X_i \times P_i \tag{2.12}$$
>
> where E is the statistical expectation operator for a probability mass function, and P_i is the likelihood of occurrence of X_i [2].

As an MTTHE example, consider the simple model of **Figure 2.1**, with a constant fault coverage C and constant failure rate λ. In this case, the MTTHE expression for the system is:

$$MTTHE = \frac{1}{\lambda(1-C)} \tag{2.13}$$

Note that MTTHE is equivalent to the metrics Mean Time Between Hazardous Events (MTBHE) and Mean Time To Unsafe Failure (MTTUF), which have been documented in literature [2][4]. As such, MTTHE will be used throughout the remainder of this report, knowing that the results apply equally to MTBHE or MTTUF.

2.7. Derivation of Mean Time to Fail Dangerous (MTTFD) Metric

As with the Probability of Failure on Demand metric, the Mean Time To Fail Dangerous (MTTFD) metric is particularly useful for quantifying the safety of protection systems (e.g. nuclear reactor protection systems), which perform safety-critical interventions in the event of a system failure to bring the system to a safe state. From the model of **Figure 2.2**, the MTTFD metric is defined as:

> **Definition 7:** : The **Mean Time To Fail Dangerous (MTTFD)** is the expected time that a system will operate before the first dangerous failure occurs [6].

Mathematically, the MTTFD can be defined the same manner as MTTHE in **Equation (2.12)**.

As an MTTFD example, consider the simple model of **Figure 2.2**, with a constant fault coverage C and constant failure rate λ. In this case, the MTTFD expression for the system is:

$$MTTFD = \frac{1}{\lambda(1-C)} \tag{2.14}$$

Note that the MTTFD is equivalent to the MTTHE metric presented earlier. This metric is also equivalent to the MTBHE and MTTUF. As such, MTTHE will be used throughout the remainder of this report.

2.8. Other Important Metrics

Other important reliability metrics commonly used in safety-critical applications are shown below. In contrast to safety studies, reliability studies are typically concerned with all types of failures, not just those that can affect the safety of the system. Thus, whereas the failure rate function for safety studies is limited to consideration of only those failures that can potentially result in an unsafe system state, the failure rate function for reliability studies will also reflect other types of failures in the system, especially those that interrupt system service.

> **Definition 8:** The **reliability**, $R(t)$, of a system is a function of time, defined as the conditional probability that the system will perform correctly throughout the interval $[t_0,t]$, given that the system was performing correctly at time t_0 [5].
>
> **Definition 9:** The **unreliability**, $Q(t)$, of a system is a function of time, defined as the conditional probability that the system will perform *incorrectly* during the interval $[t_0,t]$, given that the system was performing *correctly* at time t_0 [5].

Note that the unreliability and reliability of a system are directly related. Specifically, the probability that the system is failed or operational must always equal 1.0. This relationship is expressed as:

$$R(t) + Q(t) = 1.0 \tag{2.15}$$

> **Definition 10:** The **Mean Time To Failure (MTTF)** [5] is the expected time that a system will operate before the *first* failure occurs.

Note that based on this definition, it can be shown that the MTTF for a system can be computed if the reliability of the system, $R(t)$, is known, and $R(t)$ approaches zero as $t \rightarrow \infty$ (from [5]):

$$MTTF = \int_0^\infty R(t)dt \tag{2.16}$$

3. Overview of Numerical Safety Evaluation Process

One quantitative method to demonstrate the MTTHE compliance of a system is the evaluation of the system fault processing capabilities using fault injection. The general approach is to inject randomly selected faults into the system (on a working prototype, for example) to determine if the fault processing capabilities mitigate the faults. This safety quantification approach builds upon the University of Virginia's extensive research experience in this area. The methodology is outlined in **Figure 3.1** and consists of the following steps:

1. An analytical safety model is developed from the system architecture and inter-component dependencies to derive an expression for the MTTHE. The MTTHE expression is a function of *Critical Model Parameters* such as the fault coverage values and failure rates for the various components of the system. The development of the analytical safety model is the subject of this document.
2. A statistical model is then developed based on those found in published literature and is used to estimate the *Critical Model Parameters* that are required by the analytical safety model. This statistical model is also used to calculate the number of fault injection experiments required to meet the numerical safety target for a given confidence level.
3. A high-level generic processor fault model is defined to specify the types of faults (and their associated probabilities) that will be injected into the system under analysis. This process builds upon recent research at UVA in generic processor fault modeling, undertaken with the goal of characterizing low-level internal processor faults at the higher register-transfer level, in order to demonstrate that faults injected on the actual hardware are representative of the low-level processor faults of concern.
4. One or more operational profiles are then defined which will be used to drive the inputs to the system under analysis during the fault injection process. The operational profiles must be representative of the different system configurations and workloads that would be experienced in actual field operation.
5. A fault-free execution trace is created for each selected operational profile that will be used to generate the list of faults to inject into the system under analysis. This trace will also be used during the analysis of the fault injection experimental results.
6. Using the fault-free execution trace and the fault categories from the generic processor fault model, a fault list construction algorithm is applied to generate a list of possible faults that can be injected into the system under analysis, and which are likely to have an effect on the system. From this complete set of responsive faults, a fault list selection algorithm is then applied to randomly select a list of faults to be injected into the system, using the fault categories and associated occurrence probabilities from the generic processor fault model.
7. A fault equivalence algorithm is applied to each fault list to identify those sub-sets of faults which will have the same effect on the system, known as fault equivalence classes. The set of faults to be injected into the system is then reduced using the fault equivalence information, since only one fault from each class needs to be injected. This reduces the amount of time required to perform the evaluation of the system under analysis by reducing the total number of fault injection experiments which must be performed.

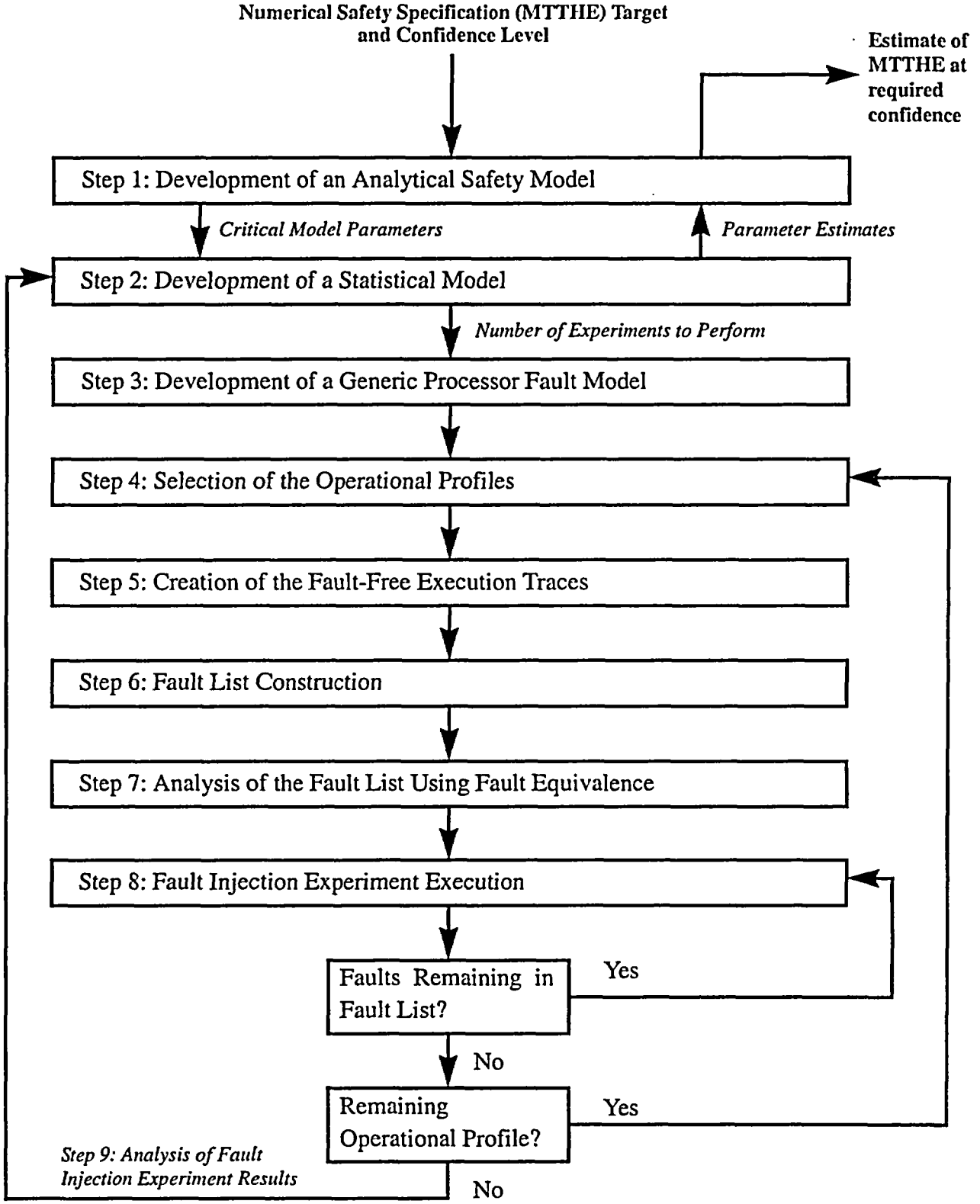


Figure 3.1. Process for quantitative safety evaluation using fault injection

8. Each fault from each reduced fault list is then injected into the system under analysis, using one of four fault injection techniques: (1) hardware-based, (2) software-based, (3) simulation-based, or (4) a hybrid approach.
9. The system is monitored during each fault injection experiment, and each fault is classified as covered, uncovered, or non-responsive by comparing the actual execution trace to the fault-free execution trace. Once all of the results of the fault injection experiments have been collected, they are fed back into the statistical model to calculate the estimates of the *Critical Model Parameters*. The *Parameter Estimates* are then in turn fed back into the analytical model to calculate the estimate of the MTTHE, as shown in **Figure 3.1.**

3.1. Development of an Analytical Safety Model

This document describes the first step of the evaluation process outlined in **Figure 3.1**, the development of a high-level analytical safety model. The purpose of this model is to provide a mathematical framework for calculating the estimate of the MTTHE. The input to the model is the MTTHE allocation along with the confidence level required in the estimate. For the purposes of a safety evaluation, the analytical safety model is used to represent, at a high level, the faulty behavior of the system under analysis. Suitable analytical safety models include Markov models [7][8], Petri nets [7][9][10], and fault trees [11]. The model uses various critical parameters such as failure rates, fault detection latencies, and fault coverages to describe the faulty behavior of the system under analysis. This is shown in **Figure 3.1** as the arrow labeled *Critical Model Parameters*, shown as an input to the box that represents the statistical model. Once the faulty behavior has been sufficiently described, an analytical safety model is converted to a corresponding mathematical representation that can be numerically solved to provide a quantitative estimate of dependability measures, such as safety. The solution of the model to generate the quantitative estimate is highly dependent upon the parameters used to describe the faulty behavior. The estimates to these parameters are shown in **Figure 3.1** as the arrow labeled *Parameter Estimates*. As such, estimating numerical values for these parameters becomes the crux of the evaluation process. Of all the parameters that typically appear in an analytical safety model used as part of a safety evaluation, the fault coverage is the most difficult to estimate.

4. Analytical Safety Modeling

4.1. Example System

To illustrate the process of building an analytical safety model, consider an example system consisting of n independent Line Replaceable Units (LRUs), each of which is capable of failing in an unsafe manner leading directly to an unsafe system failure (i.e. from a reliability block diagram perspective, the LRUs are considered to be in "series"). Each LRU can be thought of as a single physical assembly that is directly replaced during maintenance after a failure, and can represent a range of complexities from a simple relay to a complex processor-based controller assembly. Each of the LRUs in this example system has an associated failure rate λ_i and fault coverage C_i, and are assumed to fail in an independent fashion.

4.2. Development of the System Markov Model

Markov models are a useful mechanism for modeling system failure and recovery in a straightforward fashion. A Markov model consists of a set of system states with defined transitions between states. These state transition rates and therefore the holding times in each state are assumed to be exponentially distributed. As mentioned previously, other good techniques for analytical safety modeling include fault trees and Petri nets, which can in most cases be directly converted to Markov models for numerical solution.

Consider the development of a Markov model to represent the behavior of the example system described above. The first step in the development of a Markov model is the definition of system states. For this model, let us assume the system resides in one of three states:

1. O: operational, where the system is operating in a fault-free fashion,
2. FS: failed-safe, where the system has failed in a safe manner, and
3. FU: failed-unsafe, where the system has failed in an unsafe manner.

The next modeling step is the definition of transition rates between the system states. The key parameters here are the fault coverage C_i, the conditional probability of the i^{th} LRU failing safely given that a failure has occurred during a given operational cycle, and the failure rate λ_i representing the rate that failures occur for the i^{th} LRU. Since we are deriving a safety expression, the failure rate considers only those failures that can potentially result in an unsafe system state. Thus, LRUs and portions of the circuitry whose failure cannot affect system safety are not included in the model.

Note that λ_i represents the total failure rate for a given LRU, consisting of the failure rates for both transient and permanent faults. Literature has shown that transient faults contribute to 80-90% of the system failures experienced in the field [12][13][14]. However, transient faults are extremely difficult to account for in a typical field reliability calculation process based on returns of failed components to a repair facility. This is due to the fact that the effects of transient faults are limited and as such may not allow fault isolation during field maintenance and further are typically not repeatable at the repair facility (the "no problem found" syndrome).

For this effort, we assume that the failure rates derived for the LRUs (e.g. from field reliability data) are the failure rates associated with permanent faults,

$$(\lambda_i)_p = \lambda_i(1-\alpha), \tag{4.1}$$

where α is the fraction of total faults that are assumed to be transient, $0.0 \le \alpha \le 1.0$. Based on the values of $(\lambda_i)_p$ and α, a value for each λ_i will be computed, and the computed value of λ_i will be used in the model. Rearranging this equation yields:

$$\lambda_i = \frac{(\lambda_i)_p}{(1-\alpha)} \tag{4.2}$$

As an example, if a given LRU has a permanent failure rate of 10×10^{-6} failures/hour, and the ratio of transient faults is 80%, the "adjusted" failure rate used in the model would be 50×10^{-6} failures/hour.

The resulting Markov model for the example system is shown in **Figure 4.1.** An additional parameter, the system repair rate, μ, is also included. This parameter, which is the rate at which a system is repaired and returned to its operational state, is typically not included in fault trees. Generally speaking, fault trees are constructed either for non-repairable systems, or for systems with independent repair for each component (in which the availability serves as an input to the fault tree instead of a failure probability), and Markov models are generally used for repairable systems.

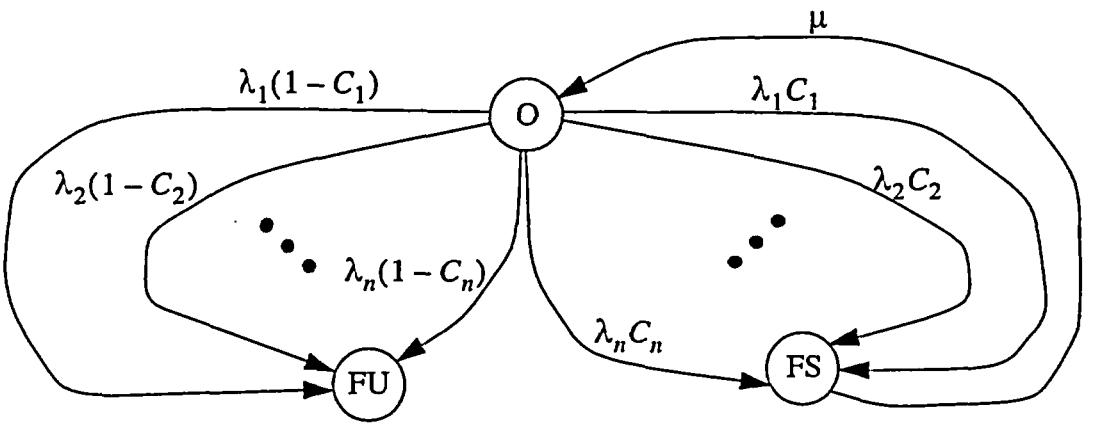


Figure 4.1. Markov model with a separate fault coverage and failure rate for each LRU

The following assumptions are made in deriving the Markov model for the system shown in **Figure 4.1:**

1. The failure rate of the replicated modules is constant,
2. The system begins operation at time $t = 0$ in a fault free mode,
3. The individual LRUs fail independently, and
4. Only one LRU will be failed (either safe or unsafe) at any given time.

4.2.1. Near Coincident Faults

Assumption four listed above, commonly referred to as the assumption of near coincident faults, deserves some further rationale to support it as an assumption. Consider a system described by the Markov model shown in **Figure 4.1**. The probability that a given LRU with a constant failure rate is operational (using assumptions one and two) is given as (from [5])

$$P(t) = e^{-\lambda_i t} \tag{4.3}$$

where λ_i is the rate that failures occur for the i^{th} LRU and t is the time of operation. A conditional probability expression determines the probability that a system is operational at time $t+\Delta$ given it was operational at time t. Using assumption three and the expression in **Equation (4.3)**, the conditional probability expression can be computed as (from [15])

$$P(\text{operation at time } (t+\Delta) | \text{operational at time } t) = \frac{P(\Delta)P(t)}{P(t)} = e^{-\lambda_i \Delta} \tag{4.4}$$

Thus, the conditional probability only depends on the time window represented by Δ and does not depend on when the time window begins (represented in **Equation (4.4)** by the operational time t). The probability of a fault event arriving sometime during the time window Δ for the i^{th} LRU is given as (from [15])

$$P_{f_1} = 1 - P(\Delta) = 1 - e^{-\lambda_i \Delta} \tag{4.5}$$

where P_{f_1} is the probability of the fault event for the i^{th} LRU. Likewise, the probability of a fault event arriving sometime during the time window Δ for the j^{th} LRU is

$$P_{f_2} = 1 - P(\Delta) = 1 - e^{-\lambda_j \Delta} \tag{4.6}$$

where P_{f_2} is the probability of the fault event. Since the fault occurrence events are independent (again using assumption three), the probability of both fault events occurring in the time window Δ is obtained by taking the product of **Equation (4.5)** and **Equation (4.6)**. If $\lambda_j = \lambda_i$, performing this operation provides

$$P_{ncf} = P_{f_1} P_{f_2} = (1 - e^{-\lambda_i \Delta})^2 \tag{4.7}$$

where P_{ncf} is the near coincident fault probability.

To illustrate the impact of two near coincident faults, **Equation (4.7)** is evaluated for several time windows, using a representative overall system failure rate of

$$\lambda_i = 50x10^{-6} \text{ failures/hour} \ . \tag{4.8}$$

The results are shown in **Table 4.1** where the first column indicates Δ and the second column denotes P_{ncf}. Typically, safety-critical systems are designed such that the fault handling mechanisms are executed within a short time period (on the order of a few seconds). Thus, the probability of near coincident faults is typically negligible for these systems, and the results from **Equation (4.7)** can be used to justify assumption four.

Time window Δ	Probability of Near Coincident Fault P_{ncf}
100 milliseconds	1.93×10^{-18}
1 sec	1.93×10^{-16}
1 min	6.94×10^{-13}
1 hour	2.50×10^{-9}
1 day	1.44×10^{-6}

Table 4.1 Probability of near coincident fault table for a given system failure rate

4.3. The Differential Equations for Markov Models

One of the attractive features of Markov models is that numerical results can be obtained without an extensive knowledge of probability theory and techniques. Markov models are characterized by the manner in which the probabilities change in time. The rates give the *flow* for changes in probabilities. The flow for the constant rate processes is smooth enough that the change in probabilities can be described by a set of first-order, linear differential equations. Once a model is constructed, solving the differential equations gives the probabilities of being in the states of the model. The solution does not require any additional knowledge of probability.

Writing the differential equations for a model is done on a per state basis. The equation for the derivative of a state depends only on the single step transitions into that state and the single step transitions out of that state. For example, consider the Markov model shown in **Figure 4.2.** There are m transitions from states $\{A_1, A_2, ..., A_m\}$ into state B with rates $\{a_1, a_2, ..., a_m\}$ respectively; these are the only transitions into state B. Also, there are n transitions from state B to states $\{C_1, C_2, ..., C_n\}$ with rates $\{c_1, c_2, ..., c_n\}$, respectively; these are the only transitions out of state B. The differential equation for state B for **Figure 4.2** is

$$\frac{d}{dt}p_B(t) = a_1 p_{A_1}(t) + a_2 p_{A_2}(t) + ... + a_m p_{A_m}(t) - (c_1 + c_2 + ... + c_n)p_B(t) \tag{4.9}$$

where $p_B(t)$ is the probability of being in state B, and $p_{A_i}(t)$ is the probability of being in state A_i. While writing the formula for state B, it is possible to ignore all the other states and transitions. The ability to ignore all other states and transitions is directly related to the memoryless property associated with Markov models. Specifically, since state B has no memory of how the system arrived at this state then the equation used to describe the state B depends only on:

1. the probability of transitioning into state B, and
2. the probability of transitioning out of state B.

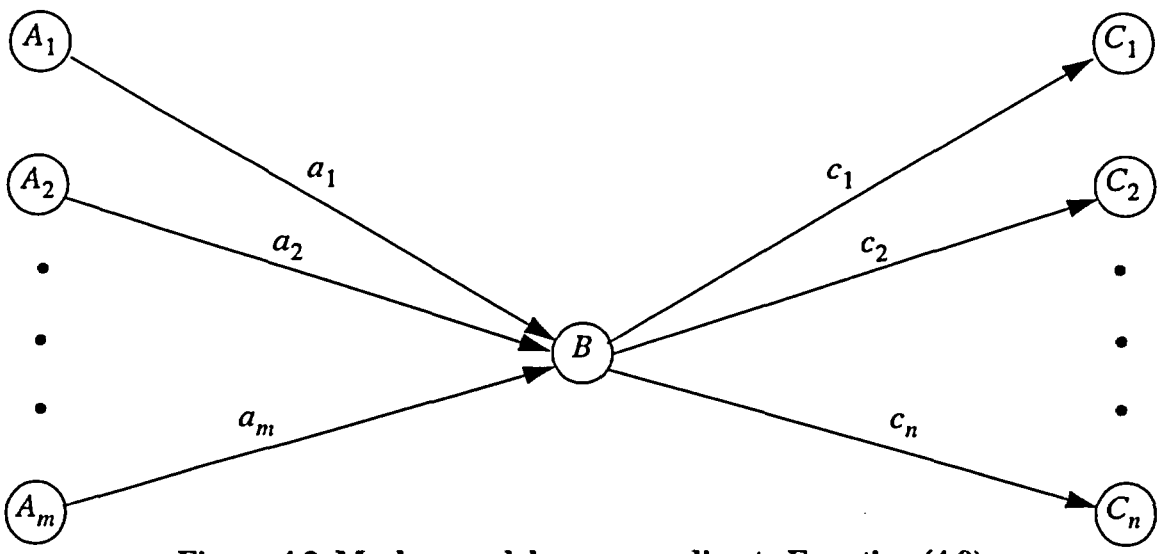


Figure 4.2. Markov model corresponding to Equation (4.9)

The positive terms on the right half of **Equation (4.9)** represent entering state B while the negative terms on the right half of **Equation (4.9)** represent the leaving of state B.

The relationship between a state in a Markov model such as B in **Figure 4.2** and the differential equation used to mathematically describe the probability of the state such as **Equation (4.9)** for the probability of state B in **Figure 4.2** can be applied to every state in the Markov model. If the Markov model contains n states then the state evaluation process used to derive a first order differential equation must be performed n times. Typically the set of n equations is written in matrix form. For instance, if the n vector equations associated with the Markov model in **Figure 4.2** are combined into a single matrix equation, then the following expression is obtained

$$\begin{bmatrix} \frac{d}{dt}p_{A_1}(t) \\ \dots \\ \frac{d}{dt}p_{A_m}(t) \\ \frac{d}{dt}p_B(t) \\ \frac{d}{dt}p_{C_1}(t) \\ \dots \\ \frac{d}{dt}p_{C_n}(t) \end{bmatrix} = \begin{bmatrix} -a_1 & 0 & \dots & 0 & 0 \dots 0 \\ 0 & \dots & 0 & 0 & 0 \dots 0 \\ 0 & \dots & -a_m & 0 & 0 \dots 0 \\ a_1 & \dots & a_m & -(c_1 + \dots + c_n) & 0 \dots 0 \\ 0 & \dots & 0 & c_1 & 0 \dots 0 \\ 0 & \dots & 0 & \dots & 0 \dots 0 \\ 0 & \dots & 0 & c_n & 0 \dots 0 \end{bmatrix} \begin{bmatrix} p_{A_1}(t) \\ \dots \\ p_{A_m}(t) \\ p_B(t) \\ p_{C_1}(t) \\ \dots \\ p_{C_n}(t) \end{bmatrix} \tag{4.10}$$

A more standard way of describing a Markov model is to number the individual states from 1 to n. The probability associated with the i^{th} state is given as $p_i(t)$. Likewise the transition rate of leaving the i^{th} state and entering the j^{th} state is given as α_{ji}. Conversely, the transition rate out of the j^{th} state is given as β_j. Assuming that a transition from any state to any other state is possible, the result presented by **Equation (4.10)** can be generalized to

$$\begin{bmatrix} \frac{d}{dt}p_1(t) \\ \frac{d}{dt}p_2(t) \\ \dots \\ \frac{d}{dt}p_n(t) \end{bmatrix} = \begin{bmatrix} -\beta_1 & \alpha_{12} & \dots & \alpha_{1n} \\ \alpha_{21} & -\beta_2 & \dots & \alpha_{2n} \\ \dots & \dots & \dots & \dots \\ \alpha_{n1} & \alpha_{n2} & \dots & -\beta_n \end{bmatrix} \begin{bmatrix} p_1(t) \\ p_2(t) \\ \dots \\ p_n(t) \end{bmatrix} \tag{4.11}$$

where $\{p_1(t), p_2(t), \dots, p_n(t)\}$ are the Markov model state probabilities. Note that for a given row, j, the rate in the diagonal, β_j, will equal the sum of the other rates in the column. Stating this concept in mathematical terms yields

$$\beta_j = \sum_{i \neq j} \alpha_{ij} \tag{4.12}$$

A more compact form of **Equation (4.11)** is

$$\frac{d}{dt}P(t) = QP(t) \tag{4.13}$$

where

$$P(t) = \left[p_1(t)\ p_2(t)\ \ldots\ p_n(t)\right]^T \tag{4.14}$$

and

$$Q = \begin{bmatrix} -\beta_1 & \alpha_{12} & \ldots & \alpha_{1n} \\ \alpha_{21} & -\beta_2 & \ldots & \alpha_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ \alpha_{n1} & \alpha_{n2} & \ldots & -\beta_n \end{bmatrix} \tag{4.15}$$

The compact set of equations given by **Equation (4.13)** is referred to as the Chapman-Kolmogorov equation [16][17][8].

Metrics such as reliability, safety, MTTF, and MTTHE associated with a given Markov model are obtained by first solving the set of n differential equations contained in **Equation (4.11)**. The probabilities of the Markov model states are then used to determine the metrics for the given system.

4.4. Calculating Reliability and Safety from a Markov Model

Now that a set of metrics of interest has been defined, and a generalized approach to solving for these metrics has been developed, the approach is applied to the Markov model in **Figure 4.1** to compute the metrics defined in Section 2.1 - Section 2.8.

Begin with the Chapman-Kolmogorov equation:

$$\frac{d}{dt}P(t) = QP(t) \tag{4.16}$$

A general solution to **Equation (4.16)** can be obtained through the use of basic linear algebra. The general solution begins by taking the Laplace transform of **Equation (4.16)** which yields

$$sP(s) - P(0) = QP(s) \tag{4.17}$$

where $P(0)$ is the initial probability of the states of the Markov model. Solving **Equation (4.17)** for $P(s)$ provides

$$P(s) = (sI - Q)^{-1}P(0) \tag{4.18}$$

Taking the inverse Laplace transform of $P(s)$ to find $P(t)$ yields

$$P(t) = e^{Qt}P(0) \tag{4.19}$$

where $e^{Qt} = L^{-1}[(sI - Q)^{-1}]$ and L^{-1} [ ] is the inverse Laplace transform operator.

Expressing the behavior of the system depicted in **Figure 4.1** in the form of **Equation (4.11)**, we have

$$\begin{bmatrix} \frac{d}{dt}p_O(t) \\ \frac{d}{dt}p_{FS}(t) \\ \frac{d}{dt}p_{FU}(t) \end{bmatrix} = \begin{bmatrix} -(\lambda_1 + \lambda_2 + \ldots + \lambda_n) & \mu & 0 \\ \lambda_1 C_1 + \lambda_2 C_2 + \ldots + \lambda_n C_n & -\mu & 0 \\ \lambda_1(1 - C_1) + \lambda_2(1 - C_2) + \ldots + \lambda_n(1 - C_n) & 0 & 0 \end{bmatrix} \begin{bmatrix} p_O(t) \\ p_{FS}(t) \\ p_{FU}(t) \end{bmatrix} \tag{4.20}$$

Now, define

$$a = \sum_{i=1}^{n} \lambda_i C_i \tag{4.21}$$

and

$$\lambda_s = \sum_{i=1}^{n} \lambda_i \tag{4.22}$$

Then,

$$\sum_{i=1}^{n} \lambda_i(1-C_i) = \sum_{i=1}^{n} (\lambda_i - \lambda_i C_i) = \sum_{i=1}^{n} \lambda_i - \sum_{i=1}^{n} \lambda_i C_i = \sum_{i=1}^{n} \lambda_i - a = \lambda_s - a \tag{4.23}$$

Using the definitions given in **Equation (4.21)** and **Equation (4.22)**, the Markov model shown in **Figure 4.1** can be represented as the model shown in **Figure 4.3**, where the individual LRU transitions from the operational state, O, to the two failed states, failed-safe (FS) and failed-unsafe (FU) have been combined.

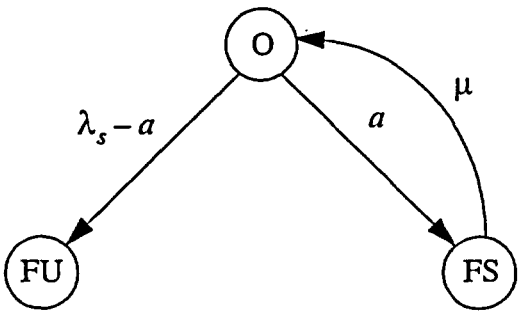


Figure 4.3. Markov model using the definitions in Equation (4.21) and Equation (4.22).

Applying the definitions from **Equation (4.21)** and **Equation (4.22)** to **Equation (4.20)** yields

$$\begin{bmatrix} \frac{d}{dt}p_O(t) \\ \frac{d}{dt}p_{FS}(t) \\ \frac{d}{dt}p_{FU}(t) \end{bmatrix} = \begin{bmatrix} -\lambda_s & \mu & 0 \\ a & -\mu & 0 \\ \lambda_s - a & 0 & 0 \end{bmatrix} \begin{bmatrix} p_O(t) \\ p_{FS}(t) \\ p_{FU}(t) \end{bmatrix} \tag{4.24}$$

Taking the Laplace transform yields

$$\begin{bmatrix} sP_O(s) - p_O(0) \\ sP_{FS}(s) - p_{FS}(0) \\ sP_{FU}(s) - p_{FU}(0) \end{bmatrix} = \begin{bmatrix} -\lambda_s P_O(s) + \mu P_{FS}(s) \\ aP_O(s) - \mu P_{FS}(s) \\ (\lambda_s - a)P_O(s) \end{bmatrix} \tag{4.25}$$

For $p_O(0) = 1$ and $p_{FS}(0) = p_{FU}(0) = 0$, we have

$$\begin{bmatrix} sP_O(s) - 1 \\ sP_{FS}(s) \\ sP_{FU}(s) \end{bmatrix} = \begin{bmatrix} -\lambda_s P_O(s) + \mu P_{FS}(s) \\ aP_O(s) - \mu P_{FS}(s) \\ (\lambda_s - a)P_O(s) \end{bmatrix} \tag{4.26}$$

Solving this set of equations yields

$$\begin{bmatrix} P_O(s) \\ P_{FS}(s) \\ P_{FU}(s) \end{bmatrix} = \begin{bmatrix} \left(\frac{\mu + r_1}{r_1 - r_2}\right)\left(\frac{1}{s - r_1}\right) - \left(\frac{\mu + r_2}{r_1 - r_2}\right)\left(\frac{1}{s - r_2}\right) \\ \left(\frac{a}{r_1 - r_2}\right)\left(\left(\frac{1}{s - r_1}\right) - \left(\frac{1}{s - r_2}\right)\right) \\ \left(\frac{\lambda_s - a}{r_1 - r_2}\right)\left(\left(\frac{r_1}{\mu + r_1}\right)\left(\frac{1}{s - r_1} - \frac{1}{s}\right) - \left(\frac{r_2}{\mu + r_2}\right)\left(\frac{1}{s - r_2} - \frac{1}{s}\right)\right) \end{bmatrix} \tag{4.27}$$

where

$$r_1 = -\left(\frac{\mu + \lambda_s}{2}\right) + \left(\frac{(\mu + \lambda_s)^2 - 4\mu(\lambda_s - a)}{4}\right)^{1/2} \tag{4.28}$$

and

$$r_2 = -\left(\frac{\mu + \lambda_s}{2}\right) - \left(\frac{(\mu + \lambda_s)^2 - 4\mu(\lambda_s - a)}{4}\right)^{1/2} \tag{4.29}$$

Taking the inverse Laplace transform yields

$$\begin{bmatrix} p_O(t) \\ p_{FS}(t) \\ p_{FU}(t) \end{bmatrix} = \begin{bmatrix} \left(\frac{\mu + r_1}{r_1 - r_2}\right)e^{r_1 t} - \left(\frac{\mu + r_2}{r_1 - r_2}\right)e^{r_2 t} \\ \left(\frac{a}{r_1 - r_2}\right)(e^{r_1 t} - e^{r_2 t}) \\ \left(\frac{\lambda_s - a}{r_1 - r_2}\right)\left(\left(\frac{r_1}{\mu + r_1}\right)(e^{r_1 t} - 1) - \left(\frac{r_2}{\mu + r_2}\right)(e^{r_2 t} - 1)\right) \end{bmatrix} \tag{4.30}$$

From **Definition 8**, we can state that the reliability is the probability of being in the operational state, O, (shown in **Figure 4.3**) when there is no repair (and thus $\mu = 0$). Thus,

$$R(t) = p_O(t) = e^{-\lambda_s t} \tag{4.31}$$

Likewise, the safety is the probability of being in the operational state, O, or the failed-safe state, FS, shown in **Figure 4.1**. Thus, the safety of the system is

$$S(t) = p_O(t) + p_{FS}(t) = \left(\frac{\mu + r_1 + a}{r_1 - r_2}\right)e^{r_1 t} - \left(\frac{\mu + r_2 + a}{r_1 - r_2}\right)e^{r_2 t} \tag{4.32}$$

4.5. Calculating MTTHE from a Markov Model

Now, to calculate the MTTHE, we assume that the system will see a given number of safe failures, each of which is followed by a system reset, before an unsafe failure occurs [2]. However, as indicated in **Figure 4.1**, there are multiple transitions from the operational state to the failed-safe or failed-unsafe states, respectively. Thus, the MTTHE will be computed by considering the i^{th} pair of transitions to the failed-safe and failed-unsafe states (defined by λ_i and C_i), and then summing over i. An example of the system behavior for the i^{th} pair of transitions depicted in **Figure 4.4**, where a sequence of scenarios to the failed-unsafe state is depicted. Here, M_{FS_i} and M_{FU_i} represent the i^{th} mean arrival times to the failed state, FS, and the failed-unsafe state, FU, respectively. To compute the MTTHE, we recognize that each scenario in **Figure 4.4** represents the i^{th} mean time to the first unsafe failure, given that $(m-1)$ safe failures followed by $(m-1)$ repairs have occurred. The mean time to fail safe failure, fail unsafe failure, and repair time are given as M_{FS_i}, M_{FU_i}, and M_R, respectively. We can calculate the expected value of this mean time to first unsafe failure by multiplying by the likelihood of occurrence and summing over all possible scenarios in accordance with **Equation (2.12)** and **Definition 6**. Thus, we have that

$$MTTHE = \sum_{i=1}^{n}\left(\sum_{m=1}^{\infty}[(M_{FS_i}+M_R)(m-1)+M_{FU_i}]C_i^{m-1}(1-C_i)\right) \tag{4.33}$$

Recognizing the expression on the right in **Equation (4.33)** (within the inner summation) as a power series, we can compute the MTTHE as:

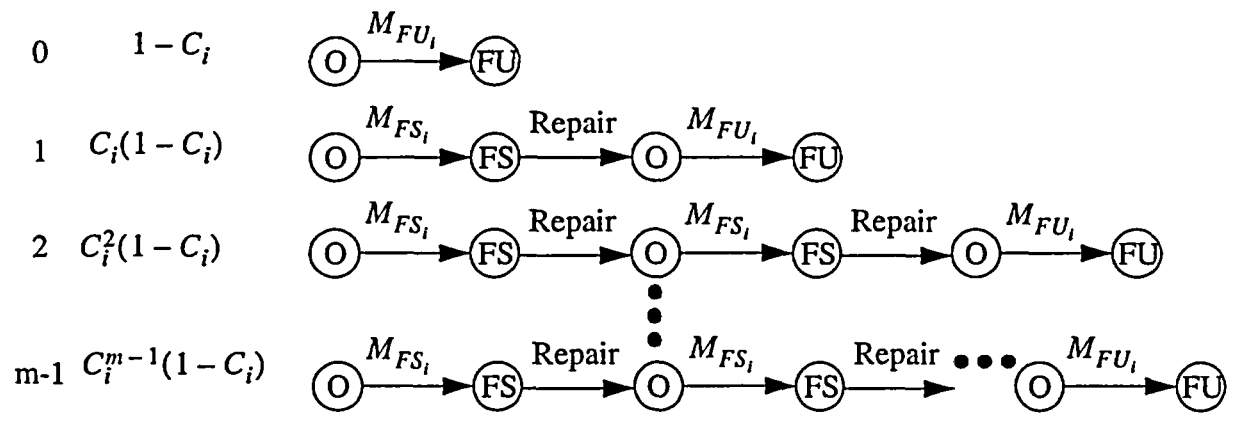


Figure 4.4. Process to calculate the MTTHE

$$\sum_{m=1}^{\infty}[(M_{FS_i}+M_R)(m-1)+M_{FU_i}]C_i^{m-1}(1-C_i)$$

$$(M_{FS_i}+M_R)(1-C_i)\sum_{m=1}^{\infty}(m-1)C_i^{m-1}+M_{FU_i}(1-C_i)\sum_{m=1}^{\infty}C_i^{m-1}$$

$$(M_{FS_i}+M_R)(1-C_i)\left[\sum_{m=1}^{\infty}mC_i^{m-1}-\sum_{m=1}^{\infty}C_i^{m-1}\right]+M_{FU_i}(1-C_i)\sum_{m=0}^{\infty}C_i^m$$

$$(M_{FS_i}+M_R)(1-C_i)\left[\sum_{m=1}^{\infty}\frac{d}{dC_i}(C_i^m)-\sum_{m=0}^{\infty}C_i^m\right]+M_{FU_i}(1-C_i)\sum_{m=0}^{\infty}C_i^m$$

$$(M_{FS_i}+M_R)(1-C_i)\left[\frac{d}{dC_i}\left(\sum_{m=1}^{\infty}C_i^m\right)-\sum_{m=0}^{\infty}C_i^m\right]+M_{FU_i}(1-C_i)\sum_{m=0}^{\infty}C_i^m \qquad (4.34)$$

$$(M_{FS_i}+M_R)(1-C_i)\left[\frac{d}{dC_i}\left(-1+\sum_{m=0}^{\infty}C_i^m\right)-\sum_{m=0}^{\infty}C_i^m\right]+M_{FU_i}(1-C_i)\sum_{m=0}^{\infty}C_i^m$$

$$(M_{FS_i}+M_R)(1-C_i)\left[\frac{d}{dC_i}\left(-1+\left(\frac{1}{1-C_i}\right)\right)-\frac{1}{(1-C_i)}\right]+M_{FU_i}(1-C_i)\left(\frac{1}{1-C_i}\right)$$

$$(M_{FS_i}+M_R)(1-C_i)\left[\frac{1}{(1-C_i)^2}-\frac{1}{(1-C_i)}\right]+M_{FU_i}$$

$$(M_{FS_i}+M_R)\left[\frac{1}{(1-C_i)}-1\right]+M_{FU_i}$$

$$MTTHE = \sum_{i=1}^{n}\left((M_{FS_i}+M_R)\left[\frac{C_i}{(1-C_i)}\right]+M_{FU_i}\right) \qquad (4.35)$$

A more useful form of **Equation (4.35)** can be obtained by noting that the i^{th} MTTF of the system is a function of M_{FS_i} and M_{FU_i}; that is

$$MTTF_i = C_iM_{FS_i}+(1-C_i)M_{FU_i} \qquad (4.36)$$

Rearranging **Equation (4.35)** and using the result provided by **Equation (4.36)** yields

$$\begin{aligned} MTTHE &= \sum_{i=1}^{n}\left((M_{FS_i}+M_R)\left[\frac{C_i}{(1-C_i)}\right]+M_{FU_i}\right) \\ &= \sum_{i=1}^{n}\left(\frac{C_iM_{FS_i}+(1-C_i)M_{FU_i}}{(1-C_i)}+M_R\left[\frac{C_i}{(1-C_i)}\right]\right) \\ &= \sum_{i=1}^{n}\left(\frac{MTTF_i}{(1-C_i)}+M_R\left[\frac{C_i}{(1-C_i)}\right]\right) \\ &= \sum_{i=1}^{n}\left(\frac{MTTF_i+M_RC_i}{(1-C_i)}\right) \end{aligned} \tag{4.37}$$

If the mean time to repair from the failed safe state is equal to the system mean time to repair then

$$MTTR = M_R \tag{4.38}$$

Typically the system MTTR is equal to the mean time to repair from the failed safe state if the unsafe system failure is a rare event. Combining the result from **Equation (4.38)** with **Equation (4.37)** provides

$$MTTHE = \sum_{i=1}^{n}\left(\frac{MTTF_i+MTTR(C_i)}{(1-C_i)}\right) \tag{4.39}$$

The expression given by **Equation (4.39)** is general in nature. Prior work in calculating MTTHE is done making a worst case analysis [3][4]; that is, it is assumed that the repair when the system enters the failed safe state is performed instantaneously. This zero time repair produces the minimum possible MTTHE, which is the most conservative approach. A zero repair time provides

$$MTTR = M_R = 0 \tag{4.40}$$

Evaluating **Equation (4.39)** using the result provided by **Equation (4.40)** yields

$$MTTHE = \sum_{i=1}^{n}\left(\frac{MTTF_i}{(1-C_i)}\right) \tag{4.41}$$

The i^{th} MTTF for the system is found by evaluating **Equation (2.16)** with the $R(t)$ equal to $e^{-\lambda_i t}$. Evaluating this expression provides

$$MTTF_i = \int_0^{\infty} e^{-\lambda_i t}dt = \frac{1}{\lambda_i} \tag{4.42}$$

Thus, the MTTHE is obtained by evaluating **Equation (4.41)** using the results provided by **Equation (4.42)**:

$$MTTHE = \sum_{i=1}^{n}\left(\frac{MTTF_i}{(1-C_i)}\right) = \sum_{i=1}^{n}\left(\frac{\frac{1}{\lambda_i}}{(1-C_i)}\right) = \frac{n}{\sum_{i=1}^{n}\lambda_i(1-C_i)} \tag{4.43}$$

4.6. Calculating Steady-State Safety (Sss) from a Markov Model

The general expression for steady-state safety S_{ss} is given by **Equation (2.4)**, repeated here:

$$S_{ss} = \lim_{t \to \infty} S(t) \tag{4.44}$$

Consider again the example system of **Figure 4.1**, as simplified by **Figure 4.3**. The steady-state safety S_{ss} is calculated using the safety expression given by **Equation (4.32)** with $\mu = 0$ from **Equation (4.40)**. Performing this operation yields

$$S_{ss} = \lim_{t \to \infty} S(t) = \lim_{t \to \infty} \left[e^{-\lambda_s t} + \left(\frac{a}{\lambda_s}\right)(1 - e^{-\lambda_s t}) \right] = \frac{a}{\lambda_s} \tag{4.45}$$

Performing the substitution for the variables in the above using **Equation (4.21)** and **Equation (4.22)** yields:

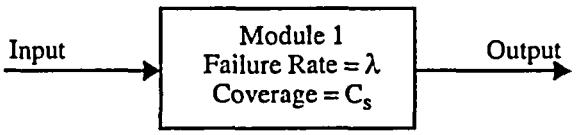
$$S_{ss} = \frac{a}{\lambda_s} = \frac{\sum_{i=1}^{n} \lambda_i C_i}{\sum_{i=1}^{n} \lambda_i} \tag{4.46}$$

5. Analytical Safety Models for Common Architectures

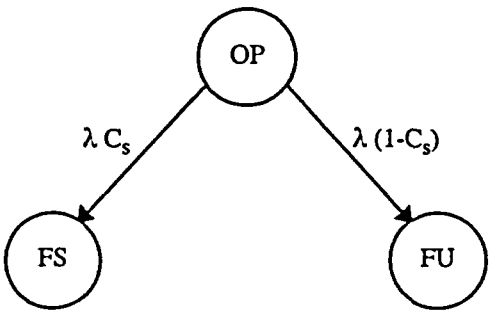
Using the mathematical analysis techniques presented in the previous section, it is possible to derive MTTHE and S_{ss} expressions for a variety of common architectures used in safety-critical systems. This chapter provides an overview of common architectures and their associated MTTHE and S_{ss} expressions (from [3]).

5.1. Simplex System

In a Simplex system, there is no form of architectural redundancy in the system to achieve improvements in either reliability or safety. Any failures of the single module in the system will directly result in an unsafe system failure, if not properly detected and reacted to by the coverage mechanism, which is typically some form of diagnostics and self-checking.



(a) System Architecture



(b) Markov Model

$$MTTHE = \frac{1}{\lambda(1 - C_s)} \qquad S_{ss} = C_s$$

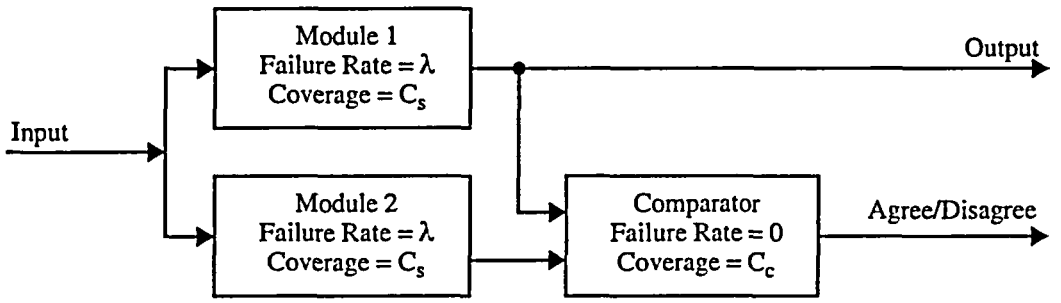
(c) Safety Metrics

Figure 5.1. Simplex System Architecture, Markov Model, and Safety Metrics

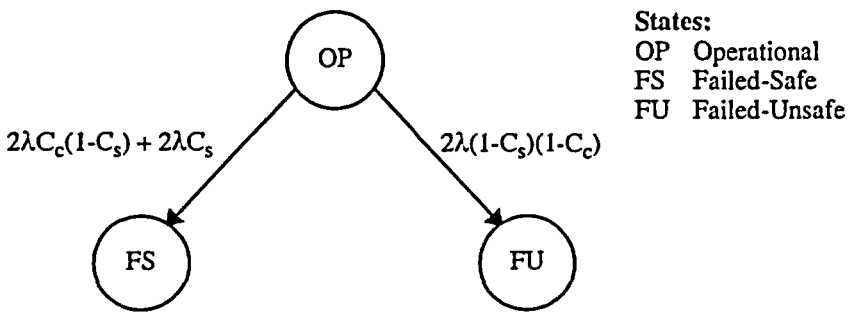
Note that if the Simplex system contains multiple independent modules in series (i.e. failure of any module will result in system failure), the MTTHE and S_{ss} results from the example of this type in Section 4.5 and Section 4.6 can be used.

5.2. Duplication with Comparison (DWC) System

In the Duplication with Comparison (DWC) architecture, two modules operate in synchrony while performing the same computation on the same set of inputs. The outputs of the two modules are either constantly or periodically compared by a Comparator and any discrepancy in the outputs indicates the presence of a fault. Each module also possesses self-diagnostic routines which attempt to detect any faults within the module itself. The system is considered failed in an unsafe manner if an incorrect output is delivered, due to a failure of either module which is not detected by either the module diagnostics or the comparison process.



(a) System Architecture



(b) Markov Model

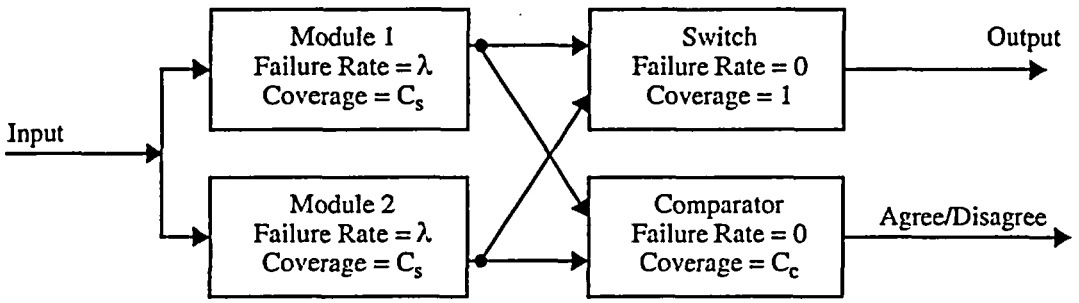
$$MTTHE = \frac{1}{2\lambda(1-C_s)(1-C_c)} \qquad S_{ss} = C_c(1-C_s) + C_s$$

(c) Safety Metrics

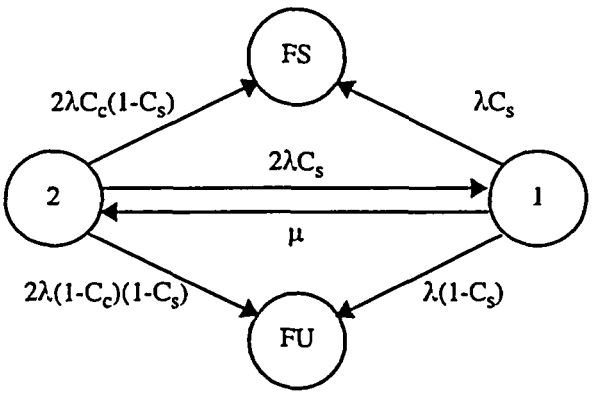
Figure 5.2. DWC System Architecture, Markov Model, and Safety Metrics

5.3. Reconfigurable Duplication with Comparison (RDWC) System

The Reconfigurable Duplication with Comparison (RDWC) architecture is a modification of the DWC architecture to incorporate active redundancy. As in DWC, two modules operate in synchrony while performing the same computation on the same set of inputs. The outputs of the two modules are either constantly or periodically compared by a Comparator and any discrepancy in the outputs indicates the presence of a fault. Each module also possesses self-diagnostic routines which attempt to detect any faults within the module itself. The different feature in RDWC is that if a module detects an internal fault, it will remove itself from the system via a switch mechanism and system operation will continue in Simplex mode using the other module. There may also be a concept of repair such that a failed module may recover and return the system to operation in full RDWC mode.



(a) System Architecture



States:
1 Single Unit Operational
2 Both Units Operational
FS Failed-Safe
FU Failed-Unsafe

(b) Markov Model

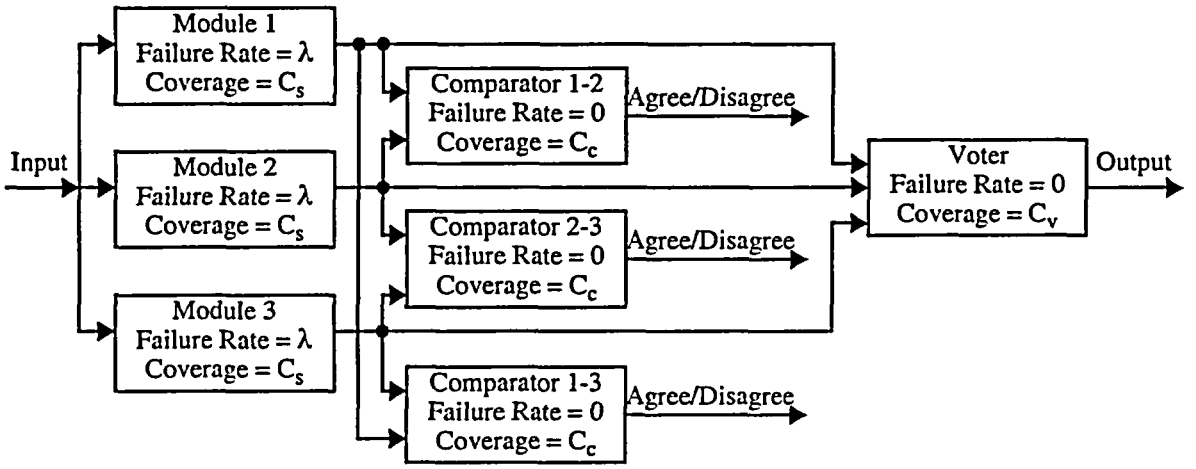
$$MTTHE = \frac{1+2C_s}{2\lambda(1-C_s)[1-C_c+C_s]} \qquad S_{ss} = C_c(1-C_s)+C_s^2$$

(c) Safety Metrics

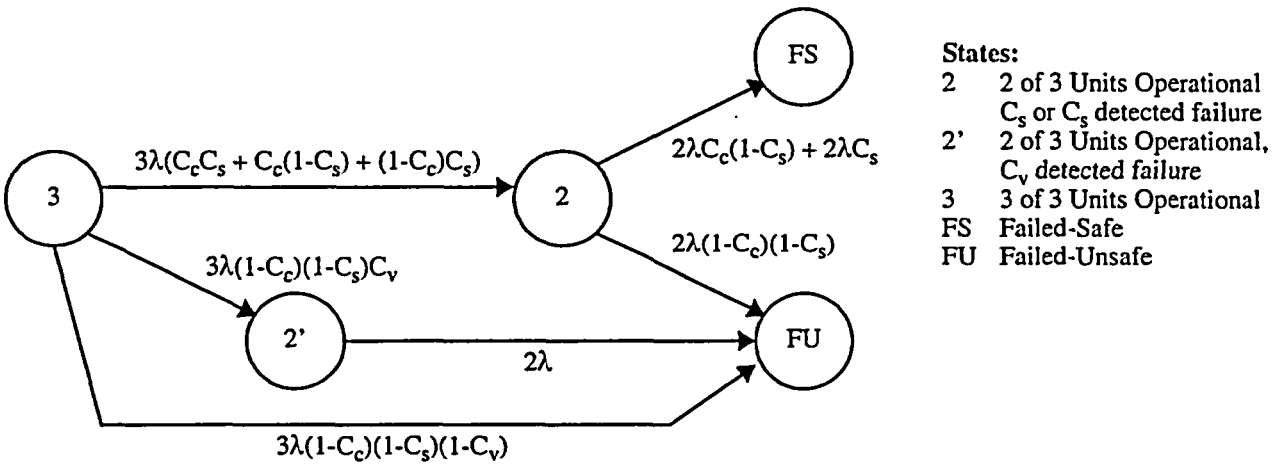
Figure 5.3. RDWC System Architecture, Markov Model, and Safety Metrics

5.4. Triplication with Comparison (TWC) System

The Triplication with Comparison (TWC) architecture is similar to the DWC architecture except that there are three modules operating in parallel and beyond the comparators there is an additional voting mechanism which performs majority voting on the three outputs. The majority voting is a passive redundancy technique which will allow the system to continue operation with one of the three modules failed. Once the second module fails, the system will fail. In this example, no repair is considered.



(a) System Architecture



(b) Markov Model

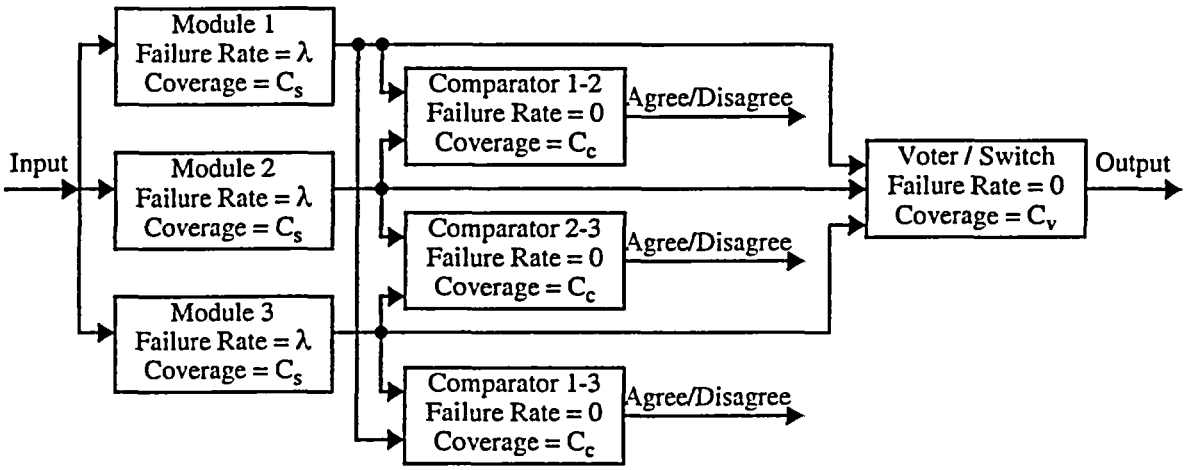
$$MTTHE = \frac{2 + 3[C_c(1-C_s) + C_s + (1-C_c)(1-C_s)C_v]}{6\lambda(1-C_c)(1-C_s)[1 + C_s + C_c(1-C_s)]} \qquad S_{ss} = [C_s^2 + C_cC_s(1-C_s)]^2$$

(c) Safety Metrics

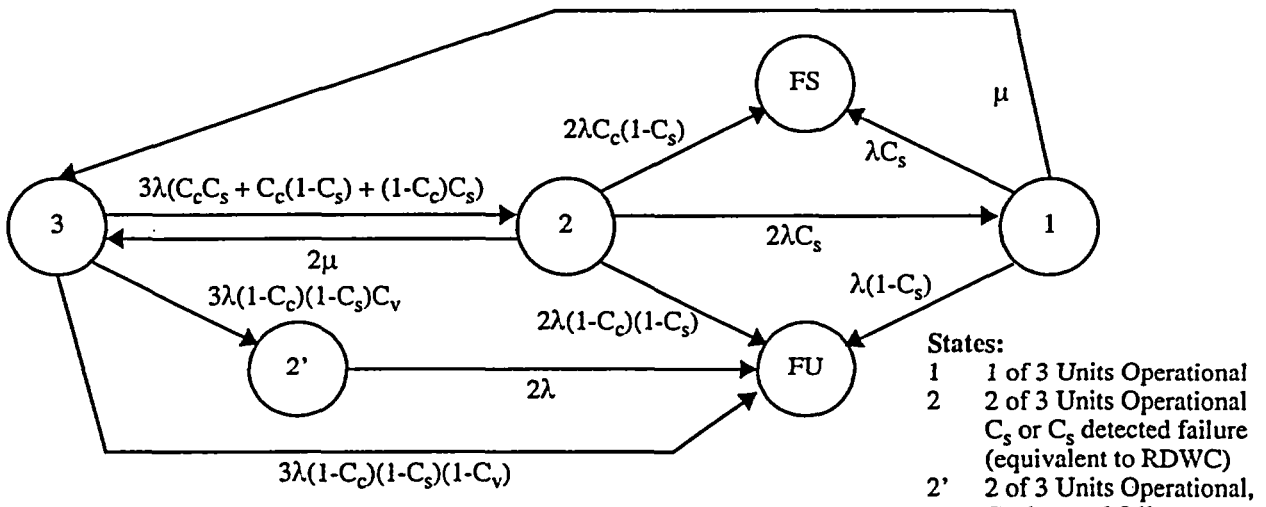
Figure 5.4. TWC System Architecture, Markov Model, and Safety Metrics

5.5. Reconfigurable Triplication with Comparison (RTWC) System

The Reconfigurable Triplication with Comparison (RTWC) architecture augments the standard TWC architecture with the active redundancy reconfiguration capabilities of the RDWC architecture. In this architecture, each of the three modules may remove themselves from the system upon detection of internal failures, and the system can operate in Simplex mode after two of the modules have failed.



(a) System Architecture



(b) Markov Model

$$S_{ss} = C_s^3 + C_cC_s(1 - C_s^2) + C_c^2(1 - C_s)^2$$

$$MTTHE = \frac{2 + 3C_s + 6C_s^2 + 3(1 - C_c)(1 - C_s)(2C_s + 1) + 3C_v(1 - C_s)(1 - C_c)}{6\lambda[1 - C_s^3 - (C_cC_s)^2 - C_cC_s(1 - C_s^2)]}$$

(c) Safety Metrics

Figure 5.5. RTWC System Architecture, Markov Model, and Safety Metrics

5.6. Safety Metric Summary

Table 5.1 below provides a concise summary of the safety metrics for the architectures considered in this report.

Architecture	MTTHE	Sss
Simplex (single module)	$\frac{1}{\lambda(1-C_s)}$	C_s
Simplex (multiple independent modules in series)	$\frac{n}{\sum_{i=1}^{n} \lambda_i(1-C_i)}$	$\frac{\sum_{i=1}^{n} \lambda_i C_i}{\sum_{i=1}^{n} \lambda_i}$
Duplication with Comparison (DWC)	$\frac{1}{2\lambda(1-C_s)(1-C_c)}$	$C_c(1-C_s)+C_s$
Reconfigurable Duplication with Comparison (RDWC)	$\frac{1+2C_s}{2\lambda(1-C_s)[1-C_c+C_s]}$	$C_c(1-C_s)+C_s^2$
Triplication with Comparison (TWC)	$\frac{2+3[C_c(1-C_s)+C_s+(1-C_c)(1-C_s)C_v]}{6\lambda(1-C_c)(1-C_s)[1+C_s+C_c(1-C_s)]}$	$[C_s^2+C_cC_s(1-C_s)]^2$
Reconfigurable Triplication with Comparison (RTWC)	$\frac{2+3C_s+6C_s^2+3(1-C_c)(1-C_s)(2C_s+1)+3C_v(1-C_s)(1-C_c)}{6\lambda[1-C_s^3-(C_cC_s)^2-C_cC_s(1-C_s^2)]}$	$C_s^3+C_cC_s(1-C_s^2)+C_c^2(1-C_s)^2$

Table 5.1 Safety Metrics Summary

6. Summary

This document has described the first step in the numerical safety quantification process - the development of an analytical safety model. System safety is expressed in a metric termed Mean Time To Hazardous Event (MTTHE). The analytical safety model results in a mathematical expression for the MTTHE based on the critical safety-related parameters in the system, which may include failure rates, fault detection latencies, repair rates, and fault coverages. The estimation of these critical parameters is the focus of the subsequent steps in the safety quantification methodology. Once parameter estimates are obtained, the values are fed back into the analytical safety model in order to derive a numerical estimate for the system MTTHE at the desired confidence level.

This document has presented the details of the analytical safety modeling approach as illustrated with Markov model representations of the systems under study. It was shown how Mean Time To Hazardous Event (MTTHE) and Steady-State Safety (S_{SS}) safety metrics may be derived from the Markov model, and these metrics were provided for several common architectures used in safety-critical systems.

7. References

[1] Cutright, E., DeLong, T., Johnson, B., *Numerical Safety Evaluation Process for Safety-Critical Systems*, UVA Technical Report UVA-CSCS-NSE-001.

[2] Smith, D. Todd, Allan White, Todd A. DeLong, Barry W. Johnson, and Ted C. Giras, *A Tutorial on Architectural Analysis Using Reliability and Safety*, Technical Report No. 990609, Center for Safety Critical Systems, University of Virginia, 1999.

[3] Choi, Charles Y., Barry W. Johnson, and Joseph A. Profeta III, "Safety Issues in the Comparative Analysis of Dependable Architectures," *IEEE Transactions on Reliability*, Vol. 46, No. 3, September 1997, pp 316-322.

[4] Choi, H., W. Wang, and K.S. Trivedi, "Conditional MTTF and its Computation in Markov Reliability Models," *Proc. 1993 Annual Reliability and Maintainability Symposium*, January 25-28, 1993, pp. 55-63.

[5] Johnson, B.W., *Design and Analysis of Fault Tolerant Digital Systems*, Addison Wesley, New York, 1989.

[6] Goble, W.M., *Control Systems Safety Evaluation and Reliability*, Instrument Society of America, 1998.

[7] Cassandras, C.G., *Discrete Event Systems: Modeling and Performance Analysis*, Aksen Associates Inc. and Richard D. Irwin Inc., Boston, Massachusetts, 1993.

[8] Karlin, S. and H.M. Taylor, *A First Course in Stochastic Processes*, Second Edition, Academic Press, New York, 1975.

[9] Murata, T., "Petri Nets: Properties, Analysis and Applications," Proceedings of the IEEE, Vol. 77, pp. 541-580, 1989.

[10] Peterson, J.L., *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[11] Modarres, M., *What Every Engineer Should Know about Reliability and Risk Analysis*, Marcel Dekker, New York, 1993.

[12] M. Ball and F. Hardie, "Effects and Detection of Intermittent Failures in Digital Systems," *AFIPS Fall Joint Computer Conference Proceedings*, pp. 329-335, 1969.

[13] R. K. Iyer and D. J. Rossetti, "A Measurement-Based Model for Workload Dependence of CPU Errors," *IEEE Transactions on Computers*, Vol. C-35, pp. 511-519, June 1986.

[14] R. K. Iyer and P. Velardi, "Hardware-Related Software Errors: Measurement and Analysis," *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 2, Feb. 1985, pp. 223-231.

[15] Ang, Alfredo H.-S. and Wilson H. Tang, *Probability Concepts in Engineering Planning and Design*, John Wiley & Sons, New York, 1975.

[16] Feller, W., *An Introduction to Probability Theory and Its Applications*, Volume 1, Wiley, New York, 1957.

[17] Hoel, P., S. Port, and C. Stone, *Introduction to Stochastic Processes*, Houghton Mifflin Company, Boston, 1972.

Statistical Model

Technical Report UVA-CSCS-NSE-003
Revision 00
August 01, 2003

Author(s)	Contact Info
Eric Cutright	edc2u@virginia.edu
Marco Pescosolido	--
Todd DeLong	tad2x@virginia.edu
Barry Johnson	bwj@virginia.edu

University of Virginia Center for Safety-Critical Systems
Department of Electrical and Computer Engineering
Thornton Hall
351 McCormick Road
Charlottesville, VA 22904-4743

Prepared for:
Project Manager: John Calvert
Division of Engineering Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555

Table of Contents

List of Figures

List of Tables

Glossary of Acronyms and Abbreviations

Acronyms and Abbreviations

C	Fault coverage
CSCS	Center for Safety-Critical Systems
E	Event Space
F	Fault Space
FS	Failed-Safe state
FU	Failed-Unsafe state
MTBHE	Mean Time Between Hazardous Event
MTTF	Mean Time To Failure
MTTFD	Mean Time To Fail Dangerous
MTTHE	Mean Time To Hazardous Event
MTTR	Mean Time To Repair
MTTUF	Mean Time To Unsafe Failure
O	Operational state
PFD(t)	Probability of Failure on Demand
PFD_{ss}	Steady-State Probability of Failure on Demand
Q(t)	Unreliability
R(t)	Reliability
S_{ss}	Steady-State Safety
S(t)	Safety
UVA	University of Virginia
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

1. Introduction

The University of Virginia (UVA) Center for Safety-Critical Systems (CSCS) has developed a methodology to quantify system safety for complex processor-based safety-critical systems [20]. The approach is built upon a campaign of extensive fault injection which is used to derive a numerical expression of system safety known as the Mean Time To Hazardous Event (MTTHE) [30] (also known alternatively as the Mean Time To Unsafe Failure (MTTUF) [30] or Mean Time Between Hazardous Events (MTBHE) [21][22]). A separate risk assessment methodology is employed at the overall system level to derive MTTHE allocations for individual processor-based systems. The UVA safety quantification methodology is then applied to the individual processor-based systems in order to demonstrate compliance with their MTTHE allocations.

This document describes the second step in the MTTHE compliance process - the development of a statistical model. The statistical model allows the mathematical estimation of fault coverage (with an associated confidence level) to be derived from the set of fault injection experiments for a given system. The statistical model is also used to calculate the number of fault injection experiments required to meet the fault coverage target for a given confidence level.

This document consists of fourteen sections. Following this introduction, Section 2 presents important background and terminology information, including the development of the MTTHE and fault coverage concepts. Section 3 then provides an overview of the UVA numerical safety evaluation process to demonstrate where the statistical modeling fits into the overall methodology. Next, Section 4 presents an overview of some key mathematical concepts in statistical modeling, including a discussion of fault coverage parameters and estimation processes. Section 5 through Section 12 then describe different statistical modeling techniques which are described in the literature. Next, Section 13 provides a short summary for the document. Finally, Section 14 presents a set of important references for this document.

Note that the vast majority of the information in this report is taken from [25].

2. Background and Terminology

This section provides background and terminology information which is useful to the understanding of the numerical safety evaluation process. The concept of Mean Time To Hazardous Event (MTTHE) is developed along with several related safety and reliability metrics and their dependent factors.

2.1. Modeling of System Safety

A discussion on the modeling of system safety naturally begins with the definition of safety:

> **Definition 1: Safety,** $S(t)$, is the probability that a system will either perform its functions correctly or will discontinue its functions in a manner that does not disrupt the operation of other systems or compromise the safety of any people associated with the system [24].

There are two important things to note about this definition. First, we define safety as a probability, which implies that safety is a quantitative measure that can take on values between zero and one, inclusive.

$$0.0 \leq S(t) \leq 1.0 \tag{2.1}$$

Second, there is an inherent notion of system states corresponding to whether the system is safe or unsafe. The first state is associated with the system functioning correctly, and we will refer to this state as the Operational State. The second state is associated with the system discontinuing "... its functions in a manner that does not disrupt the operation of other systems or compromise the safety of any people associated with the system." In other words, the system ceases to perform its function, but it does so in a safe manner. We will refer to this state as the Failed-Safe State because the system has failed to perform its function but in a safe manner. The third and final state is essentially all the scenarios that are not included in the first two states. That is, the third state is associated with the system not performing its function correctly, and in addition possibly jeopardizing the safety of the system. We will refer to this third state as the Failed-Unsafe State because like the Failed-Safe State the system has failed to perform its function but has done so in an unsafe manner. The relationship between these three states is shown in **Figure 2.1.**

Referring to **Figure 2.1,** we will assume that the system begins in the Operational state, where it is performing its intended function correctly. According to **Figure 2.1,** the system can transition to either the Failed-Safe state or the Failed-Unsafe state (indicated by the arrows in **Figure 2.1**). Associated with each state transition is a parameter that indicates the rate at which the transition occurs, and as such is referred to as the transition rate. The state transition rate depends on two parameters: (1) the failure rate function, $\lambda(t)$, and (2) the fault coverage, $C(t)$, defined below.

> **Definition 2: Failure rate function,** $\lambda(t)$, represents the rate at which the system fails. For safety studies, this rate considers only those failures that can potentially result in an unsafe system state. The failure rate function may be time-varying or may be a constant. The typical approach is to assume that the failure rate function follows an exponential distribution, resulting in a constant failure rate, λ. One assumption that we will make regarding the failure rate function is that the value of the failure rate function is always

greater than zero. As such, the system will eventually transition from the Operational state to either the Failed-Safe state or the Failed-Unsafe state (that is, the system will eventually fail as $t \rightarrow \infty$).

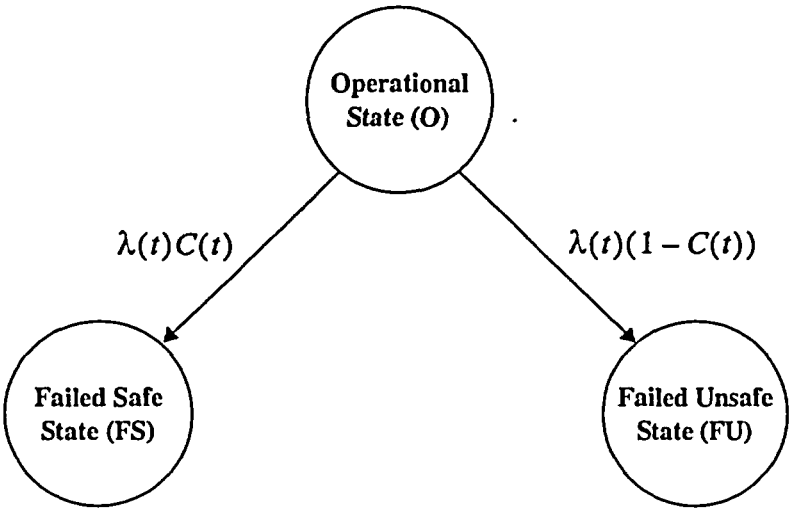


Figure 2.1. Three State System Model used to Calculate Safety

Definition 3: Fault coverage, $C(t)$, is the conditional probability that a system correctly handles a fault, given that a fault has occurred. The fault coverage function may be time-varying or may be a constant. The typical approach is to assume that the fault coverage is a constant, C. And, as with the safety, because fault coverage is a probability, it will take on values between zero and one, inclusive.

$$0.0 \leq C(t) \leq 1.0 \tag{2.2}$$

Referring to **Figure 2.1**, the transition rate from the Operational state to the Failed-Safe state is $\lambda(t)C(t)$, representing the case where a potentially unsafe fault has occurred and the system has correctly handled the fault and transitioned to a Failed-Safe state. The transition rate from the Operational state to the Failed-Unsafe state is $\lambda(t)(1-C(t))$, representing the case where a potentially unsafe fault has occurred and the system has NOT correctly handled the fault and has transitioned to a Failed-Unsafe state. For safety studies, it is assumed that a potentially unsafe fault which remains undetected by the system will place the system into a Failed-Unsafe state.

From the model of **Figure 2.1**, there are two general classes of safety and reliability metrics which can be computed:

1. The probability that a system is operating in a given state at time t, and
2. The expected time to some event of interest

Common "state probability"-based metrics from the first class include:

- Safety, $S(t)$
- Steady-State Safety, S_{ss}
- Probability of Failure on Demand, $PFD(t)$
- Steady-State Probability of Failure on Demand, PFD_{ss}

Common "expected time to event"-based metrics from the second class include:

- Mean Time To Hazardous Event, *MTTHE*
- Mean Time to Unsafe Failure, *MTTUF*
- Mean Time To Fail Dangerously, *MTTFD*
- Mean Time To Failure, *MTTF*
- Mean Time To Repair, *MTTR*

Each of these metrics will be discussed in the following sections.

2.2. Derivation of Safety (S(t)) Metric

To derive an expression for safety, *S(t)*, refer back to Definition 1. From this definition, we can calculate an expression for safety by determining the probability of being in either the Operational or Failed-Safe states for the simple three state model of **Figure 2.1**. Assuming a constant fault coverage C and constant failure rate λ, the safety expression for this model is given by:

$$S(t) = p_{OP}(t) + p_{FS}(t) = e^{-\lambda t} + (C - C \cdot e^{-\lambda t}) = C + (1 - C) \cdot e^{-\lambda t} \qquad (2.3)$$

where $p_{OP}(t)$ and $p_{FS}(t)$ indicate the probability of being in the Operational and Failed-Safe states, respectively, at time t.

2.3. Derivation of Steady-State Safety (S_{SS}) Metric

Another useful and common metric for quantifying system safety is the steady-state safety, S_{ss}. The steady-state safety may be calculated from the expression for system safety, $S(t)$. Again, from Definition 1, we conclude that the safety for a system which can be represented by **Figure 2.1** is the probability of either residing in the Operational State or the Failed-Safe State.

> **Definition 4:** The **Steady-State Safety, S_{SS},** is a probability representing the evaluation of safety as a function of time, in the limiting case as time approaches infinity:

$$S_{ss} = \lim_{t \to \infty} S(t) \tag{2.4}$$

Note that S_{ss} represents the probability that the system will operate safely after it is placed into service. S_{ss} represents the minimum safety for a safety-critical system and as such provides a lower bound for the system safety under most conditions. It is also a useful metric in that the fault coverage represents a conservative lower bound on steady-state safety, such that this bound can be expressed independently of the system failure rate (which may be difficult to estimate, especially for design faults), in contrast to MTTHE which requires estimation of the system failure rate.

As an S_{ss} example, consider the simple model of **Figure 2.1**, with a constant fault coverage C and constant failure rate λ. In this case, the S_{ss} expression for the system can be derived from the safety expression of **Equation (2.3)** as:

$$S_{ss} = \lim_{t \to \infty} S(t) = \lim_{t \to \infty} (C + (1 - C) \cdot e^{-\lambda t}) = C \tag{2.5}$$

2.4. Derivation of Probability of Failure On-Demand (PFD(t)) Metric

A useful metric for quantifying system safety is the Probability of Failure on Demand, $PFD(t)$. This metric represents the probability that the system has failed in an dangerous manner at a given instant of time. In other words, this metric represents the probability that the system has failed dangerously at the time it is needed to respond to a demand [23]. This metric is particularly useful for quantifying the safety of protection systems (e.g. nuclear reactor protection systems), which perform safety-critical interventions in the event of a system failure to bring the system to a safe state.

The formulation of this metric requires a slight modification to the three-state safety model as illustrated in **Figure 2.2**. Comparing this to **Figure 2.1**, the three states are Operational, Failed-Safe and Failed-Dangerous instead of Operational, Failed-Safe, and Failed-Unsafe. The definition for the Operational state is the same as stated in **Section 2.1**. In this case, the Failed-Safe state refers to the situation where the system shuts down even when there is no potentially dangerous situation. A failure of this type would be considered a safe failure, a false alarm or a false trip. The Failed-Dangerous state covers the case when the system would be unable to respond to a demand, a potentially dangerous condition [23]. Referring to **Figure 2.1**, the failed-unsafe state of this model is replaced with the failed-dangerous state. The remaining states remain unchanged.

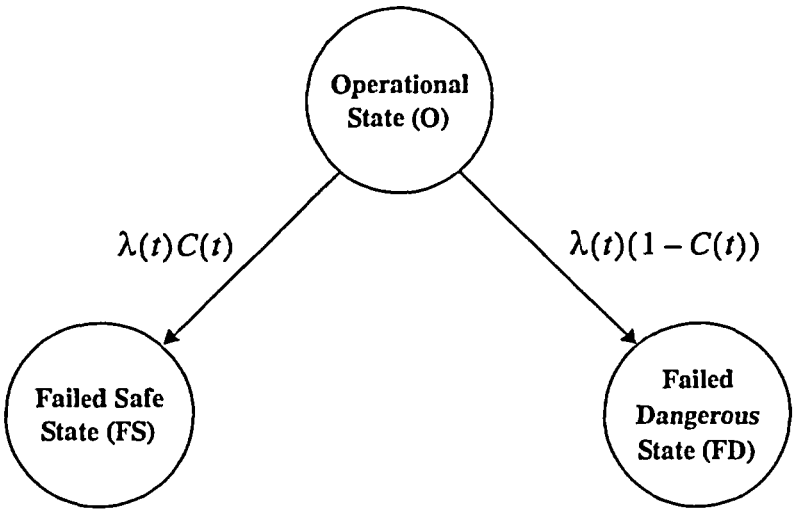


Figure 2.2. Modified Three State System Model used to Calculate MTTFD and PFD

Referring to **Figure 2.2**, we can calculate $PFD(t)$ as the following:

$$PFD(t) = p_{FD}(t) = 1 - p_O(t) - p_{FS}(t) \tag{2.6}$$

As an example of the $PFD(t)$ metric, consider the simple model of **Figure 2.2**, with a constant fault coverage C and constant failure rate λ. In this case, the $PFD(t)$ expression for the system is:

$$PFD(t) = p_{FD}(t) = 1 - p_O(t) - p_{FS}(t) = 1 - e^{-\lambda t} - (C - C \cdot e^{-\lambda t}) = (1 - C) + (1 - C) \cdot e^{-\lambda t} \tag{2.7}$$

2.5. Derivation of Steady-State Probability of Failure On-Demand (PFD_{SS}) Metric

Another useful metric for quantifying system safety is the steady-state Probability of Failure on Demand, PFD_{ss}, which can be calculated from the expression for $PFD(t)$. From **Figure 2.2** we conclude that PFD(t) is the likelihood of residing in the failed-dangerous state.

> **Definition 5:** : The **Steady-State Probability of Failure On-Demand, PFD_{SS},** is a probability representing the evaluation of PFD as a function of time, in the limiting case as time approaches infinity:
>
> $$PFD_{ss} = \lim_{t \to \infty} PFD(t) \tag{2.8}$$

As an PFD_{ss} example, consider the simple model of **Figure 2.2**, with a constant fault coverage C and constant failure rate λ. In this case, the PFD_{ss} expression for the system is:

$$PFD_{ss} = \lim_{t \to \infty} PFD(t) = \lim_{t \to \infty} [(1 - C) + (1 - C) \cdot e^{-\lambda t}] = (1 - C) \tag{2.9}$$

Note that $PFD(t)$ is related to $S(t)$ (sometimes referred to as Safety Availability, *SA(t)* [23]) by the following relationship:

$$PFD(t) = 1 - S(t) = 1 - SA(t) \tag{2.10}$$

In addition, PFD_{ss} is related to S_{ss} by:

$$PFD_{ss} = 1 - S_{ss} \tag{2.11}$$

2.6. Derivation of Mean Time To Hazardous Event (MTTHE) Metric

For safety studies, a useful mean time metric is the Mean Time to Hazardous Event (MTTHE), also known as the Mean Time To Unsafe Failure (MTTUF) [30]. Here the "event" is an unsafe system failure (as opposed to a system failure for the MTTF). The MTTHE is the primary safety metric that will be used in the UVA numerical safety evaluation process and is defined below:

> **Definition 6:** The **Mean Time To Hazardous Event (MTTHE)** is the expected time that a system will operate before the first *unsafe* failure occurs. Mathematically, the MTTHE is defined as the expected value of a random variable, X, which represents the time to the first unsafe failure. It can be defined as a continuous-time or discrete-time function, but will be defined as a discrete-time function here since that is how it is used later in this document. Given this, then
>
> $$E[X] = MTTHE = \sum X_i \times P_i \tag{2.12}$$
>
> where E is the statistical expectation operator for a probability mass function, and P_i is the likelihood of occurrence of X_i [30].

As an MTTHE example, consider the simple model of **Figure 2.1**, with a constant fault coverage C and constant failure rate λ. In this case, the MTTHE expression for the system is:

$$MTTHE = \frac{1}{\lambda(1-C)} \tag{2.13}$$

Note that MTTHE is equivalent to the metrics Mean Time Between Hazardous Events (MTBHE) and Mean Time To Unsafe Failure (MTTUF), which have been documented in literature [30][22]. As such, MTTHE will be used throughout the remainder of this report, knowing that the results apply equally to MTBHE or MTTUF.

2.7. Derivation of Mean Time to Fail Dangerous (MTTFD) Metric

As with the Probability of Failure on Demand metric, the Mean Time To Fail Dangerous (MTTFD) metric is particularly useful for quantifying the safety of protection systems (e.g. nuclear reactor protection systems), which perform safety-critical interventions in the event of a system failure to bring the system to a safe state. From the model of **Figure 2.2**, the MTTFD metric is defined as:

> **Definition 7:** : The **Mean Time To Fail Dangerous (MTTFD)** is the expected time that a system will operate before the first dangerous failure occurs [23].

Mathematically, the MTTFD can be defined the same manner as MTTHE in **Equation (2.12)**.

As an MTTFD example, consider the simple model of **Figure 2.2**, with a constant fault coverage C and constant failure rate λ. In this case, the MTTFD expression for the system is:

$$MTTFD = \frac{1}{\lambda(1-C)} \tag{2.14}$$

Note that the MTTFD is equivalent to the MTTHE metric presented earlier. This metric is also equivalent to the MTBHE and MTTUF. As such, MTTHE will be used throughout the remainder of this report.

2.8. Other Important Metrics

Other important reliability metrics commonly used in safety-critical applications are shown below. In contrast to safety studies, reliability studies are typically concerned with all types of failures, not just those that can affect the safety of the system. Thus, whereas the failure rate function for safety studies is limited to consideration of only those failures that can potentially result in an unsafe system state, the failure rate function for reliability studies will also reflect other types of failures in the system, especially those that interrupt system service.

> **Definition 8:** The **reliability,** $R(t)$, of a system is a function of time, defined as the conditional probability that the system will perform correctly throughout the interval $[t_0,t]$, given that the system was performing correctly at time t_0 [24].
>
> **Definition 9:** The **unreliability,** $Q(t)$, of a system is a function of time, defined as the conditional probability that the system will perform *incorrectly* during the interval $[t_0,t]$, given that the system was performing *correctly* at time t_0 [24].

Note that the unreliability and reliability of a system are directly related. Specifically, the probability that the system is failed or operational must always equal 1.0. This relationship is expressed as:

$$R(t) + Q(t) = 1.0 \tag{2.15}$$

> **Definition 10:** The **Mean Time To Failure (MTTF)** [24] is the expected time that a system will operate before the *first* failure occurs.

Note that based on this definition, it can be shown that the MTTF for a system can be computed if the reliability of the system, $R(t)$, is known, and $R(t)$ approaches zero as $t \rightarrow \infty$ (from [24]):

$$MTTF = \int_0^\infty R(t)dt \tag{2.16}$$

3. Overview of Numerical Safety Evaluation Process

One quantitative method to demonstrate the MTTHE compliance of a system is the evaluation of the system fault processing capabilities using fault injection. The general approach is to inject randomly selected faults into the system (on a working prototype, for example) to determine if the fault processing capabilities mitigate the faults. This safety quantification approach builds upon the University of Virginia's extensive research experience in this area. The methodology is outlined in **Figure 3.1** and consists of the following steps:

1. An analytical safety model is developed from the system architecture and inter-component dependencies to derive an expression for the MTTHE. The MTTHE expression is a function of *Critical Model Parameters* such as the fault coverage values and failure rates for the various components of the system. The development of the analytical safety model is the subject of this document.
2. A statistical model is then developed based on those found in published literature and is used to estimate the *Critical Model Parameters* that are required by the analytical safety model. This statistical model is also used to calculate the number of fault injection experiments required to meet the numerical safety target for a given confidence level.
3. A high-level generic processor fault model is defined to specify the types of faults (and their associated probabilities) that will be injected into the system under analysis. This process builds upon recent research at UVA in generic processor fault modeling, undertaken with the goal of characterizing low-level internal processor faults at the higher register-transfer level, in order to demonstrate that faults injected on the actual hardware are representative of the low-level processor faults of concern.
4. One or more operational profiles are then defined which will be used to drive the inputs to the system under analysis during the fault injection process. The operational profiles must be representative of the different system configurations and workloads that would be experienced in actual field operation.
5. A fault-free execution trace is created for each selected operational profile that will be used to generate the list of faults to inject into the system under analysis. This trace will also be used during the analysis of the fault injection experimental results.
6. Using the fault-free execution trace and the fault categories from the generic processor fault model, a fault list construction algorithm is applied to generate a list of possible faults that can be injected into the system under analysis, and which are likely to have an effect on the system. From this complete set of responsive faults, a fault list selection algorithm is then applied to randomly select a list of faults to be injected into the system, using the fault categories and associated occurrence probabilities from the generic processor fault model.
7. A fault equivalence algorithm is applied to each fault list to identify those sub-sets of faults which will have the same effect on the system, known as fault equivalence classes. The set of faults to be injected into the system is then reduced using the fault equivalence information, since only one fault from each class needs to be injected. This reduces the amount of time required to perform the evaluation of the system under analysis by reducing the total number of fault injection experiments which must be performed.

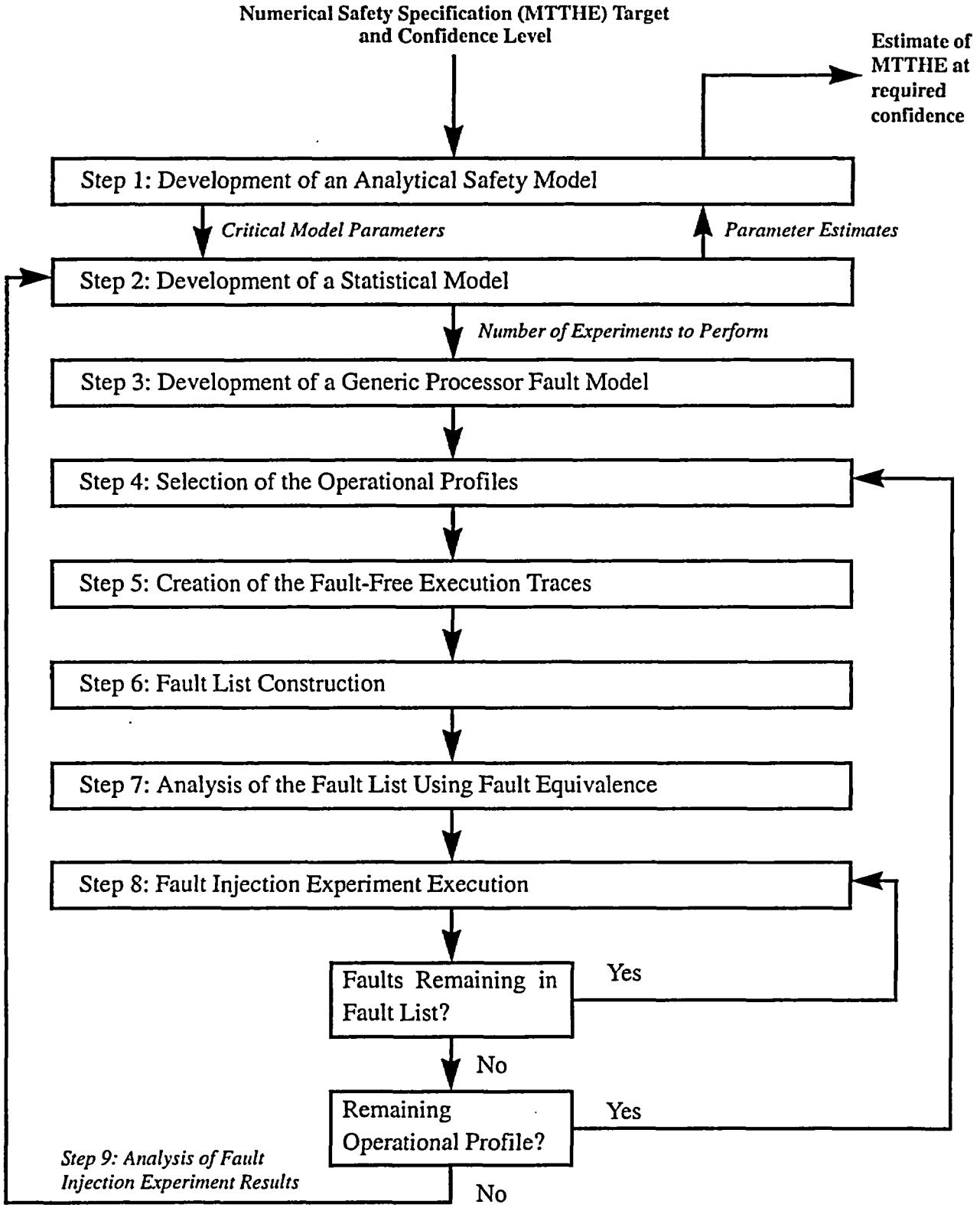


Figure 3.1. Process for quantitative safety evaluation using fault injection

8. Each fault from each reduced fault list is then injected into the system under analysis, using one of four fault injection techniques: (1) hardware-based, (2) software-based, (3) simulation-based, or (4) a hybrid approach.
9. The system is monitored during each fault injection experiment, and each fault is classified as covered, uncovered, or non-responsive by comparing the actual execution trace to the fault-free execution trace. Once all of the results of the fault injection experiments have been collected, they are fed back into the statistical model to calculate the estimates of the *Critical Model Parameters*. The *Parameter Estimates* are then in turn fed back into the analytical model to calculate the estimate of the MTTHE, as shown in **Figure 3.1.**

3.1. Development of a Statistical Model

This document describes the second step of the evaluation process outlined in **Figure 3.1**, the development of a statistical model. The statistical model allows the mathematical estimation of fault coverage (with an associated confidence level) to be derived from the set of fault injection experiments for a given system. The statistical model is also used to calculate the number of fault injection experiments required to meet the fault coverage target for a given confidence level..

4. Statistical Modeling Concepts

4.1. Introduction

Several issues are involved in the problem of statistical estimation of fault coverage. It is generally believed that if perfect and exhaustive testing of the system could be performed, the value of coverage estimated experimentally would be exactly equal to the system's fault coverage. However, this is normally not the case for fault coverage estimation.

Clearly, it is impossible to achieve perfect testing, due to the limitations inherent with the experimental process. It is also evident that exhaustive testing on real systems is impractical. Fault injection campaigns normally require a large number of experiments to be performed to obtain meaningful data, due to the very high fault tolerance capabilities of dependable systems. Assuming that imperfect testing is a negligible issue, performing a sufficiently large number of injection experiments would result in high estimation accuracy. However, a major issue in fault tolerance estimation is the ability to achieve high precision, which requires detailed knowledge of how faults occur in the system, as will be clarified later.

Different statistical models available in the literature are analyzed in this report and compared in order to determine their validity and applicability to fault coverage estimation. The validation of statistical models for coverage estimation is in itself a very complex issue. Intuitively, it should be sufficient to compare the numerical results provided by different models to determine which one provides the best accuracy and precision at the lowest cost. However, numerical results can be misleading if they are not supported by an adequate mathematical formulation of the estimation process itself.

For this reason, this section is dedicated to a formal and mathematical description of the concepts that are fundamental in statistical estimation of fault coverage. Although the formulation of the problem presented here lays no claims to being the most general and formal possible, it does provide a sufficient and adequate approach that very well applies to a wide variety of statistical models for fault coverage estimation.

4.2. Fault Space

The basic formalism required for the development of a statistical model for fault coverage estimation lies in the definition of the set of all possible faults that are considered in the model of the experimental process. Only when this set is completely defined can the statistical model be developed. The set of all faults is usually called the *fault space*, and will be indicated as $\mathfrak{F}$ in this report.

It is important to first mark a distinction between the fault space and what is here defined as the *fault universe U*, the set of all possible faulty states of the system. The definition of the fault universe derives from the simple consideration that the occurrence of a fault is the transition to a system state which should not be reached, ideally, according to the transition rules imposed by the system's design. Therefore, a fault can be regarded as the faulty system state itself, and the experimental injection of a fault as the forced transition to that state.

It is practically impossible to fully describe the fault universe, as a complete definition of the state of a system is out of reach. In fact, the fault universe is normally regarded as a continuous multi-

dimensional space, thus also infinite and uncountable. Theoretically, the full description of a system's state, thus also of a faulty state, would require a distributed-parameter model of some characteristic physical quantities.

The limited ability to fully describe a system's state is normally overcome by means of concentrated-parameter models (for example, the digital values stored in a register or present on a line of a microprocessor at a certain time). The transition from a distributed- to a concentrated-parameter model corresponds to the reduction of the fault universe to the fault space. The fault space provides a simplified description of the fault universe, and it is usually discrete and finite, yet, in general, multi-dimensional. Nonetheless, the fault space is normally a very large set, impossible to exhaustively test for fault coverage. Therefore, it is the fault space that is considered as the statistical population from which the faults are sampled and injected into the system.

A fault in the fault space can be identified by a set of characteristic attributes, such as location, value, timing, etc. The type and number of attributes that characterize faults intrinsically depend on how the fault space has been constructed from the fault universe [1][5].

In general, the fault space must be regarded as a multi-dimensional space with d dimensions, where each fault is identified through a set of d coordinates, i.e. the attributes of the fault. Then the fault space can be described as the Cartesian product of the sets of values that can be assumed by the attributes, which are the axes of the multi-dimensional fault space. Indicating with α_j the axis of attribute A_j and with a_j the particular value assumed by A_j, we can write:

$$\Im = \alpha_1 \times \ldots \times \alpha_d \quad \text{and} \quad f(a_1, \ldots, a_d) \in \Im \tag{4.1}$$

where $f(a_1, \ldots, a_d)$ is the fault for which $A_j = a_j$.

One of the attributes that characterize the fault must somehow describe the operational profile of the system when the fault occurs. For example, one way of doing this is to consider the attribute system activity, which can be modeled as a trajectory in the system state space starting from the system state at the moment of fault occurrence and evolving in function of the sequence of system inputs [8][10][11]. Generally, the operational profile is implicit because it is too complex to describe. However, it is always at least implicitly assumed that faults, and a system's response to them, strongly depend on the operational profile of the system at the moment of occurrence.

4.3. Fault Distribution

The occurrence of a fault in a system can be regarded as a random event, and can therefore be described in a probabilistic framework. The axiomatic theory of probability, due to Kolmogorov [14][15], first requires the definition of the space of elementary events. The elementary events that must be considered in a fault model are the single faulty system states, thus the space of elementary events is the fault universe U.

Then the σ-algebra E of events e is determined by considering the subsets of the space of elementary events such that: $\varnothing$ belongs to E; U belongs to E; if a finite number or a countable infinity of subsets of U belong to E then also their union and their intersection belongs to E.

Finally, a probability function *Prob (e)* is defined on E that satisfies Kolmogorov's axioms. The definition of the probability function in a fault model is quite dependent on the purpose for which it is defined. For fault coverage in Markov models, the probability of the event e should be defined as the relative frequency of occurrence of the transitions of the systems' state into any of the faulty states in $e \subseteq U$. With these definitions, it is then possible to define a random variable F and a function $\Phi(F)$ such that:

$$\Phi(F):\Im \rightarrow E \tag{4.2}$$

Notice that the function $\Phi(F)$ is univocal but not bi-univocal, i.e. the counter-image of e into $\Im$ through $\Phi^{-1}(e)$ can be a generic subset of $\Im$, including the empty set. Finally, a probability function is defined on the random variable F such that:

$$P(F=f) = Prob(\Phi(f)) \quad \text{and} \quad \sum_{f \in \Im} P(F=f) = 1 \tag{4.3}$$

The fault distribution is described in a fault model by the probability function of **Equation (4.3)**.

4.3.1. Typical Fault Distributions

The fault distribution deserves careful consideration in the development of a statistical model, as the quality of the results can significantly be affected by the assumptions made about it. Real fault distributions can never be described or known completely, therefore in general an approximation of the fault distribution is considered instead.

The simplest approximation is provided by the uniform distribution, which assigns exactly the same relative probability of occurrence to each and every fault in the fault space:

$$P(F=f) = \frac{1}{|\Im|} \quad \forall (f \in \Im) \tag{4.4}$$

where $|\Im|$ indicates the cardinality of the fault space. The uniform distribution is very commonly assumed in statistical models for fault coverage estimation, but it certainly is a rough approximation of reality that must be avoided whenever sufficient information about the system is available.

A better approximation of the real fault distribution can be obtained if the fault space $\Im$ can be partitioned into M classes $\Im_i$ for which the relative probability of occurrence is known. Within each class

$\Im_i$ the uniform distribution must be assumed in absence of further information. This distribution is called the class-wise uniform distribution, as it assigns constant probabilities of occurrence to all faults belonging to the same class:

$$P(F=f) = \sum_{i=1}^{M} \frac{\delta_i(f)}{|\Im|} P(F \in \Im_i) \tag{4.5}$$

where the indicator function $\delta_i(f)$ has the value 1 or 0 if f does or does not, respectively, belong to $\Im_i$. The probabilities of occurrence of each of the classes:

$$P(F \in \Im_i) = \sum_{f \in \Im_i} P(F=f) \quad \text{with} \quad \sum_{i=1}^{M} P(F \in \Im_i) = 1 \tag{4.6}$$

are called the *weighting factors* for reasons that will be clearer later.

The class-wise uniform distribution must always be preferred to the uniform distribution, for reasons that will be discussed later. Notice that if the real fault distribution is known, then there is no need to resort to the uniform or class-wise uniform distributions. On the other hand, if the real fault distribution is unknown, then the weighting factors cannot be determined from **Equation (4.5)**, because the actual $P(F=f)$ is unknown.

Many different strategies can be employed to determine a partition of the fault space and the corresponding set of weighting factors. For example, if the relative failure rates of the single components of the system are known, the fault distribution can be modeled accordingly by attributing probabilities to the faults that are proportional to the failure rates of the components in which the faults occur. This leads to the following values of the weighting factors:

$$P(F \in \Im_i) = \lambda_i \Big/ \left(\sum_{k=1}^{M} \lambda_k \right) \tag{4.7}$$

where λ_i indicates the failure rate of the component corresponding to the fault class $\Im_i$. Another example is provided by a system for which it can be assessed that all components have the same failure rate, which, however, is proportional to the workload assigned to the component. Then the relative measure of the workload can be used as the weighting factor for the class of faults occurring in the corresponding component. Instead, if the global failure rate of the system is known to be proportional to the workload of the system, then the relative frequency of occurrence of each workload during the system's operational lifetime can be used as the weighting factor for the class of faults that occur when the corresponding workload is executed by the system.

Moreover, it is possible that the distribution of specific attributes of the system are known. For example, if the relative probability of occurrence of *stuck-at-1* and *stuck-at-0* faults is known, then the fault space can be partitioned in two classes, each associated with the corresponding weighting factor. The same concept can be generalized to the case when multiple attributes of the faults have a known

distribution [8]. The probabilities of fault occurrence can then be expressed as the product of a set of probabilities defining the distributions of the attributes of the fault, for example:

$$P(F = f(a_1, a_2, a_3, a_4, \ldots, a_n)) = P(\text{system activity} = A_1 = a_1) \cdot P(\text{location} = A_2 = a_2) \cdot P(\text{value} = A_3 = a_3) \cdot P(\text{timing} = A_4 = a_4) \cdot P(\ldots) \quad (4.8)$$

Information concerning the distribution of the attributes can be obtained from data collected from real systems, from experiments or fault simulations, or from engineering judgment. Only the information on the system activity distribution is practically impossible to determine, as it depends on the considered target system, its functional input profile, and the moment when the fault occurs. Therefore, one can neglect the system activity distribution, consider the partition of the system as determined by all the other attributes for which the distribution is known, and use the probabilities calculated in **Equation (4.8)** (without the term relative to the system activity) as the weighting factors of a class-wise uniform distribution.

The same exact procedure can also be carried out for all other attributes A_j, other than the system activity, that can be assumed to have a uniform distribution over the range of possible values α_j. Each class of the induced partition of the fault space will contain all faults with the attributes for which the distribution is known set to specific values and the other attributes undefined. Therefore, all classes will have the same cardinality $|\alpha_{u1}| \cdot |\alpha_{u2}| \cdot \ldots \cdot |\alpha_{uk}|$, where α_{ux} are the axes of the k attributes with unknown distribution.

The distribution determined with this procedure can provide a very accurate approximation of the real fault distribution, because the partition induced generally contains a large number of classes.

4.4. Definition of Fault Coverage

The term fault coverage refers to the constant quantity mathematically defined as:

$$C = P[(\text{proper handling of fault})|(\text{occurrence of a fault} \in \mathfrak{S})] \tag{4.9}$$

i.e., the conditional probability that correct fault handling is performed given that a fault belonging to the fault space $\mathfrak{S}$ has occurred in the system. However, it must be pointed out that the mentioned probability is more properly referred to as the coverage factor, which is the limit as $\tau \to \infty$ of the coverage function [8][9]:

$$C = P[(\text{proper fault handling at time } t = t_c \in (t_0, t_0 + \tau))|(\text{occurrence of a fault} \in \mathfrak{S} \text{ at } t = t_0)] \tag{4.10}$$

where t_c is the coverage latency, i.e. the delay between the occurrence of the fault and its handling by the fault-tolerance mechanisms. Thus, the coverage factor is:

$$C = C_\infty = C(\tau)|_{\tau \to \infty} \tag{4.11}$$

and the coverage latency is a random variable with cumulative distribution function:

$$F_{tc}(\tau) = C(\tau) = C_\infty \tag{4.12}$$

The time-dependent coverage function is not discussed in this report, and the term fault coverage will be used to indicate only the asymptotic value of the fault coverage function of **Equation (4.10)**, defined by **Equation (4.9)** and **Equation (4.11)**.

4.4.1. Fault Coverage

The random event described by the predicate "proper handling of fault | occurrence of a fault $\in \mathfrak{S}$" can be associated to a binary random variable Y, which assumes the value 1 when the predicate is true, and the value 0 when the predicate is false. The variable Y is then distributed like a Bernoulli distribution of parameter C, as evident from the definition of coverage provided in **Equation (4.9)** and from the definition of the parameter of the Bernoulli distribution. It is well known that for a Bernoulli variable the parameter of the distribution equals the mean of the variable. In this case, we have:

$$\begin{aligned} C = E[Y] &= 1 \cdot P(Y = 1) + 0 \cdot P(Y = 0) = P(Y = 1) \\ &= \sum_{f \in \mathfrak{S}} P(Y = 1 | F = f) P(F = f) \end{aligned} \tag{4.13}$$

The last expression is obtained applying the theorem of total probability [14]. It is generally assumed that the fault tolerance mechanism will always either cover or not cover a certain fault f, the probability $P(Y = 1|F = f)$ is either 0 or 1, and can be expressed as a function of the fault:

$$y(f) = P(Y = 1|F = f) = \{1 \text{ if f is covered; } 0 \text{ if f is not covered}\} = Y \tag{4.14}$$

then, from **Equation (4.13)**, we obtain the following expression for the fault coverage:

$$C = \sum_{f \in \mathfrak{S}} y(f) P(F = f) \tag{4.15}$$

The validity of assumption stated in **Equation (4.14)** depends on how accurately the constructed fault space $\mathfrak{S}$ describes the real fault space of the system. In many systems, the ability of the fault tolerance mechanism to tolerate a certain fault strongly depends on the operational profile of the system at the moment of occurrence. As an example, software diagnostic routines that execute during the processor's idle time can be inhibited if the workload is very high: for such a system, the workload must be considered as one of the attributes of faults, i.e. one of the coordinates in the fault space $\mathfrak{S}$. The same concept extends to the other attributes as well, meaning that any attribute to which the fault tolerance mechanism is sensitive must be used as one of the dimensions of the fault space $\mathfrak{S}$.

4.4.2. Forced Coverage

Imagine that one could force a certain distribution of the fault space. Then we would have to associate the event occurrence of a fault $\in \mathfrak{S}$ to a new random variable $\underline{F}$, different from F, which is distributed according to the new probability function $P(\underline{F}=f)$.

A new Bernoulli random variable $\underline{Y}$, different from Y, must then be introduced to describe the fault handling event. The distribution parameter $\underline{C}$, different from C, of the variable $\underline{Y}$ is called the forced coverage and is calculated as in **Equation (**, **Equation (4.14)**, and **Equation (4.15)**:

$$\underline{C} = E[\underline{Y}] = \sum_{f \in \mathfrak{S}} y(f)P(\underline{F}=f) \tag{4.16}$$

Although the two variables Y and $\underline{Y}$ have different distributions, they are related by the fact that the fault tolerance mechanism is the same for both, that is $Y = \underline{Y} = y(f)$ whether the fault f occurs with the fault distribution or with the forced distribution. Therefore, the values of C and $\underline{C}$ must also be related. In order to determine the relationship between the two distribution parameters, a new random variable P is introduced, which is also a function of the variable $\underline{F}$ that assumes the values:

$$P = p(f) = \frac{P(F=f)}{P(\underline{F}=f)} \tag{4.17}$$

when the event $\underline{F} = f$ occurs. The mean of P can be calculated by:

$$E[P] = \sum_{f \in \mathfrak{S}} p(f)P(\underline{F}=f) = \sum_{f \in \mathfrak{S}} P(F=f) = 1 \tag{4.18}$$

that is, when faults occur in the system with the forced distribution the ratio between the probabilities that the fault occurs according to the fault distribution and the forced distribution is equal, on the average, to unity.

The covariance ρ between the random variables $\underline{Y}$ and P, defined as the mean of the cross-products of the two variables with their means removed, is:

$$\rho = E[(\underline{Y}-E[\underline{Y}])(P-E[P])] = E[(\underline{Y}-\underline{C})(P-1)] = E[\underline{Y}P]-E[\underline{Y}]-\underline{C}E[P]+\underline{C}$$

$$= \sum_{f \in \mathfrak{S}} y(f)p(f)P(\underline{F}=f)-\underline{C}-\underline{C}+\underline{C} = \sum_{f \in \mathfrak{S}} y(f)P(F=f)-\underline{C} = C-\underline{C} \tag{4.19}$$

The relationship between the forced coverage and the fault coverage of the system can therefore be expressed by:

$$C = \underline{C} + \rho \tag{4.20}$$

In order to better understand the meaning of the covariance ρ, the special case of the uniform forced distribution $P(\underline{F} = f)$ is considered here:

$$P(\underline{F} = f) = \frac{1}{|\mathfrak{S}|} \quad \forall (f \in \mathfrak{S}) \tag{4.21}$$

In this case, the value of the forced coverage $\underline{C}$, i.e. the mean of $\underline{Y}$, is just the total fraction of faults that are covered by the fault tolerance mechanism, called the *coverage proportion*, and indicated as $\dot{C}$ [8][10][11]:

$$\dot{C} = \underline{C} = E[Y] = \frac{1}{|\mathfrak{S}|} \sum_{f \in \mathfrak{S}} y(f) \tag{4.22}$$

The random variable P assumes the values:

$$P = p(f) = |\mathfrak{S}| \cdot P(F = f) \tag{4.23}$$

The variable P expresses the ratio between the relative probability of occurrence of the fault f and the same probability for the case when all faults are equally likely. The covariance between $\underline{Y}$ and P is then a measure of how fairly the fault tolerance mechanism behaves when it attempts to cover faults with higher or lower probability of occurrence than the average. An unfair fault tolerance mechanism that covers the most likely faults better than the least likely faults will lead to a positive covariance, while an unfair fault tolerance mechanism that covers the least likely faults better than the most likely faults will determine a negative covariance. If $\rho = 0$ then the fault tolerance mechanism is said to be *fair*.

In the more general case of a non-uniform forced distribution $P(\underline{F} = f)$, the covariance is a measure of how fairly faults occurring with probability $P(\underline{F} = f)$ will be tolerated as compared to faults occurring with probability $P(F = f)$. If $\rho = 0$, the fault tolerance mechanism is said to be *equally fair* for both distributions.

4.5. Sampling Process

The goal of fault injection experiments is that of statistically evaluating as accurately and precisely as possible the fault coverage as defined in **Equation (4.15)**. The problem is statistically equivalent to the estimation of the distribution parameter, that is the mean, of the Bernoulli random variable Y, given that faults occur with the distribution $P(F = f)$.

Ideally, one could calculate the fault coverage by observing the system during its normal operation for an infinite time, and calculating the limit of the ratio between the number of times that the fault coverage mechanism covers a fault and the total number of faults that occurred in the system.

For the purpose of statistical estimation, only a finite number of observations are made: from the statistical population Y, distributed as a Bernoulli random variable with unknown mean C, the fault coverage, a sample of size n is selected, that is a collection of independent realizations of the random variable Y, indicated as $Y_1, \ldots, Y_n$. As it was assumed before, each realization of the random variable Y is a function of a fault that has occurred in the system, that is:

$$Y_k = y(f_k) \tag{4.24}$$

Since it would take an extremely long time to observe enough occurrences of faults in the system, faults are instead sampled from the fault space $\mathfrak{S}$ and injected on purpose into the system. Indicating with $\underline{F}$ the random variable associated with the event "fault $\underline{F} = f$ has been sampled and injected", the *sampling distribution* is defined by the values of:

$$P(\underline{F} = f) \tag{4.25}$$

Notice that the fault injection experiment forces the event "occurrence of a fault $\in \mathfrak{S}$" in the system with the forced distribution in **Equation (4.25)**. That is, sampling and injecting a fault from the fault space with a certain sampling distribution is equivalent to forcing a fault distribution on the fault space. The Bernoulli random variable $\underline{Y}$ observed during fault injection experiments is not distributed like the variable Y observed during the operational life of the system. Instead, the distribution of the variable $\underline{Y}$ is Bernoulli with parameter $\underline{C}$, as determined in the previous section.

In most cases, it is assumed that sampling is performed *with replacement*. When this is not true, it is assumed that the size of the sample is very small compared to the cardinality of the fault space, therefore Bernoulli analysis is still possible [14].

4.6. Estimation

Estimating a parameter of a statistical population requires the definition of a function of the observations collected during the sampling process, properly called a *statistic* [14], which is also a random variable. The distribution of the statistic must provide meaningful information about the estimated parameter. The statistic employed depends on the estimation strategy that is desired. The most common types of estimation are: *point estimation*, *confidence intervals*, *distribution fitting*, and *test of hypotheses*. The models presented in this report all fall into the first three categories, which are described in the following section.

4.6.1. Point Estimation

The statistic used in point estimation strategies is simply called *estimator*, and it provides a value based on the sample that is simply used as the estimate of the parameter. An estimator is supposed to be distributed around the value of the estimated parameter with very high probability.

In the previous section it has been shown how the experimental process influences the estimation of fault coverage, by forcing a sampling distribution that is different from the fault distribution. An estimator for fault coverage, therefore, attempts to infer the distribution parameter of the variable Y from the observations of the variable $\underline{Y}$. This is possible because, as it has been shown in Section 4.4, the two variables are related to each other by **Equation (4.20)**. Estimators of the fault coverage must be defined according to the assumptions made about the fault space, the fault distribution and the sampling distribution.

The quality of an estimator can be assessed in many different ways. The two properties that are normally used to characterize the quality of an estimator are its *precision* and its *accuracy*. The precision of an estimator is its ability to provide estimates that fall, on the average, around the estimated values, and it is quantified by the *bias*, that is, the difference between its mean and the estimated parameter. The accuracy of an estimator is its ability to provide estimates that are, on the average, very close to each other, and it is quantified by the *variance*, that is, the mean square error of the estimator from its mean.

A reasonable balance of precision and accuracy is generally required from statistical estimators. Unfortunately, in the case of statistical estimation of fault coverage, achieving high precision is much more complicated than achieving high accuracy. This is due to the fact that the fault distribution is seldom known, and simplistic assumptions must be made about it.

4.6.2. Confidence Intervals

Rather than a point estimate of the distribution parameter, a *confidence interval* can be determined from the sample observations. Confidence intervals are determined based on the observations so that it can be assessed that the probability of the estimated parameter lying within the interval is at least equal to a certain confidence level. The extremes of a confidence interval are therefore functions of the observations, i.e. statistics.

Typically, the extremes are determined by applying a pivotal transformation to a point estimator of the parameter, from which a pivotal quantity of known distribution is derived. Then, the inverse of the pivotal transformation is applied to an ad hoc percentile, depending on the confidence level required, of the known distribution to obtain the required statistics.

In fault coverage estimation, two types of confidence intervals are generally used: single-sided, which define a lower limit for the fault coverage, and double-sided, that define a symmetric interval around the value of the fault coverage. Other methods to determine confidence intervals are also possible, as will be seen in the following sections.

4.6.3. Distribution Fitting

In the case of distribution fitting, the observed data are used to determine the distribution of the sampled population. Generally, it is assumed that the *parent distribution*, that is, the distribution of the population, is of a known type, and the values of the distribution parameters are determined from the data. There are several methods available to implement distribution fitting, among which two examples will be discussed in this report.

In fault coverage estimation, distribution fitting is used by interpreting the fault coverage not as a constant parameter, but rather as a random variable with values in the interval [0 : 1]. From the data, an a posteriori distribution is then determined.

4.7. Number of Experiments

In the design of fault tolerant systems the level of dependability, expressed by any of the common metrics, is a requirement that must be met and verified by the use of behavioral models of the system under development. The fault coverage factor necessary to achieve the required level of dependability is either itself a predetermined parameter, or it is determined as a function of the dependability metrics provided.

During or after the development of the system, fault injection campaigns on computer models or system prototypes are performed in order to verify that the requirement for the fault coverage has been met. The number of experiments necessary to assess whether or not the requirement was met strongly depends on the desired value of the fault coverage and on the degree of confidence assigned.

Dependability assessment can take up a large portion of the design time, especially if simulated fault injection is used. Therefore, it is desirable that an *a priori* estimate of how many experiments should be performed in the fault injection campaign is available, so that the cost and time necessary can be calculated accordingly. How this estimation is performed depends on the specific statistical model employed.

4.8. No-Response Problem

The occurrence of a fault in a system does not necessarily result in an error. Some types of faults, such as transient faults, might not affect the normal operation of the system under testing, in which case the fault-tolerance mechanisms might not be exercised by their occurrence. This may be caused, for example, by the occurrence of the fault in some part of the system that is currently unused, or by the practical inability to inject a fault that was previously selected for fault injection. However, the same fault may, during the system's operational lifetime, result in an error that will either be detected by the fault-tolerance mechanism or result in a system failure.

The problem is rather analogous to the "no-reply" response commonly encountered in opinion polls, which evidently affects the accuracy of the estimates [8]. The experimental process and the resulting estimations might be more or less affected by this problem, depending on the particular strategy employed for fault injection experiments. For example, if the number of faults to be injected is predetermined and the faults are all sampled before the fault injection campaign, simply discarding the no-response events would not be a reasonable strategy, because not enough meaningful results would have been obtained. Neither would be a reasonable strategy to simply assume that the fault is covered when it results in a no-response event simply because it did not cause the system to fail, as it would probably lead to optimistic estimates.

4.9. Statistical Modeling Techniques

This section discusses an array of statistical techniques which are applicable for use in fault coverage estimation from a set of fault injection experiments [18][19][26][27][28][29][31]. The various statistical models in this report are analyzed using, whenever possible, the same concepts and terminology presented here. The qualitative characteristics of each model will be summarized at the end of each section considering the following properties:

- **Estimation**: specifies the type of estimate produced by the model, according to the discussion in Section 4.6. Specifically it will be indicated if the model provides: a point estimator for fault coverage/non-coverage; confidence intervals for fault coverage/non-coverage (also the type of such intervals, and how such intervals are determined); an estimated distribution for the fault coverage/non-coverage (also how such distribution is determined).
- **Precision**: extending the discussion in Section 4.6.1, this attribute specifies the ability of the estimation process to provide estimates that fall, on the average, around the actual value of the system's fault coverage. For example, for point estimators, this attribute is quantified by the bias, calculated as the difference between the mean of the estimator and the estimated parameter, that is, the system's fault coverage/non-coverage; for confidence intervals determined using the pivotal transformation, the precision is quantified by the bias of the transformed estimator; for estimated distributions of the fault coverage/non-coverage, it is quantified by the mean of the estimated distribution.
- **Accuracy**: extending the discussion in Section 4.6.1, this attribute specifies the ability of the estimation process to provide estimates that, on the average, have a limited spreading of values. That is, a very accurate estimation process would lead to very similar estimates if, ideally, the experiment could be repeated numerous times. The accuracy is quantified, for point estimators or confidence intervals derived from point estimators, by the variance of the estimator, that is the mean square error of the estimator from its mean. It must be stressed that very accurate estimators can be useless if they are not precise, because the limited spreading of values would occur around an estimated quantity that is not equal to the system's fault coverage/non-coverage.
- **Assumptions**: will indicate the stated/unstated assumptions on which the model relies and will provide qualitative comments about the formal development of the statistical model in the referenced papers.
- **No-Response Problem Strategy**: will briefly summarize the strategy adopted in the referenced papers as a solution to the no-response problem, and describe how the whole estimation process is affected by the adopted strategy.
- **Required number of experiments**: provides information on how an *a priori* estimate of the necessary number of experiments can be calculated. It must be noted that often this problem is neglected in the referenced papers. Whenever possible, however, some viable options may nonetheless be discussed in the apposite section.

- **Applicability**: attempts to extrapolate, from all the previously discussed properties, the pros & cons of using the model to estimate fault coverage for specific types of systems, coverage requirements, etc. Typical cases when the model is used, or should be used, will also be indicated whenever possible.

The listed properties will be briefly summarized at the end of each section and finally summarized in a composite table, so as to provide an easily accessible reference to establish the optimal approach to statistical estimation of fault coverage in each specific case.

5. Simple Bernoullian Model

5.1. Introduction

This section describes the simplest model for fault coverage estimation, based on the standard statistical approach to the estimation of the mean of a Bernoulli population, which makes use of simple random sampling. The simple Bernoulli model provides the most generic approach to the problem of fault coverage estimation. As such, its applicability is extended throughout a very wide range of situations, although its results are generally inadequate.

Because of its simplicity and wide applicability, this statistical model is often the ready choice to designers of dependable systems. The cost and time of experimental setup and data analysis, which can be determining factors for designers of dependable systems, are drastically minimized by the application of this model.

Although this model is not completely described in any of the referenced papers, it provides, as it is, a very useful introduction to the classical statistical approach on which most of the models described in this report are based. The description of the simple Bernoulli model presented in this section is the result of various information collected from similar models described in [4][5][8][10][11][12].

5.2. Fault Space and Distribution

The fault space considered in a simple Bernoulli model cannot be described in any more detail than the analysis discussed in Section 4.2. In fact, this model applies to any possible configuration of the fault space, that is single or multi-dimensional, non-partitioned or partitioned, finite or infinite, discrete or continuous. However, some of the definitions provided in Section 4 would have to be slightly modified in order to account for infinite and/or continuous fault spaces, therefore, it will still be assumed that the fault space is finite and discrete, which is the approach commonly adopted.

The only condition that must be satisfied by the fault space is that it is possible to randomly sample a fault from it. This is necessary because the sampling process used in this model is the simple random sampling strategy, in which every fault is equally likely to be selected for injection in the system. In other words, using this model forces the uniform distribution on the system.

Known characteristics of the system, such as the failure rates, workloads, sensitivity to external agents, etc. of the internal components are completely ignored by the simple random sampling process, unless such parameters are considered as dimensions of the fault space.

This model is normally used when the fault distribution of the system is completely unknown. In general, the restrictive assumption of a uniform fault distribution is stated when this model is used. However, a uniform fault distribution is never found in real systems, although some systems might be characterized by fault occurrence probabilities that are very close to a constant value. Only in these cases, the uniform distribution might be a good approximation of reality. Otherwise, it will be shown that the uniform distribution is only sufficient for the validity of this model, but it is not necessary.

5.3. Point Estimator

This model provides a point estimator for the fault coverage, given by the sample average, that is the arithmetic average of the n observations of the random variable $\underline{Y}$:

$$\hat{C} = \frac{1}{n} \cdot \sum_{k=1}^{n} \underline{Y}_k \tag{5.1}$$

5.3.1. Precision of the Estimator

The mean of the estimator is calculated as:

$$E[\hat{C}] = E\left[\frac{1}{n} \cdot \sum_{k=1}^{n} \underline{Y}_k\right] = \frac{1}{n} \cdot \sum_{k=1}^{n} E[\underline{Y}] = \dot{C} \tag{5.2}$$

that is, the *coverage proportion*, which was defined in **Equation (4.22)**. The relationship of the coverage proportion with the coverage factor has been introduced in Section 4.4, and is repeated here for clarity:

$$C = \dot{C} + \rho \tag{5.3}$$

Therefore, the estimation provided by the sample average in this model is *optimistic* for $\rho < 0$, and it is *pessimistic* for $\rho > 0$. For values of the covariance close to zero, the estimator can be considered unbiased. This is always verified for a *fair* fault tolerance mechanism.

The assumption of an almost uniform fault distribution is then only a very special case of the condition $\rho \approx 0$, thus it is not necessary that faults are all equally likely to occur. This is true for any statistical models that is based on the arithmetic average of samples collected using a uniform sampling distribution.

5.3.2. Accuracy of the Estimator

The variance of the estimator is easily calculated as:

$$V[\hat{C}] = V\left[\frac{1}{n} \cdot \sum_{k=1}^{n} \underline{Y}_k\right] = \frac{1}{n^2} \cdot \sum_{k=1}^{n} V[\underline{Y}] = \frac{\dot{C}(1-\dot{C})}{n} = \frac{(C-\rho)(1-C+\rho)}{n}$$

$$= \frac{C(1-C)}{n} + \frac{\rho(2C-1-\rho)}{n} \tag{5.4}$$

and its unbiased estimator is:

$$\hat{S}[\hat{C}] = \frac{\hat{C}(1-\hat{C})}{n-1} \tag{5.5}$$

If the fault tolerance mechanism is fair, then $\rho \approx 0$ and the variance is the same that would be obtained if the variable Y were observed rather than $\underline{Y}$, i.e. if the coverage were calculated from data

collected during the operational lifetime of the system rather than the fault injection campaign. To determine whether the variance of the estimator is increased when $\rho \neq 0$ the following inequality is evaluated:

$$\rho(2C - 1 - \rho) > 0 \tag{5.6}$$

substituting using **Equation (5.3)**:

$$\rho(C + \dot{C} - 1) > 0 \tag{5.7}$$

which is typically verified for all the possible values of $\rho > 0$, since $C + \dot{C}$ will always be larger than 1 for typical values of the fault coverage and the coverage proportion. Therefore, the variance is increased or reduced, respectively, for positive or negative values of the covariance ρ. If this model is applied to a system with an unfair fault tolerance mechanism that favors the most likely faults, the variance of the estimator is larger than the one it would have observing the real system during its operational lifetime. On the other hand, if it is applied to a system with an unfair fault tolerance mechanism that favors the least likely faults, the variance of the estimator will be smaller than the one it would have observing the real system.

5.4. Confidence Intervals

Usually, confidence intervals for the fault coverage are defined using the technique of the pivotal transformation applied to the point estimator introduced in the previous section. From the definition of confidence intervals:

$$P(C \in [\hat{C}_1 : \hat{C}_2]) = \gamma \tag{5.8}$$

and using the pivotal quantity:

$$\frac{\hat{C} - E[\hat{C}]}{V[\hat{C}]} \tag{5.9}$$

it can be observed that the estimator is distributed like a binomial random variable, which can be approximated by the normal distribution with mean $E[\hat{C}]$ and variance $V[\hat{C}]$, so that the pivotal quantity in **Equation (5.9)** has the standard normal distribution. The opportunity of using the normal approximation will be discussed in detail in Section 10.

The symmetric double-sided $100\gamma\%$ confidence interval is then given by:

$$(\hat{C} - K_\gamma \sqrt{\hat{S}[\hat{C}]}) < \dot{C} < (\hat{C} + K_\gamma \sqrt{\hat{S}[\hat{C}]}) \tag{5.10}$$

where K_γ indicates the $[100(1+\gamma)/2]^{th}$ standard normal percentile, and the estimated variance has been used rather than the unknown variance.

When the variance is unknown, the t-Student distribution should be considered rather than the normal distribution. However, for typical sample sizes of fault injection experiments, with $n \gg 100$, the two distributions are equivalent. The single sided $100\gamma\%$ confidence interval is instead given by:

$$\dot{C} > \hat{C} - z_\gamma \sqrt{\hat{S}[\hat{C}]} \tag{5.11}$$

where z_γ indicates the $100\gamma^{th}$ standard normal percentile.

It must be noted that the confidence intervals have been defined for the mean of the estimator, i.e. the coverage proportion $\dot{C}$, which is different from the real coverage factor C if the fault tolerance mechanism is *unfair*. Therefore, the indicated confidence intervals should only be used when it can be assessed that $\rho \approx 0$.

5.5. Number of Experiments

An order of magnitude of the number of experiments necessary to achieve a reasonable estimation of the coverage factor can be calculated by imposing a constraint on the variance of the estimator, assuming that the coverage factor is approximately equal to the coverage proportion, that is in the case of a fair fault tolerance mechanism:

$$V[\hat{C}] < v \tag{5.12}$$

which leads to the estimate:

$$n > \frac{C(1-C)}{v} \tag{5.13}$$

Otherwise, the constraint can be applied on the standard deviation in terms of the desired non-coverage:

$$\sigma < 100x\%(1\text{-}C) \tag{5.14}$$

which leads to:

$$n > \frac{C}{x^2(1-C)} \tag{5.15}$$

For example, if it is desired that the standard deviation of the estimate is within 10% of the non-coverage, with a required coverage of 0.999, the number of experiments needed is at least 99,900.

In practice [2], the number of experiments required is more conservatively chosen in the order of magnitude of 10^x if 10^{-x} is the desired non-coverage. In Section 10 it will be discussed that the number of experiments must be further increased in order to obtain meaningful confidence intervals using the normal approximation. In fact, for a confidence level of about 95%, 10^x experiments are generally more than sufficient. However, for a confidence level of 99%, the number of experiments required must be increased by about two orders of magnitude.

5.6. No-Response Problem

The no-response problem can be solved in different ways, depending on practical considerations about the experimental process. For example, faults that result in no-response can be assumed to be either all covered or uncovered to obtain, respectively, optimistic and pessimistic estimates. If a pessimistic estimate is still lower than the required coverage, it can be used as the final estimate of the fault coverage. Otherwise, faults that result in no-response can be simply discarded, leading to a more meaningful estimate of the fault coverage. However, in this case it must be kept into account that the actual number of experiments performed is lower than the sample size n, and more samples should be selected from the fault space unless the no-response injections are much less than the sample size.

The main problem with this approach is that for each no-response fault that is injected into the system, at least another fault must be injected (more than one fault will have to be injected if newly sampled faults result in no-response as well), which is a very costly operation if many no-response faults are found. Clearly, if faults are labeled as "no-response" because it is not possible to inject them, the problem of the added cost does not subsist, provided that the sampling process itself has a much lower cost than the injection experiments.

5.7. Conclusions

The model discussed in this section suffers from several limitations, yet it is usually chosen by designers of dependable systems as the most appealing approach to fault coverage estimation, either in the form that has been presented here or in some variations that will be discussed in the next three sections. The reason for this choice must be searched for in the simplicity and familiarity of the theoretical and practical development of the fault injection campaign resulting from the application of this model.

This approach is typical of estimation problems that attempt to estimate the distribution parameter of a Bernoulli population, and, as such, it is very well documented in the literature and widely applied to a range of engineering problems. However, it was discussed in Section 4 how fault coverage estimation is a fairly unique problem in its kind. The two main peculiarities are represented by the significance of the fault distribution and the typical range of values in which the fault coverage lies.

The estimation process described in this section does not make use of any information about the fault distribution. When this or similar models are discussed, typically the sources from the literature resort to the simplistic assumption that all faults are equally likely to occur, such as in [2][3][4][8], or the problem is simply ignored (that is, the same unstated assumption is made), such as in [5]. It has been shown in Section 5.3 that this kind of assumptions leads to biased estimates of the fault coverage (or the non-coverage, depending on the specific approach used in the references). The bias of the estimator is quantified by the covariance ρ, representing the unfairness of the fault tolerant mechanism. Only when the fault tolerant mechanism can be considered a fair one, i.e. $\rho \approx 0$, this assumption can lead to precise estimates (according to the definition provided in Section 4.9) of the fault coverage. Therefore, the assumption of a uniform fault distribution is only sufficient, but not necessary, for the unbiasedness of the

estimator proposed in this model. In general, all the models that resort to the assumption of a uniform fault distribution would provide equally precise estimates if only the condition $\rho \approx 0$ is verified.

Whether the fairness condition is usually verified in real systems or not is an open research problem, as no such study has been found in the literature. From an engineering perspective it could be argued that in the design of dependable systems, most of the effort is devoted to achieving high coverage for the most likely faults to occur. For example, a fault tolerance mechanism that implements error-detecting/error-correcting codes for a communication channel is generally designed such that only one or few bits can be recovered in a fixed-length word, because those are the errors that are most often observed. Clearly, it can generally be argued that the most likely faults are the ones that have been historically more often observed and studied, thus also the ones for which ready solutions are available.

From such considerations it could be concluded that in real systems ρ is greater than zero, thus the model presented in this section would provide pessimistic estimates of the fault coverage. Such a conservative result is often desirable, especially for critical applications in which safety is of major concern.

However, the conclusion that real systems have a positive covariance has only been reached from practical considerations, and is not to be interpreted as a statement of general validity. It would be interesting to carry out a detailed study of the covariance in systems for which historical data are available, so as to assess the applicability of the model presented in this section (and the other models that make use of the strict assumption of a uniform fault distribution) and to quantify the degree of conservativeness of the estimates provided by such models.

The other limitation of this model regards the applicability of the pivotal transformation used to determine the confidence intervals in practical fault injection experiments. It is common practice [14] in statistical estimation of the distribution parameter of a Bernoulli random variable to assume that the pivotal quantity of **Equation (5.9)** is distributed as a standard Gaussian. This assumption derives in general from the result provided by the Central Limit Theorem, but in the specific case it is better understood considering that the estimator of **Equation (5.1)** is distributed as a binomial random variable, which a discrete distribution that can be approximated by the normal distribution with the same mean and variance.

The appropriateness of such approximation has been discussed several times in the literature, and the general conclusion is that a large number of experiments and a relatively low value of the system's coverage are necessary conditions for the approximation to be valid [2][10]. However, often times such constraints are ignored or not interpreted correctly, which can result in unreliable estimates of the fault coverage. A more detailed discussion about the use of the normal approximation is provided in Section 10, where the numerical data provided in [11] are compared to the results obtained using the binomial distribution.

It must be stressed that since the pivotal quantity is obtained from the biased estimator of **Equation (5.1)**, the confidence intervals will always be imprecise. That is, even if an infinite number of experiments were performed, the confidence intervals would approach arbitrarily closely the value of the coverage proportion, rather than the fault coverage.

On the positive side, this model does provide high accuracy, as it is well known that the sample average is the UMVUE (uniformly minimum variance unbiased estimator) for the mean of a Bernoulli population [14]. Clearly, the estimates will always be concentrated, however accurately, around the value of the coverage proportion rather than the fault coverage, because the estimator is biased, thus imprecise.

As for the determination of the number of experiments required, a practical formula was provided in Section 5.5, which was obtained by imposing a constraint on the variance of the estimator, given a desired coverage level. This formula was not found in the referenced publications, however, for typical fault injection campaigns it is consistent with the common practice [2] of performing at least 10^x experiments if the required non-coverage level is 10^{-x}.

It must be noted that in Section 10 it is shown from the numerical data provided in [11] that the number of experiments must be further increased, according to the confidence level required, so that the normal approximation is valid.

Finally, no optimal solution could be determined to solve the problem of no-response experiments. Therefore, any of the approaches described in Section 5.6 can be used according to practical considerations made in each specific case.

These conclusions are summarized in **Table 5.1.**

Estimation	Point estimator for fault coverage. Single- and double- sided confidence intervals for the fault coverage determined with the normal approximation.
Precision	Biased by ρ (centered around the coverage proportion). Optimistic for $\rho < 0$, conservative for $\rho > 0$. Unbiased if the fault tolerance mechanism is fair.
Accuracy	Minimum variance if the fault tolerance mechanism is fair. For $\rho \neq 0$, minimum variance around the coverage proportion.
Assumptions	The uniform distribution is always indicated as the requirement for the unbiasedness of the estimator, whereas the fairness condition is necessary and sufficient. The validity of the normal approximation is only verified for low values of the coverage or for a high number of experiments.
No-Response Problem	This model does not define a specific solution to the no-response problem. Specific considerations must be made in each case.
Number of Experiments	Generally (confidence level around 95%) in the order of 10^x , if 10^{-x} is the desired coverage level. Higher by one-two orders of magnitude for confidence level around 99%.
Applicability	This model should only be employed when no information is available about the fault distribution, and the uniform distribution is the only possible assumption. It must also be possible to perform a large number of experiments in order to use the normal approximation.

Table 5.1 Properties of the Simple Bernoulli Model

6. Stratified Bernoullian Model

6.1. Introduction

This section introduces the concept of stratified estimation of fault coverage. Stratification consists in creating an ad hoc partition of the fault space and estimating the fault coverage pertinent to each of the classes in the partition, and then combining the estimated class fault coverage factors to obtain an estimate of the overall system's coverage.

Stratification has several practical advantages [11]: if the partition of the fault space is determined so that faults in the same class all occur at a certain location, e.g. within the same component or on the same equipotential line, physical fault injection can be simplified because it is not necessary to re-apply the probes at each experiment; also, the data obtained in a fault injection campaign to calculate a coverage estimate for single components can be reused in other systems where the same components are employed; finally, if quantitative information about the occurrence of the faults in different classes is available, the system's coverage estimate can be determined much more precisely than using the simple Bernoulli model.

The model presented in this section is mostly based on the stratified sampling strategy presented in [8][10][11].

6.2. Fault Space and Distribution

In this model, the assumption on the fault space is that it can be partitioned into M disjoint subsets, also called *classes*:

$$\mathfrak{I} = \bigcup_{m=1}^{M} \mathfrak{I}_m \quad \text{and} \quad \forall(i,j) \quad \text{with} \quad i \neq j \quad \mathfrak{I}_i \cap \mathfrak{I}_j = \varnothing \tag{6.1}$$

The probability of occurrence of a fault can then be rewritten as:

$$P(F=f) = P[(F=f)|(F \in \mathfrak{I}_i)]P(F \in \mathfrak{I}_i) \tag{6.2}$$

where the theorem of the total probability has been used together with the obvious relation:

$$P[(F=f)|(F \in \mathfrak{I}_i)] = 0 \quad \forall(f \notin \mathfrak{I}_i) \tag{6.3}$$

The quantity

$$P(F \in \mathfrak{I}_i) \tag{6.4}$$

expresses the relative probability that a fault that has occurred in the system belongs to class $\mathfrak{I}_i$. Ideally, this quantity could be calculated as the limit of the ratio between the number of faults belonging to the class $\mathfrak{I}_i$ that have occurred in a system over the total number of faults that have occurred after an infinite observation time. Clearly, it is impossible to calculate the exact value of such probabilities, and more practical considerations must be made. In Section 4.3.1 several approaches to determine an approximate class-wise uniform fault distribution were described, which apply to his case as well. In fact, the

$P(F \in \mathfrak{I}_i)$ are the identical to the *weighting factors* that were there introduced. If the probabilities in **Equation (6.4)** cannot be determined, and it can always be assumed that:

$$P(F \in \mathfrak{I}_i) = \frac{1}{M} \tag{6.5}$$

or that:

$$P(F \in \mathfrak{I}_i) = \frac{|\mathfrak{I}_i|}{|\mathfrak{I}|} \tag{6.6}$$

The condition expressed in **Equation (6.5)** is a stronger condition than **Equation (6.6)**: the latter is obtained when all faults are equally likely to occur (uniform distribution), whereas **Equation (6.5)** implies that the cumulative probability of all faults belonging to a certain class is constant for all classes. The two statements are significantly different: as a clarifying example, consider a fault space with ten faults uniformly distributed and partitioned in only two classes, the first containing only one fault, the second containing nine faults; using **Equation (6.5)** the relative probabilities of both classes would be 0.5, whereas using **Equation (6.6)** one would obtain 0.1 and 0.9, respectively. Clearly, it is impossible to state which of the two assumptions should be made in absence of quantitative information on the fault distribution. However, the same practical considerations that usually lead designers of dependable systems to assume a uniform distribution of the fault space would suggest that **Equation (6.6)** should always be preferred to **Equation (6.5)**.

6.3. Definition of Fault Coverage

Following the development presented in [8], the definition of fault coverage can be modified, using **Equation (4.15)** and **Equation (6.2)**, as follows:

$$C = \sum_{f \in \mathfrak{I}} y(f)P(F=f) = \sum_{f \in \mathfrak{I}} y(f)P[(F=f)|(F \in \mathfrak{I}_i)]P(F \in \mathfrak{I}_i)$$

$$= \sum_{i=1}^{M} \sum_{f \in \mathfrak{I}_i} y(f)P[(F=f)|(F \in \mathfrak{I}_i)]P(F \in \mathfrak{I}_i) = \sum_{i=1}^{M} P(F \in \mathfrak{I}_i) \sum_{f \in \mathfrak{I}_i} y(f)P[(F=f)|(F \in \mathfrak{I}_i)]$$

$$= \sum_{i=1}^{M} P(F \in \mathfrak{I}_i)C_i \tag{6.7}$$

where C_i is called the *class fault coverage* of $\mathfrak{I}_i$, that is:

$$C_i = \sum_{f \in \mathfrak{I}_i} y(f)P[(F=f)|(F \in \mathfrak{I}_i)] \tag{6.8}$$

Equation (6.7) allows expressing the coverage factor as a function of the individual coverage factors of each of the classes in the partitioned fault space. This property will later be used to determine an estimator that combines the class coverage factors into an estimator for the system's fault coverage.

6.4. Sampling

The stratified sampling strategy requires the selection of a predetermined number n_i of samples from each class $\mathfrak{I}_i$ of the partitioned fault space.

If the values of the probabilities of **Equation (6.4)** are known, the number of faults sampled from each class is chosen to be:

$$n_i = P(F \in \mathfrak{I}_i) \cdot n \tag{6.9}$$

for reasons that will be clarified later. Clearly, the n_i so determined must be rounded to the nearest integer. This sampling strategy is referred to as *stratified sampling with representative allocation* [8].

If the values of the probabilities of **Equation (6.4)** are not known, different strategies can be employed. Usually similar considerations to those discussed in Section 6.2 are made to calculate the sample sizes. Substituting expressions **Equation (6.5)** and **Equation (6.6)** into **Equation (6.9)**, the following two relations are obtained, respectively:

$$n_i = \frac{n}{M} \tag{6.10}$$

called *homogeneous allocation* [8], or:

$$n_i = n \cdot \frac{|\mathfrak{I}_i|}{|\mathfrak{I}|} \tag{6.11}$$

which will be here termed *uniform allocation.*

In general, given a certain assumption on the value of the weighting factors of **Equation (6.4)**, the size of the sample extracted from the corresponding classes are obtained by substituting the assumed value into **Equation (6.9)**.

Within each class, however, the samples are selected using simple random sampling, that is the sampling distribution within each class is given by:

$$P[(\underline{F} = f) | (\underline{F} \in \mathfrak{I}_i)] = \frac{1}{|\mathfrak{I}_i|} \tag{6.12}$$

and the overall sampling distribution can be expressed as in **Equation (4.5)**:

$$P(\underline{F} = f) = \sum_{i=1}^{M} \frac{\delta_i(f)}{|\mathfrak{I}_i|} P(\underline{F} \in \mathfrak{I}_i) \tag{6.13}$$

Therefore, stratified sampling forces a class-wise uniform distribution on the fault space.

6.5. Point Estimator

With the definition of fault coverage given in **Equation (6.7)** and given that samples are extracted using a simple random sampling strategy within each class, the estimation of the class fault coverage of class $\Im_i$ can be performed as it has been described in Section 5, using the estimator:

$$\hat{C}_i = \frac{1}{n_i} \sum_{k=1}^{n_i} Y_k \tag{6.14}$$

that is, the sample average of the observations extracted from class $\Im_i$, with mean:

$$E[\hat{C}_i] = \dot{C}_i \tag{6.15}$$

that is, the *class coverage proportion*. This estimator is completely equivalent to the one introduced in Section 5 to estimate the system's fault coverage. The same considerations discussed in the previous section apply, therefore, unchanged to the estimation of the class coverage factors. However, it must be kept into account that the size of each class will generally be much smaller than the size of the fault space, thus a proportionally lower number of samples must be selected from each class to obtain an accurate estimate. The precision of the estimator of **Equation (6.14)** will still be affected by the fairness of the fault tolerant mechanism within each class, but it will be discussed how the combined estimate of the system's fault coverage can be determined more precisely in this case.

Also, it must be pointed out that in order to calculate the estimator as in **Equation (6.14)**, the sample sizes n_i cannot be zero, otherwise $\hat{C}_i$ is undetermined.

From the class fault coverage estimates, the estimate of the system's fault coverage can be calculated using the estimator:

$$\hat{C} = \sum_{i=1}^{M} \hat{C}_i P(F \in \Im_i) \tag{6.16}$$

which is the weighted average of the estimated coverage factors of all the classes in the partitioned fault space. Notice that substituting **Equation (6.7)** and **Equation (6.9)** into **Equation (6.16)**, the following is verified:

$$\hat{C} = \sum_{i=1}^{M} \frac{1}{n_i} \sum_{k=1}^{n_i} Y_k = \frac{1}{n} \sum_{i=1}^{n} Y_i \tag{6.17}$$

that is, the estimator can still be expressed as the arithmetic average of all the samples collected. Although similar properties can be expected for this estimator to the one described in the previous section, it must be noted that the sampling strategy adopted in this case is not simple random sampling.

Therefore, different characteristics should be expected, both for what concerns the precision of the estimator and for what concerns its accuracy.

6.5.1. Precision of the Estimator

Analogously to what was done in Section 4.4, we can define the random variable $\underline{Y}_i$, describing the fault tolerance event for faults occurred in class $\mathfrak{I}_i$, and the random variable P_i as a function of a fault from class $\mathfrak{I}_i$ that has occurred in the system with probability $P[(\underline{F}=f)|(\underline{F}\in\mathfrak{I}_i)]$:

$$P_i = p_i(f) = \frac{P[(F=f)|(F\in\mathfrak{I}_i)]}{P[(\underline{F}=f)|(\underline{F}\in\mathfrak{I}_i)]} \tag{6.18}$$

We can then define the class covariance ρ_i between the variables $\underline{Y}_i$ and P_i:

$$\rho_i = \sum_{f\in\mathfrak{I}_i}[(\underline{Y}_i-\underline{C}_i)(P_i-1)] = C_i-\underline{C}_i \tag{6.19}$$

It is easy to show that:

$$\rho = \sum_{i=1}^{M}\rho_i P(F\in\mathfrak{I}_i) \tag{6.20}$$

$$\underline{C} = \sum_{i=1}^{M}\underline{C}_i P(F\in\mathfrak{I}_i) \tag{6.21}$$

The mean of estimator in **Equation (6.16)** is finally calculated as:

$$E[\hat{C}] = E\left[\sum_{i=1}^{M}\hat{C}_i P(F\in\mathfrak{I}_i)\right] = \sum_{i=1}^{M}E[\hat{C}_i]P(F\in\mathfrak{I}_i) = \sum_{i=1}^{M}\underline{C}_i P(F\in\mathfrak{I}_i)$$

$$= \sum_{i=1}^{M}(C_i-\rho_i)P(F\in\mathfrak{I}_i) = C-\rho = \underline{C} \tag{6.22}$$

which shows that the estimate is still biased by the quantity ρ.

However, the value assumed by the covariance ρ is different, in general, from the covariance that would be obtained using the simple Bernoulli model of Section 5. In fact, in this case the forced distribution, that is, the sampling distribution, is not uniform, but class-wise uniform, with weighting factors depending on the particular sampling strategy chosen, as discussed in Section 6.2 and Section 6.4.

Stating the fairness condition of the fault tolerance mechanism with respect to the fault and forced distributions can be just as impractical as it is in the simple Bernoulli model described in the previous section.

Generally, when using a stratified approach it is assumed that faults within a single class are uniformly distributed. Therefore, in general, the partition of the fault space should be defined so that faults lying within the same class have an approximately constant probability of occurrence. However,

making the same considerations as in Section 5, only the fairness of the fault tolerance mechanism within each class is required in order to obtain an unbiased estimate of the fault coverage. That is, an unbiased estimate is obtained when the variable Y is uncorrelated with the relative probability of occurrence of faults distributed as in the fault distribution and the sampling distribution.

Obviously, this condition is verified if the classes are designed so that faults within each class occur approximately all with equal probability (class-wise uniform distribution), because in this case the variable P_i, defined in **Equation (6.18)**, always assumes values very close to unity. By simple inspection of **Equation (6.19)**, it is clear that $\rho_i \approx 0$, and from **Equation (6.20)** it is also verified that $\rho \approx 0$.

However, the values of ρ_i can be almost zero even if the distribution is not uniform within each class, as was discussed in Section 5. Achieving the fairness condition is easier in this case because many considerations can be made so as to approximate the fault distribution using the sampling distribution, which were discussed in Section 4.3.1.

6.5.2. Accuracy of the Estimator

The variance of the estimator is derived as a linear combination of the variances of the class coverage estimators $\hat{C}_i$ using the definition of class coverage proportion, the result obtained in **Equation (5.4)**, and **Equation (6.9)**:

$$V[\hat{C}] = \sum_{i=1}^{M} [P(F \in \mathfrak{I}_i)]^2 V[\hat{C}_i] = \sum_{i=1}^{M} [P(F \in \mathfrak{I}_i)]^2 \left(\frac{\dot{C}_i(1-\dot{C}_i)}{n_i}\right)$$

$$= \sum_{i=1}^{M} \frac{P(F \in \mathfrak{I}_i)\dot{C}_i(1-\dot{C}_i)}{n} \tag{6.23}$$

which can be estimated using [8]:

$$\hat{S}(\hat{C}) = \sum_{i=1}^{M} \frac{[P(F \in \mathfrak{I}_i)]^2 \hat{C}_i(1-\hat{C}_i)}{n_i - 1} \tag{6.24}$$

The variance depends on the pre-determined number n_i of faults sampled from each class, and can be minimized by choosing an appropriate allocation strategy. It is stated in [8] that using the Lagrange multiplier method it can be proven that the variance is minimum if the sample sizes are chosen so that:

$$n_i = n \left(\frac{P(f \in \mathfrak{I}_i)\sqrt{\dot{C}_i(1-\dot{C}_i^2)}}{\sum_{k=1}^{M} P(f \in \mathfrak{I}_k)\sqrt{\dot{C}_k(1-\dot{C}_k^2)}} \right) \tag{6.25}$$

which, in absence of any *a priori* knowledge of the fault coverage, must be approximated with **Equation (6.9)**, the representative allocation, to obtain:

$$V[\hat{\dot{C}}] = \frac{\dot{C}}{n - \sum_{i=1}^{M} \frac{P(F \in \Im_i)\dot{C}_i^2}{n}} \tag{6.26}$$

Compared to the results obtained in **Equation (5.4)**, the variance of this estimator is reduced if:

$$\sum_{i=1}^{M} P(F \in \Im_i)\dot{C}_i^2 > \dot{C}^2 \tag{6.27}$$

which is not always verified. One case when **Equation (6.27)** is valid for any values of the class fault coverage factors is when $P(F \in \Im_i) = 1/M$, which is, as discussed in Section 6.2, a quite strong requirement. Obviously, it is only possible to verify **Equation (6.27)** for each specific system. In Section 10, numerical results from simulated fault injection campaigns provided in [11] will be analyzed to reach the conclusion that the variance of the estimator obtained with this model is almost identical to the variance of the simple Bernoulli model and the generalized Bernoulli model (which is presented in Section 9).

6.6. Confidence Intervals

Using the Central Limit Theorem, double- and single-sided confidence intervals can be defined exactly as in **Equation (5.10)** and in **Equation (5.11)**, using the mean and the variance estimator calculated in **Equation (6.22)** and **Equation (6.24)**, respectively.

The imprecision of the confidence intervals is still a problem in the stratified case, however, using the same considerations made about the estimator, it is generally possible to minimize the bias by choosing an appropriate partition of the fault space.

Since the use of the Central Limit Theorem implies approximating the binomial distribution of the estimator in **Equation (6.17)** with the normal distribution, the same considerations made in Section 5 apply in this case. In Section 10, the validity of the approximation will be discussed in more detail, and a numerical method to calculate confidence intervals for the binomial distribution in the stratified case will be described, although the results are not encouraging.

6.7. Number of Experiments

The number of experiments necessary to estimate the fault coverage with the stratified model can be calculated applying the constraint on the variance for each of the classes in the partitioned fault space as it was shown in Section 5.

The order of magnitude of the required number of experiments is still the same as discussed in Section 5. Therefore, generally it is assumed that 10^x experiments are sufficient if 10^{-x} is the desired non-coverage. This number must be further increased according to the coverage level desired and the confidence degree imposed to determine the confidence intervals. This will be discussed in Section 10 in more detail.

6.8. No-Response Problem

A technique called a posteriori stratification is proposed in [8] as a solution to the no-response problem, which can only be applied in the case of physical fault injection and when very detailed information about the system is available, that is, quoting from [8], only when "the detailed wiring diagram of the target system is available".

From this statement, it appears that the proposed method lacks of generality, and is therefore beyond the scope of this report. Therefore, only similar considerations to those made in Section 5 can be made, in the general case, as a solution to the no-response problem.

In stratified sampling, however, special attention must be posed to the fact that the sample sizes n_i are pre-determined according to the fault distribution, therefore the estimator, both in the form of **Equation (6.16)** and **Equation (6.17)**, must be modified according to the specific strategy used.

If no-response data are simply discarded, undetermined situations can be created, especially if all faults in one class result in no-response (for example, if the class corresponds to faults occurring in a non-testable component). In such a case, the estimator of the class coverage factor of **Equation (6.14)** is meaningless, since both the numerator and denominator are zero.

6.9. Conclusions

The model discussed in this section, like the simple Bernoulli model described in Section 5, is very commonly used in the assessment of the fault coverage of dependable systems, because of the practical advantages that it provides [11].

It was also demonstrated that if quantitative information about the system is available, which allows determining an ad hoc partition of the fault space, the precision of the estimator provided by the stratified model is greatly improved compared to the simple Bernoulli model.

Otherwise, the stratified approach can only be motivated by practical considerations, because if no information about the fault distribution is available, this model is equivalent, if not worse, to the simple Bernoulli model.

In fact, in absence of information about the fault distribution the sampling strategy can only be chosen as in **Equation (6.10)** and **Equation (6.11)**. The former represents a very strict condition that does not appear to be justified in real systems and does not lead to any advantage in precision or accuracy. The latter is represents the uniform distribution assumption, which was discussed in Section 5.

Another problem with this model, which occurs only when a relatively small number of experiments can be performed or when some class has very low probability of occurrence, is that since the sample sizes of each class are pre-determined using **Equation (6.9)**, these can assume the value zero, in which case the estimator for the class fault-coverage of **Equation (6.14)** would be undetermined. Obviously, one could decide to impose that $n_i \geq 1$, but this would necessarily affect the precision and accuracy of the estimate. It must be noted, however, that this problem generally does not subsist, since the sample sizes are usually much greater than 1.

From a theoretical point of view this result is still very interesting. Considering an ideal case in which the number of classes is comparable to the cardinality of the fault space, it is natural to believe that the stratified approach would lead to a very precise and accurate estimate of the fault coverage, provided that the weighting factors of the classes are known. However, the model presented in this section does not allow such an ideal case, as it would lead to undetermined situations. This problem will be solved in Section 9, using a very simple, yet powerful device.

The precision of the estimator is still affected by the value of the covariance ρ. However, it was discussed in Section 6.5 that if some information about the fault distribution is available, the sampling distribution can be chosen so as to reduce the value of the covariance, thus increasing the precision of the estimate.

It must be noted that in [8][10][11] it is stated that the estimator is unbiased. This statement can be misleading, as it is based on the implied assumption that the real fault distribution of the system is identical to a class-wise uniform distribution. This is clearly never the case, although a class-wise uniform distribution can very well approximate the real fault distribution. Here an even looser condition than the class-wise uniform distribution has been stated for the unbiasedness of the estimator, i.e. the fairness of the fault tolerance mechanism within each class, or, equivalently, the fairness of the fault tolerance mechanism with respect to the real fault distribution and the forced class-wise uniform distribution.

As for the validity of the fairness condition in real systems, the same considerations made in the previous sections apply here. Apart from engineering considerations that apply to specific cases, further research on this matter is necessary.

The variance of the estimator varies with the sampling strategy adopted and the particular system under analysis. However, it will be concluded in Section 10, from numerical data, that the accuracy of the stratified model is very close to the accuracy of the simple Bernoulli model, which is very close to the minimum variance achievable.

The number of experiments required has been discussed to be in the same order of magnitude as for the simple Bernoulli model in Section 6.7.

Finally, no optimal solution could be determined to solve the problem of no-response experiments. Special attention must be devoted to how the chosen strategy affects the estimation process, because the sample sizes for each class are pre-determined using **Equation (6.9)**. Undetermined situations can be created if all faults in a certain class result in no-response.

These conclusions are summarized in **Table 6.1.**

Estimation	Point estimator for fault coverage. Single- and double- sided confidence intervals for the fault coverage determined with the normal approximation.
Precision	Biased by ρ (centered around the forced coverage). Unbiased if the fault tolerance mechanism is fair with respect to the fault and forced distributions. ρ generally smaller than in the simple model.
Accuracy	Not necessarily minimum variance, but very close to the variance of the simple model.
Assumptions	The class-wise uniform fault distribution is generally assumed as the requirement for the unbiasedness of the estimator, whereas the fairness condition is necessary and sufficient. The validity of the normal approximation is only verified for low values of the coverage or for a high number of experiments.
No-Response Problem	This model does not define a specific solution to the no-response problem. Specific considerations must be made in each case. Problems of indeterminacy can occur.
Number of Experiments	Generally (confidence level around 95%) in the order of 10^{x}, if 10^{-x} is the desired coverage level. Higher by one-two orders of magnitude for confidence level around 99%. Same as the simple model.
Applicability	This model can be employed when practical considerations demand a stratified approach or when some information about the fault distribution is available. It must also be possible to perform a large number of experiments in order to use the normal approximation.

Table 6.1 Properties of the Stratified Bernoulli Model

7. Multi-Stage Stratified Sampling

7.1. Introduction

The model described in this section, which employs multi-stage sampling with stratification, is based on the work presented in [5]. The main limitation of the model, not stated but implied in the referenced paper, is that it assumes a uniform fault distribution, and as such, it does not improve on the precision or the accuracy of the estimator. However, for systems of unknown fault distribution, for which only the uniform distribution can be assumed, it provides useful recursive expressions for the coverage estimator from samples extracted from the fault space using multi-stage and combined multi-stage and stratified sampling. The other limitation, indicated by the author of [5], is the approximation of the estimator's distribution to the normal distribution.

7.2. Fault Space

The fault space considered in this model is a finite multi-dimensional fault space. The attributes that characterize a fault are the coordinates that identify a fault in the space. Such attributes can be location, type, instant of occurrence, duration, workload, input profile, etc. Furthermore, the fault space can also be partitioned into classes in order to cope with the large size of the fault space. Some practical considerations are exposed in [5] that usually lead to only consider three-dimensional spaces in fault injection experiments, however, the theoretical development is valid for a generic number of dimensions.

7.3. Fault Distribution and Definition of Fault Coverage

The basic assumption of this model is that the fault distribution is uniform, that is all faults are equally likely to occur. Therefore, the probabilities associated to the fault classes are determined as the ratio between the elements in each class and the total number of faults. As discussed in the previous sections, this assumption leads to considering the coverage factor identical to the coverage proportion, which is not correct by definition of fault coverage, and is usually not verified in real systems.

7.4. Sampling

The sampling process considered in this model consists of successive selection along each dimension of the fault space, of the fault attributes belonging to the sample. The selection of the attributes for the sampled fault is done using a simple random sampling strategy in each stage, i.e. along each dimension. The sampling process will, for example, first determine a sub-population of faults corresponding to a certain set of locations, then determine a subset of the sub-population that has a certain set of temporal characteristics, etc. The sample consists therefore of all the faults belonging to the Cartesian product of the subsets sampled along the various dimensions.

When stratification is also used, each class of the fault space is considered separately, and a multi-stage sampling campaign is carried out within each class. Stratification is used only as a device to simplify the sampling process, but given the assumed uniform fault distribution, it does not affect the resulting estimate of the system's fault coverage.

7.5. Terminology

In the generic situation of a k-dimensional case, let $\mathbf{D}_i$ indicate the i-th dimension, and let $\vec{D}_i$ indicate the set of possible values along dimension $\mathbf{D}_i$. We can then define the vector:

$$\vec{D} = [\vec{D}_1, \vec{D}_2, \ldots, \vec{D}_k] = \Im \tag{7.1}$$

which will coincide with the fault space $\Im$. Therefore a generic fault $f \in \Im$ will be identified by a vector $\vec{d} = [\vec{d}_1, \vec{d}_2, \ldots, \vec{d}_k]$ where $\vec{d}_i \in \vec{D}_i$ and the fault will be indicated as $f_{\vec{d}}$. The sampling process as previously described determines the sets D_i:

$$D_i \subseteq \vec{D}_i \tag{7.2}$$

of possible values along dimension $\mathbf{D}_i$. The sampled faults will be identified by a vector of attributes $d = [d_1, d_2, \ldots, d_k]$, where $d_i \in D_i$ and the sampled faults can be indicated as f_d. Finally, the sample sizes along each direction will be indicated as s_i. The observations of the injected faults are indicated as $Y_d = y(f_d)$.

7.6. Estimator

The coverage of the system is estimated from the sample using the estimator:

$$\hat{C} = \frac{\sum_{d \in D} y(f_d)}{\prod_{i=1}^{k} s_i} \tag{7.3}$$

It is easy to show that this estimator is a very simple generalization of the estimator introduced in Section 5 to the case of multi-stage random sampling. The estimator is still the sample average of the results of the fault injection experiments. Therefore, it is unbiased if the fault tolerance mechanism is fair. All the other considerations made in Section 5 about the precision and the accuracy of the estimator apply unchanged to this case. A recursive expression of the variance (and of an unbiased estimator of the variance) of estimator of **Equation (7.3)** is provided in [5], but they are not reported here because of their complexity.

If stratification is used, the coverage for each class is determined using **Equation (7.3)**, and then a weighted average of the classes' coverage is computed to estimate the system coverage. The weighting factors are given by the relative sizes of the classes in which the fault space has been partitioned.

7.7. Confidence Intervals

The model proposed in [5] defines double-sided confidence intervals using the normal approximation. Obviously, single sided confidence intervals could be defined as well. The same considerations exposed in the previous sections about confidence intervals determined with the pivotal transformation of the biased estimator of **Equation (7.3)** and the normal approximation apply to this model as well.

7.8. Number of Experiments

Since the model described here is equivalent for the most part to the simple Bernoulli model, the same number of experiments is required as discussed in Section 5.5.

7.9. No-Response Problem

The no-response problem is simply faced in [5] by considering two confidence intervals, one in the optimistic assumption that all no-response faults are covered, the other in the pessimistic assumption that all no-response faults are not covered. Clearly, this is the only practically viable procedure in this model because of the peculiar sampling strategy employed. It would be too complex to repeat the sampling process to collect the missing information, and the recursive expressions would not apply.

7.10. Conclusions

The model presented in this section has very similar characteristics and the same limitations as the simple Bernoulli model presented in Section 5. However, it provides recursive formulas to estimate coverage when the sampling strategy used is multi-staged or stratified and multi-stage, and to calculate the variance of the estimator and the variance's unbiased estimate. The model was not discussed in deep detail because it does not add much to the theory and the quality of the results that were obtained in Section 5.

These comments are not intended to diminish the value of the referenced article [5], since it discusses in much detail and with very adequate formalism many of the concepts that have been used in Section 4 and Section 5.

The properties of this model are summarized in **Table 7.1.**

Estimation	Point estimator for fault coverage. Single- and double- sided confidence intervals for the fault coverage determined with the normal approximation.
Precision	Biased by ρ (centered around the coverage proportion). Optimistic for $\rho < 0$, conservative for $\rho > 0$. Unbiased if the fault tolerance mechanism is fair.
Accuracy	Minimum variance if the fault tolerance mechanism is fair. For $\rho \neq 0$, minimum variance around the coverage proportion.
Assumptions	The uniform distribution is always indicated as the requirement for the unbiasedness of the estimator, whereas the fairness condition is necessary and sufficient. The validity of the normal approximation is only verified for low values of the coverage or for a high number of experiments.
No-Response Problem	Both optimistic and pessimistic estimators and confidence intervals are determined considering the no-response faults covered and uncovered respectively.
Number of Experiments	Generally (confidence level around 95%) in the order of 10^{x}, if 10^{-x} is the desired coverage level. Higher by one-two orders of magnitude for confidence level around 99%. Same as the simple model.
Applicability	This model should only be employed when no information is available about the fault distribution, apart from its multi- dimensional structure. It must also be possible to perform a large number of experiments in order to use the normal approximation.

Table 7.1 Properties of the Multi-Stage Bernoulli Model

8. Fault Equivalence Model

8.1. Introduction

Fault equivalence is a fault coverage estimation strategy that attempts to reduce the number of fault injection experiments required by partitioning the fault space into equivalence classes. The presentation of the fault equivalence model provided here is based on [2], where a uniform fault distribution is assumed. Therefore, it must be once more remarked that the estimator provided is biased.

However, it is shown in [2] that under certain conditions the number of experiments required can be reduced compared to the simple Bernoulli model, to obtain estimates with the same or lower variance. Under such conditions the accuracy of the estimate is therefore increased compared to the simple Bernoulli model.

The extension of the model presented in [2] to the case of a generic fault distribution is not immediate. However, when possible equivalent expressions valid for the generic case will be indicated for completeness.

8.2. Fault Space

The fault space considered in this model is partitioned into *equivalence classes*, such that any two faults belong to the same class if and only if it can be assessed that the fault tolerance mechanism will either cover or not both faults. Thus, the equivalence relation inducing the fault space partition can be expressed as:

$$f_h \sim f_k \quad \Leftrightarrow \quad y(f_h) = y(f_k) \tag{8.1}$$

A more flexible definition of the partition relation could account for the randomness associated with the actual behavior of the fault tolerance mechanism during the fault injection campaign, or for the uncertainty associated with the imperfect capability to assess the equivalence of faults. For example, one could define a distance function of fault f_j from fault f_i:

$$|f_i, f_j| = P[(Y_j = 1)|(Y_i = 0)] \cdot P[(Y_i = 0)] + P[(Y_j = 0)|(Y_i = 1)] \cdot P[(Y_i = 1)] \tag{8.2}$$

which expresses the probability that the fault tolerance mechanism behaved for fault f_j like it did for fault f_i (notice that the order of the operands must be preserved). Although it is not possible to define a useful partition of the fault space using the distance relation only, given a fault f_i it is always possible to define a subset $\Im_i$ of the fault space such that the following condition is satisfied by all the elements of the subset:

$$|f_i, f_j| \leq \gamma \qquad \forall (f \in \Im_i) \tag{8.3}$$

where γ is a confidence parameter.

The model presented in [2] only uses **Equation (8.1)** to induce a pre-determined partition of the space, however, **Equation (8.3)** can also be used in this model without any change in the results, leaving more flexibility in the setup of the fault injection experiments. This will be discussed in more detail in Section 8.4.

8.3. Fault Distribution

The fault distribution is assumed to be uniform in the model proposed in [2]. It is possible, however, to generalize the model to the case of a generic fault distribution, but this would be beyond the scope of this report. When readily available, formulas that are valid for the general case will be provided, together with the corresponding version in the assumption of uniform distribution.

8.4. Sampling

The sampling strategy is properly referred to as the fault equivalence process. The process starts with the random selection of a fault f_1 from the complete fault space. The equivalence class to which the fault belongs, which is predetermined using **Equation (8.1)**, is removed from the fault space, and another fault is randomly selected from the so reduced space. The process continues until M samples have been collected.

Given the described process, it is clear that **Equation (8.3)** could also be used to perform the fault equivalence process. Each time a fault f_i is sampled, the subset $\Im_i$ of faults distant from f_i less than the confidence level γ are removed from the population from which the sample f_{i+1} will be randomly selected. Therefore, the population from which sample f_i is selected is determined by the following:

$$f_1 \in \Im \quad \text{and} \quad f_i \in \Im \backslash \Im_{i-1} \backslash \ldots \backslash \Im_{n-1} \quad \forall i = 1 \ldots M \tag{8.4}$$

Notice that the following conditions are always satisfied by the sets $\Im_i$ determined using the fault equivalence process:

$$\bigcup_{i=1}^{M} \Im_i \subseteq \Im \quad \text{and} \quad \forall i, j = 1 \ldots M, \Im_i \cap \Im_j = \varnothing \tag{8.5}$$

Although these conditions do not define a partition of the fault space, the second one is sufficient for the validity of this model.

The number of faults contained in $\Im_i$ and the number of expanded faults are indicated as:

$$n_i = |\Im_i| \quad \text{and} \quad n = \sum_{i=1}^{n} n_i \tag{8.6}$$

which are, in this model, random variables, as their actual value depends on the specific realization of the sampling process.

8.5. Point Estimator

Notice that with the terminology introduced in the Section 8.4, this case is equivalent to the case of stratified sampling, and the same estimator previously used can be applied here:

$$\hat{C} = \sum_{i=1}^{M} \hat{C}_i P(F \in \mathfrak{I}_i) \tag{8.7}$$

with:

$$\hat{C}_i = y(f_i) \tag{8.8}$$

The estimator of **Equation (8.7)** is always unbiased, even if the assumption of uniform distribution does not hold. When no information about the fault distribution is available, one can do no better than assuming a uniform distribution and set:

$$P(F \in \mathfrak{I}_i) = \frac{n_i}{n} \tag{8.9}$$

to obtain the estimator proposed in [2]:

$$\hat{C} = \frac{1}{n} \sum_{i=1}^{M} \hat{C}_i n_i \tag{8.10}$$

Notice that in this case the estimated class fault coverage is always either 1 or 0. This is due to the assumption that is it possible to assess with certainty that all faults in a class will either be covered or uncovered if only one fault in the same class is, respectively, covered or uncovered.

8.5.1. Precision of the Estimator

It is immediate to show that the estimator of **Equation (8.7)** is unbiased contrarily to the other estimators encountered in the previous sections:

$$E[\hat{C}] = E\left[\sum_{i=1}^{M} \hat{C}_i P(F \in \mathfrak{I}_i)\right] = \sum_{i=1}^{M} \sum_{f \in \mathfrak{I}_i} y(f) P[(F = f) | (F \in \mathfrak{I}_i)] P[(F \in \mathfrak{I}_i)]$$

$$= \sum_{f \in \mathfrak{I}} y(f) P(F = f) = C \tag{8.11}$$

However, the estimator of **Equation (8.10)** is still biased, unless the assumption of **Equation (8.9)** is verified in the real system, that is, unless the uniform distribution is assumed, which implies the condition $\rho = 0$. Therefore, the estimator of **Equation (8.10)** leads in general to unbiased estimates of the coverage proportion $\dot{C}$.

8.5.2. Accuracy of the Estimator

It is shown in [2] that the variance:

$$\hat{S} = \frac{1}{n}\sum_{i=1}^{M}[y(f_i) - \hat{C}]^2 n_i \tag{8.12}$$

is lower than the variance that would be obtained using the simple Bernoulli model if the following condition applies:

$$\frac{n_u}{M_u} < \frac{n}{M} \tag{8.13}$$

where n_u indicates the total number of expanded uncovered faults, and M_u is the number of uncovered faults in the sample. Therefore, if the condition of **Equation (8.13)** applies, a reduction of the sample variance is achieved and the use of fault equivalence is justified. Clearly, this condition can only be verified after the fault injection campaign has been carried out, therefore it is not guaranteed *a priori* that fault equivalence will actually result in variance reduction.

A different approach to determining the variance reduction achieved via fault equivalence has been carried out in [13], where it is assumed that the estimator is exponentially distributed (which is a typical approximation used in the estimation of the distribution parameters of a Bernoulli population that is known to be very high or very low [14]), and it is argued that in usual systems the condition of **Equation (8.13)** is not very often verified. The model presented in [2] introduces a cost function that allows determining the convenience of using fault equivalence given the uncertainty of the variance reduction that can be achieved.

8.6. Confidence Intervals

Confidence intervals are determined using the normal approximation seen in the previous sections. Therefore, the same cautions discussed before are necessary in this case. A formula that allows calculating a lower confidence limit for coverage in the case when no uncovered faults are found in the sample is also provided in [2]:

$$\widehat{C}_\gamma = 1 - z_\gamma \sqrt{\frac{(n-1)M^2}{(M-1)n^4}} \tag{8.14}$$

which can be used when $M \gg 1$ and $n \gg 1$. In [2] it is also reported, from other references, that a number of experiments greater than 100 is sufficient for the application of the normal approximation. This statement has no general validity, as will be discussed in Section 10.

8.7. Number of Experiments

The number of experiments necessary to estimate coverage C with a confidence level of γ using the fault equivalence model presented in [2] is estimated using the relation:

$$n \approx \frac{z_\gamma}{(1-C)} \tag{8.15}$$

where n is the size of the expanded fault set considered in the worst case scenario when all sampled faults are covered and no useful information can be determined from the data. The number of actual experiments performed M obviously varies proportionally to the average size of each expanded class.

8.8. No-Response Problem

The fault equivalence model simply rejects the faults that result in no-response. Given the description of the sampling process provided in Section 8.4, this strategy is appropriate because faults are sampled and injected sequentially.

8.9. Conclusions

The fault equivalence model described in this section uses an estimator that, given the assumption of a uniform fault distribution, has a precision comparable to the precision of the estimator considered in the simple Bernoulli model. However, if the condition expressed by **Equation (8.13)** is verified, the accuracy of the estimator is guaranteed to be higher than the one obtained in the simple Bernoulli model. This is only an a posteriori condition, that must be verified after the fault injection campaign has been carried out.

When the condition of **Equation (8.13)** is verified, the higher accuracy of the estimate allows to interrupt the sequential sampling process when a desired number of fault injection experiments have been performed, which can be, in general, much lower than the number of experiments that must be performed in the simple Bernoulli model, depending proportionally on the average size of the expanded fault classes.

Therefore, the cost of fault injection experiments can be significantly reduced by using the fault equivalence model. A cost function is introduced in [2] that allows to determine the advantage produced by fault equivalence as a function of the achieved variance reduction.

It must be finally pointed out that the results obtained in [13] suggest that fault equivalence does not, for real systems, provide an advantage in most cases, because the condition of **Equation (8.13)** is not often verified, also given that the fault equivalence that can be achieved normally is relatively small.

These conclusions are summarized in **Table 8.1.**

Estimation	Point estimator for fault coverage. Single- and double- sided confidence intervals for the fault coverage determined with the normal approximation. Confidence interval in the case of no uncovered faults found.
Precision	Biased by ρ (centered around the coverage proportion). Optimistic for $\rho < 0$, conservative for $\rho > 0$. Unbiased if the fault tolerance mechanism is fair.
Accuracy	Variance reduced if the a posteriori condition of **Equation (8.13)** is verified.
Assumptions	The uniform distribution is indicated as the requirement for the unbiasedness of the estimator, whereas the fairness condition is necessary and sufficient. The validity of the normal approximation is only verified for low values of the coverage or for a high number of experiments.
No-Response Problem	This model discards the no-response faults, consistently with the sequential sampling process.
Number of Experiments	Reduced proportionally to the achievable fault equivalence.
Applicability	This model can only be employed when it is possible to define an equivalence relation in the fault space. Moreover, condition of **Equation (8.13)** must be verified for the model to provide any advantage compared to the simple model.

Table 8.1 Properties of the Fault Equivalence Model

9. Generalized Bernoullian Model

9.1. Introduction

This section introduces a generalization of the Bernoulli models presented in the previous sections, that can be only applied, however, to the case of known fault distribution. This model provides an unbiased and, under certain conditions, minimum variance estimator for the fault coverage.

The presentation is based on the widely available literature on sampling theory. This model has been considered and evaluated for fault coverage estimation in [8][10][11].

9.2. Fault Space and Distribution

The requirement imposed by this model is the knowledge of the fault distribution of the system. No particular assumption is made on the shape of the fault distribution. That is, the probability function:

$$P(F = f) \tag{9.1}$$

must be known $\forall(f \in \mathfrak{I})$. Although for most systems it is impossible to determine an exact fault distribution, some considerations that can be made to obtain a reasonable approximation have already been presented in Section 4.3.

9.3. Sampling

This model does not require using any specific sampling strategy, as long as the sampling distribution is a known function, expressed as in **Equation (4.25)**. However, it has been pointed out in Section 4.3 that the information on the distribution of the operational profile is practically impossible to determine, as it depends on the considered target system, its functional inputs, and the point in time when the fault occurs. The knowledge of the distribution of the operational profile is not necessary, however, if the representative sampling strategy is used, that is:

$$P(\underline{F} = f) = P(F = f) \tag{9.2}$$

Similarly to the considerations made in [8], it can be argued that a fault can occur at any random instant of the system's operational lifetime, so the probability of it occurring in coincidence with a particular operational profile is dependent only on the frequency at which the operational profile recurs.

Therefore, designing the experiments so that the system emulates the type of activity that it will typically perform during its lifetime, and so that the fault injection is independent from the particular operational profile (for example, by injecting the fault with a random delay from the beginning of the experiment and with random inputs imposed on the system) is equivalent to setting the sampling distribution of the operational profile equal to its real distribution. Therefore, it can always be assumed that, with this device:

$$P(\underline{A} = a) = P(A = a) \tag{9.3}$$

where A and $\underline{A}$ are random variables representing, respectively, the operational profile with which the fault occurs in the system and the one that has been "sampled" by randomizing the injection process.

9.4. Point Estimator

The estimator of the fault coverage considered in this model is given by:

$$\hat{C} = \frac{1}{n}\sum_{k=1}^{n} \underline{Y}_k \frac{P(F = f_k)}{P(\underline{F} = f_k)} \tag{9.4}$$

Notice that substituting **Equation (9.1)** into **Equation (9.3)**, that is in the case of representative sampling, the estimator is just the sample average, unbiased in this case because of the particular sampling strategy adopted.

9.4.1. Precision of the Estimator

The mean of the estimator is evaluated as:

$$E[\hat{C}] = E\left[\frac{1}{n}\sum_{k=1}^{n} y(f_k)\frac{P(F = f_k)}{P(\underline{F} = f_k)}\right] = \frac{1}{n}\sum_{k=1}^{n} E\left[y(f_k)\frac{P(F = f_k)}{P(\underline{F} = f_k)}\right]$$

$$= \frac{1}{n}\sum_{i=1}^{n}\sum_{f \in \mathfrak{I}} P(\underline{F} = f)y(f)\frac{P(F = f)}{P(\underline{F} = f)} = \frac{1}{n}\sum_{k=1}^{n}\sum_{f \in \mathfrak{I}} y(f)P(F = f) = C \tag{9.5}$$

that is, the estimator of **Equation (9.5)** is always unbiased. Obviously the validity of this statement is only guaranteed by the ability to accurately evaluate the fault distribution. The same result is obtained with representative sampling, which is an obvious conclusion, given that the forced distribution is identical to the fault distribution, thus the variables $\underline{Y}$ and Y are identically distributed.

In general, the fault distribution cannot be completely known, and only approximated fault distributions can be determined. Generally, such approximations are class-wise uniform functions like the ones described in Section 4.3.1. In this case, the generalized model described in this section is similar to the stratified model presented in Section 6. However, it must be noted that whereas the stratified model has very specific applications, and it requires that the size of the samples extracted from the various classes is pre-determined, in this case it is only required that the probability of sampling faults from a certain class is proportional to the relative frequency of faults occurring in a certain class. Therefore, **Equation (6.9)** does not apply in this case, and the problem of undetermined class coverage estimates is removed, since the system's coverage is always calculated as in **Equation (9.3)**.

9.4.2. Accuracy of the Estimator

The variance of the estimator is (the proof is easily derived, and is omitted for brevity):

$$V[\hat{C}] = \frac{1}{n}\left[\sum_{f \in S}\left(y(f)\frac{[P(F=f)]^2}{P(F=f)}\right) - C^2\right] \tag{9.6}$$

which, in the case of representative sampling is minimum and is equal to:

$$V[\hat{C}] = \frac{C(1-C)}{n} \tag{9.7}$$

and can be estimated from the sample by the corrected sample variance:

$$\hat{S}[\hat{C}] = \frac{C(1-C)}{n-1} \tag{9.8}$$

The estimator of **Equation (9.3)** is therefore the UMVUE (*uniformly minimum variance unbiased estimator*) for the fault coverage if and only if the sampling distribution equals the fault distribution.

9.5. Confidence Intervals

Confidence intervals for the fault coverage can be determined following the same procedure used in the previous sections. Since the estimator is unbiased, as the sample size is increased the extremes of the interval approach the real value of the fault coverage, no matter what the fault and sampling distributions are. However, if the two distributions match, the extremes will converge more rapidly, since the variance is the lowest possible. The appropriateness of the normal approximation will be discussed in the following section.

9.6. Number of Experiments

The number of experiments necessary to estimate the fault coverage with the generalized model can be determined as it was done for the simple Bernoulli model in Section 5.5, and the same considerations apply for the normal approximation.

9.7. No-Response Problem

As was discussed for the simple Bernoulli model, any particular strategy can be employed in the generalized model when no-response faults are injected. In this case, however, the fault distribution is known. Therefore, it is possible to determine how no-response data will influence the final estimate. If the faults resulting in no-response are relatively likely to occur, a new fault with comparable probability of occurrence should be sampled from the fault space for injection, so as to prevent that the no-response event affects the estimate significantly. If a fault with lower probability of occurrence results in no-response, the corresponding datum can, in general, be discarded. Clearly, if too many faults result in no-response, even if they have a low probability of occurrence, new faults should be sampled to avoid obtaining biased estimates.

9.8. Conclusions

The estimator for fault coverage provided by the generalized model is always unbiased, as long as the fault distribution is known with sufficient detail. Generally, designers of dependable systems can have no better knowledge about the fault distribution than a class-wise uniform distribution, which can be determined, if sufficient information about the system is available, as described in Section 4.3.1. Therefore, generally the estimate provided by the generalized model might still be biased, but the value of the bias (which is still represented, as in the stratified model, by ρ) can be assumed to be very small if the fault distribution is a good approximation of the real fault distribution of the system.

The generalized model can be seen as an extension of the stratified model in which each class contains only one fault. The size of the sample extracted from each class is not, however, predetermined using **Equation (6.9)**, which would lead to $n_i = 1$ for every i, i.e. exhaustive testing. Instead, faults are selected for injection with a certain probability given by any desired sampling distribution.

Moreover, the simple Bernoulli model can be seen as a special case of the generalized model, in which the fault and sampling distributions are both uniform.

If the sampling distribution equals the fault distribution the estimator has also minimum variance, and is therefore the UMVUE of the fault coverage. This property is the most desirable condition of an estimator, therefore the generalized model must be the preferred choice for fault coverage estimation, as it provides estimates with the highest possible precision and accuracy.

Obviously, specific needs or requirements might induce the designer of the fault injection experiment to resort to other models, which justifies the several different approaches that are described in the literature. However, for what concerns the precision and accuracy of the estimate, the generalized model provides the best solution.

One main limitation, which the generalized model shares with all the other models based on Bernoulli theory, is the definition of the confidence intervals resorting to the normal approximation, which imposes constraints on the minimum number of experiments to be performed that are more severe than the one obtained by imposing an upper bound to the variance of the estimator. This limitation will be overcome in Section 10, where an analytical solution for the equation that defines the upper confidence limit for the non-coverage will be provided.

The generalized model also allows to make more specific considerations about the strategy that must be adopted in the event of no-response faults, which can be based on the known relative probability of occurrence of the faults.

These conclusions are summarized in **Table 9.1.**

Estimation	Point estimator for fault coverage. Single- and double- sided confidence intervals for the fault coverage determined with the normal approximation (exact distribution can also be used).
Precision	Unbiased estimator. If the fault distribution is assumed class-wise uniform, the bias is equal to the one of the stratified model.
Accuracy	Minimum variance if representative sampling is used.
Assumptions	The fault distribution must be known, or at least a good approximation. The validity of the normal approximation is only verified for low values of the coverage or for a high number of experiments.
No-Response Problem	This model does not define a specific solution to the no-response problem. Specific considerations must be made in each case.
Number of Experiments	Generally (confidence level around 95%) in the order of 10^{x}, if 10^{-x} is the desired coverage level. Higher by one-two orders of magnitude for confidence level around 99%. Same as the simple model. If the exact distribution (Section 10) is used, the constraint on the variance is sufficient (required experiments generally much lower). Confidence intervals always meaningful with the exact distribution.
Applicability	This model can be employed only when detailed information about the fault distribution is available. Otherwise, only the uniform distribution can be assumed and the model is completely equivalent to the simple model.

Table 9.1 Properties of the Generalized Bernoulli Model

10. Binomial Model

10.1. Introduction

The model presented in this section is an extension to the various Bernoulli models discussed in the previous sections, which was introduced in [11] in order to determine single-sided confidence intervals for coverage using the actual binomial distribution of the fault coverage estimator rather than the normal approximation. Since most of the properties and the assumptions used in this model are equivalent to those stated in the previous sections, specific information omitted here can be found in the respective sections dedicated to the referenced models.

This section is also intended to provide a broader, critical view of the models presented so far, as it concludes the first part of this report dedicated to classical statistical models.

10.2. Fault Distribution, Sampling and Estimators

It has been pointed out that the estimators provided in the first four sections are biased because they neglect the real fault distribution of the system. Therefore, only if the fault tolerance mechanism of the system is fair can an unbiased estimate of the fault coverage be obtained. Otherwise, the estimators only provide an estimate of the coverage proportion or of the forced coverage. This is due to the adoption of the simple random sampling strategy for selecting faults from the whole fault space or from classes within the fault space described in the stratified model, and to the use of the sample average for estimating, respectively, the system's fault coverage or the class fault coverage. The estimates have been shown, in all cases, to be unbiased if and only if the fault tolerance mechanism can be considered fair, that is for $\rho \approx 0$.

With the generalized model, instead, the fault distribution has been assumed to be known, and an unbiased estimator for the fault coverage could be determined, no matter the sampling strategy employed. It was also noted that if the sampling distribution is chosen so that it is identical to the fault distribution, the estimator has also minimum variance.

Table 10.1 shows the mean of various estimators for non-stratified strategies under different assumptions about the fault distribution and the sampling distribution. When the forced coverage $\underline{C}$ is indicated, the estimator provides an estimate of the coverage that would be obtained if the sampling distribution were forced in the real system.

Fault Distribution $P(F=f)$	Estimator $\hat{C}$	Sampling Distribution $P(\underline{F}=f)$		
		Simple Random Sampling $1/\|\mathfrak{I}\|$	Representative Sampling $P(F=f)$	Generic Sampling $P(\underline{F}=f)$
Unknown	$\frac{1}{n}\sum_{i=1}^{n} y(f_i)$	$\underline{C} = \dot{C}$	n.a.	n.a.
Uniform $1/\|\mathfrak{I}\|$	$\frac{1}{n}\sum_{i=1}^{n} y(f_i)$	$\underline{C} = C$		$\underline{C}$
	$\sum_{i=1}^{n} y(f_i)P(F=f_i)$			n.a.
	$\frac{1}{n}\sum_{i=1}^{n} y(f_i)\frac{P(F=f_i)}{P(\underline{F}=f_i)}$			C
Known $P(F=f)$	$\frac{1}{n}\sum_{i=1}^{n} y(f_i)$	$\dot{C}$	C	$\underline{C}$
	$\frac{1}{n}\sum_{i=1}^{n} y(f_i)\frac{P(F=f_i)}{P(\underline{F}=f_i)}$	C		

Table 10.1 Estimators of Fault Coverage and their Mean

In [11] four different models using a stratified approach are described, obtained as the combination of arithmetic or weighted average and homogeneous or representative allocation. The extension of the table to stratified approaches to fault coverage estimation is immediate, by interpreting the type of allocation used as a device to impose a certain sampling distribution. The following approaches are described in [11]:

1. arithmetic average with homogeneous allocation
2. arithmetic average with representative allocation
3. weighted average with homogeneous allocation
4. weighted average with representative allocation

Some considerations have been made in the previous sections about the accuracy and the precision of these models: approach 1 is not interesting for the purpose of the analysis in this report because it produces biased estimates, as it was discussed in Section 6.5; approaches 2 and 3 are equivalent to the generalized model, assuming that in no case values of the samples sizes n_i equal to zero are obtained, as

it was discussed in Section 9.8, but they don't produce minimum variance estimates; approach 4 is also equivalent to the generalized model, using, however, a representative sampling strategy, which produces estimates with minimum variance, as discussed in the previous section, and is therefore interesting for the purposes of this report.

Therefore, the only cases that will be discussed in this section is approach 4, both from the perspective of the stratified model (that is, with pre-determined sample sizes), and from the perspective of the generalized model (that is, with sampling distribution equal to the fault distribution, but no pre-determined sample sizes).

The estimates will be provided in the form of upper confidence limits for the non-coverage, determined using both the normal approximation and the binomial distribution.

10.3. Terminology

In this section the quantity under analysis will not be the coverage, but rather the *non-coverage*, defined as:

$$G = 1 - C \tag{10.1}$$

The problem of estimating the non-coverage of a system is identical to estimating fault coverage, if the variable Y, and thus also the function $y(f)$, are replaced, in the formulations presented so far, by the variable Z and the function $z(f)$ respectively, defined as:

$$Z = 1 - Y \tag{10.2}$$

$$z(f) = 1 - y(f) \tag{10.3}$$

With this substitution, the estimators discussed so far will be estimators for the non-coverage, and will be indicated as $\hat{G}$. The corresponding mean, variance, and variance estimator will be indicated as $E[\hat{G}]$, $V[\hat{G}]$, $\hat{S}[\hat{G}]$ respectively. All these quantities have the exact same properties studied for the estimators of fault coverage.

In order to simplify the symbolism, the class non-coverage factors and the class occurrence probabilities can be grouped into two vectors of M elements, one for each class of the partition:

$$\vec{G} = [G_1, G_2, \ldots, G_M] \quad \text{and} \quad \vec{P} = [P_1, P_2, \ldots, P_M] \tag{10.4}$$

where $P_i = P(F \in \Im_i)$.

It is easy to show, from **Equation (6.7)**, that the following relation is satisfied:

$$G = \vec{P} \bullet \vec{G} \tag{10.5}$$

where the dot product indicates the scalar product of the two vectors.

Now, for each class a variable X_i can be defined that is equal to the summation of the samples Z_k resulting from sampling in the class $\Im_k$:

$$X_i = \sum_{i=1}^{n_i} Z_k \tag{10.6}$$

and the various X_i, called the deficiency numbers, can be grouped in a vector, defined as:

$$\vec{X} = [X_1, X_2, \ldots, X_M] \tag{10.7}$$

Since the value of an estimator depends on the sample, it can be indicated as $\hat{G} = \hat{G}(\vec{X})$.

A $100\gamma\%$ confidence region for the non-coverage vector $\vec{G}$ is a function $I_\gamma(\vec{X})$ of the sample vector such that for any given value of the non-coverage $\vec{G}$:

$$P[\vec{G} \in I_\gamma(\vec{X})] = \gamma \tag{10.8}$$

while an upper $100\gamma\%$ confidence limit for the system non-coverage G is a function $\widehat{G}_\gamma(\vec{X})$ of the sample vector such that for any given value of the non-coverage G:

$$P[G < \widehat{G}_\gamma(\vec{X})] = \gamma \tag{10.9}$$

Using the relations of **Equation (10.5)**, **Equation (10.8)** and **Equation (10.9)**, the function $\widehat{G}_\gamma(\vec{X})$ can be defined as:

$$\widehat{G}_\gamma(\vec{X}) = max_{\vec{G} \in I_\gamma(\vec{X})} [\vec{P} \bullet \vec{G}] \tag{10.10}$$

With these definitions, it will always suffice to define a confidence region for the non-coverage vector to obtain an upper confidence limit for the system non-coverage.

10.4. Normal Confidence Limits

The distribution of the estimator $\hat{G}$, which is the arithmetic average of the samples collected, both in the stratified and non-stratified case, can be approximated using the normal distribution, as seen in the previous sections. A rule-of-thumb that entitles one to approximate the binomial distribution of the estimator $B(n, G)$ with the normal distribution $N[nG;nG(1-G)]$ requires that:

$$nG \geq \alpha \quad \text{and} \tag{10.11}$$

$$n(1-G) \geq \alpha \tag{10.12}$$

where α depends on the desired accuracy of the approximation. For coverage estimation only **Equation (10.12)** can be used, since **Equation (10.11)** is generally automatically satisfied; vice-versa for non-coverage estimations. Coverage estimations with a confidence level of about 99% typically require that $\alpha \approx 50$, as will be discussed in Section 10.6.

If the condition of **Equation (10.11)** is satisfied, an approximate upper $100\gamma\%$ confidence limit for the system non-coverage can be defined as:

$$\widehat{G}_\gamma(\vec{X}) = \hat{G}(\vec{X}) + z_\gamma \sqrt{S[\hat{G}]} \tag{10.13}$$

which applies to both the estimator of the system non-coverage of the stratified model and to the estimator of the generalized representative model as well.

Indicating the vector of the samples Z_k within each class with the vector:

$$\vec{Z}_i = [Z_1, Z_2, \ldots, Z_{n_i}] \tag{10.14}$$

it will suffice to substitute $\vec{Z}_i$ in place of $\vec{X}$, and $\hat{G}_i$ in place of $\hat{G}$ into **Equation (10.13)** to obtain a confidence limit for the non-coverage of class $\Im_i$.

10.5. Binomial Confidence Limits

Following the discussion presented in [11], two methods to determine upper confidence limits in the stratified perspective using the binomial distribution rather than its normal approximation will be now introduced. The first method is used to determine a confidence limit for the system non-coverage using the estimator $\hat{G}$ for the system non-coverage. The second method is used to determine a confidence limit for the system non-coverage by first determining a confidence limit for the non-coverage of each class of the partitioned space.

10.5.1. Confidence Limit using the System Non-Coverage

The estimator $\hat{G}$ for the system non-coverage is a positive function of the random vector $\vec{X}$. As such, one can associate a cumulative distribution function $F_{\hat{G}}$ to the estimator:

$$F_{\hat{G}}(g) = P[\hat{G}(\vec{X}) \leq g] \tag{10.15}$$

Then, a function $\hat{g}_\gamma(\vec{G})$ of the non-coverage vector can be defined so that it is a solution to the equation:

$$F_{\hat{G}}[\hat{g}_\gamma(\vec{G})] = 1-\gamma \tag{10.16}$$

where γ is a fixed parameter. Solving the equation corresponds, therefore, to finding the function $\hat{g}_\gamma(\vec{G})$ for which the cumulative distribution of the estimator is constant and equal to $(1-\gamma)$, regardless of the value of the non-coverage vector $\vec{G}$.

Equation (10.16) defines a confidence region as defined in **Equation (10.8)**, such that the function $g_\gamma(\hat{G})$ is a $100\gamma\%$ confidence limit, where $g_\gamma(\hat{G})$ is defined as:

$$F_{\hat{G}}[g_\gamma(\hat{g}_\gamma(\vec{G}))] = 1-\gamma \tag{10.17}$$

that is, $g_\gamma(\hat{G})$ is the inverse of the function $\hat{g}_\gamma(\vec{G})$. Therefore, the confidence limit is provided by the solution of the following maximization problem [11]:

Maximization of $G = \vec{P} \bullet \vec{G}$ under the constraints:

- given by the confidence region frontier $g_\gamma(\hat{G})$ for G:

$$\vec{G}: \sum_{X' | (\hat{G}(X') \le \hat{G}(X))} \prod_{i=1}^{M} \binom{n_i}{X'_i} G_i^{X'_i} (1 - G_i)^{n_i - X'_i} = 1 - \gamma$$

- given by the limits of the parameter space:

$$\forall i = 1 \ldots M \quad \text{and} \quad G \in [0, 1]$$

which must be solved numerically in most cases. However, the maximization problem is intractable [11] for more than three classes in the partitioned fault space, therefore an alternative formulation is determined based on vectorial statistics, which is introduced in the following section.

The maximization problem, however, has an analytical solution for the special case of a single class in the partitioned fault space:

$$\widehat{G}_\gamma(X) = \frac{1}{1 + \dfrac{n - X}{(X+1) F_{2(X+1), 2(n-X), \gamma}}} \tag{10.18}$$

where $F_{\nu_1, \nu_2, \gamma}$ is the 100γ percentile of an F distribution with ν_1, ν_2 degrees of freedom.

10.5.2. Confidence limit for the representative generalized model

Equation (10.18) can be used to determine the upper confidence limit in the case of the representative generalized model. In fact, the case of a single class in the partitioned fault space is equivalent to applying the stratified approach to the entire fault space at once. However, in the stratified approach faults are selected within the classes using a uniform sampling distribution, while the representative generalized model uses a sampling distribution equal to the fault distribution of the system.

Observing that the maximization problem reduces to the evaluation of the percentile of the binomial distribution of the estimator when only one class is present in the partitioned fault space, it is clear that the upper confidence limit thereof determined does not resort to any approximation.

Since representative sampling is equivalent to observing the variable Z and not the generic variable $\underline{Z}$ observed when a distribution is forced into the system, it can be shown [14] that **Equation (10.18)** is actually an unbiased confidence limit for the system's non-coverage.

Moreover, since the estimator used in the maximization problem has minimum variance, the upper limit determined with **Equation (10.18)** is the most accurate possible, that is, it converges to the mean of the estimator in the fastest possible way.

10.5.3. Confidence Limit using Vectorial Statistics

The method presented here makes use of vectorial statistics, which allows determining a confidence limit for the system coverage by selecting appropriate independent confidence levels for each class.

The $100\gamma_i\%$ upper confidence limit can be defined for each class by using the binomial distribution of each estimator $\hat{G}_i$. Choosing the values of the class non-coverage confidence level γ_i so that their product equals the desired confidence level γ, it can be proven that the $100\gamma\%$ upper confidence limit is the solution of the following minimization problem [11]:

Minimization of $G = \vec{P} \bullet \vec{G}$ under the constraints:

- given by the global confidence :

$$\vec{G}: \prod_{i=1}^{M}\left[1-\sum_{X_i=1}^{X_i}\binom{n_i}{X_i}G_i^{X_i}(1-G_i)^{n_i-X_i i}\right] = \gamma$$

- given by the limits of the parameter space:

$$\forall i = 1 \ldots M \quad \text{and} \quad G \in [0, 1]$$

10.6. Comparison of the Results

In [11] the data collected from simulated fault injection experiments are analyzed using the normal approximation and the vectorial statistics, in order to compare the results obtained with the three approaches for a confidence level of 99%. Three different ideal systems are analyzed, which are characterized by a class-wise uniform fault distribution. The interested reader should refer to [11] for the actual data.

10.6.1. Non-Stratified Approach

10.6.1.1Normal Approximation

For low coverage estimations the normal approximation can be used even for relatively small sample sizes, while very large sample sizes are necessary for high coverage estimations. The rule-of-thumb presented in Section 10.4 provides a very reasonable criterion with a value of $\alpha \approx 50 \rightarrow 100$.

10.6.1.2Exact Distribution

The confidence limit obtained using **Equation (10.18)** is very accurate for a wide range of sample sizes if the coverage is relatively low. For high coverage estimations, this confidence limit proves to be somewhat more conservative than the one obtained with the normal approximation, but generally within the same order of magnitude. Moreover, in the case of high coverage estimations for critical systems it is often preferable to obtain a conservative limit with high confidence than an optimistic estimate with low confidence.

10.6.2. Stratified Approach

10.6.2.1Normal Approximation

The stratified approach produces almost the same results as the non-partitioned approach, although slightly lower confidence limit and a small loss of confidence is observed. This reaffirms that the representative stratified approach and the representative generalized approach are equivalent for what concerns that precision and accuracy of the estimators. The small differences must be attributed to the pre-determined sample sizes used in the stratified approach.

10.6.2.2Vectorial Statistics

The results obtained from vectorial statistics are very conservative, setting the confidence limit a few orders of magnitude higher than the real non-coverage.

10.7. Comments

The simulated experiments for which the data is shown in [11] lead to the following conclusions:

1. if the normal approximation is used, the number of experiments must be consistent with the rule-of-thumb;
2. the normal approximation leads to optimistic confidence limits if the number of experiments is not consistent with the rule-of-thumb;
3. using the exact distribution in the representative generalized approach leads to slightly more conservative confidence limits;
4. using vectorial statistics in the stratified approach leads to extremely conservative results, and should therefore be avoided.

The proposed method using the binomial distribution proved acceptable only in the non-stratified approach. The explanation to this result is to be searched for in the constraints applied to the minimization problem of the vectorial statistics method, which sets the global confidence level equal to the product of the class confidence levels. Since the confidence level is less than 1, each class is constrained by a much higher confidence level than the global one. For example, to obtain a global confidence level of 99%, the class confidence levels, supposed all equal in this example, would have to be almost 99.98% when the fault space is partitioned in 50 classes. It would be interesting to repeat the same experiments applied to a slightly modified vectorial statistics model, where the global confidence level equals the geometric average of the class confidence levels, that is:

$$\sqrt[M]{\prod_{i=1}^{M} \gamma_i} = \gamma \tag{10.19}$$

which should intuitively provide much more reasonable upper confidence limits. However, it should first be assessed whether using the geometric average would be theoretically justifiable or not.

Also, the maximization problem based on the estimator, which could not be evaluated in the simulated experiments, would intuitively lead to more reasonable the confidence limits, equivalent to those obtained in the non-stratified case.

10.8. Conclusions

The analysis of several statistical methods carried out in [11] provides very useful information on the type and quality of the results obtained using different statistical approaches for estimating fault coverage in systems with different characteristics. However, only the results for a few of the analyzed methods have been reported here, that is only for the unbiased and minimum variance estimators in both the non-stratified and stratified approaches. In all the other cases, the negative qualities expected of the biased and non-minimum variance estimators that have been formally proven in the previous sections are verified by the data provided in [11].

From the same results, it has also been possible to verify that the normal approximation of the distribution of the estimators should be used exclusively when either the coverage is not too high, or the number of experiments is sufficiently high. From the data reported in [11], it has been possible to argue a reasonable range of values for the parameter a of the rule-of-thumb, which proved to be a valid tool to evaluate the opportunity of applying the normal distribution.

The model based on vectorial statistics proved to be inadequate in most cases, as it provides overly conservative estimates.

The analytical expression of **Equation (10.18)** for the upper confidence limit of the non-coverage based on the binomial distribution that can always be applied to the case of the representative generalized model has been provided in [11]. This expression leads to very realistic estimates that are slightly more conservative than the ones obtained using the normal approximation, but are honored in a percentage of cases that is very close to the confidence level with which the limit was determined.

Finally, it must be concluded that the representative generalized model is the best statistical model based on the Bernoulli scheme available for fault coverage estimation. The only limitation of the model is the required knowledge of the fault distribution, which, however, has been shown to be a solvable problem if some practical considerations can be made about the system under analysis, or if historical data are available about the occurrence of faults with specific characteristics. Since an analytical expression for the upper confidence limit is available in this case, the inaccuracy introduced by using the normal approximation is also removed.

The properties of the generalized model have already been summarized in **Table 7.1. Table 10.2** summarizes the properties of the vectorial statistics method introduced here as an attempt to use the binomial distribution of the class non-coverage factors in a stratified perspective, which, however, does not produce acceptable results.

Estimation	Upper confidence limit for non-coverage determined with minimization problem of vectorial statistics.
Precision	Unbiased estimator if the fault distribution is close to class-wise uniform.
Accuracy	High variance, overly conservative estimates obtained (a few orders of magnitude lower than with the generalized model).
Assumptions	The fault distribution must be approximated by a class-wise uniform distribution. The confidence levels for the classes are determined so that their product equals the global confidence level required. This is not justified and leads to overly conservative estimates.
No-Response Problem	This model does not define a specific solution to the no-response problem. Specific considerations must be made in each case.
Number of Experiments	Upper confidence limits overly conservative even for a very high number of experiments. Impossible to determined an adequate number of experiments *a priori*.
Applicability	This model can be applied when a stratified approach is necessary and conservative estimates are desired. However, in such case the Bayesian approach leads to more reasonable conservativeness. Therefore, this model should generally be avoided

Table 10.2 Properties of the Vectorial Statistics Model

11. Bayesian Model

11.1. Introduction

The Bayesian model for statistical evaluation of fault coverage with a stratified approach is proposed in [10], which also reports the results of simulated fault injection campaigns with the intent of comparing the estimations obtained via the Bayesian approach with the ones obtained using the method of vectorial statistics. Although the comparison favors the Bayesian model, the results are not too encouraging compared to the ones obtained in the non-stratified case, as will be discussed at the end of this section.

11.2. Fault Space, Fault Distribution and Sampling

The fault space considered in this model is partitioned into classes of faults for which a class coverage can be defined as it was done in Section 6. Similarly, the fault distribution that must be assumed is a class-wise uniform distribution, with assigned weighting factors equal to the probability of occurrence of any of the faults belonging to each class.

Samples from each class are selected using a simple random sampling strategy. The number of samples that must be selected from each class is pre-determined using the same criteria previously discussed. In particular, two allocation strategies are considered: homogeneous and representative. It has already been argued that the results that one can obtain with homogeneous allocation will always be less accurate than those obtained with representative allocation, from considerations on the variance of the estimators obtained in the two cases. Similar conclusions are reached in [10] by analyzing the results obtained in both cases. Therefore, the case of homogeneous allocation will not be considered in this section.

11.3. Estimation

The estimation process used in this model is quite original in comparison to the models based on the Bernoulli scheme seen in the previous sections. Here, the class non-coverage factors are considered as random variables of unknown distribution in the interval [0;1] rather than constant parameters of the system. The information provided by the observed data is then used to determine the a posteriori distribution. Since the distribution of the class non-coverage is unknown, a conservative assumption is made *a priori* about it, i.e. that the non-coverage is uniformly distributed in the whole interval. This is expressed by assigning a beta distribution to the class non-coverage. This choice is motivated by the fact that the beta distribution with values of both parameters equal to 1 is identical to the uniform distribution. Also, the beta distribution belongs to a family of conjugate prior distributions, that always leads to a posteriori distributions within the same family.

The distribution assumed for the class non-coverage can then be expressed by the probability density function:

$$f_{G_i}(g) = \frac{g^{k_i-1}(1-g)^{l_i-1}}{\beta(k_i, l_i)} \tag{11.1}$$

where $\beta(k_i, l_i)$ is the beta function with parameters k_i and l_i. The posterior distribution based on the observation of the deficiency number X_i in each class is then given by:

$$f_{G_i}(g) = \frac{g^{X_i + k_i - 1}(1-g)^{n_i - X_i + l_i - 1}}{\beta[(X_i + k_i), (n_i - X_i + l_i)]} \tag{11.2}$$

11.3.1. Moments of the System Non-Coverage Distribution

The system non-coverage is expressed as the weighted average of the class non-coverage factors. Therefore, the distribution of the system non-coverage is obtained from the joint distribution of the class non-coverage factors.

It is well known [14] that a distribution can be fully described by its moment generating function rather than its density or cumulative probability function. It is also well known [14] that two distributions well approximate each other if the first few moments (generally the first four moments are more sufficient) are identical for both.

In order to determine the joint distribution, two methods can be used, the first based on the moment generating functions, the second based on moments determined using the assumption of independence of the class non-coverage factors.

11.3.1.1Moment Generating Functions

The moment generating function of a beta distribution can be expressed as a confluent hyper-geometric function. Therefore, the moment generating function for the class non-coverage posterior distribution can be expressed as:

$$\Phi_{G_i}(t) = {}_1F_1[(X_i + k_i), (n_i + k_i + l_i), t] \tag{11.3}$$

Combining the moment generating functions of the different class non-coverage factors, an expression for the moment generating function of the system non-coverage is obtained:

$$\Phi_G(t) = \prod_{i=1}^{M} {}_1F_1[(X_i + k_i), (n_i + k_i + l_i), t] \tag{11.4}$$

from which the moments of the posterior distribution of the system non-coverage can be determined by differentiation. The expressions of the resulting moments are omitted here for conciseness.

11.3.1.2Independence Assumption

Assuming that the different powers of the class non-coverage are mutually independent variables, a simple expression for the r-th moment of the posterior joint distribution is derived in [10]:

$$\mu_r = \frac{\beta[(X_i + k_i + r), (n_i - X_i + l_i)]}{\beta[(X_i + k_i), (n_i - X_i + l_i)]} \tag{11.5}$$

11.3.2. The Pearson Distribution System

Once the first four moments have been determined using one of the two proposed methods, a distribution that adequately describes the data can be determined using the Pearson distribution system. The probability density function of the system non-coverage $f_G(g)$ can be obtained as the solution to the differential equation:

$$\frac{1}{f_G(g)}\frac{df_G(g)}{dg} = -\frac{a+g}{b_0+b_1 g+b_2 g^2} \tag{11.6}$$

where a, b_0, b_1 and b_2 are calculated from the moments (the expression is omitted here). The Pearson distribution system contains seven types of distribution, such as the beta, gamma, inverse Gaussian, t-Student. The distribution type and parameters that best describe the distribution of the system non-coverage is determined by numerical solution of **Equation (11.6)**, using the first four moments calculated from the data as shown in the previous section.

11.3.3. Confidence Intervals

The expression used to define confidence intervals for the system non-coverage:

$$P(G \in [\hat{G}_1 ; \hat{G}_2]) = \gamma \tag{11.7}$$

can be solved given the distribution of G that has been determined from the data. In [10] upper confidence limits for the non-coverage are calculated as:

$$\widehat{G}_\gamma = F_G^{-1}(\gamma) \tag{11.8}$$

where $F_G(g)$ is the cumulative distribution function of the type and with parameters determined using the procedure described.

11.4. Analysis of the Results

The upper confidence limits obtained with the Bayesian method are compared in [10] with the ones obtained with the vectorial statistics method. In general, it is observed that between the independence assumption and the moment generating functions, the latter method always leads to more reliable confidence limits, which are always less conservative than the ones obtained with the vectorial statistics and more conservative than the ones obtained in the non-stratified representative case.

These results confirm that the vectorial statistics method is not appropriate for upper confidence limit estimation in the stratified case and demonstrate that the Bayesian approach can be used as a valid alternative.

However, the confidence limits resulting from the Bayesian model are still more conservative, by about an order of magnitude, than the ones obtained with the non-stratified representative case. Therefore, the non-stratified approach must still be preferred to the stratified one.

It must be noted that the sample systems used in the simulated fault injection campaigns all have a negative value of the covariance. It would be interesting to observe the behavior of the Bayesian model in the opposite case.

11.5. Conclusions

The Bayesian model described in this section is substantially different from the Bernoulli models described in the previous sections. The estimation process is, in fact, based on a distribution fitting approach that describes the non-coverage of a system as a random variable distributed in the interval [0;1]. The distribution of such random variable is determined from the data using a Bayesian method, that leads from an *a priori* distribution, which is assumed to be uniform, to an *a posteriori* distribution, which, in most cases, is determined to be the Beta distribution.

Measures of the precision and accuracy of this estimation process should be given by the mean and variance of the a posteriori distribution. However, such measures are not provided in [10], and cannot be easily derived. Therefore, the overall quality of the estimate cannot be assessed based on the formal development of the model, but only by the analysis of the results of the simulated fault injection experiments provided in [10]. However, since the estimates produced are only single-sided confidence intervals, it is also impossible to characterize separately the precision and the accuracy of the estimate, i.e. the mean and the variance of the distribution. Such an analysis is further complicated by the fact that the results are provided through diagrams rather than by numerical tables.

Empirical arguments would suggest that since the fault distribution is not considered in the model, a bias at least in the order of magnitude of the covariance ρ should be expected. Unfortunately, giving general validity to such arguments is not an easy matter.

However, it is possible to assess that the Bayesian approach provides estimates that are more conservative than the ones obtained using the representative generalized model by about an order of magnitude, and much less conservative than the ones obtained using the vectorial statistics model described in the previous section.

Therefore, if practical considerations (such as reuse of previously obtained data from fault injection campaigns) lead to the choice of a stratified approach for fault coverage estimation, then the Bayesian approach must be preferred to the vectorial statistics model. However, if the stratified approach is not necessary, the representative generalized model must still be preferred, as it is known that it provides the UMVUE estimator for the fault coverage.

It can also be argued that the conservativeness of the estimates provided by this model is due to the assumption that the prior distribution is uniform. Generally, it can be assumed, for real systems, that the probability of the fault coverage being close to zero is very low. Therefore, more biased prior distributions could be employed instead that assign higher probability to values of the fault coverage closer to unity. However, the conservative approach has been chosen in [10] so as to minimize inaccuracies due to unverifiable assumptions.

The properties of this model are summarized in **Table 11.1.**

Estimation	Upper confidence limit for non-coverage determined from the estimated distribution of the deficiency number.
Precision	This information is not provided. It can be argued that since the fault distribution is not used at all, the model leads to biased estimates.
Accuracy	This information is not provided. However, from the data it can be argued that conservative estimates are obtained (within an order of magnitude lower than with the generalized model).
Assumptions	The prior distribution of the fault non-coverage is set to the uniform distribution. A class-wise uniform distribution is assumed.
No-Response Problem	No-response faults can only be discarded. The problem is ignored by the model.
Number of Experiments	An estimate of the required number of experiments is not provided.
Applicability	This model can be applied when a stratified approach is necessary. Otherwise, the representative generalized model must be preferred, given that at least a class-wise uniform distribution is known.

Table 11.1 Properties of the Bayesian Model

12. Statistics of the Extremes Model

12.1. Introduction

This section introduces a new approach to fault coverage estimation, based on Statistics of the Extremes, a well-developed theoretical framework that has been successfully applied to many engineering problems in which rare events play a fundamental role [6][7]. Dependability assessment problems often demand using reasonably conservative approaches to modeling and estimation processes, especially in the case of high-safety systems. The applicability of Statistics of the Extremes to dependability assessment problems has been already demonstrated in different instances, e.g. for software reliability analysis [3]. The application of Statistics of the Extremes to fault coverage estimation has been first introduced in [4] and then proposed in a revised version in [17], which will be the reference for the discussion presented here. Such work is motivated by the need to obviate making *a priori* assumptions about the distribution of the random variables involved in the estimation process, as is done in most of the models presented in the previous sections.

12.2. Motivation

In the previous sections, it was pointed out that in order to estimate fault coverage, precisely detailed knowledge of the fault distribution is necessary; otherwise, one must assume that the fault tolerance mechanism is equally fair with faults occurring as they would in the real system and in the fault injection process.

The practical inability to assess whether the fairness condition is verified or not demands statistical models that provide conservative, though not overly conservative, estimates. The statistical models that have been presented in this report make implicit use of two assumptions, directly deriving from the classical definition of coverage:

- The fault tolerance mechanism behaves deterministically for a given fault, and
- The fault distribution is a well-defined characteristic of the system

The non-deterministic behavior of the fault tolerance mechanism has been so far described as related to the randomness of the operational state of the system at the moment of occurrence of the fault. By reproducing in the experimental process the typical activities that will be performed by the system during its operational lifetime, one is able to isolate the dimension of the fault space describing the system's operational profile, thus also keep into account the associated randomness.

However, this device still does not account for other forms of randomness of the fault tolerance process, which might be due to different reasons, such as temporary failure of the components implementing the fault tolerance capabilities, fluctuations in their behavior, environmental conditions, etc. Furthermore, inaccuracies inherent with the experimental process, which only attempts, with unknown success, to mimic the system's behavior, should also be considered. This might be particularly true for fault injection experiments performed on early-stage systems prototypes or on computer simulated models of the system.

As for the fault distribution, similar considerations apply that justify the belief that faults will not occur, during the system's operational lifetime, according to some partially or fully predictable

distribution reproducible during testing. Multiple instances of the same system's design might show arbitrarily different fault distributions, for example, if the environmental conditions vary or if the functional input profiles are consistently different. This may be true, for example, for semi-custom designs that find application in multiple fields for which the same system is subject to different operating conditions.

If some or all of the previous considerations apply, then the two aforementioned assumptions do not hold, thus the system's fault coverage cannot be considered as a fixed parameter of the system, but rather as a random variable of, in general, unknown distribution. It must be noted that the Bayesian approach described in Section 11 does make use of a similar statement, which, however, is only related to the peculiarity of the statistical estimation process, and not, as discussed here, to the non-applicability of the assumptions above. In addition, the conservative character of the results produced by the Bayesian model is not voluntarily elaborated, but only passively and practically observed.

Finally, it must be noted that specific applications for which a cautious approach to dependability assessment is critical, such as high safety systems, demand a justifiably conservative estimation of fault coverage. That is, unjustified overly or insufficiently conservative estimates resulting from inadequacies of the statistical model employed, which is the case for the vectorial statistics and the Bayesian models, are not desirable.

12.3. Introduction to Statistics of the Extremes

Statistics of the extremes theory studies the statistical distribution of the extreme values found in a sample extracted from a population with a generic distribution. In this section, the basic terminology and the most significant results useful for the discussion are briefly introduced. For more detailed information refer to [6] or similar texts.

Consider a sample of n values extracted from a population X of generic distribution $F_X(x)$, called the parent distribution. Indicating with X_i the i-th order statistic of the sample (that is, the random variable representing the i-th smallest values of X observed), the minimum and the maximum observed values will be, respectively, X_1 and X_n, distributed as:

$$F_{X_1}(x) = 1 - [1 - F_X(x)]^n \tag{12.1}$$

$$F_{X_n}(x) = [F_X(x)]^n \tag{12.2}$$

In the limit when the sample size is increased to infinity, the distributions of **Equation (12.1)** and **Equation (12.2)** may or may not converge to a limit distribution. If the limit distributions exist, they can only belong to one of three domains of attraction: the Frechet, Weibull and Gumbel distributions for minima and maxima. It can also be proven that a parent distribution with finite end-point in the tail of interest cannot lie in the Frechet domain of attraction.

The Frechet, Weibull and Gumbel distributions can be written in a unified form, called the Von Mises form, from which they are obtained by simply setting the value of a parameter c, called the Von Mises constant, such that, respectively, $c > 0$, $c < 0$, and $c = 0$. For example, it is easy to show that the Von

Mises constant $c = -1$ for a uniform parent distribution, therefore the latter belongs to the domain of attraction for both minima and maxima of the Weibull distribution.

Empirical methods to determine from observed data the domain of attraction to which a population of unknown parent distribution belongs are available. By plotting the observed data on Frechet, Weibull and Gumbel probability paper for minima or maxima, the corresponding domain is given by the plot that most closely approximates a straight line. Once the domain of attraction has been determined, the data can be used to determine the parameters of the limit distribution.

12.4. A Short Dissertation on the Viable Approach

The main motivations that justify using Statistics of the Extremes have been identified in Section 12.2 in the need for a conservative approach for fault coverage estimation. It was then pointed out in Section 12.3 that Statistics of the Extremes have the practical advantage of allowing analytical and graphical analysis of the experimental data in a fairly independent manner from the parent distributions of the random variables involved in the experimental process.

This section discusses the problem of estimating the fault coverage of a dependable system from a Statistics of the Extremes perspective. The double intent of this brief discussion is to suggest possible directions for future research on the matter, while at the same time providing an informational background that should help understand the analysis of the model proposed in [17], which will be carried out in Section 12.5.

12.4.1. Statistics of the Extremes of a Bernoulli Variable

For a dependable system it is expected that the fault coverage will have a value very close to unity, although it is always assumed that the coverage factor does never actually reach the value 1. Obviously, it is also assumed that it is very unlikely that the coverage assumes values any close to 0. This is due to the fact that, for dependable and ultra-dependable systems, the occurrence of a non-covered fault is a rare event. The random variable associated with the tolerance/non-tolerance of a fault is the Bernoulli variable Y, which can only assume the values 0 and 1, with probability $(1 - C)$ and C, respectively. For such a variable, it is not easy to define a tail of the distribution. In fact, the left tail would correspond to simply the value 0, and the right tail to simply the value 1.

Neither in [6] nor in [7] discrete distributions (and certainly not Bernoulli) are considered, and it is not clear if the results provided by Statistics of the Extremes theory can be applied to the study of the Bernoulli random variable Y. From **Equation (12.1)** and **Equation (12.2)**, it appears that for an infinite sample size the distribution of the maximum and the minimum of the Bernoulli variable Y would degenerate to total coverage and total non-coverage for the maximum and the minimum, respectively.

This fact simply means that if an infinite number of faults occur in the system (or are injected into the system), at least one fault will be uncovered (unless $C = 1$) and at least one fault will be covered (unless $C = 0$). Therefore the minimum and the maximum of the variable Y will be 0 and 1, respectively, with probability 1. This result is not very interesting, since an infinite number of fault occurrences will never be observed, neither during the system's lifetime (which is, in real-life, limited), nor during a fault injection campaign.

However, using **Equation (12.1)** and **Equation (12.2)** in the case of a finite sample size, one can determine the probability of observing at least one covered fault and at least one uncovered fault. This result is quite interesting. It is a very common problem, in fact, that no uncovered faults are revealed during fault injection campaigns. This generally occurs when an insufficient number of fault injection experiments are performed to estimate the coverage of a very dependable system.

In order to estimate a reasonable number of experiments to be performed, **Equation (12.1)** can be used. The probability that in a sample of size n the minimum value of the observations of Y is 0, i.e. $F_{X_1}(0)$, is the probability that at least one fault in the sample was uncovered.

Therefore, given a desired coverage level, one can assess the number of experiments necessary to observe at least one uncovered fault with a certain confidence $\gamma = F_{X_1}(0)$ as:

$$n = \log_C(1-\gamma) \tag{12.3}$$

This result is in agreement with the conclusions reached in [16] using a different but equivalent approach. In Section 5 it was mentioned that in order to estimate the fault coverage, a number of experiments of 10^x is generally performed if the required non-coverage is 10^{-x}. From **Equation (12.3)** it can be calculated that with such approach, the probability of revealing at least one uncovered fault is only about 0.7, which is usually not sufficient. However, if the number of experiments is increased by just one order of magnitude, the probability of revealing at least one uncovered fault is increased to about 0.99999, which guarantees with very high confidence that at least one or more uncovered faults will in fact be detected.

12.4.2. Statistics of the Extremes of the Estimator

Referring to the estimator proposed in the representative generalized model (or the one in the simple Bernoulli model if no information about the fault distribution is available), it is expected that the estimated value of the fault coverage lies with high probability around the actual value of the system's fault coverage, since the estimator is unbiased (in the case of simple random sampling, the estimate will fall with high probability around the value of the coverage proportion). It is clear that on rare occasions one might find that the estimator assumes extreme values that approach either the value 1 or the value 0. For example, if no uncovered faults are found during the fault injection campaign, the estimator would assume the value 1 (which is actually not a very rare case); in the extremely unlikely situation (which is, however, possible) that all sampled faults are not covered, the estimator would assume the value 0.

This is perfectly described by the fact that the estimator, which is a linear combination of Bernoulli random variables (the observations of Y), is binomially distributed. For the same reason, the number of covered faults found in the fault injection campaign is also a binomial random variable, as is the number of uncovered faults (which, keeping the terminology introduced in Section 10, will be called the deficiency number and indicated as X).

Since the estimator, the number of covered faults, and the deficiency number are random variables that assume specific values only when the whole fault injection campaign has been completed, a whole fault injection campaign can be seen as a probabilistic "random experiment," the outcome of which is the

value assumed by the mentioned random variables. Therefore, the fault coverage estimate obtained from a single fault injection campaign is just one of the possible realizations.

This simple consideration explains and extends the discussion presented in Section 12.2, where it was said that the fault coverage observed in different instances, e.g. in several identical systems all operating for a certain amount of time, will be, in general, different for the various systems. In the same way, the fault coverage of the system calculated from experimental data will vary if multiple fault injection campaigns are carried out. In Section 12.2 it was concluded that, in some applications, one estimate of the fault coverage, i.e. only one fault injection campaign, can be insufficient to assess with some degree of certainty that the experimental results will be verified during the actual operational lifetime of the system. Therefore, a more conservative approach should be used in those applications that require a high level of confidence.

One such conservative approach could be to perform multiple fault injection campaigns, each campaign obviously resulting in a different estimate of the fault coverage. This is, ideally, the same principle that is used to define confidence intervals for the fault coverage, except that only one campaign is carried out, and an error margin is determined from the known distribution of the estimator so that the estimate is reliable with a given confidence level.

If multiple fault injection campaigns were actually carried out, the data obtained (that is, a series of fault coverage estimates) could be analyzed using Statistics of the Extremes.

Given the discreteness of the fault tolerance/non-tolerance events, a whole fault injection campaign only provides one observation of the random variable "fault coverage estimate". The rare events that can be considered in this case are, respectively, a fault coverage estimate equal or extremely close to 1, and a fault coverage estimate extremely lower than 1 or even equal to 0, which are found, in fact, in the tails of the parent distribution.

The main conceptual difficulty of applying Statistics of the Extremes to the problem of fault coverage estimation lies in the fact that normally only one fault injection campaign is carried out (or can practically be carried out, considering the cost associated with fault injection experiments). Instead, Statistics of the Extremes requires that multiple fault injection campaigns are carried out to obtain a certain number of estimates of the fault coverage. This is due to the fact that the random variables to which Statistics of the Extremes theory is applied are generally continuous ones, for which each observation leads naturally to a real number. Therefore, if Statistics of the Extremes is applied to the fault coverage estimator, it must be kept in mind that a single fault injection campaign represents only one observation of the random variable.

As mentioned before, the estimator is known to have a binomial distribution, for which it is reasonable to believe that a non-degenerate limit distribution exists. However, such a statement cannot be supported by the theory presented in [6] and [7], as both texts only refer to continuous distributions. Assuming that the limit distribution exists, then only an approach based on multiple fault injection campaigns could be used, if Statistics of the Extremes were to be applied to the fault coverage estimator. In such case, both [6] and [7] provide several estimation methods based on graphical and analytical approach (e.g. Gumbel probability paper, Pickand method, etc.).

Such an approach is probably worth investigating. However, since multiple fault injection campaigns would be needed, either the total number of experiments required must be very large, or each observation of the fault coverage estimator would have a quite large variance. Notice that given the conservativeness inherent with a process that uses Statistics of the Extremes, a larger variance would not necessarily be of concern.

In order to elaborate an estimate from the observed fault coverage estimates, the first step would have to be determining the domain of attraction of the estimator's distribution. The second step would have to be determining the distribution parameters for minima and maxima from the data. Finally, the resulting distribution could be used to calculate confidence intervals, or its mean could simply be used as the final estimate.

Both the limit distribution for the minimum and the maximum can be calculated. However, only the one for the minimum of the observed fault coverage is actually interesting, as it would lead to the desired conservative estimate; the maximum's distribution would instead always lead to an overly optimistic estimate.

Unfortunately, the same comments that were made in Section 11 about the inability to formally determine, or even choose, the degree of conservativeness that would be achieved apply to the described approach. However, simulated fault injection campaigns could be carried out, as in [8][10][11], to validate or discard the model based on the numerical results that would be obtained.

12.5. The Kaufman et al. Model

As mentioned in Section 12.1, the research work presented in [17] is motivated by the constant need, in practically all the statistical models for fault coverage estimation, of some *a priori* assumptions about the distributions involved.

The authors point out, specifically, that the binomial distribution of the point estimator is assumed, in different instances, to be well approximated by the normal distribution [2][5][8], the exponential distribution [13], and the beta/uniform distribution [10]. It would therefore be desirable to develop a statistical model that does not need to resort to such assumptions.

As was discussed in Section 12.3, Statistics of the Extremes theory does indeed lead to limit distributions of only three types independently of the parent distribution, and such limit distributions can often be easily determined using analytical or graphical methods.

12.5.1. Assumptions

The probabilistic and statistical framework on which the model is based is similar to the one described in Section 4, although the fault distribution is assumed to be uniform, i.e. all faults in the system are equally likely to occur. Thereinafter, the estimation process is described equivalently to the simple Bernoulli model presented in Section 5. This means that the sampling distribution is also chosen to be uniform.

The discussion of the model articulates around a series of assumptions, some of which are quoted below for reference, and will be commented in the following sections:

i. the occurrence of an uncovered fault is a rare event, and as a result, this information is found within the tail of the parent distribution

ii. the maximum number of uncovered faults that can ever occur in a given system is d. After each set of fault injection experiments is completed, one uncovered fault is removed from the system so that the resulting upper limit on the number of uncovered faults in subsequent fault injection experiments is d - 1; Hence, the data for R_f consists of $\{1, 2, \ldots, d-1, d\}$ uncovered faults; that is, the number of uncovered faults contained within a given system approaches a lower limit or bound during its design and test period

iii. [...] during fault injection experiments for ultra-dependable systems, few if any uncovered faults are discovered

iv. [...] this upper limit can be adjusted if the designer believes that the selected number of 100 uncovered faults is too unrealistic; that is, d can be set to any specified value

12.5.2. Statistics of the Extremes for Fault Coverage Estimation

The core of the model consists in the application of Statistics of the Extremes to the deficiency number. The process is described as follows. The "data," that is, the realizations of the deficiency number, are first plotted on Gumbel probability paper in order to determine the domain of attraction. A quick inspection of the plot represented in [17] shows that the function is linearly increasing from 0 to $\underline{d}$. In other words, the distribution of the deficiency number X is implicitly assumed to be uniform in the interval $[0; \underline{d}]$. Note that this assertion is not equivalent to the stated assumption of equally-likely occurrence of faults.

The plot is interpreted as a proof that the data belongs to the Weibull family. This information is then used to determine the probability that the deficiency number X exceeds values x in same interval $[0; \underline{d}]$, which is then plotted on linear paper. From this data it is concluded that the lowest probability of exceeding a certain number of uncovered faults is obtained for $X = \underline{d}$. Notice that if the implied assumption of a uniform distribution of the deficiency number is taken into account, this result is straightforward and does not require using Statistics of the Extremes.

12.5.3. Analysis

Although the use of Statistics of the Extremes is original and worth investigating, there are concerns to be expressed about the formal development of the model. Several arguments can be raised regarding the assumptions listed before. The assumption of faults being equally likely to occur in the system, for example, may drastically limit the applicability of the model, although it could be immediately removed by simply stating that the sampling distribution is equal to, or approximates, the system's fault distribution. In this case, the model could be applied to such systems for which detailed information about the fault distribution is available, thus reducing the bias of the estimator given by the covariance ρ.

In the statement of assumption *i*, it is not clear to what the term parent distribution refers. If it refers to the distribution of the random variable Y (which describes the tolerance/non-tolerance of a fault that has occurred in the system), then, as it was discussed in Section 12.4.1, the concept of tail for a Bernoulli

distribution is not well defined, and does not lead to useful results in the application of Statistics of the Extremes to the estimation of fault coverage in the limit of an infinite sample size.

If the term refers to the distribution of the estimator, then, as it was discussed in Section 12.4.2, the "rare event" to be considered is the event that after a whole fault injection campaign the value assumed by the estimator is lower, or much lower, than the mean of the estimator, i.e. the system's fault coverage, and not the occurrence of an uncovered fault.

If, finally, it refers to the fault distribution, then it is not clear why [17] should resort to a graphical method to determine the domain of attraction. The uniform distribution belongs, in fact, to the Weibull family, as can be determined analytically by calculating the Von Mises constant $c = -1$ [6].

In assumption *ii*, the data for variable R_f is said to be equal to the set of natural numbers between 0 and $\underline{d}$. However, R_f is defined in [17] as a Bernoulli random variable, thus it cannot assume any values other than 1 or 0. The meaning of the variable R_f is therefore unclear. The mentioned data seems to represent the possible values of the deficiency number. On the other hand, from the description of the process, it appears that such data are obtained in a deterministic manner by removing one uncovered fault from the system every time one is found through fault injection. While a similar process appears to be applicable to a software analysis problem as described in [3], in the more general case faults can result from a combination of design errors and, most often, unpredictable operating states, deteriorating components, fluctuations of the environmental conditions; it is therefore not clear how faults could in practice be "removed" from the system. A similar issue is also raised by assumption *iii*, which, in contrast with the previous assumption, states that at the end of each set of fault injection experiments few if any uncovered faults are found.

12.5.4. Results of the Model

From the conclusion reached in [17] and indicated in Section 12.5.2, $\underline{X} = \underline{d}$ is chosen as the most conservative value that should be used to estimate the non-coverage. Using the relationship:

$$\hat{C} = 1 - \underline{X}/n \tag{12.4}$$

an estimate of the fault coverage is determined.

In the introduction to the model, in [17], it is stated that this model provides an estimate of the fault coverage in the case when testing reveals no failure. It appears, however, that **Equation (12.4)** cannot comfortably be used as such estimate, for the reasons expressed below.

The relation $\underline{X} = \underline{d}$ indicates that the value of the deficiency number, according to this model, is not observed as a result of the fault injection campaign, but is instead set equal to the value $\underline{d}$, which is determined from engineering judgment, as is clearly indicated by assumption *iv*.

Setting the value of $\underline{d}$ based on engineering judgment may not be considered completely arbitrary, but rather analogous to what is done in other models by setting confidence levels. In both cases, in fact, one parameter is set to a desired value and an estimate is obtained in the absence of data (see, for example, the model presented in [16]). However, it must be argued that although confidence levels are subjective measures, they can be expressed independently from the other quantitative information

available for a particular system. That is, for example, a 99% confidence level has the same meaning applied to any particular problem, whereas the value of $\underline{d}$ does not have a character of general applicability.

Another quantity determined in [17] using a similar approach is the number of required fault injection experiments, given a desired fault coverage. This is calculated as:

$$n = \frac{1 - C_{desired}}{\underline{X}} \tag{12.5}$$

which is also linearly dependent on the value $\underline{X} = \underline{d}$. Once again, given the type of relationship between the value of n and $\underline{X}$, this appears practically equivalent to simply setting the value of n based on engineering judgment. If the parameter expressed a relative measure, independent from the system (such as a confidence level or some other probability), this approach would be acceptable. Instead $\underline{X} = \underline{d}$ is a quantity that lacks the necessary generality.

12.6. Conclusions

From the considerations made in the previous sections, it must be concluded that the Statistics of the Extremes model proposed in [17], in its current formulation, cannot be applied in a straightforward manner to the problem of statistical estimation of the fault coverage.

Although it is stated that Statistics of the Extremes are used to model coverage when testing reveals no failure, it is not clear how this information is used in the application of the theory to the problem in question.

The two main conclusions reached by the model, regarding the estimated fault coverage and the required number of experiments, could have been reached without resorting to Statistics of the Extremes. The implied assumption that the deficiency number is uniformly distributed in the interval [0; $\underline{d}$] (from which the Weibull domain of attraction was determined) was not formally justified and is in contrast with the results of the Bernoulli theory. The probability of exceedance calculated from the Weibull distribution is therefore not significant to the problem, unless otherwise proven.

It is possible that Statistics of the Extremes could be successfully applied to fault coverage estimation problems (for example, as it was suggested here in Section 12.4), but such a study is beyond the scope of this report.

13. Summary

This report provided a detailed analysis of several statistical models for fault coverage estimation found in the literature. A major effort has been devoted to achieving a consistent presentation throughout the whole report, so as to simplify a direct comparison of the proposed models.

In Section 2 the concept of fault coverage and its significance in the assessment of the dependability of critical systems were introduced. Statistical estimation of fault coverage was then introduced as related to the more general problem of parameter estimation for behavioral models. The need for high precision and accuracy, due to the sensitivity of the characteristic dependability metrics, was pointed out. Finally, the main research objectives have been identified. The objectives can be summarized as follows:

1. To verify the formal correctness of the statistical models and identify the stated and unstated assumptions on which they rely.
2. To characterize each statistical model based on such properties as the type of estimate provided, its precision and accuracy, the required number of experiments, the strategy to be adopted in case of no-response events.
3. To compare, based on such properties, the proposed models.

This conclusive section is intended to verify how these objectives were all achieved by the analysis carried out in this report.

Section 4 provided a formally well defined probabilistic and statistical framework for the problem of fault coverage estimation. The concepts of fault universe and fault space were introduced and formally related by the use of the axiomatic theory of probability. The fault distribution was then introduced, and several approaches to its estimation were proposed. A mathematical definition of the fault coverage was then provided, which highlighted the importance of the fault distribution in the estimation process. The concepts of forced and sampling distribution were then defined, and their effect on the estimation process was quantified by a characteristic quantity, the covariance ρ. Finally, the set of properties used in the comparison of the statistical models was defined. The concepts described in Section 4, therefore, formed the basis for the formal analysis and comparison of the statistical models presented in the subsequent sections. The most significant results that were obtained are discussed in detail in the conclusions to each section, and are summarized here.

It was determined that all the models that rely on the assumption of a uniform fault distribution lead to estimates of the coverage proportion rather than the system's fault coverage. Otherwise stated, the estimators used in these models are biased by a quantity equal to the covariance ρ. In order for the estimator to be unbiased, it is not necessary but only sufficient that the fault distribution is uniform. The concept of fairness of the fault tolerance mechanism was introduced as the necessary and sufficient condition for the unbiasedness of the estimator.

It was shown that if quantitative information about the system is available, the fault distribution can be approximated by a class-wise uniform distribution and the bias of the estimate greatly reduced. This approach can be used in both the stratified and generalized model. The concept of relative fairness was then stated as the necessary and sufficient condition for the unbiasedness of the estimators used in these models.

It was shown that the simple Bernoulli model and the multi-stage model can be seen as special cases of the generalized model. Also, the stratified approach was shown to provide equivalent results to the generalized model. It was pointed out that the stratified approach, however, can lead to undetermined situations, thus it should only be chosen if it is required for the practical advantages that it produces (such as the possibility of re-use of fault injection data). Finally, since the estimator provided by the generalized model is the uniformly minimum variance unbiased estimator for fault coverage, the generalized model must always be chosen as the best approach to statistical estimation of fault coverage.

An analytical expression for the confidence limit has been provided, which can be applied to the generalized model and its special cases. This avoids using the normal approximation, which has been shown to provide reliable confidence limits only if the rule-of-thumb of **Equation (10.11)** is satisfied, that is only if the number of experiments is sufficiently large or if the coverage is not too high. The analytical expression, instead, always leads to reliable confidence limits.

The required number of experiments has been determined using both the constraint on the variance (useful when the analytical expression for the confidence limits is employed), and the rule-of-thumb (useful when the normal approximation is employed). It has been pointed out that if the normal approximation is used, the common practice of performing 10^x experiments to estimate a non-coverage of 10^{-x} is only sufficient if the desired confidence is about 95%, while for a 99% confidence this number must be increased by one-two orders of magnitude. It was discussed that fault equivalence can reduce the variance of the estimate, and can therefore be employed to reduce the required number of experiments proportionally to the fault equivalence achieved. However, in Section 12 Statistics of the Extremes theory was used to determine the probability of finding at least one uncovered fault in the whole fault injection campaign. It was shown that in order to obtain meaningful data, 10^x experiments are not sufficient, even if the analytical expression is used, as it is only guaranteed with a 0.7 confidence that at least one uncovered fault will be found. However, it was also determined that 10^{x+1} experiments ensures that meaningful data will be obtained.

A model based on vectorial statistics was described for the stratified approach, the results of which appear to be overly conservative. A possible explanation has been identified in the very strict constraint that is chosen to determine the class confidence levels. It was concluded that if the stratified approach is used, the normal approximation must be preferred, provided that the rule-of-thumb is satisfied.

A model based on Bayesian statistics was also discussed for the stratified approach. The results proved to be overly conservative, although much more reliable than the ones obtained with the vectorial statistics model. It was concluded that if the stratified approach is used, the normal approximation must sill be preferred, provided that the rule-of-thumb is satisfied.

It was commented that although numerical analysis through simulated fault injection is an aid to the quantitative and practical understanding of the results produced by different models, a formal justification of such results is always preferable. For the Bernoulli models, in fact, the numerical data always verify the conclusions that can be reached from the formal analysis, while in the case of vectorial and Bayesian statistics the numerical results are not supported by an adequate theoretical development. The formal analysis specifically allows a distinction between errors due to the variance of the estimator and its bias. From numerical results, such distinction is not easy.

In Section 12, the possible use of Statistics of the Extremes for fault coverage estimation was discussed, and it was argued that the model proposed in [17] does not appear to be well formulated.

Finally, for each model a summary table was provided that describes each of the attributes identified in Section 4 to characterize the properties of the statistical models. These tables can be used as a quick reference to determine the best approach to statistical estimation of fault coverage. Typical application examples have also been provided.

14. References

[1] Dhiraj K. Pradhan, *Fault-Tolerant Computer System Design*, Prentice Hall PTR

[2] D. Todd Smith, Barry W. Johnson, Nikos Andrianos, Joseph A. Profeta III, "A Variance-Reduction Technique via Fault-Expansion for Fault-Coverage Estimation", in *IEEE Transactions on Reliability*, vol.46, no. 3, September 1997, pp. 366-374

[3] Lori M. Kaufman, Joanne Bechta Dugan, Barry W. Johnson, "Using Statistics of the Extremes for Software Reliability Analysis", in *IEEE Transactions on Reliability*, vol.48, no. 3, September 1999, pp. 292-299

[4] Lori M. Kaufman, *Dependability Analysis for Ultra-Dependable Systems Using Statistics of the Extremes*, Ph.D. Dissertation, Department of Electrical Engineering, University of Virginia, May 1997

[5] Cristian Costantinescu, "Using Multi-Stage & Stratified Sampling for Inferring Fault-Coverage Probabilities", in *IEEE Transactions on Reliability*, vol. 44, no. 4, December 1995, pp. 632-639

[6] Enrique Castillo, *Extreme Value Theory in Engineering*, Academic Press, Inc.

[7] Alfredo H-S. Ang, Wilson H. Tang, *Probability Concepts in Engineering Planning and Design*, Vol. II, John Wiley & Sons

[8] David Powell, Eliane Martins, Jean Arlat, Yves Crouzet, in "Estimators for Fault Tolerance Coverage Evaluation", in *IEEE Transactions on Computers*, vol. 44, no. 2, February 1995, pp. 261-274

[9] David Powell, Michel Cukier, Jean Arlat, Yves Crouzet, "Estimation of Time-Dependent Coverage", *LAAS-CNRS*, Toulouse 1997

[10] Michel Cukier, Jean Arlat, David Powell, "Frequentist and Bayesian Coverage Estimations for Stratified Fault-Injection", in *Proceeding of DCCA-6*, March 1997, Grainau, Germany, pp. 38-57

[11] David Powell, Michel Cukier, Jean Arlat, "On Stratified Sampling for High Coverage Estimations", in *Proceedings of EDDC-2*, October 2-4 1996, Taormina, Italy, pp. 37-54

[12] W. G. Bouricius, W. C. Carter, P. R. Schneider, "Reliability Modeling techniques for Self-Repairing Computer Systems", in *Proc. 24 th Nat. Conf., ACM*, 1969, pp. 295-309

[13] W. Wang, K. S. Trivedi, B. V. Shah, J. A. Profeta III, "The impact of fault expansion on the interval estimate for fault detection coverage", in *24 th International Symposium on Fault Tolerant Computing*, 1994, pp. 330-337

[14] Fausto Ricci, *Statistica ed elaborazione statistica delle informazioni*, Zanichelli, 1975

[15] A. N. Kolmogorov, *Grund begriffe der Wahrscheinlichkeitsrechnung*, Springer, 1944

[16] K. W. Miller, L. J. Morrel, R. E. Noonan, S. K. Park, D. M. Nichol, "Estimating the Probability of Failure When Testing Reveals No Failure", in *IEEE Transactions on Software Engineering*, Vol. 18, no. 1, January 1992, pp. 33-43

[17] Lori M. Kaufman, Barry W. Johnson, Joanne Bechta Dugan, "Coverage Estimation for an Ultra-Dependable System using Statistics of the Extremes", submitted to *IEEE Transactions on Computers* for publication in 2000.

[18] Constantinescu, C., "Using Multi-Stage and Stratified Sampling for Inferring Fault Coverage Probabilities," *IEEE Transactions on Reliability*, Vol. 44, No. 4, Dec. 1995, pp. 632-639.

[19] Cukier, M., J. Arlat and D. Powell, "Frequentist and Bayesian Estimations with Stratified Fault Injection," *Proceedings of the Sixth IFIP International Working Conference on Dependable Computing for Critical Applications (DCCA-6)*, Mar. 1997, pp. 38-57.

[20] Cutright, E., DeLong, T., Johnson, B., *Numerical Safety Evaluation Process for Safety-Critical Systems*, UVA Technical Report UVA-CSCS-NSE-001.

[21] Choi, Charles Y., Barry W. Johnson, and Joseph A. Profeta III, "Safety Issues in the Comparative Analysis of Dependable Architectures," *IEEE Transactions on Reliability*, Vol. 46, No. 3, September 1997, pp 316-322.

[22] Choi, H., W. Wang, and K.S. Trivedi, "Conditional MTTF and its Computation in Markov Reliability Models," *Proc. 1993 Annual Reliability and Maintainability Symposium*, January 25-28, 1993, pp. 55-63.

[23] Goble, W.M., *Control Systems Safety Evaluation and Reliability*, Instrument Society of America, 1998.

[24] Johnson, Barry W., *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Wesley Publishing Company, 1989.

[25] Pescosolido, Marco, *Comparative Analysis of Statistical Models for Fault Coverage Estimation*, Master's Thesis, University of Virginia Department of Electrical and Computer Engineering, May 2002.

[26] Powell, D., E. Martins, J. Arlat and Y. Crouzet, "Estimators for Fault Tolerance Coverage Evaluation," *The 23rd Annual International Symposium on Fault Tolerant Computing*, June 1993, pp. 229-237.

[27] Powell, D., E. Martins, J. and Y. Crouzet, "Estimators for Fault Tolerance Coverage Evaluation," *IEEE Transaction on Computers*, Vol. 44, No. 2, Feb. 1995, pp. 261-274.

[28] Smith, D., B. Johnson, J. Profeta and D. Bozzolo, "A Method to Determine Equivalent Fault Classes for Permanent and Transient Faults," *Proceedings of the Reliability and Maintainability Symposium*, Jan. 1995, pp. 418-424.

[29] Smith, D. Todd, Barry W. Johnson, Nikos Andrianos, and Joseph A. Profeta III, "A Variance Reduction Technique via Fault Expansion for Fault Coverage Estimation", *IEEE Transactions on Reliability*, Vol. 46, No. 3, September 1997, pp. 366-374.

[30] Smith, D. Todd, Allan White, Todd A. DeLong, Barry W. Johnson, and Ted C. Giras, *A Tutorial on Architectural Analysis Using Reliability and Safety*, Technical Report No. 990609, Center for Safety Critical Systems, University of Virginia, 1999.

[31] Wang, W., K. Trivedi, B. Shah, and J. Profeta III, "The Impact of Fault Expansion on the Internal Estimate for Fault Detection Coverage," *Proceedings of the 24th Annual International Symposium on Fault-Tolerant Computing*, June 1994, pp. 330-337.

Generic Processor Fault Model

Technical Report UVA-CSCS-NSE-004
Revision 00
August 1, 2003

Author(s)	Contact Info
Eric Cutright	edc2u@virginia.edu
Todd DeLong	tad2x@virginia.edu
Barry Johnson	bwj@virginia.edu

University of Virginia Center for Safety-Critical Systems
Department of Electrical and Computer Engineering
Thornton Hall
351 McCormick Road
Charlottesville, VA 22904-4743

Prepared for:
Project Manager: John Calvert
Division of Engineering Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555

Foreword

This generic document presents a Generic Fault Model applicable for a variety of processors commonly used in safety-critical applications. For the Calvert Cliffs Digital Feedwater Control System (DFWCS) safety quantification effort under the Nuclear Regulatory Commission project, the AMD 486DX4 processor discussed in Section 11 of the document is used in the Main and Back-up Azonics Controllers in the DFWCS system. Since no internal detailed design information is available from AMD on the AMD486DX4 processor, the weighting factors for the fault selection process associated with the AMD 486DX4 will be derived based on a uniform distribution as presented in Section 12.2.

Table of Contents

List of Figures

List of Tables

Glossary of Acronyms and Abbreviations

Acronyms and Abbreviations

ADC	Analog to Digital Converter
ALU	Arithmetic and Logic Unit
C	Fault coverage
CISC	Complex Instruction Set Computer
CPU	Central Processing Unit
CS	Chip Select
CSCS	Center for Safety-Critical Systems
E	Event Space
EBI	External Bus Interface
F	Fault Space
FIE	Fault Injection Experiment
FS	Failed-Safe state
FU	Failed-Unsafe state
GPT	General Purpose Timer
HDL	Hardware Description Language
IC	Integrated Circuit
IMB	InterModule Bus
I/O	Input/Output
LIVE	Low Intrusion Verification Environment
MTBHE	Mean Time Between Hazardous Event
MTTF	Mean Time To Failure
MTTFD	Mean Time To Fail Dangerous
MTTHE	Mean Time To Hazardous Event
MTTR	Mean Time To Repair
MTTUF	Mean Time To Unsafe Failure
MUX	Multiplexer
O	Operational state
PC	Program Counter
PFD(t)	Probability of Failure on Demand
PFD_{ss}	Steady-State Probability of Failure on Demand
PLA	Programmable Logic Array
QSM	Queued Serial Module
Q(t)	Unreliability
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
ROM	Read-Only Memory
R(t)	Reliability
SIM	System Integration Module
SRAM	Standby Random Access Memory
SRC	Simple RISC Computer

S_{ss}	Steady-State Safety
S(t)	Safety
TPU	Timer Processor Unit
UVA	University of Virginia
VCU	Virginia Commonwealth University
VHSIC	Very High Speed Integrated Circuit
VHDL	VHSIC Hardware Description Language
XOR	Exclusive OR
$\leftarrow$	an assignment of the value on the right-hand side to the entity on the left-hand side
$\Rightarrow$	the replacement of the expression on the left-hand side with the expression on the right-hand side
$\oplus$	the logical XOR function
$\langle \ldots \rangle$	standard register transfer notation for referring to a bit vector
$(i \ @ \ 0)$	standard register transfer notation for replication, and indicates that the value to the right of the @ is to be repeated i times
#	concatenation: bits on the right are appended to bits on the left

1. Introduction

The University of Virginia (UVA) Center for Safety-Critical Systems (CSCS) has developed a methodology to quantify system safety for complex processor-based safety-critical systems [9]. The approach is built upon a campaign of extensive fault injection which is used to derive a numerical expression of system safety known as the Mean Time To Hazardous Event (MTTHE) [47] (also known alternatively as the Mean Time To Unsafe Failure (MTTUF) [47] or Mean Time Between Hazardous Events (MTBHE) [6][7]). A separate risk assessment methodology is employed at the overall system level to derive MTTHE allocations for individual processor-based systems. The UVA safety quantification methodology is then applied to the individual processor-based systems in order to demonstrate compliance with their MTTHE allocations.

This document describes the third step in the MTTHE compliance process - the development of a generic processor fault model. The high-level generic processor fault model specifies the types of faults (and their associated probabilities) that will be injected into the system under analysis. This process builds upon recent research at UVA in generic processor fault modeling, undertaken with the goal of characterizing low-level internal processor faults at the higher register-transfer level, in order to demonstrate that faults injected on the actual hardware are representative of the low-level processor faults of concern.

Note that for the purposes of this document, the term ***processor*** refers to a microprocessor or a microcontroller. The primary difference between the two is that a microcontroller contains a CPU along with memory-mapped peripherals that are external to the CPU yet are still internal to the microcontroller integrated circuit. In contrast, a microprocessor only contains a CPU; any memory-mapped peripherals will be external to the microprocessor integrated circuit.

This document consists of fifteen sections. Following this introduction, Section 2 presents important background and terminology information, including the development of the MTTHE and fault coverage concepts. Section 3 then provides an overview of the UVA numerical safety evaluation process to demonstrate where the generic processor fault modeling fits into the overall methodology. Then Section 4 presents an overview of the generic processor fault modeling process. Next, Section 5 presents a generic, implementation-independent processor model. Section 6 then presents a behavioral-level fault model for the generic, implementation-independent processor model, and describes the fault space that is generated for this fault model. Next, Section 7 describes how simulation-based fault injection experiments are used to demonstrate the sufficiency of the behavioral-level fault model. The application of the behavioral-level fault model to actual processors commonly used in safety-critical systems is then illustrated in the next sections. The application of the model to the MC68332, MC6809, MC68HC16, and AMD486 processors is presented in Section 8, Section 9, Section 10, and Section 11 respectively. Next, Section 12 illustrates how fault selection weighting factors are derived in general and for a given processor, using the MC68332 processor as an example. Section 13 then describes how faults within the fault space defined by the behavioral-level fault model map to the fault injection environment for injection into the actual system. Next, Section 14 provides a short summary of the document, and finally, Section 15 provides the relevant references published in the literature.

2. Background and Terminology

This section provides background and terminology information which is useful to the understanding of the numerical safety evaluation process. The concept of Mean Time To Hazardous Event (MTTHE) is developed along with several related safety and reliability metrics and their dependent factors.

2.1. Modeling of System Safety

A discussion on the modeling of system safety naturally begins with the definition of safety:

> **Definition 1: Safety,** $S(t)$, is the probability that a system will either perform its functions correctly or will discontinue its functions in a manner that does not disrupt the operation of other systems or compromise the safety of any people associated with the system [23].

There are two important things to note about this definition. First, we define safety as a probability, which implies that safety is a quantitative measure that can take on values between zero and one, inclusive.

$$0.0 \leq S(t) \leq 1.0 \tag{2.1}$$

Second, there is an inherent notion of system states corresponding to whether the system is safe or unsafe. The first state is associated with the system functioning correctly, and we will refer to this state as the Operational State. The second state is associated with the system discontinuing "... its functions in a manner that does not disrupt the operation of other systems or compromise the safety of any people associated with the system." In other words, the system ceases to perform its function, but it does so in a safe manner. We will refer to this state as the Failed-Safe State because the system has failed to perform its function but in a safe manner. The third and final state is essentially all the scenarios that are not included in the first two states. That is, the third state is associated with the system not performing its function correctly, and in addition possibly jeopardizing the safety of the system. We will refer to this third state as the Failed-Unsafe State because like the Failed-Safe State the system has failed to perform its function but has done so in an unsafe manner. The relationship between these three states is shown in **Figure 2.1.**

Referring to **Figure 2.1**, we will assume that the system begins in the Operational state, where it is performing its intended function correctly. According to **Figure 2.1**, the system can transition to either the Failed-Safe state or the Failed-Unsafe state (indicated by the arrows in **Figure 2.1**). Associated with each state transition is a parameter that indicates the rate at which the transition occurs, and as such is referred to as the transition rate. The state transition rate depends on two parameters: (1) the failure rate function, $\lambda(t)$, and (2) the fault coverage, $C(t)$, defined below.

> **Definition 2: Failure rate function,** $\lambda(t)$, represents the rate at which the system fails. For safety studies, this rate considers only those failures that can potentially result in an unsafe system state. The failure rate function may be time-varying or may be a constant. The typical approach is to assume that the failure rate function follows an exponential distribution, resulting in a constant failure rate, λ. One assumption that we will make regarding the failure rate function is that the value of the failure rate function is always

greater than zero. As such, the system will eventually transition from the Operational state to either the Failed-Safe state or the Failed-Unsafe state (that is, the system will eventually fail as $t \rightarrow \infty$).

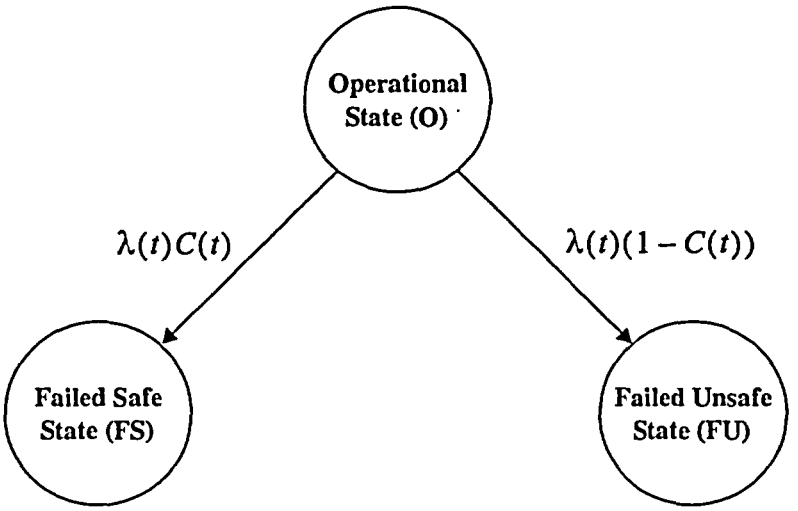


Figure 2.1. Three State System Model used to Calculate Safety

Definition 3: Fault coverage, $C(t)$, is the conditional probability that a system correctly handles a fault, given that a fault has occurred. The fault coverage function may be time-varying or may be a constant. The typical approach is to assume that the fault coverage is a constant, C. And, as with the safety, because fault coverage is a probability, it will take on values between zero and one, inclusive.

$$0.0 \leq C(t) \leq 1.0 \qquad (2.2)$$

Referring to **Figure 2.1**, the transition rate from the Operational state to the Failed-Safe state is $\lambda(t)C(t)$, representing the case where a potentially unsafe fault has occurred and the system has correctly handled the fault and transitioned to a Failed-Safe state. The transition rate from the Operational state to the Failed-Unsafe state is $\lambda(t)(1-C(t))$, representing the case where a potentially unsafe fault has occurred and the system has NOT correctly handled the fault and has transitioned to a Failed-Unsafe state. For safety studies, it is assumed that a potentially unsafe fault which remains undetected by the system will place the system into a Failed-Unsafe state.

From the model of **Figure 2.1**, there are two general classes of safety and reliability metrics which can be computed:

1. The probability that a system is operating in a given state at time t, and
2. The expected time to some event of interest

Common "state probability"-based metrics from the first class include:

- Safety, $S(t)$
- Steady-State Safety, S_{ss}
- Probability of Failure on Demand, $PFD(t)$
- Steady-State Probability of Failure on Demand, PFD_{ss}

Common "expected time to event"-based metrics from the second class include:

- Mean Time To Hazardous Event, *MTTHE*
- Mean Time to Unsafe Failure, *MTTUF*
- Mean Time To Fail Dangerously, *MTTFD*
- Mean Time To Failure, *MTTF*
- Mean Time To Repair, *MTTR*

Each of these metrics will be discussed in the following sections.

2.2. Derivation of Safety (S(t)) Metric

To derive an expression for safety, *S(t)*, refer back to Definition 1. From this definition, we can calculate an expression for safety by determining the probability of being in either the Operational or Failed-Safe states for the simple three state model of **Figure 2.1**. Assuming a constant fault coverage C and constant failure rate λ, the safety expression for this model is given by:

$$S(t) = p_{OP}(t) + p_{FS}(t) = e^{-\lambda t} + (C - C \cdot e^{-\lambda t}) = C + (1 - C) \cdot e^{-\lambda t} \qquad (2.3)$$

where $p_{OP}(t)$ and $p_{FS}(t)$ indicate the probability of being in the Operational and Failed-Safe states, respectively, at time t.

2.3. Derivation of Steady-State Safety (S_{SS}) Metric

Another useful and common metric for quantifying system safety is the steady-state safety, S_{ss}. The steady-state safety may be calculated from the expression for system safety, $S(t)$. Again, from Definition 1, we conclude that the safety for a system which can be represented by **Figure 2.1** is the probability of either residing in the Operational State or the Failed-Safe State.

> **Definition 4:** The **Steady-State Safety, S_{SS},** is a probability representing the evaluation of safety as a function of time, in the limiting case as time approaches infinity:

$$S_{ss} = \lim_{t \to \infty} S(t) \tag{2.4}$$

Note that S_{ss} represents the probability that the system will operate safely after it is placed into service. S_{ss} represents the minimum safety for a safety-critical system and as such provides a lower bound for the system safety under most conditions. It is also a useful metric in that the fault coverage represents a conservative lower bound on steady-state safety, such that this bound can be expressed independently of the system failure rate (which may be difficult to estimate, especially for design faults), in contrast to MTTHE which requires estimation of the system failure rate.

As an S_{ss} example, consider the simple model of **Figure 2.1**, with a constant fault coverage C and constant failure rate λ. In this case, the S_{ss} expression for the system can be derived from the safety expression of **Equation (2.3)** as:

$$S_{ss} = \lim_{t \to \infty} S(t) = \lim_{t \to \infty} (C + (1 - C) \cdot e^{-\lambda t}) = C \tag{2.5}$$

2.4. Derivation of Probability of Failure On-Demand (PFD(t)) Metric

A useful metric for quantifying system safety is the Probability of Failure on Demand, $PFD(t)$. This metric represents the probability that the system has failed in an dangerous manner at a given instant of time. In other words, this metric represents the probability that the system has failed dangerously at the time it is needed to respond to a demand [18]. This metric is particularly useful for quantifying the safety of protection systems (e.g. nuclear reactor protection systems), which perform safety-critical interventions in the event of a system failure to bring the system to a safe state.

The formulation of this metric requires a slight modification to the three-state safety model as illustrated in **Figure 2.2**. Comparing this to **Figure 2.1**, the three states are Operational, Failed-Safe and Failed-Dangerous instead of Operational, Failed-Safe, and Failed-Unsafe. The definition for the Operational state is the same as stated in **Section 2.1.** In this case, the Failed-Safe state refers to the situation where the system shuts down even when there is no potentially dangerous situation. A failure of this type would be considered a safe failure, a false alarm or a false trip. The Failed-Dangerous state covers the case when the system would be unable to respond to a demand, a potentially dangerous condition [18]. Referring to **Figure 2.1**, the failed-unsafe state of this model is replaced with the failed-dangerous state. The remaining states remain unchanged.

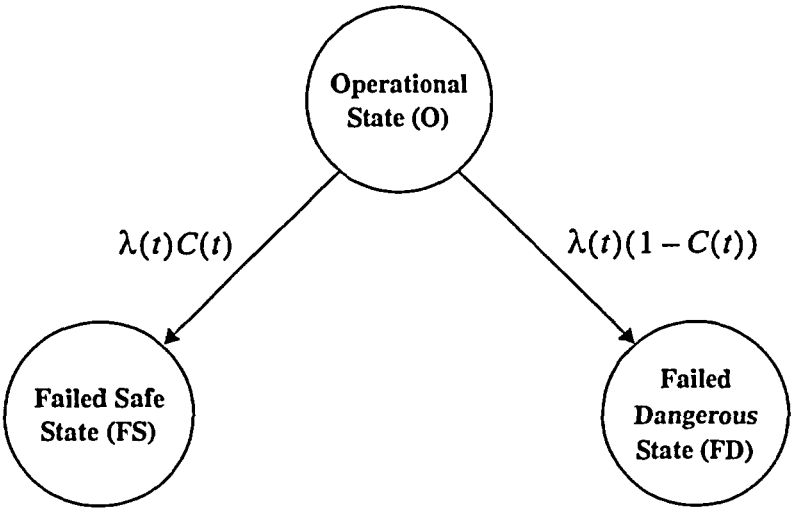


Figure 2.2. Modified Three State System Model used to Calculate MTTFD and PFD

Referring to **Figure 2.2**, we can calculate $PFD(t)$ as the following:

$$PFD(t) = p_{FD}(t) = 1 - p_O(t) - p_{FS}(t) \tag{2.6}$$

As an example of the $PFD(t)$ metric, consider the simple model of **Figure 2.2**, with a constant fault coverage C and constant failure rate λ. In this case, the $PFD(t)$ expression for the system is:

$$PFD(t) = p_{FD}(t) = 1 - p_O(t) - p_{FS}(t) = 1 - e^{-\lambda t} - (C - C \cdot e^{-\lambda t}) = (1 - C) + (1 - C) \cdot e^{-\lambda t} \tag{2.7}$$

2.5. Derivation of Steady-State Probability of Failure On-Demand (PFD_{SS}) Metric

Another useful metric for quantifying system safety is the steady-state Probability of Failure on Demand, PFD_{ss}, which can be calculated from the expression for $PFD(t)$. From **Figure 2.2** we conclude that PFD(t) is the likelihood of residing in the failed-dangerous state.

> **Definition 5:** : The **Steady-State Probability of Failure On-Demand, PFD_{SS}**, is a probability representing the evaluation of PFD as a function of time, in the limiting case as time approaches infinity:
>
> $$PFD_{ss} = \lim_{t \to \infty} PFD(t) \tag{2.8}$$

As an PFD_{ss} example, consider the simple model of **Figure 2.2**, with a constant fault coverage C and constant failure rate λ. In this case, the PFD_{ss} expression for the system is:

$$PFD_{ss} = \lim_{t \to \infty} PFD(t) = \lim_{t \to \infty} [(1 - C) + (1 - C) \cdot e^{-\lambda t}] = (1 - C) \tag{2.9}$$

Note that $PFD(t)$ is related to $S(t)$ (sometimes referred to as Safety Availability, *SA(t)* [18]) by the following relationship:

$$PFD(t) = 1 - S(t) = 1 - SA(t) \tag{2.10}$$

In addition, PFD_{ss} is related to S_{ss} by:

$$PFD_{ss} = 1 - S_{ss} \tag{2.11}$$

2.6. Derivation of Mean Time To Hazardous Event (MTTHE) Metric

For safety studies, a useful mean time metric is the Mean Time to Hazardous Event (MTTHE), also known as the Mean Time To Unsafe Failure (MTTUF) [47]. Here the "event" is an unsafe system failure (as opposed to a system failure for the MTTF). The MTTHE is the primary safety metric that will be used in the UVA numerical safety evaluation process and is defined below:

> **Definition 6:** The **Mean Time To Hazardous Event (MTTHE)** is the expected time that a system will operate before the first *unsafe* failure occurs. Mathematically, the MTTHE is defined as the expected value of a random variable, X, which represents the time to the first unsafe failure. It can be defined as a continuous-time or discrete-time function, but will be defined as a discrete-time function here since that is how it is used later in this document. Given this, then
>
> $$E[X] = MTTHE = \sum X_i \times P_i \tag{2.12}$$
>
> where E is the statistical expectation operator for a probability mass function, and P_i is the likelihood of occurrence of X_i [47].

As an MTTHE example, consider the simple model of **Figure 2.1**, with a constant fault coverage C and constant failure rate λ. In this case, the MTTHE expression for the system is:

$$MTTHE = \frac{1}{\lambda(1-C)} \tag{2.13}$$

Note that MTTHE is equivalent to the metrics Mean Time Between Hazardous Events (MTBHE) and Mean Time To Unsafe Failure (MTTUF), which have been documented in literature [47][7]. As such, MTTHE will be used throughout the remainder of this report, knowing that the results apply equally to MTBHE or MTTUF.

2.7. Derivation of Mean Time to Fail Dangerous (MTTFD) Metric

As with the Probability of Failure on Demand metric, the Mean Time To Fail Dangerous (MTTFD) metric is particularly useful for quantifying the safety of protection systems (e.g. nuclear reactor protection systems), which perform safety-critical interventions in the event of a system failure to bring the system to a safe state. From the model of **Figure 2.2**, the MTTFD metric is defined as:

> **Definition 7:** : The **Mean Time To Fail Dangerous (MTTFD)** is the expected time that a system will operate before the first dangerous failure occurs [18].

Mathematically, the MTTFD can be defined the same manner as MTTHE in **Equation (2.12)**.

As an MTTFD example, consider the simple model of **Figure 2.2**, with a constant fault coverage C and constant failure rate λ. In this case, the MTTFD expression for the system is:

$$MTTFD = \frac{1}{\lambda(1-C)} \tag{2.14}$$

Note that the MTTFD is equivalent to the MTTHE metric presented earlier. This metric is also equivalent to the MTBHE and MTTUF. As such, MTTHE will be used throughout the remainder of this report.

2.8. Other Important Metrics

Other important reliability metrics commonly used in safety-critical applications are shown below. In contrast to safety studies, reliability studies are typically concerned with all types of failures, not just those that can affect the safety of the system. Thus, whereas the failure rate function for safety studies is limited to consideration of only those failures that can potentially result in an unsafe system state, the failure rate function for reliability studies will also reflect other types of failures in the system, especially those that interrupt system service.

> **Definition 8:** The **reliability,** $R(t)$, of a system is a function of time, defined as the conditional probability that the system will perform correctly throughout the interval $[t_0,t]$, given that the system was performing correctly at time t_0 [23].
>
> **Definition 9:** The **unreliability,** $Q(t)$, of a system is a function of time, defined as the conditional probability that the system will perform *incorrectly* during the interval $[t_0,t]$, given that the system was performing *correctly* at time t_0 [23].

Note that the unreliability and reliability of a system are directly related. Specifically, the probability that the system is failed or operational must always equal 1.0. This relationship is expressed as:

$$R(t) + Q(t) = 1.0 \tag{2.15}$$

> **Definition 10:** The **Mean Time To Failure (MTTF)** [23] is the expected time that a system will operate before the *first* failure occurs.

Note that based on this definition, it can be shown that the MTTF for a system can be computed if the reliability of the system, $R(t)$, is known, and $R(t)$ approaches zero as $t \rightarrow \infty$ (from [23]):

$$MTTF = \int_0^\infty R(t)dt \tag{2.16}$$

3. Overview of Numerical Safety Evaluation Process

One quantitative method to demonstrate the MTTHE compliance of a system is the evaluation of the system fault processing capabilities using fault injection. The general approach is to inject randomly selected faults into the system (on a working prototype, for example) to determine if the fault processing capabilities mitigate the faults. This safety quantification approach builds upon the University of Virginia's extensive research experience in this area. The methodology is outlined in **Figure 3.1** and consists of the following steps:

1. An analytical safety model is developed from the system architecture and inter-component dependencies to derive an expression for the MTTHE. The MTTHE expression is a function of *Critical Model Parameters* such as the fault coverage values and failure rates for the various components of the system. The development of the analytical safety model is the subject of this document.
2. A statistical model is then developed based on those found in published literature and is used to estimate the *Critical Model Parameters* that are required by the analytical safety model. This statistical model is also used to calculate the number of fault injection experiments required to meet the numerical safety target for a given confidence level.
3. A high-level generic processor fault model is defined to specify the types of faults (and their associated probabilities) that will be injected into the system under analysis. This process builds upon recent research at UVA in generic processor fault modeling, undertaken with the goal of characterizing low-level internal processor faults at the higher register-transfer level, in order to demonstrate that faults injected on the actual hardware are representative of the low-level processor faults of concern.
4. One or more operational profiles are then defined which will be used to drive the inputs to the system under analysis during the fault injection process. The operational profiles must be representative of the different system configurations and workloads that would be experienced in actual field operation.
5. A fault-free execution trace is created for each selected operational profile that will be used to generate the list of faults to inject into the system under analysis. This trace will also be used during the analysis of the fault injection experimental results.
6. Using the fault-free execution trace and the fault categories from the generic processor fault model, a fault list construction algorithm is applied to generate a list of possible faults that can be injected into the system under analysis, and which are likely to have an effect on the system. From this complete set of responsive faults, a fault list selection algorithm is then applied to randomly select a list of faults to be injected into the system, using the fault categories and associated occurrence probabilities from the generic processor fault model.
7. A fault equivalence algorithm is applied to each fault list to identify those sub-sets of faults which will have the same effect on the system, known as fault equivalence classes. The set of faults to be injected into the system is then reduced using the fault equivalence information, since only one fault from each class needs to be injected. This reduces the amount of time required to perform the evaluation of the system under analysis by reducing the total number of fault injection experiments which must be performed.

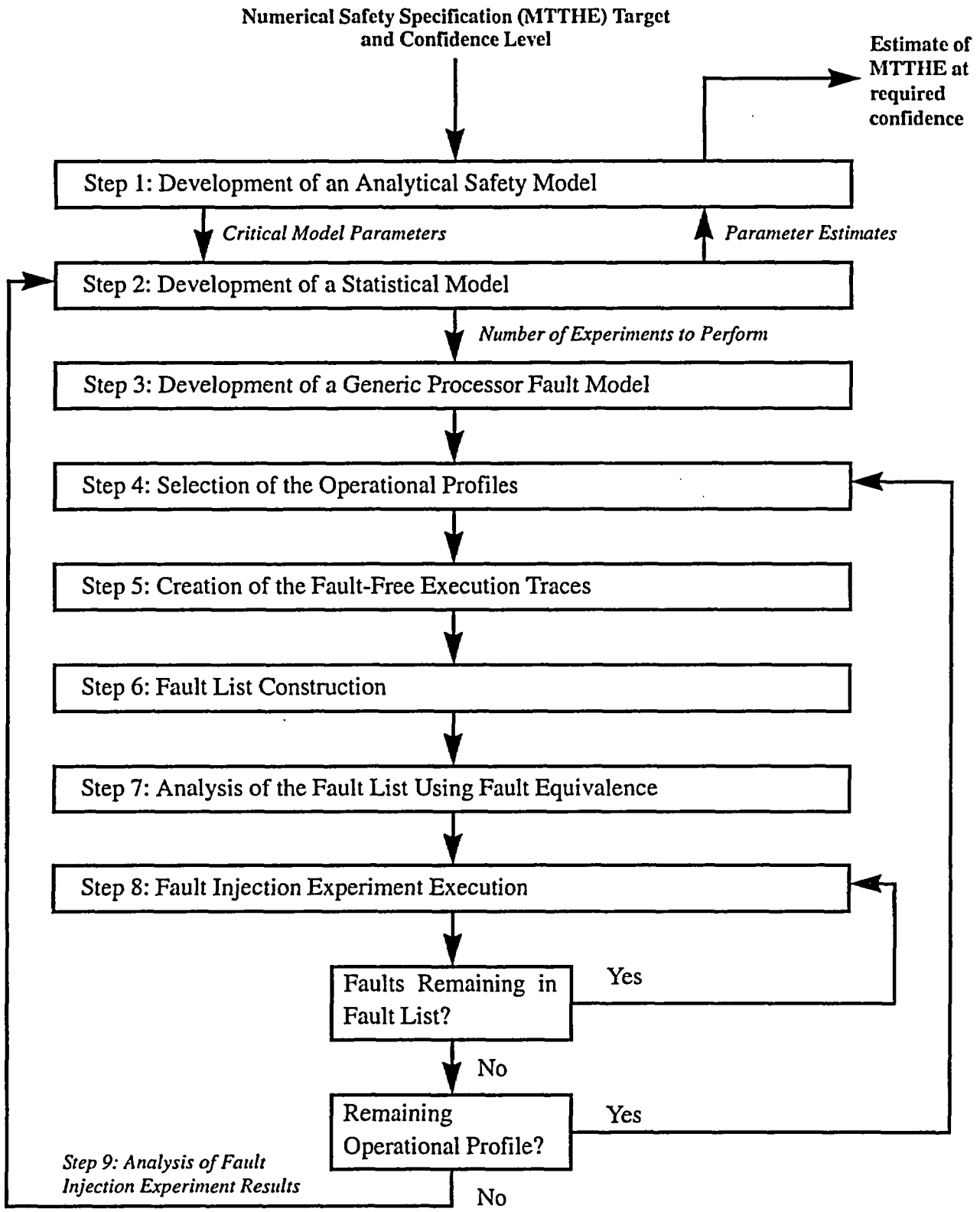


Figure 3.1. Process for quantitative safety evaluation using fault injection

8. Each fault from each reduced fault list is then injected into the system under analysis, using one of four fault injection techniques: (1) hardware-based, (2) software-based, (3) simulation-based, or (4) a hybrid approach.
9. The system is monitored during each fault injection experiment, and each fault is classified as covered, uncovered, or non-responsive by comparing the actual execution trace to the fault-free execution trace. Once all of the results of the fault injection experiments have been collected, they are fed back into the statistical model to calculate the estimates of the *Critical Model Parameters*. The *Parameter Estimates* are then in turn fed back into the analytical model to calculate the estimate of the MTTHE, as shown in **Figure 3.1.**

3.1. Development of a Generic Processor Fault Model

This document describes the third step of the evaluation process outlined in **Figure 3.1,** the development of a generic processor fault model. The purpose of this model is to accurately represent the types of faults that can occur within the system under analysis. The fault model is used to generate the fault space, *F*. *F* is usually a multi-dimensional space whose dimensions can include fault characteristics such as location, time, and value, as shown in **Figure 3.2** [23]. Here, time represents the time of occurrence and duration, location is where the fault occurs within the system under analysis, and value represents the form of the corruption. Note that value can be something as simple as a mask, or something more complex that is state dependent.

In general, completely proving the sufficiency of the fault model used to generate *F* is usually very difficult. The fault modeling of the applied processor is obviously the most problematic, due to the large fault space of a complex processor and the interactions with software. It is more traditional to assume that the fault model is sufficient, justifying this assumption to the greatest extent possible with experimental data, historical data, or results published in literature.

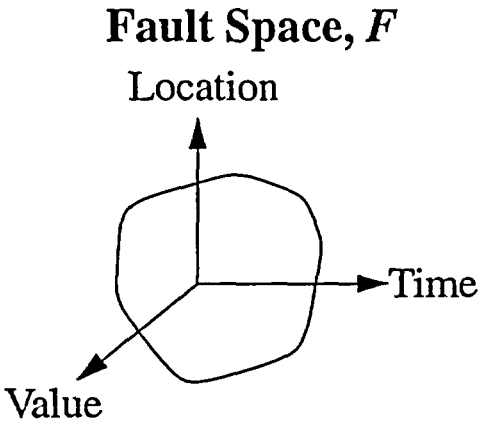


Figure 3.2. The system fault space, *F*, characterized by location, value, and time

To this end, UVA has developed a behavioral-level generic processor fault model that represents the faulty behavior of a general-purpose, implementation-independent microprocessor. The generic processor considered performs a basic fetch-execute instruction cycle typical of a von Neumann architecture. As

such, it contains a control unit, which operates as a synchronous finite state machine, and a datapath, consisting mainly of combinational logic and some storage elements, which performs the information processing within the processor. The datapath will typically contain some sort of register file that contains general-purpose and special-purpose registers, a program counter, an Arithmetic and Logic Unit (ALU), and some sort of fetch and decode logic block. In addition, the processor contains an interface that allows for communication with entities external to the processor. And finally, the processor contains internal signals that allow for communication between the datapath and control unit.

The effectiveness of the generic processor fault model will be demonstrated by developing a gate-level model of an actual 32-bit RISC processor (with a structure similar to the generic processor) and performing simulation-based fault injection experiments to demonstrate that all faulty behaviors that are produced by gate-level faults (for instance, stuck-at faults) can be represented by the behavioral-level fault model. Any unusual faulty behavior discovered as a result of these fault injection experiments will be used to augment the behavioral-level generic processor fault model.

Once the generic processor fault model has been developed, an analysis will be performed to ensure that the experimental environment that will be used to perform the fault injection experiments fully supports the fault model.

4. Generic Processor Fault Modeling Process

The Generic Processor Fault Modeling process consists of two distinct steps: (1) the development of a generic processor fault model to define the fault space in terms of location, time, and value (as shown in **Figure 3.2**) for a generic processor, and (2) the application of the generic processor fault model to the actual processor used in the system under evaluation. These steps are described in the two sections below.

4.1. Development of a Generic Processor Fault Model

Section 5 and Section 6 of this document present the development of a generic processor fault model. As its name implies, this processor fault model is generic and therefore applicable to all processors following the basic architecture presented in Section 5. The generic processor fault model defines the fault space in terms of location, time, and value (as shown in **Figure 3.2**) for a generic processor.

The development of the generic processor fault model will proceed in the following steps:

1. Development of a generic, implementation-independent processor representation containing functional blocks typical of a von Neumann architecture (see Section 5),
2. Definition of the fault space (time, location, and value) for each functional block within the generic processor (see Section 6.2.1 - Section 6.2.6),
3. Validation of the generic processor fault model using a simulation-based approach (see Section 7.3), and
4. Augmentation of the generic processor fault model based on simulation results (see Section 7.4)

The Generic Processor Fault Model provides the following information used in performing fault injection on a given system:

- The identification of the key functional blocks and their associated fault space for a generic processor,
- The probability distributions for corruption types for a given functional block (e.g. single location corruption, multiple location corruptions, terminate fetch/execute, etc.),
- The probability distribution for bitwise corruptions in a given address or data value (scalable to different lengths)

4.2. Application of a Generic Processor Fault Model

The application of the generic processor fault model to generate a set of faults to be injected into a given system will proceed in the following steps:

1. Analysis of specific processor functional blocks and the mapping of these blocks to the generic processor fault model,
2. Derivation of weight factors based on estimated failure rate for the identified processor blocks to be used in the fault selection process,
3. Analysis of the total potentially unsafe failure rate in a given system to determine the percentage of the failure rate to be allocated to the processor and to the remaining components which are external to the processor. These percentages will be used as weighting factors in the fault selection process,
4. Analysis of the fault space to be used in the fault selection process, consisting of duration (transient/permanent), start time, functional block, location within the block (within the block use equal weight for selection) and the corruption values,
5. Derivation of the faults for each location of each block, and definition of the "corruption values" (that is, sequence of actions and masks needed to represent the fault), and
6. Analysis of permanent gate-level faults in a processor model to validate the patterns necessary to inject permanent faults, and
7. Mapping of the derived patterns to the mechanisms for injection of faults by means of the fault injection environment.

The application of the Generic Processor Fault Model to an actual processor requires the following information in order to generate a fault list for a given system:

- The total potentially unsafe failure rate in a given system,
- The percentage of the failure rate to be allocated to the processor and to the remaining components which are external to the processor,
- Derivation of weight factors for individual processor blocks based on either (1) uniform selection if the internal processor architecture details are not known, or (2) weighted selection based on complexity factors if the internal processor architecture details are known,
- The register set and memory addresses used in the actual system, and
- The desired fault injection time window to be used for the experiments.

5. Generic Processor Representation

In general, completely proving the sufficiency of a fault model is usually very difficult. It is more traditional to assume that a fault model is sufficient, justifying this assumption to the greatest extent possible with experimental data, historical data, or results published in literature. To this end, a behavioral-level fault model will be developed that models the faulty behavior of a general-purpose, implementation-independent processor like the one shown in **Figure 5.1.** The processor performs a basic fetch-execute instruction cycle typical of a von Neumann architecture. As such, the **CPU** contains a **Control Unit**, which operates as a synchronous finite state machine, and a **Datapath**, consisting mainly of combinational logic and some storage elements, which performs the information processing within the processor. The **Datapath** will typically contain a **Register File** that contains general-purpose and special-purpose registers, a **Program Counter**, an Arithmetic and Logic Unit (**ALU**), and a **Fetch and Decode Logic Block**. In addition, the **CPU** will contain internal signals that allow for communication between the **Datapath** and **Control Unit**. These are labeled as **Control Signals** and **Status Information** in **Figure 5.1.** Also, the processor may contain one or more **Memory-Mapped Peripheral Functional Blocks** that are external to the **CPU** (but still internal to the processor) that perform such functions as serial communication, timing-related operations, and so forth. Finally, the processor will contain an **External Bus Interface** that allows for communication with entities external to the processor using the **Data Bus**, **Address Bus**, and **Control Bus**.

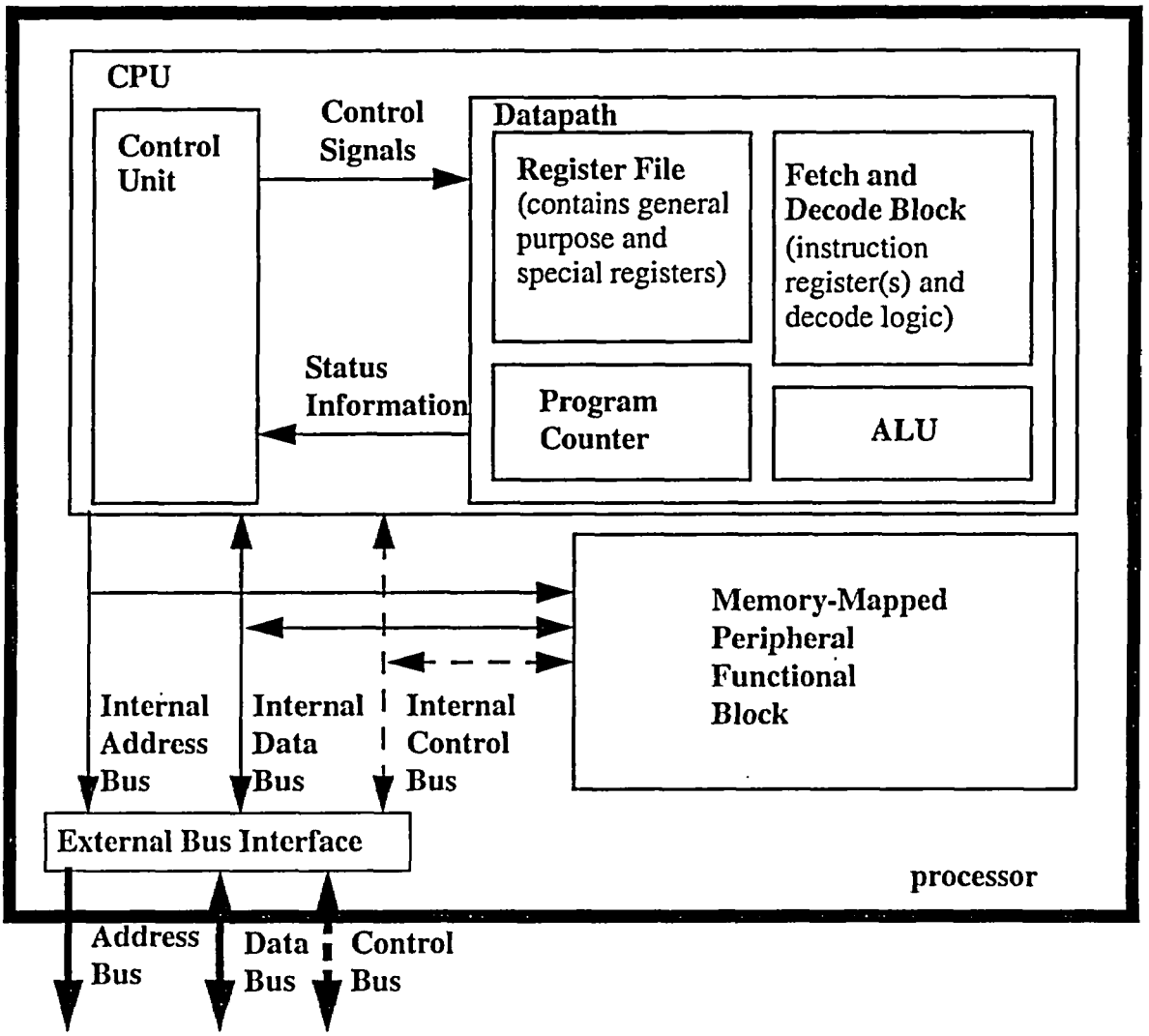


Figure 5.1. Generic, implementation-independent architecture for a general-purpose processor

6. Generic Processor Fault Model Development

6.1. Historical Background

For at least 20 years, researchers have attempted to develop a behavioral-level fault model that accurately represents the faulty behavior of a processor like the one shown in **Figure 5.1.** Most of the research focused on using the fault models for developing test scenarios for detecting the various faulty behaviors defined by the fault model. One of the more significant early efforts was by Thatte and Abraham [52][53][54][55]. They developed an approach to model a processor at a behavioral level, independent of the implementation. They modeled the information flow among the registers within the processor, as controlled by the instruction set, with no consideration of the structural features of the processor. Their work was based on earlier efforts by Robach and Saucier [40][41][42], who developed behavioral fault modeling techniques for a processor control unit, and Sridhar and Hayes [49][50], who developed fault modeling techniques for bit-sliced processors. The fault model developed by Thatte and Abraham was extended by Brahme and Abraham in 1984 [4]. They expanded the fault model for the instruction sequencing to consider the faulty behavior of the individual microinstructions and microsequences that constitute a given processor instruction. They also addressed the issue of the size of the test vector sets that were generated by the fault model developed by Thatte and Abraham. All of this work has been adapted and extended by many researchers, but the work by Yount and Siewiorek [57][58] seems to be the more relevant to this effort. The behavioral-level fault model developed in [57][58] will form the basis for the generic processor fault model for the UVa numerical safety evaluation process for safety-critical systems.

Finally, there is a tremendous amount of published literature that describes other efforts in the area of processor fault modeling. While the efforts might not be directly applicable to this effort, some of them are referenced here for completeness [1][2][8][13][14][15][16][17][19][24][25][26][27][28][29] [32][33] [34][35][36][39][43][44][45][46][48][51][56][59].

6.2. Proposed Fault Model

The most reasonable means available for corrupting the information flow through a processor-based system is through the programmer's model of the processor. This limits corruptions to entities within the processor that are visible within the scope of the programmer's model. For this effort, the primary candidates for corruption are the CPU registers. The challenge is to be able to represent corruptions within the other entities of the processor (for instance, the ALU) by means of the register corruptions. In the past, researchers have attempted to develop such fault models by modeling the various internal entities at a gate or transistor level and performing single-event fault injection experiments. The purpose was to characterize the faulty behavior of the entities at a higher abstraction level in the hope that the errors produced by the single fault events could be represented by corruptions of the inputs to or outputs from the various entities.

While these efforts did produce fault models that represented some aspect of the faulty behavior of the entities, there are two inherent problems with the approach. The first problem has to do with the gate- or transistor-level representation of the internal entities. The researchers had to assume an implementation (or a set of implementations) with no way to verify their assumptions. In general, this is

always going to be true because one seldom has access to the implementation of a design, especially for a commercial processor. Therefore, given that the implementation is unknown and probably not accessible, the developed fault model that is based on an assumed implementation will always be limited in its ability to represent the true faulty behavior of the processor.

The second problem has to do with the assumptions that are made for the gate- or transistor-level model, especially with regard to the fault occurrence. All the researchers assumed a single fault occurrence at the lower abstraction level. While this may be a well-accepted assumption, primarily based on the wide acceptance of the stuck-at fault model [23], the fact of the matter is that this is not necessarily a good assumption to make with the way that today's processors are implemented [37]. In fact, one of the goals for the work by Fung and Patel [37][38] in modeling the faulty behavior of an ALU was to eliminate having to make this assumption. However, they still had to constrain the problem to faulty behavior within a single slice of the ALU in order to develop any type of meaningful results.

Given all of this, the fault model for this effort will assume that fault events (single as well as multiple fault events) that occur within the processor model shown in **Figure 5.1** will manifest as corruptions of the information flow within the model shown in **Figure 5.1**.

As discussed in [9] and [11], the faults within the fault space that will be defined by this fault model have three characteristics - time, location, and value. Time includes the time of occurrence as well as the duration. For time of occurrence, in general real faults can occur at any time. However, the only time faults can affect the information is when the information is read from memory, loaded into a register, and so forth. As such, the two possible times of occurrence for this fault model are an instruction boundary, which includes register references, or a memory reference. For duration, the two durations are transient and permanent, with the fault space apportioned appropriately as discussed in [10] to account for the fact that a majority of the faults that occur within a system are transient in nature. Note that in the case of transient faults, an assumption is being made that the injected fault has a duration of zero, and will instantly produce an error before becoming inactive. That is, in the case of register corruptions, subsequent instructions will not be directly affected by the transient fault, even though they may be affected by the propagation of the effect of the transient. In the case of bus-injected transients, subsequent memory cycles will not be directly affected by the transient. Also note that in the case of modeling a permanent fault, a unique (possibly data-dependent) corruption might have to be performed after every instruction execution (or memory reference) to properly represent the permanent fault. While this may be time-consuming, there is a very good likelihood that most of these situations will be detected very quickly. Thus, even though a corruption is being performed after each instruction execution (or memory reference), very few would need to be performed before errors will be detected, and the experiment will be terminated.

Location refers to any location within the system containing information that is visible and can be corrupted. With regard to the actual fault injection experiments, the possible locations for fault injection are limited to those locations within the processor that are accessible using the programmer's model of the processor and the memory map, and those locations that are visible at the interface to the processor.

Finally, value refers to the value of the corruption mask, where *mask* is a bit vector that is combined with the uncorrupted information to produce the corrupted information. For registers inside the processor

and memory locations in the memory map, the number of bits that will be corrupted by a given mask will be based on the simulation results presented in Section 7.3. While there are approaches described in the literature for determining the number of bits to corrupt (for instance, the 50-30-20 approach described in [3]), there is no data available to justify them. So, it seems more reasonable to use the data that is available from the simulation-based fault injection experiments.

The following sections describe the approach for corrupting the information flow for each entity within the model shown in **Figure 5.1**.

6.2.1. Register File Fault Model

Yount and Siewiorek [57][58] developed a very generic fault model for the register file within a processor. However, due to limited time and resources, they confined their fault injection experiments to a subset of the generic fault model - a register fault model, a read register selection fault model, and a write register selection fault model.

6.2.1.1. Register Fault Model

The register fault model represents corruptions that occur within the registers of the processor as defined by the programmer's model as well as any special-purpose registers that might be part of the memory map.

6.2.1.1.1. Time

The time of occurrence can only be on an instruction boundary.

6.2.1.1.2. Location

Any register that is accessible using the programmer's model of the processor, or any register that appears within the memory map is a potential candidate for corruption.

6.2.1.1.3. Value

With regard to value, there are three scenarios that are considered:

1. Missed load - all or part of a register is not loaded when it should be.
2. Level change in storage - the correct value of one or more bits in the register are complemented.
3. Assign the value of all zeros and all ones to the register.

Scenario 1 can be accomplished by determining a mask value, based on the value of **expr**, that yields the current value of R_k when the mask is XORed with R_k after it receives the value of **expr** as shown in **Equation (6.1)**.

$$R_k \leftarrow \text{expr} \Rightarrow R_k \leftarrow (\text{expr} \oplus (mask\langle(w-1)\ldots 0\rangle)) \tag{6.1}$$

Here, $\leftarrow$ represents an assignment of the value on the right-hand side to the entity on the left-hand side, $\Rightarrow$ represents the replacement of the expression on the left-hand side with the expression on the right-hand side, and $\oplus$ represents the XOR function. The expression, $mask\langle(w-1)\ldots 0\rangle$ is the mask value, computed as a function of the value of **expr** and the current value of R_k. Also, w is equal to the register width, where the w bits of the register are numbered from 0 to $(w-1)$. The notation, $\langle\ldots\rangle$, is standard register transfer notation for referring to a bit vector, as defined in [21].

Scenario 2 is often referred to as the bit-flip fault model in the literature. This scenario can be accomplished by selecting a mask and XORing the mask with the current value in the register as shown in **Equation (6.2)**.

$$R_k \Rightarrow R_k \oplus (mask\langle(w-1)\ldots 0\rangle) \tag{6.2}$$

Scenario 3 can be accomplished by XORing the current value of R_k with itself to create the all zeros corruption value or the complement of itself to create the all ones corruption value, as shown in **Equation (6.3)**.

$$R_k \Rightarrow R_k \oplus R_k$$
$$R_k \Rightarrow R_k \oplus \overline{R_k} \tag{6.3}$$

6.2.1.2. Read Register Selection Fault Model

The read register selection fault model represents faults within the decoding logic that erroneously select a given register (other than the correct register) to be used as an input operand to the current operation, as shown in **Equation (6.4)**.

$$(R_k \leftarrow R_i \text{ op } R_j) \Rightarrow (R_k \leftarrow R_x \text{ op } R_j) \exists (x \neq i)$$
$$(R_k \leftarrow R_i \text{ op } R_j) \Rightarrow (R_k \leftarrow R_i \text{ op } R_x) \exists (x \neq j) \tag{6.4}$$

6.2.1.2.1. Time

The time of occurrence can only be on an instruction boundary.

6.2.1.2.2. Location

Any register that is accessible using the programmer's model of the processor, or any register that appears within the memory map is a potential candidate for corruption.

6.2.1.2.3. Value

With regard to value, the fault scenario can be represented by XORing the portion of the instruction that contains the input register selection information with a selected mask before the instruction is executed, as shown in **Equation (6.5)**,

$$\text{instr\_fetch(addr)} \Rightarrow \text{instr\_fetch(addr)} \oplus (((x-a-i) \ @ \ 0) \# mask\langle (i+a-1)\ldots i \rangle \# (i \ @ \ 0)), \tag{6.5}$$

where x is the width of the instruction, i is the starting bit position of the input register selection field, and a is the width of the field. The notation, $(i \ @ \ 0)$, is register transfer notation for replication, and indicates that the value to the right of the @ is to be repeated i times and concatenated, which is represented by #.

6.2.1.3. Write Register Selection Fault Model

The write register selection fault model represents faults within the decoding logic that erroneously selects a given register (other than the correct register) to be used as the output operand for the current operation, as shown in **Equation (6.6)**.

$$(R_k \leftarrow R_i \text{ op } R_j) \Rightarrow (R_x \leftarrow R_i \text{ op } R_j)\exists(x \neq k) \tag{6.6}$$

6.2.1.3.1. Time

The time of occurrence can only be on an instruction boundary.

6.2.1.3.2. Location

Any register in the programmer's model of the processor that can be designated as a destination register or any register that appears within the memory map that can be written to is a potential candidate for corruption.

6.2.1.3.3. Value

With regard to value, the fault scenario can be represented by XORing the portion of the instruction that contains the output register selection information with a selected mask before the instruction is executed, as shown in **Equation (6.7)**,

$$\text{instr\_fetch(addr)} \Rightarrow \text{instr\_fetch(addr)} \oplus (((x-b-j) \ @ \ 0)\#mask\langle(j+b-1)\ldots j\rangle\#(j \ @ \ 0)), \tag{6.7}$$

where j is the starting bit position of the output register selection field, and b is the width of the field.

6.2.2. Program Counter Fault Model

The Program Counter (PC) fault model represents corruptions that occur within the PC of the processor.

6.2.2.1. Time

The time of occurrence can only be on an instruction boundary.

6.2.2.2. Location

The PC, which is accessible using the programmer's model of the processor and the instruction set defined for the processor. For instance, if the PC is not directly accessible using the programmer's model of the processor, then it can be corrupted in a variety of ways using the instruction set of the processor. First, branch/jump instructions could be inserted into the source code to manipulate the PC to the desired corrupted value. Or, another approach would be to create an interrupt, and then corrupt the return address that gets pushed onto the stack (which is loaded into the PC upon return from the interrupt). Finally, the PC can be corrupted when it is placed on the address bus to fetch an instruction.

6.2.2.3. Value

With regard to value, there are three scenarios that are considered:

1. Missed load - all or part of a register is not loaded when it should be.
2. Level change in storage - the correct value of one or more bits in the register are complemented.
3. Assign the value of all zeros and all ones to the register.

Scenario 1 can be accomplished by determining a mask value, based on the value of **expr**, that yields the current value of the PC when the mask is XORed with the PC after it receives the value of **expr** as shown in **Equation (6.8)**.

$$PC \leftarrow \text{expr} \Rightarrow PC \leftarrow (\text{expr} \oplus (mask\langle (w-1)\ldots 0\rangle)) \tag{6.8}$$

Scenario 2 is often referred to as the bit-flip fault model in the literature. This scenario can be accomplished by selecting a mask and XORing the mask with the current value in the register as shown in **Equation (6.9)**.

$$PC \Rightarrow PC \oplus (mask\langle (w-1)\ldots 0\rangle) \tag{6.9}$$

Scenario 3 can be accomplished by XORing the current value of PC with itself to create the all zeros corruption value or the complement of itself to create the all ones corruption value, as shown in **Equation (6.10)**.

$$\begin{aligned} PC &\Rightarrow PC \oplus PC \\ PC &\Rightarrow PC \oplus \overline{PC} \end{aligned} \tag{6.10}$$

6.2.3. Control Unit/Instruction Decode Fault Model

There are at least two instances where the results of the instruction decode/fetch operations can be corrupted. The first is in the fetching of the instruction from memory, and the second is the decoding of the instruction that is performed by a decoder block within the processor. In both cases, an appropriate fault model is to select a mask that is XORed with the uncorrupted instruction, similar to the bus fault model.

6.2.3.1. Time

The time of occurrence could occur on an instruction boundary or a memory reference.

6.2.3.2. Location

With regard to location, any location that contains an instruction, as defined by the system memory map, is a potential candidate for a corruption. Note that this also includes the instruction register within the processor (see **Fetch and Decode Block** in **Figure 5.1**).

6.2.3.3. Value

With regard to value, the fault model for the instruction fetch/decode block is shown in **Equation (6.11)**,

$$\text{instr\_fetch(addr)} \Rightarrow \text{instr\_fetch(addr)} \oplus (((x-c-k) \ @ \ 0)\#mask\langle(k+c-1)\ldots k\rangle\#(k \ @ \ 0)), \quad (6.11)$$

where k is the starting position of the operation code field, and c is the width of the operation code. Note that values for k and c may vary, depending on the format of a given instruction. For instance, in a RISC processor, the instruction format is fixed, and thus the values for k and c would be fixed. However, for a CISC processor, the instruction format will vary, depending on the operation, number of parameters, and so forth. A graphical depiction of this is shown in **Figure 6.1**. Thus, the fault model described by **Equation (6.11)** is applicable as long as its understood that values for k and c will have multiple values for a given set of fault injection experiments, the values being determined by the formats of the instructions that are to be corrupted.

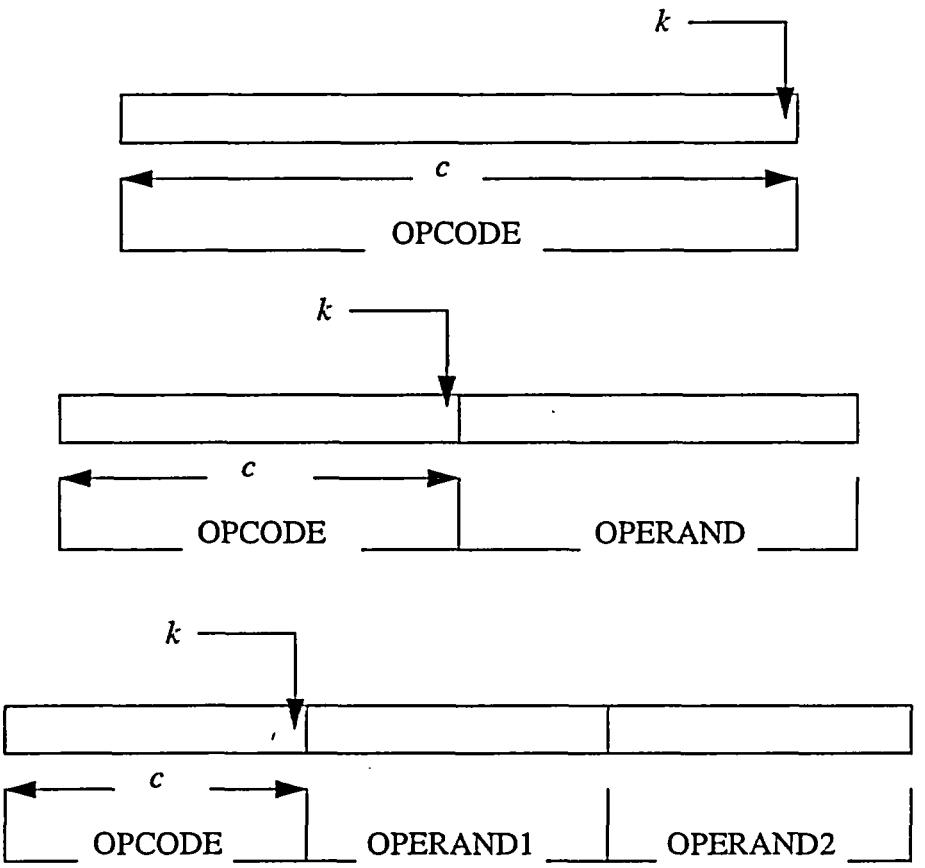


Figure 6.1. Values for *k* and *c* for various CISC processor instruction formats.

6.2.4. Bus Fault Model

The bus fault model represents corruptions that occur on the **Data Bus** and **Address Bus** for the processor model shown in **Figure 5.1**, both internal and external.

6.2.4.1. Time

The time of occurrence will be on a reference to a location within the memory map.

6.2.4.2. Location

Any location within the memory map is a potential candidate for corruptions.

6.2.4.3. Value

With regard to value, the correct value of one or more bits on the data bus or address bus will be complemented to represent the corruption. This is often referred to as the bit-flip fault model in the literature. For the address bus, the address value will be XORed with a selected mask before a memory or I/O read or write, as shown in **Equation (6.12) - Equation (6.15)**,

$$\text{mem\_write(addr,data)} \Rightarrow \text{mem\_write(addr} \oplus (mask\langle(y-1)\ldots 0\rangle)\text{,data)} \tag{6.12}$$

$$\text{mem\_read(addr)} \Rightarrow \text{mem\_read(addr} \oplus (mask\langle(y-1)\ldots 0\rangle)) \tag{6.13}$$

$$\text{output(addr,data)} \Rightarrow \text{output(addr} \oplus (mask\langle(y-1)\ldots 0\rangle)\text{,data)} \tag{6.14}$$

$$\text{input(addr)} \Rightarrow \text{input(addr} \oplus (mask\langle(y-1)\ldots 0\rangle)) \tag{6.15}$$

where y is the width of the address bus.

For the data bus, the data value will be XORed with a selected mask before a memory or I/O write or after a read, as shown in **Equation (6.16) - Equation (6.19)**,

$$\text{mem\_write(addr,data)} \Rightarrow \text{mem\_write(addr,(data} \oplus (mask\langle(z-1)\ldots 0\rangle)) \tag{6.16}$$

$$\text{mem\_read(addr)} \Rightarrow \text{mem\_read(addr)} \oplus (mask\langle(z-1)\ldots 0\rangle) \tag{6.17}$$

$$\text{output(addr,data)} \Rightarrow \text{output(addr,(data} \oplus (mask\langle(z-1)\ldots 0\rangle)) \tag{6.18}$$

$$\text{input(addr)} \Rightarrow \text{input(addr)} \oplus (mask\langle(z-1)\ldots 0\rangle) \tag{6.19}$$

where z is the width of the data bus.

Note that in **Equation (6.12)-Equation (6.15)** and **Equation (6.16)-Equation (6.19)**, the multiple faulty behaviors that are described represent behaviors that are exclusive to one another. Thus, as a for instance, for a given bus access, a corruption on the address bus will be represented by one (and only one) of the faulty behaviors listed in **Equation (6.12)-Equation (6.15)**, depending on the function being performed by the system. If a write to memory is being performed, then **Equation (6.12)** is used. If a read from memory is being performed, then **Equation (6.13)** is used. If a write to an output device is being performed, then **Equation (6.14)** is used. And, finally if a read from an input device is being performed, then **Equation (6.15)** is used. Likewise for corruptions on the data bus and **Equation (6.16)-Equation (6.19)**.

6.2.5. ALU Fault Model

Consider a model for a general-purpose instruction format for arithmetic and logic operations, as shown in **Equation (6.20)**,

$$D \leftarrow S_1 \text{ op } S_2, \tag{6.20}$$

where D is the destination of the result for the operation, equal to

$$D = R_i \text{ for } i = 1, \ldots, \text{number of registers}$$

or

$$D = \text{memory\_write(addr,data)}.$$

S_1 and S_2 are the sources for the operation, equal to

$$S_1, S_2 = R_i \text{ for } i = 1, \ldots, \text{number of registers}$$

or

$$S_1, S_2 = \text{memory\_read(addr)}.$$

Possible locations for corruptions include D, S_1, and S_2, as well as the operation, which could be corrupted to another valid instruction or an invalid instruction. In general, the corruption of these various locations is accounted for in the fault models that have been discussed previously. For instance, source and destination register corruptions are accounted for in the register fault model described in Section 6.2.1. Likewise, memory access corruptions are accounted for with the bus fault model described in Section 6.2.4, and operation corruption is accounted for with the instruction decode fault model described in Section 6.2.3. Given all of this, the faulty behavior of the ALU is just a subset of the faulty behavior defined for other entities within the processor, and as such has been accounted for by the fault models defined for the other entities.

6.2.6. Memory-Mapped Peripheral Functional Block Fault Model

Since any memory-mapped peripheral functional block is visible to the CPU via registers defined in the memory map, either the register fault model defined in Section 6.2.1 or the bus fault model in Section 6.2.4 can be used to model the faulty behavior of any peripheral functional blocks that exist within the processor model shown in **Figure 5.1**. If the register fault model is used, it is necessary to identify the individual registers which are used by a given memory-mapped device access. If the bus fault model is used, it is only necessary to perform corruptions at the bus interface level. As both approaches are equally correct and the bus fault model is the simpler approach, the bus fault model is the preferred approach.

7. Validation of the Generic Processor Fault Model

Given the fault model described in Section 6, the approach to demonstrate the sufficiency of the fault model involves demonstrating that the faulty behavior defined by the generic processor fault model represents all of the faulty behavior of an actual processor. One approach for demonstrating the sufficiency of the fault model is to perform simulation-based fault injection experiments on a model of the system under analysis. The purpose of the experiments is to demonstrate that all of the faulty behavior for a realistic processor is represented within the fault model. Unfortunately, simulation models of actual processors that could be used to perform the fault injection experiments are generally not publicly available. Thus, the approach is to use a model of a general-purpose processor that is available, which contains the same architectural features, and use it to perform the simulation-based fault injection experiments to validate the generic processor fault model.

Note that demonstrating that the fault model represents all of the faulty behavior of interest is related to the assumptions that are made regarding the faulty behavior of the system under analysis. For this validation effort, the primary assumption being made is that the gate-level fault model that is used for the simulation-based fault injection experiments is sufficient to represent all of the faulty behavior of interest. The gate-level fault model that will be used is the stuck-at fault model [23], which is prevalent in industry as the most commonly used and accepted fault model.

7.1. Overview of Validation Approach

The first step in the effort to demonstrate the sufficiency of the fault model is to demonstrate that the processor model used for the simulation-based fault injection experiments is capable of correctly executing a test program before the model has been synthesized (to a gate-level description). The purpose of this step is to demonstrate that the processor model (before synthesis) contains the correct functionality (that is, the correct fault-free behavior). The test program will be developed such that it exercises, to the greatest extent possible, the architectural features and resources of the processor.

Once it has been demonstrated that the processor model correctly executes the test program before synthesis, the next step is to demonstrate that the same test program is correctly executed after the model has been synthesized. The primary difference between the pre- and post-synthesis versions of the processor model (other than level of abstraction) is that timing is added to the post-synthesis version, based on the propagation delay of the gate-level components contained in the processor model.

Once it has been demonstrated that the synthesized, gate-level processor model correctly executes the test program, the next step is to demonstrate that the model can be used to perform fault injection experiments. A small number of fault injection experiments will be performed on signals within the processor model that will definitely produce an error if corrupted. The purpose of performing these experiments is to demonstrate that the processor model, which has been modified to support fault injection, is capable of correctly modeling the selected gate-level faults (as defined by the stuck-at fault model) within the processor model.

Finally, this same small number of fault injection experiments will be used to demonstrate that the fault injection experiments are capable of being performed in an automated fashion using tools developed at UVa. In the previous step, the fault injection experiments were performed in an interactive fashion

using the graphical tools provided by the VHDL simulator. For example, determining if a given signal has been correctly corrupted with a stuck-at fault is a fairly simple thing to do. One need only view the signal (in a waveform, for instance) before and after the fault is injected to demonstrate that it is, indeed, corrupted to the desired faulty value. However, because of the large number of fault injection experiments that is required for this effort, it is not feasible (or practical) to perform them all in such a fashion. As such, the fault simulation tool developed at UVa provides a means to perform fault injection experiments in an automated fashion. The crux to such an endeavor is to develop a means to determine when the fault produces an error by a means other than visual confirmation. This is usually accomplished by defining a correct set of outputs (for a given time period), and comparing the current outputs to the defined correct outputs. The correct set of outputs will be determined during the second step where the correct execution of the test program on the synthesized, gate-level model is demonstrated.

Once these four steps have been performed, then the sufficiency of the behavioral-level processor fault model will be demonstrated using simulation-based fault injection experiments. That is, fault injection experiments using the stuck-at fault model will be performed, and the results will be analyzed to demonstrate that the behavioral-level fault model represents the faulty behavior of the processor model shown in **Figure 5.1.** If a faulty behavior is identified from the stuck-at fault injection experiments that is not represented in the behavioral-level fault model, then the fault model will be augmented to support the additional faulty behavior.

7.2. Description of Available VHDL Model

The processor model that will be used to demonstrate the sufficiency of the fault model is a model of a general-purpose, 32-bit Reduced Instruction Set Computer (RISC) known as the Simple RISC Computer (SRC), described in detail in [21]. The SRC contains all of the entities within the CPU model shown in **Figure 5.1.** A synthesizable version of the model, described using a Hardware Description Language (HDL) known as the Very High Speed Integrated Circuit (VHSIC) HDL (VHDL) [22], has been obtained for this effort [30].

Note that a simulation model of the SRC is being used to demonstrate the sufficiency of the fault model, as opposed to a simulation model of a CISC processor like the MC68332 (models for which are not publicly available). As such, an assumption is being made in using the SRC that the results of the FIEs on the RISC processor are representative of the types of functional faults that occur with a Complex Instruction Set Computer (CISC) processor like the MC68332.

7.3. Analysis of Simulation FIE Results

A total of 10,900 simulation-based FIEs were performed using the VHDL model of the SRC RISC processor described in Section 7.2. Each experiment represents one gate-level, stuck-at fault, permanently injected into the hardware. The decomposition of the FIEs according to the generic processor model shown in **Figure 5.1** is shown in **Table 7.1.** As can be seen in **Table 7.1**, the FIEs are comprised of 934 experiments within the control unit (8.6%), 630 experiments within the program counter (5.8%), 5,261 experiments within the ALU (48.3%), 699 experiments within the fetch/decode logic (6.4%), and 3,376 experiments within the bus interface, which includes both the internal and external data, address, and control busses, as well as the memory address register and the memory data register (30.9%).

Processor Component from Figure 5.1	Number of FIEs	Percentage of total
Program Counter	630	5.8%
Control Unit	934	8.6%
Fetch and Decode Logic Block	699	6.4%
Bus Interface (including Data Bus, Address Bus, and Control Bus)	3,376	31.0%
ALU	5,261	48.3%
Total	10,900	100%

Table 7.1. Decomposition of FIEs according to generic processor in Figure 5.1.

Note that there were no FIEs performed within the register file of the processor model. Our argument for not performing FIEs within the register file is that we know that faults in the register file will affect one or more of the registers. Thus, our belief is that any results obtained from FIEs within the register file will not produce any additional faulty behaviors that are not already accounted for within the fault model definition described in Section 6.2. As such, the decision was made to perform FIEs on the other components since our belief is that they will provide more meaningful results. Also note that this same argument is being used to justify not performing any FIEs in any memory-mapped peripherals as well.

7.3.1. FIE Results

The results of the FIEs can be categorized into the following four categories:

1. No Response
2. Single corruption
3. Multiple corruptions
4. Terminate Fetch and Execute Process

The decomposition of the FIE results into these four categories is summarized in **Table 7.2. Table 7.3** presents the distribution of all of the FIE results, both those supported by the fault model and those not supported by the fault model. Here, *supported* means that the first corruption produced by the injected fault corresponds to the proposed faulty behavior as described by the appropriate fault model in

Section 6.2, for a given functional block. Likewise, *not supported* means that the first corruption produced by the injected fault does not correspond to the proposed faulty behavior in Section 6.2. These unsupported faulty behaviors include single corruptions outside the functional block where the fault was injected, multiple corruptions, and the termination of the fetch/execute cycle (see **Table 7.2**). The faulty behaviors that are not supported will be used to augment the proposed fault model in Section 6.2.

Table 7.4 presents a distribution of the FIE results for the responsive faults that are supported by the proposed fault model in Section 6.2. For instance, referring to **Table 7.4**, for the first corruption that occurred as a result of a fault injected into the ALU functional block, 99.33% correspond to the proposed fault model in Section 6.2.5. The remaining 0.67% of the corruptions correspond to faulty behaviors that are not supported by the ALU fault model.

7.3.1.1. No Response Results

The injected fault had no effect on the program being executed. That is, the fault-free and faulty execution traces matched exactly. Here, ***trace*** represents a listing of the times of occurrence and the values for the data bus, the address bus, and all of the registers within the processor, including those not visible using the programmer's model, for the duration of the simulation. With regard to the no-response faults, since these faults produced no effect on the simulation output, they require no further analysis. Note that the number of no-response faults is consistent with what others have found in published literature when they have performed these types of experiments. Reducing the number of no-response faults requires either applying additional inputs or executing additional programs to further explore the fault space of the processor.

7.3.1.2. Single Corruptions

At the first instance where the faulty trace differs from the fault-free trace, there is a difference in only one location. This faulty behavior corresponds to the respective proposed fault models listed in **Table 7.4** and described in Section 6.2. However, there are single corruptions that do not correspond to the proposed fault models described in Section 6.2. In particular, there are instances where the injected fault produces a corruption that is outside of the functional block where the fault is injected. The distribution of these single corruptions are listed in **Table 7.5**.

While these corruptions may not appear to not conform to the fault model described in Section 6.2, a further analysis of the results shows that many of the results can be explained as propagations of the initial corruption. For instance, consider the 69 corruptions listed in **Table 7.5** associated with the **Fetch and Decode Logic Block**. As shown in **Table 7.6**, all of these corruptions occur in the PC (11 corruptions), the register file (57 corruptions), or on the address bus (1 corruption). Such corruptions could occur if information contained within the current instruction stored in the instruction register (which is uncorrupted) is corrupted before it is placed in its destination (that is, the PC, a register in the register file, or the address bus). For instance, if the instruction contains information that is to be stored in a register, the information could become corrupted at the output of the instruction register. So, the

information stored within the instruction register is uncorrupted, but the information stored in the register is corrupted. This is shown in **Figure 7.1.**

Also, consider the 589 corruptions listed in **Table 7.5** associated with the **Bus Interface.** As shown in **Table 7.7,** all of these corruptions occur in either the instruction register (568 corruptions) or the register file (21 corruptions). These corruptions can be explained in a manner similar to the fetch/decode corruptions described above. For instance, processors often contain buffer registers within the bus interface (but not necessarily part of the programmer's model) that are used to place address information on the address bus or obtained data from the data bus. A scenario that could explain these corruptions is for the information to become corrupted after it is stored in these buffer registers. Thus, the information on the busses is uncorrupted, but the information stored in the buffer registers is corrupted, which is what gets stored in the instruction register or the register file. This scenario is very similar to the scenario shown in **Figure 7.1,** where the busses replace the instruction register as the source of uncorrupted information.

Finally, consider the 8 corruptions listed in **Table 7.5** associated with the **ALU.** As shown in **Table 7.8,** all of these corruptions occur on the address bus. These corruptions can be explained in a manner similar to the fetch/decode corruptions and the bus interface corruptions. For instance, processors often contain an accumulator register at the output of the ALU (that is usually not part of the programmer's model) that stores the result computed by the ALU. If corruptions occurred within this register, they would not be observed until they propagated to other units within the processor, such as the result of an address computation that is placed on the address bus. Again, this scenario is very similar to the scenario shown in **Figure 7.1,** where the ALU replaces the instruction register as the source of uncorrupted information, and the address bus replaces the register file as the destination of the corrupted information.

Of all the single corruptions shown in **Table 7.5,** the only set that cannot be explained by propagation is those associated with the control unit. As shown in **Table 7.9,** while some of the corruptions do support the fault model proposed in Section 6.2 (that is, the instruction register is corrupted), the large majority do not. In addition to the instruction register being corrupted, the FIE results shown in **Table 7.5** show that any register within the programmer's model, as well as the data and address busses, are potential locations for corruptions. Thus, the fault model for the control unit (described in Section 6.2.3) needs to be augmented to account for this observed behavior.

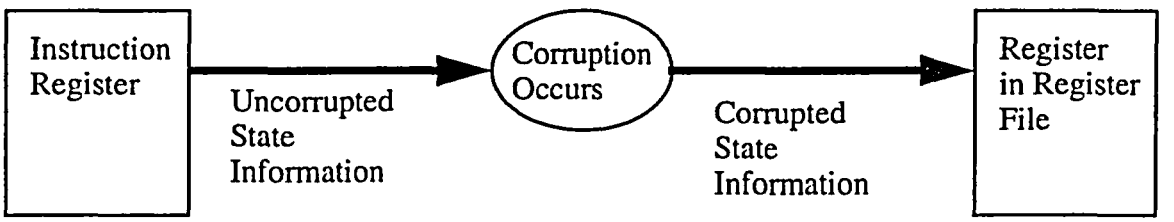


Figure 7.1. Example of register file corruption where instruction register remains uncorrupted.

7.3.1.3. Multiple Corruptions

At the first instance where the faulty trace differs from the fault-free trace, there is a difference in more than one location. **Table 7.5** shows the distribution of locations of the multiple corruptions within the processor.

An additional analysis was performed on the 350 multiple corruption results listed in **Table 7.5** to determine if the multiple corruptions were independent or correlated. The supposition was that these multiple corruptions were, in fact, instances in which a fault caused the incorrect register in the register file to be loaded, thereby corrupting two locations. That is, the correct destination register, as specified in the destination field of the instruction, was not properly updated, and an incorrect register was actually updated as a result of the effect of the fault.

In order to validate the supposition, the following four conditions must be satisfied for all 350 FIEs that produced multiple corruptions:

1. The instruction register is uncorrupted.
2. Two and only two corruptions are present, both in the register file.
3. The current instruction is updating a register within the register file.
4. The updated register is one of the two registers corrupted.

After applying condition 1, 156 of the 350 failed. Of the 194 that passed condition 1, 148 failed condition 2. Of the 46 that passed conditions 1 and 2, all 46 passed condition 3. Finally, after applying condition 4, 23 failed, resulting in 23 that passed all four conditions. Therefore, 23 of the 350 multiple corruptions could potentially be correlated and thus supported by the fault model, with the remaining 327 unsupported. Thus, it is necessary to augment the fault model to support this multiple-corruption faulty behavior, which is currently not represented within the fault model.

7.3.1.4. Terminate Fetch and Execute Process

The injected faults cause the processor to halt the fetch-execute cycle prematurely, and thus the processor system ceases to continue to perform any operation. Note, however, that the faulty and fault-free execution traces match up to the point where the faulty trace prematurely ends. **Table 7.5** shows the frequency of this behavior. The fault model does not currently contain any such faulty behavior. Thus, it will need to be augmented to account for this type of faulty behavior.

Hardware Functional Block	No Response	Single corruption	Multiple corruptions	Terminate Fetch and Execute Process	Total
Register File	na	na	na	na	na
Program Counter	257	351	0	22	630
Control Unit	136	419	245	134	934
Fetch and Decode Logic Block	50	604	45	0	699
Bus Interface	2079	1195	41	61	3376
ALU	1205	4037	19	0	5261
Memory-Mapped Peripheral Devices	na	na	na	na	na
Total	3727	6606	350	217	10900
Percentage of Total (%)	34.19	60.61	3.21	1.99	100

Table 7.2. Distribution of FIE results according to four categories listed in Section 7.3.1.

Type of Corruption	Frequency	% of total
Terminate Fetch and Execute Process	217	1.99
Single Corruption - Supported by fault model	5565	51.06
Single Corruption - Not supported by fault model	1041	9.55
Multiple Corruptions - 2	182	1.67
Multiple Corruptions - 3	53	0.49
Multiple Corruptions - 4	42	0.39
Multiple Corruptions - 5	64	0.59
Multiple Corruptions - 6	9	0.08
No Response	3727	34.19
Total	10900	100

Table 7.3. Distribution totals of the categories of FIE results.

Hardware Functional Block	Fault Model	Frequency	% Supported	% Not Supported
Register File	Section 6.2.1	na	100	0
Program Counter	Section 6.2.2	351	94.10	5.90
Control Unit	Section 6.2.3	44	5.51	94.49
Fetch and Decode Logic Block	Section 6.2.3	535	82.43	17.57
Bus Interface	Section 6.2.4	606	46.72	53.28
ALU	Section 6.2.5	4029	99.33	0.67
Memory-Mapped Peripheral Devices	Section 6.2.6	na	100	0

Table 7.4. Distribution of FIE results supported by the proposed fault model in Section 6.2.

Hardware Functional Block	Terminate Fetch and Execute Process	Single Corruption	Multiple Corruptions				
		Different Functional Block	2	3	4	5	6
Register File	na	na	na	na	na	na	na
Program Counter	22	0	0	0	0	0	0
Control Unit	134	375	88	48	42	58	9
Fetch and Decode Logic Block	0	69	39	0	0	6	0
Bus Interface	61	589	40	1	0	0	0
ALU	0	8	15	4	0	0	0
Memory-Mapped Peripheral Devices	na	na	na	na	na	na	na

Table 7.5. Distribution of FIE results not supported by the proposed fault model in Section 6.2.

First Corruption	Second Corruption	Frequency	Total
No Response		50	50
Multiple Corruption		45	45
Termination		0	0
Address Bus	No Corruption	0	1
	Address Bus	0	
	Data Bus	0	
	Instruction Register	0	
	Program Counter	0	
	Register File	0	
	Multiple Corruption	0	
	Termination	1	
Data Bus	No Corruption	0	0
	Address Bus	0	
	Data Bus	0	
	Instruction Register	0	
	Program Counter	0	
	Register File	0	
	Multiple Corruption	0	
	Termination	0	
Instruction Register	No Corruption	0	535
	Address Bus	0	
	Data Bus	0	
	Instruction Register	187	
	Program Counter	51	
	Register File	200	
	Multiple Corruption	97	
	Termination	0	
Program Counter	No Corruption	0	11
	Address Bus	11	
	Data Bus	0	
	Instruction Register	0	
	Program Counter	0	
	Register File	0	
	Multiple Corruption	0	
	Termination	0	
Register File	No Corruption	0	57
	Address Bus	2	
	Data Bus	1	
	Instruction Register	0	
	Program Counter	10	
	Register File	44	
	Multiple Corruption	0	
	Termination	0	
TOTAL			699

Note: shaded rows are those corresponding to the single corruptions in different functional blocks listed in **Table 7.5**.

Table 7.6. Instruction Register/Decode Results

First Corruption	Second Corruption	Frequency	Total
No Response		2079	2079
Multiple Corruption		41	41
Termination		61	61
Address Bus	No Corruption	0	343
	Address Bus	54	
	Data Bus	187	
	Instruction Register	6	
	Program Counter	0	
	Register File	0	
	Multiple Corruption	0	
	Termination	96	
Data Bus	No Corruption	0	263
	Address Bus	0	
	Data Bus	213	
	Instruction Register	21	
	Program Counter	0	
	Register File	28	
	Multiple Corruption	1	
	Termination	0	
Instruction Register	No Corruption	0	568
	Address Bus	1	
	Data Bus	0	
	Instruction Register	72	
	Program Counter	89	
	Register File	286	
	Multiple Corruption	120	
	Termination	0	
Program Counter	No Corruption	0	0
	Address Bus	0	
	Data Bus	0	
	Instruction Register	0	
	Program Counter	0	
	Register File	0	
	Multiple Corruption	0	
	Termination	0	
Register File	No Corruption	0	21
	Address Bus	7	
	Data Bus	6	
	Instruction Register	0	
	Program Counter	0	
	Register File	8	
	Multiple Corruption	0	
	Termination	0	
TOTAL			3376

Note: shaded rows are those corresponding to the single corruptions in different functional blocks listed in **Table 7.5**.

Table 7.7. Memory Interface Results

First Corruption	Second Corruption	Frequency	Total
No Response		1205	1205
Multiple Corruption		19	19
Termination		0	0
Address Bus	No Corruption	0	8
	Address Bus	0	
	Data Bus	8	
	Instruction Register	0	
	Program Counter	0	
	Register File	0	
	Multiple Corruption	0	
	Termination	0	
Data Bus	No Corruption	0	0
	Address Bus	0	
	Data Bus	0	
	Instruction Register	0	
	Program Counter	0	
	Register File	0	
	Multiple Corruption	0	
	Termination	0	
Instruction Register	No Corruption	0	0
	Address Bus	0	
	Data Bus	0	
	Instruction Register	0	
	Program Counter	0	
	Register File	0	
	Multiple Corruption	0	
	Termination	0	
Program Counter	No Corruption	0	894
	Address Bus	304	
	Data Bus	0	
	Instruction Register	0	
	Program Counter	62	
	Register File	465	
	Multiple Corruption	63	
	Termination	0	
Register File	No Corruption	36	3135
	Address Bus	27	
	Data Bus	4	
	Instruction Register	0	
	Program Counter	870	
	Register File	2093	
	Multiple Corruption	105	
	Termination	0	
TOTAL			5261

Note: shaded rows are those corresponding to the single corruptions in different functional blocks listed in **Table 7.5.**

Table 7.8. Arithmetic and Logic Unit Results

First Corruption	Second Corruption	Frequency	Total
No Response		136	136
Multiple Corruption		245	245
Termination		134	134
Address Bus	No Corruption	0	36
	Address Bus	9	
	Data Bus	8	
	Instruction Register	0	
	Program Counter	13	
	Register File	1	
	Multiple Corruption	0	
	Termination	5	
Data Bus	No Corruption	0	27
	Address Bus	1	
	Data Bus	5	
	Instruction Register	10	
	Program Counter	0	
	Register File	10	
	Multiple Corruption	1	
	Termination	0	
Instruction Register	No Corruption	0	44
	Address Bus	0	
	Data Bus	0	
	Instruction Register	1	
	Program Counter	1	
	Register File	7	
	Multiple Corruption	33	
	Termination	2	
Program Counter	No Corruption	0	101
	Address Bus	42	
	Data Bus	6	
	Instruction Register	1	
	Program Counter	3	
	Register File	2	
	Multiple Corruption	46	
	Termination	1	
Register File	No Corruption	3	211
	Address Bus	11	
	Data Bus	4	
	Instruction Register	4	
	Program Counter	31	
	Register File	123	
	Multiple Corruption	32	
	Termination	3	
TOTAL			934

Note: shaded rows are those corresponding to the single corruptions in different functional blocks listed in **Table 7.5.**

Table 7.9. Control Unit Results

7.3.2. Frequency of bit corruptions

Table 7.10 lists the distribution of the number of bits that were corrupted as a result of the injected faults. As mentioned in Section 6.2, this distribution will be used, when selecting a fault mask, to determine the number of bits to corrupt for a given fault injection experiment. Note that the total of 7671 is computed as the total number of single corruptions plus all the totals for a given number of multiple corruptions multiplied by the number of multiple corruptions. That is,

$$7671 = 6606 + (182 \times 2) + (53 \times 3) + (42 \times 4) + (64 \times 5) + (9 \times 6)$$

Bits Corrupted	Frequency	%
1	5939	77.42
2	452	5.89
3	96	1.25
4	128	1.67
5	47	0.61
6	136	1.77
7	129	1.68
8	94	1.23
9	44	0.57
10	58	0.76
11	32	0.42
12	53	0.69
13	57	0.74
14	30	0.39
15	18	0.23
16	13	0.17
17	7	0.09
18	14	0.18
19	11	0.14
20	28	0.37
21	10	0.13
22	12	0.16
23	25	0.33
24	12	0.16
25	27	0.35
26	22	0.29
27	14	0.18
28	20	0.26
29	9	0.12
30	46	0.60
31	13	0.17
32	75	0.98
Total	7671	100

Table 7.10. Distribution of number of corrupted bits

7.3.3. Frequency of Instruction Fields Corruptions

As discussed in Section 6.2.1.2, Section 6.2.1.3, and Section 6.2.3, certain fault scenarios will be represented with the fault model by corrupting fields within the instruction that define the opcode, source registers, and destination registers. **Table 7.11** lists the distribution of corruptions within the various fields of the instruction. From the 10,900 fault injection results, the instruction register was corrupted 1147 times. However, 215 of these corruptions did not affect the operation of the instruction, and thus had no effect upon the instruction. After these are removed, we are left with 932 corruptions of the instruction register that are show in **Table 7.11**. In lieu of these results, it seems reasonable to assume a uniform distribution for the various categories in **Table 7.11** for use with the fault list construction algorithm [12]. In fact, assuming a uniform distribution for corruptions of the instruction register will provide a more conservative estimate during the assessment because corruption categories that are zero in **Table 7.11** will be non-zero when generated by the fault list construction algorithm. For instance, the category in **Table 7.11** where the Opcode, Destination, Source 1, and Source 2 are all corrupted did not occur during the simulation-based fault injection experiments. Thus, a more conservative estimate during the assessment would result from injecting this scenario during the fault injection experiments using the fault injection environment [9].

Instruction field corrupted	Frequency	%
Opcode	166	17.81
Destination	112	12.02
Source 1	300	32.19
Source 2	290	31.12
Opcode and Destination	16	1.72
Opcode and Source 1	0	0
Opcode and Source 2	0	0
Destination and Source 1	0	0
Destination and Source 2	0	0
Source 1 and Source 2	0	0
Opcode, Destination, and Source 1	48	5.15
Opcode, Destination, and Source 2	0	0
Destination, Source 1, and Source 2	0	0
Opcode, Destination, Source 1, and Source 2	0	0
Total	932	100

Table 7.11. Distribution of corruptions within instruction fields

7.4. Augmented Fault Model

The following section redefines those fault models described in Section 6.2 that need to be augmented to more accurately reflect the results of the simulation-based FIEs. Please note that if a given fault model from Section 6.2 is not mentioned in this section, then the fault model did not require any augmentation. And as such, the description in Section 6.2 is to be used for the fault model definition.

7.4.1. Augmented Register File Fault Model

The following section redefines the register file fault model to more accurately reflect the simulation-based FIE results. In particular, the fault model is augmented to include the faulty behavior of multiple registers being corrupted simultaneously as a result of a single fault.

7.4.1.1. Register Fault Model

The register fault model represents corruptions that occur within the registers of the processor as defined by the programmer's model as well as any special-purpose registers that might be part of the memory map.

7.4.1.1.1. Time

The time of occurrence can only be on an instruction boundary.

7.4.1.1.2. Location

One or more registers that are accessible using the programmer's model of the processor, or one or more registers that appear within the memory map are potential candidates for corruption.

7.4.1.1.3. Value

With regard to value, there are four scenarios that are considered:

1. Missed load - all or part of a register is not loaded when it should be.
2. Extraneous load - all or part of a register is loaded when it should not be.
3. Level change in storage - the correct value of one or more bits in the register are complemented.
4. Assign the value of all zeros and all ones to the register.

Scenario 1 can be accomplished by determining a mask value, based on the value of **expr**, that yields the current value of R_k when the mask is XORed with R_k after it receives the value of **expr** as shown in **Equation (7.1)**.

$$R_k \leftarrow \text{expr} \Rightarrow R_k \leftarrow (\text{expr} \oplus (mask\langle(w-1)\ldots 0\rangle)) \tag{7.1}$$

Scenario 2 can be accomplished by assigning the value of **expr**, intended to be stored in R_k, into any other register, R_j, in the register file, as shown in **Equation (7.2)**.

$$R_k \leftarrow \text{expr} \Rightarrow R_j \leftarrow \text{expr}\ \exists(j \neq k) \tag{7.2}$$

Scenario 3 is often referred to as the bit-flip fault model in the literature. This scenario can be accomplished by selecting a mask and XORing the mask with the current value in the register as shown in **Equation (7.3)**.

$$R_k \Rightarrow R_k \oplus (mask\langle (w-1)\ldots 0\rangle) \tag{7.3}$$

Scenario 4 can be accomplished by XORing the current value of R_k with itself to create the all zeros corruption value or the complement of itself to create the all ones corruption value, as shown in **Equation (7.4)**.

$$\begin{aligned} R_k &\Rightarrow R_k \oplus R_k \\ R_k &\Rightarrow R_k \oplus \overline{R_k} \end{aligned} \tag{7.4}$$

7.4.2. Augmented Control Unit Fault Model

The control unit fault model described in Section 6.2.3 needs to include potential corruptions in any register visible by the programmer's model as well as the data and address bus. In essence, the control unit fault model can be augmented to include the augmented register file fault model (described in Section 6.2.1 and Section 7.4.1) and the bus fault model (described in Section 6.2.4), as well as original definition of the control unit as described in Section 6.2.3. Note that augmenting the control fault model to include such faulty behavior as that described in Section 6.2.1.2 and Section 6.2.1.3 seems reasonable given that these two sections are intended to represent faults within the decoding logic that erroneously selects a given register (other than the correct register) to be used as an input or output operand to the current instruction.

7.4.3. Fetch and Execute Process Fault Model

The termination of the fetch-execute cycle of the processor is not a faulty behavior that was part of the fault model definition in Section 6.2. However, based on the simulation-based FIE results, this is a faulty behavior that needs to be a part of the fault model so that the fault model more accurately represents the faulty behavior of a processor.

Define a processor M as a quintuple

$$M = (I, O, S, \delta, \lambda) \tag{7.5}$$

where I, O, and S are finite, non-empty sets of inputs, outputs, and states, respectively, and

$$\begin{aligned} \delta &: (I \times S \rightarrow S) \text{ is the state transition function} \\ \lambda &\text{ is the output function such that} \\ \lambda &: (I \times S \rightarrow O) \text{ for Mealy machines} \\ \lambda &: (S \rightarrow O) \text{ for Moore machines} \end{aligned} \tag{7.6}$$

The cartesian product $I \times S$ is the set containing all pairs of elements, (I_i, S_j). The state transition function, δ, associates with each pair, (I_i, S_j), an element, S_k, from S, called the *next state*. In a Mealy machine, the output function, λ, associates with each pair, (I_i, S_j), an element, O_k, from O, while in a Moore machine a correspondence exists between the states and the outputs [31].

Given this definition of a processor, the termination of the fetch-execute cycle of the processor can be represented by defining a faulty state transition function, δ':

$$\delta' : (I \times S_j \rightarrow S_j) \tag{7.7}$$

That is, the next state for the processor is always equal to the current state of the processor, regardless of the inputs applied to the processor. Thus, the fetch-execute cycle of the processor is never completed.

7.5. Summary of FIE Results

Given the analysis in Section 7.3.1.2 regarding the unsupported faults listed in **Table 7.5**, as well as the necessary augmentations to the fault model mentioned in Section 7.3.1.3 and Section 7.3.1.4, all of the FIE results can now be considered supported by the fault model described in Section 6.2, along with the appropriate augmented fault models described in Section 7.4. As such, **Table 7.2**, **Table 7.3**, and **Table 7.4** are repeated as **Table 7.12**, **Table 7.13**, and **Table 7.14** with the numbers adjusted accordingly based on the analysis. These results are also combined and summarized in **Table 7.15**; the corresponding percentages are listed in **Table 7.16**. Note that **Table 7.16** does not include the no response faults listed in **Table 7.15**.

Hardware Functional Block	No Response	Single corruption	Multiple corruptions	Terminate Fetch and Execute Process	Total
Register File	na	na	na	na	na
Program Counter	257	351	0	22	630
Control Unit	136	419	245	134	934
Fetch and Decode Logic Block	50	604	45	0	699
Bus Interface	2079	1195	41	61	3376
ALU	1205	4045	11	0	5261
Memory-Mapped Peripheral Devices	na	na	na	na	na
Total	3727	6629	327	217	10900
Percentage of Total (%)	34.19	60.82	3.00	1.99	100

Table 7.12. Results of Table 7.2, modified to include analysis in Section 7.3.1.2.

Type of Corruption	Frequency	% of total
Terminate Fetch and Execute Process	217	1.99
Single Corruption - Supported by fault model	6629	60.82
Single Corruption - Not supported by fault model	0	0
Multiple Corruptions - 2	159	1.46
Multiple Corruptions - 3	53	0.49
Multiple Corruptions - 4	42	0.39
Multiple Corruptions - 5	64	0.59
Multiple Corruptions - 6	9	0.08
No Response	3727	34.19
Total	10900	100

Table 7.13. Results of Table 7.3, modified to include the analysis in Section 7.3.1.2.

Hardware Functional Block	Fault Model	Frequency	% Supported	% Not Supported
Register File	Section 6.2.1	na	100	0
Program Counter	Section 6.2.2	351	100	0
Control Unit	Section 6.2.3	420	100	0
Fetch and Decode Logic Block	Section 6.2.3	618	100	0
Bus Interface	Section 6.2.4	1195	100	0
ALU	Section 6.2.5	4045	100	0
Memory-Mapped Peripheral Devices	Section 6.2.6	na	100	0

Table 7.14. Results of Table 7.4, modified to include the analysis in Section 7.3.1.2.

Hardware Functional Block	No Response	Terminate Fetch and Execute Process	Single Corruption	Multiple Corruptions					Total
				2	3	4	5	6	
Register File	na	na	na	na	na	na	na	na	na
Program Counter	257	22	351	0	0	0	0	0	630
Control Unit	136	134	420	87	48	42	58	9	934
Fetch and Decode Logic Block	50	0	618	25	0	0	6	0	699
Bus Interface	2079	61	1195	40	1	0	0	0	3376
ALU	1205	0	4045	7	4	0	0	0	5261
Memory-Mapped Peripheral Devices	na	na	na	na	na	na	na	na	na
Total	3727	217	6629	159	53	42	64	9	10900

Table 7.15. Summary of FIE results.

Hardware Functional Block	Terminate Fetch and Execute Process	Single Corruption	Multiple Corruptions				
			2	3	4	5	6
Register File	na	na	na	na	na	na	na
Program Counter	5.90	94.10	0.00	0.00	0.00	0.00	0.00
Control Unit	16.79	52.63	10.90	6.02	5.26	7.27	1.13
Fetch and Decode Logic Block	0.00	95.22	3.85	0.00	0.00	0.92	0.00
Bus Interface	4.70	92.14	3.08	0.08	0.00	0.00	0.00
ALU	0.00	99.73	0.17	0.10	0.00	0.00	0.00
Memory-Mapped Peripheral Devices	na	na	na	na	na	na	na

Table 7.16. Summary of FIE results (percentages).

8. Application of the Generic Processor Fault Model to the MC68332 Processor

Figure 8.1 shows a model of the MC68332 microcontroller, that corresponds to the generic processor model shown in **Figure 5.1**. The model of the MC68332 contains a **CPU**, a System Integration Module (**SIM**), an InterModule Bus (**IMB**), and three memory-mapped peripherals that are external to the **CPU** but internal to the MC68332.

With regard to **Figure 8.1**, the **IMB**, which are the internal data, address, and control busses that allow for communication among the modules within the MC68332, corresponds to the **Internal Data Bus**, **Internal Address Bus**, and **Internal Control Bus** that are shown in **Figure 5.1**. The **SIM** contains the **External Bus Interface**, which allows the MC68332 to communicate with entities external to the MC68332 using the **Data Bus**, **Address Bus**, and **Control Bus**. The three memory-mapped peripherals are the Queued Serial Module (**QSM**), the Timer Processor Unit (**TPU**), and a bank of Random Access Memory (**RAM**). The **QSM** provides the MC68332 with serial communication functionality. The **TPU** provides the MC68332 with timer functionality. Finally, the **RAM** provides the **CPU** with an internal bank of volatile memory that can be used for stacks, variable data storage, and so forth. In addition, the **RAM** can be configured to be used by the **TPU** for microcode storage if the user prefers to develop microcode for the timer functions supported by the **TPU** (as opposed to using the microcode programmed into non-volatile memory within the **TPU**). In this case, the **TPU** accesses the **RAM** using a **Direct TPU/ RAM Data/Address Bus** between the **TPU** and **RAM** as opposed to using the **IMB**. All of these modules correspond to the **Memory-Mapped Peripheral Functional Block** shown in **Figure 5.1**. Note that even though **Figure 5.1** shows only one such block, in general there are typically several such blocks, as is the case with the MC68332. As such, the one block shown in **Figure 5.1** is representative of one or more functional blocks that are external to the CPU yet still internal to the processor.

In addition, **Figure 8.1** shows a model of the MC68332 **CPU** that corresponds to the CPU model shown in the generic processor model shown in **Figure 5.1**. The **Control Unit** and the **Fetch and Decode Block** in **Figure 5.1** correspond to the **Control Unit** in **Figure 8.1** as well as the **Instruction Pipeline and Decode**, **Bus Controller**, and the **Output Operand Decode Logic** and **Input Operands Decode Logic** in the **Execution Unit** in **Figure 8.1**. These portions of the CPU shown in **Figure 8.1** are responsible for the fetching and decoding of instructions, and the fetching and storing of instruction operands. For instance, the **Instruction Pipeline and Decode** is responsible for fetching instructions and partially decoding them for the Execution Unit. The **Output Operand Decode Logic** and **Input Operands Decode Logic** are responsible for further decoding of the instruction to determine address values for fetching and storing operands. Finally, the **Bus Controller** is responsible for buffering the requests to fetch operands, store operands, and prefetch instructions. Likewise, the **Execution Unit** in **Figure 8.1** corresponds to the **Datapath** in **Figure 5.1**, both containing entities such as a **Program Counter**, a **Register File**, and an **ALU**. This information is summarized in **Table 8.1**.

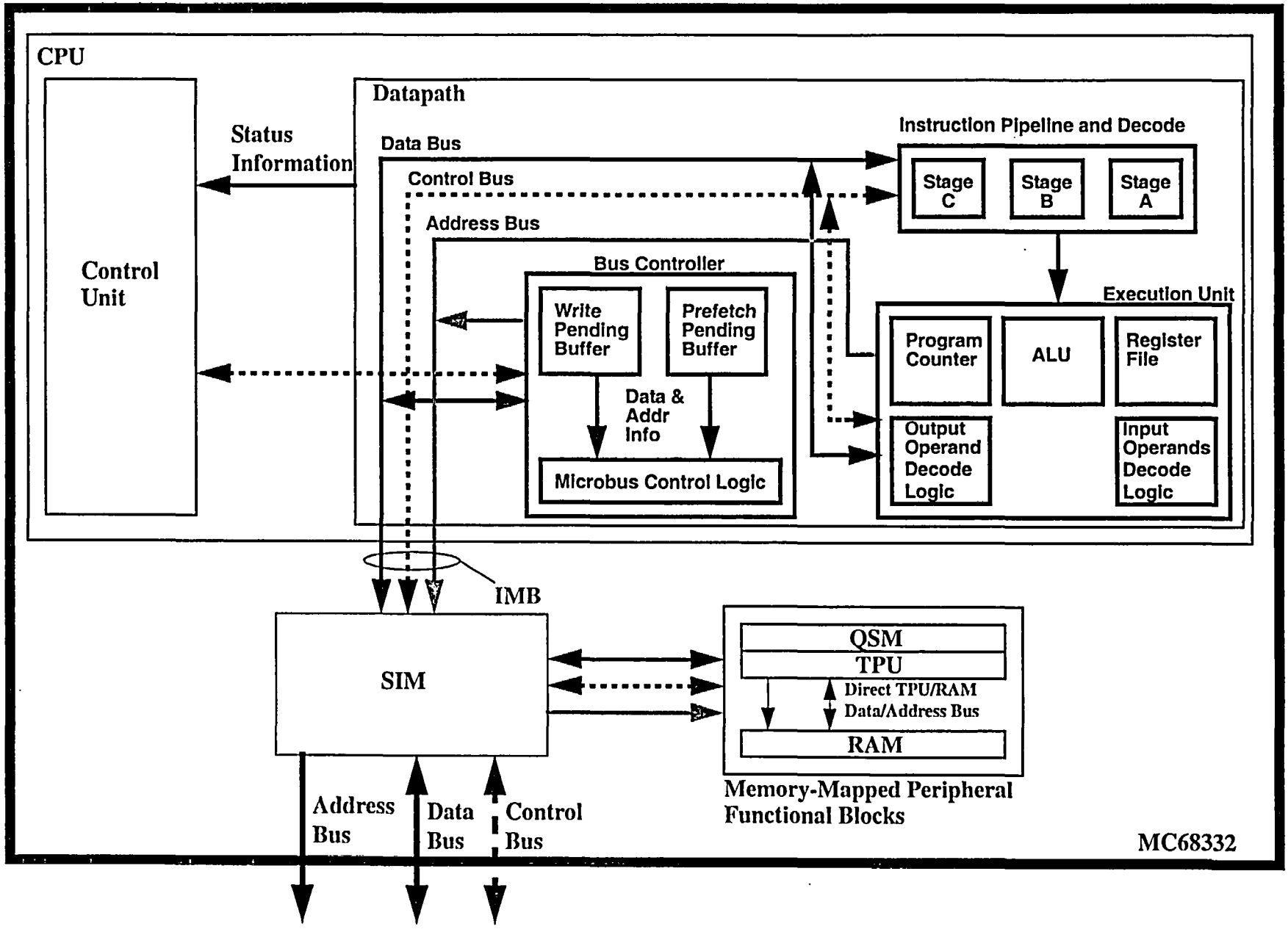


Figure 8.1. Model of the MC68332 corresponding to the generic processor model of Figure 5.1.

Hardware Functional Block from Figure 5.1	Equivalent Functional Blocks in Figure 8.1	Fault Model
Register File	Register File	Section 6.2.1 and Section 7.4.1
Program Counter	Program Counter	Section 6.2.2
Control Unit	Control Unit	Section 6.2.3 and Section 7.4.2
Fetch and Decode Logic Block	Instruction Pipeline and Decode, Bus Controller, Output Operand Decode Logic, and Input Operands Decode Logic.	Section 6.2.3
Internal Data Bus, Internal Address Bus, and Internal Control Bus, as well as the External Bus Interface.	IMB (which are the internal data, address, and control busses) and SIM (which contains the external Data Bus, Address Bus, and Control Bus).	Section 6.2.4
ALU	ALU	Section 6.2.5
Memory-Mapped Peripheral Devices	QSM, TPU, RAM	Section 6.2.6

Table 8.1. Summary of equivalence of MC68332 to generic processor in Figure 5.1.

Note that an assumption is being made here that the effects of faults can be modeled at the boundaries of the instructions, independent of the processor implementation. For instance, even in a pipelined machine, instructions use registers or memory locations, and they affect registers or memory locations. Thus, this fault model is applicable to non-pipelined and pipelined processor architectures alike.

9. Application of the Generic Processor Fault Model to the MC6809 Processor

Figure 9.1 shows a model of the MC6809 that corresponds to the generic processor model shown in **Figure 5.1**. The model of the MC6809 contains a **CPU** and an **External Bus Interface**. Note that in contrast to the MC68332, the MC6809 has no integrated memory-mapped peripherals. This is because the MC6809 is a relatively simple microprocessor, not a complex microcontroller like the MC68332.

With regard to **Figure 9.1**, the internal data, address, and control busses that allow for communication among the modules within the MC6809, correspond to the **Internal Data Bus**, **Internal Address Bus**, and **Internal Control Bus** that are shown in **Figure 5.1**. The MC6809 **Bus Controller** is responsible for overseeing this functionality. The **External Bus Interface** allows the MC6809 to communicate with entities external to the MC6809 using the **Data Bus**, **Address Bus**, and **Control Bus**. The **Timing** unit controls the clock functions for the MC6809.

In addition, **Figure 9.1** shows a model of the MC6809 **CPU** that corresponds to the **CPU** model shown in the generic processor model shown in **Figure 5.1**. The **Control Unit** and the **Fetch and Decode Block** in **Figure 5.1** correspond to the **Control Unit** in **Figure 9.1** as well as the **Instruction Register**, and the **MUX / Control** blocks in **Figure 9.1**. These portions of the CPU shown in **Figure 9.1** are responsible for the fetching and decoding of instructions, and the fetching and storing of instruction operands. Finally, the **Program Counter**, **Register File**, and **ALU** in **Figure 9.1** have direct counterparts in **Figure 5.1**. This information is summarized in **Table 9.1**.

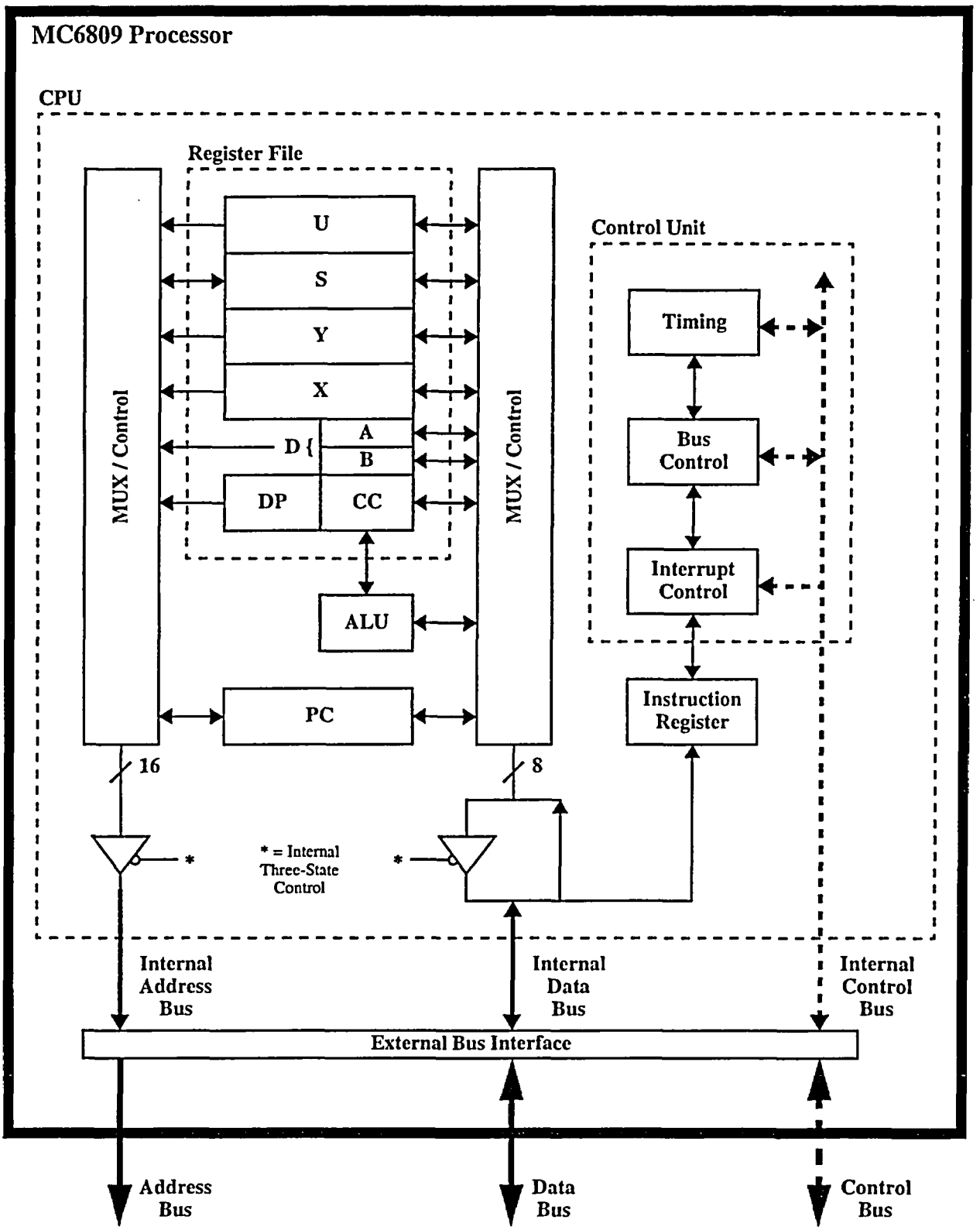


Figure 9.1. Model of the MC6809 corresponding to the generic processor model of Figure 5.1.

Hardware Functional Block from Figure 5.1	Equivalent Functional Blocks in Figure 9.1	Fault Model
Register File	Register File	Section 6.2.1 and Section 7.4.1
Program Counter	Program Counter	Section 6.2.2
Control Unit	Control Unit	Section 6.2.3 and Section 7.4.2
Fetch and Decode Logic Block	Instruction Register, MUX / Control	Section 6.2.3
Internal Data Bus, Internal Address Bus, and Internal Control Bus, as well as the External Bus Interface.	Internal Data Bus, Internal Address Bus, and Internal Control Bus, and External Bus Interface.	Section 6.2.4
ALU	ALU	Section 6.2.5
Memory-Mapped Peripheral Devices	None	n.a.

Table 9.1. Summary of equivalence of MC6809 to generic processor in Figure 5.1.

10. Application of the Generic Processor Fault Model to the MC68HC16 Processor

Figure 10.1 shows a model of the MC68HC16 that corresponds to the generic processor model shown in **Figure 5.1**. The model of the MC68HC16 contains a **CPU**, a System Integration Module / External Bus Interface (**SIM /EBI**), an InterModule Bus (**IMB**), and four memory-mapped peripherals that are external to the **CPU** but internal to the MC68HC16.

With regard to **Figure 10.1**, the **IMB**, which are the internal data, address, and control busses that allow for communication among the modules within the MC68HC16, corresponds to the **Internal Data Bus**, **Internal Address Bus**, and **Internal Control Bus** that are shown in **Figure 5.1**. The **SIM/EBI** contains the **External Bus Interface**, which allows the MC68HC16 to communicate with entities external to the MC68HC16 using the **Data Bus**, **Address Bus**, and **Control Bus**. The four memory-mapped peripherals are the Queued Serial Module (**QSM**), the General Purpose Timer unit (**GPT**), a bank of Standby Random Access Memory (**SRAM**), and an Analog-To-Digital Converter (**ADC**). The **QSM** provides the MC68HC16 with serial communication functionality. The **GPT** provides the MC68HC16 with timer functionality. Finally, the **SRAM** provides the **CPU** with an internal bank of volatile memory that can be used for stacks, variable data storage, and so forth. The **ADC** provides the microcontroller with built-in analog-to-digital conversion functionality, a convenient feature as the MC68HC16 is designed for embedded control applications. All of these modules correspond to the **Memory-Mapped Peripheral Functional Block** shown in **Figure 5.1**. Note that even though **Figure 5.1** shows only one such block, in general there are typically several such blocks, as is the case with the MC68HC16. As such, the one block shown in **Figure 5.1** is representative of one or more functional blocks that are external to the CPU yet still internal to the processor.

In addition, **Figure 10.1** shows a model of the MC68HC16 **CPU** that corresponds to the **CPU** model shown in the generic processor model shown in **Figure 5.1**. The **Control Unit** and the **Fetch and Decode Block** in **Figure 5.1** correspond to the **Control Unit** in **Figure 10.1** as well as the **Instruction Pipeline and Decode**, **Bus Controller**, and the **Output Operand Decode Logic** and **Input Operands Decode Logic** in the **Execution Unit** in **Figure 10.1**. These portions of the CPU shown in **Figure 10.1** are responsible for the fetching and decoding of instructions, and the fetching and storing of instruction operands. For instance, the **Instruction Pipeline and Decode** is responsible for fetching instructions and partially decoding them for the Execution Unit. The **Output Operand Decode Logic** and **Input Operands Decode Logic** are responsible for further decoding of the instruction to determine address values for fetching and storing operands. Finally, the **Bus Controller** is responsible for buffering the requests to fetch operands, store operands, and prefetch instructions. Likewise, the **Execution Unit** in **Figure 10.1** corresponds to the **Datapath** in **Figure 5.1**, both containing entities such as a **Program Counter**, a **Register File**, and an **ALU**. This information is summarized in **Table 10.1**.

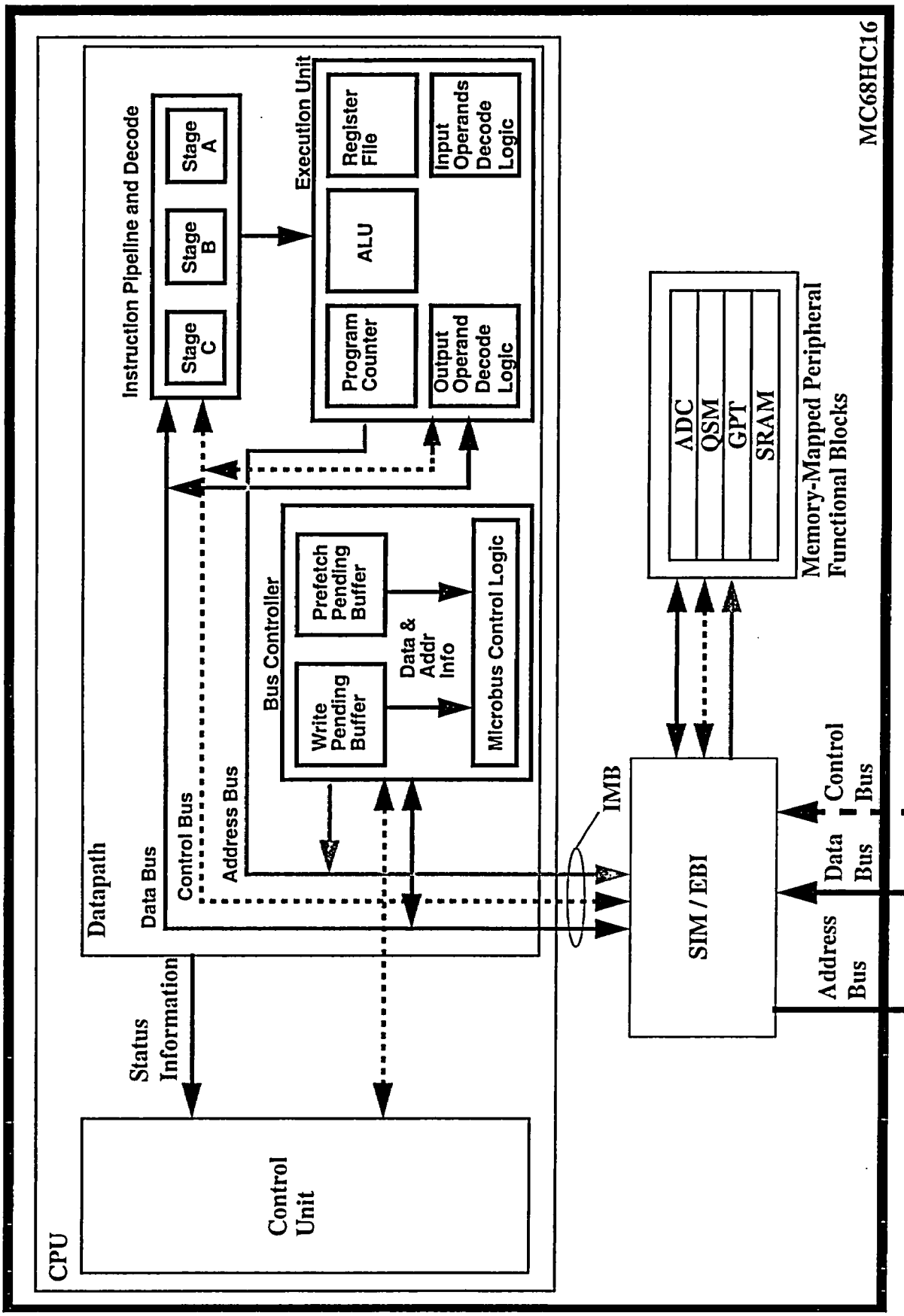


Figure 10.1. Model of the MC68HC16 corresponding to the generic processor model of Figure 5.1.

Hardware Functional Block from Figure 5.1	Equivalent Functional Blocks in Figure 10.1	Fault Model
Register File	Register File	Section 6.2.1 and Section 7.4.1
Program Counter	Program Counter	Section 6.2.2
Control Unit	Control Unit	Section 6.2.3 and Section 7.4.2
Fetch and Decode Logic Block	Instruction Pipeline and Decode, Bus Controller, Output Operand Decode Logic, and Input Operands Decode Logic.	Section 6.2.3
Internal Data Bus, Internal Address Bus, and Internal Control Bus, as well as the External Bus Interface.	IMB (which are the internal data, address, and control busses) and SIM/ EBI (which contains the external Data Bus, Address Bus, and Control Bus).	Section 6.2.4
ALU	ALU	Section 6.2.5
Memory-Mapped Peripheral Devices	ADC, QSM, GPT, SRAM	Section 6.2.6

Table 10.1. Summary of equivalence of MC68HC16 to generic processor in Figure 5.1.

11. Application of the Generic Processor Fault Model to the AMD486 Processor

Figure 11.1 shows a model of the AMD486 that corresponds to the generic processor model shown in **Figure 5.1**. The model of the AMD486 contains a **CPU**, a **Segmentation Unit**, a **Paging Unit**, a **Floating Point Unit**, a **Cache Unit**, and an **External Bus Interface**. Note that the AMD486 has no integrated memory-mapped peripherals.

With regard to **Figure 11.1**, the internal data, address, microinstruction, and displacement busses that allow for communication among the modules within the AMD486, correspond to the **Internal Data Bus**, **Internal Address Bus**, and **Internal Control Bus** that are shown in **Figure 5.1**. The **External Bus Interface** allows the AMD486 to communicate with entities external to the AMD486 using the **Data Bus**, **Address Bus**, and **Control Bus**. A **Clock Generator Unit** (not shown) controls the clock functions for the AMD486.

In addition, **Figure 11.1** shows a model of the AMD486 **CPU** that corresponds to the **CPU** model shown in the generic processor model shown in **Figure 5.1**. The **Control Unit** and the **Fetch and Decode Block** in **Figure 5.1** correspond to the **Control Unit** in **Figure 11.1** as well as the **Instruction Pipeline/ Decode Unit** in **Figure 11.1**. These portions of the CPU shown in **Figure 11.1** are responsible for the fetching and decoding of instructions, and the fetching and storing of instruction operands. Finally, the **Program Counter** (within the **Control Unit**), **Register File**, and **ALU** in **Figure 11.1** have direct counterparts in **Figure 5.1**. This information is summarized in **Table 11.1**.

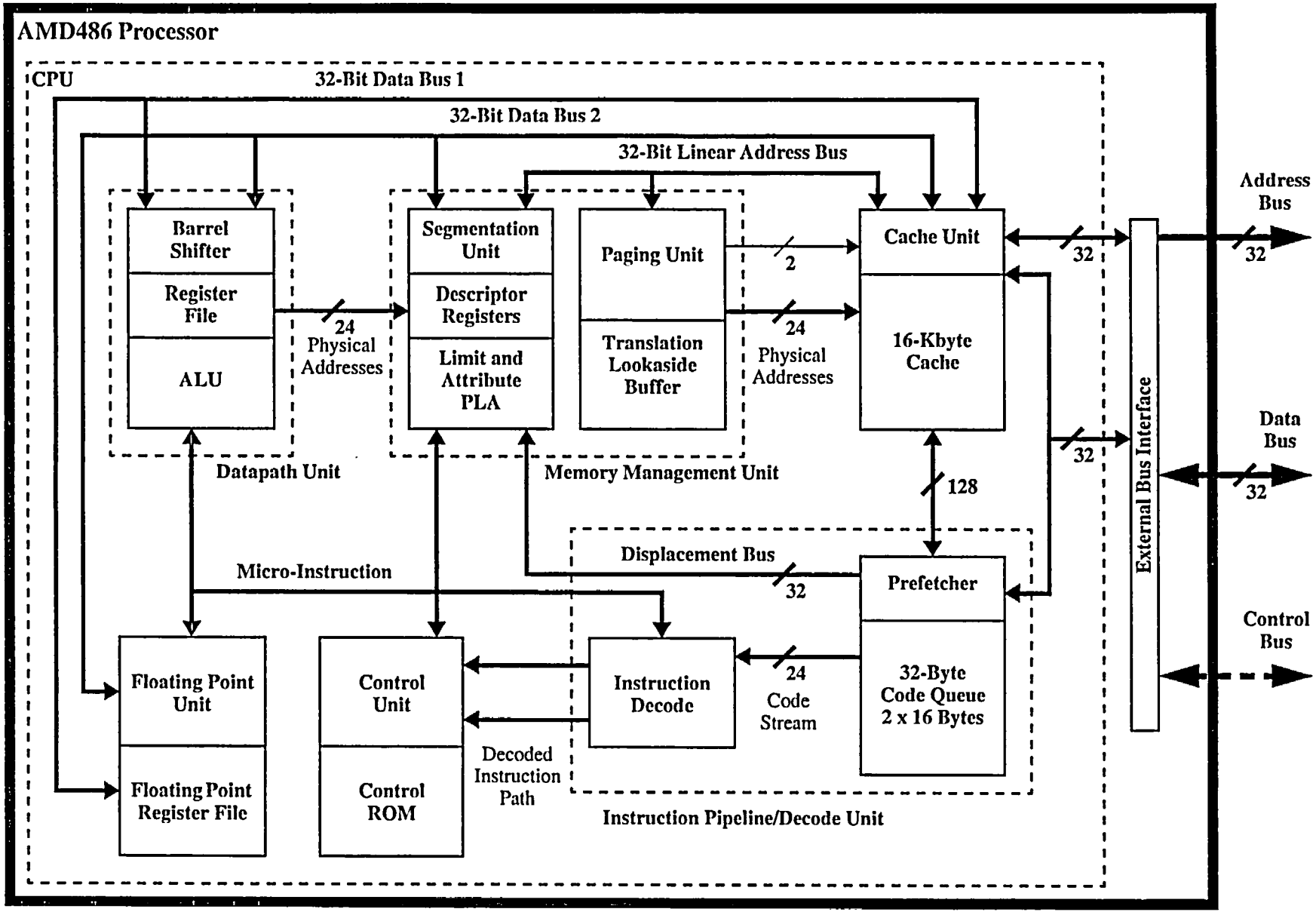


Figure 11.1. Model of the AMD486 corresponding to the generic processor model of Figure 5.1.

Hardware Functional Block from Figure 5.1	Equivalent Functional Blocks in Figure 11.1	Fault Model
Register File	Register File, FPU Register File	Section 6.2.1 and Section 7.4.1
Program Counter	Program Counter (in Control Unit)	Section 6.2.2
Control Unit	Control Unit, Control ROM	Section 6.2.3 and Section 7.4.2
Fetch and Decode Logic Block	Cache Unit, Paging Unit, Code Queue, Segmentation Unit, Instruction Decode, Shifter	Section 6.2.3
Internal Data Bus, Internal Address Bus, and Internal Control Bus, as well as the External Bus Interface.	Internal Data, Address Microinstruction, and Displacement Bus, and External Bus Interface	Section 6.2.4
ALU	ALU, FPU	Section 6.2.5
Memory-Mapped Peripheral Devices	None	n.a.

Table 11.1. Summary of equivalence of AMD486 to generic processor in Figure 5.1.

12. Derivation of Fault Selection Weighting Factors

In order to generate the fault space for a given system, it is necessary to determine weighting factors for the fault selection process. These weighting factors control the probability of selecting the individual fault behaviors from the Generic Processor Fault Model and are specific to the processor applied in the system. The two main steps in the fault selection weighting factor calculation are:

1. Analysis of the total potentially unsafe failure rate in a given system to determine the percentage of the failure rate to be allocated to the processor and to the remaining components which are external to the processor.
2. Derivation of weight factors for individual processor blocks based on either (1) uniform selection if the internal processor architecture details are not known, or (2) weighted selection based on complexity factors if the internal processor architecture details are known

The first step is application-dependent, and requires a detailed analysis of the overall system within which the processor is applied. The second step requires an analysis of the specific processor used in the system and development of either generic weighting factors or processor-specific weighting factors.

12.1. Derivation of Processor and External Weighting Factors

This process requires a reliability-based study of the overall system within which the processor is applied. The goal is to determine the percentage of faults to be considered which are internal to the processor and the percentage of faults to be considered which are external to the processor. The first step in determining these percentages is to analyze the overall system to identify those components which can fail in a potentially unsafe manner. During this process, it is conservatively assumed that all processor faults can be potentially unsafe, and that any external components about which there is some doubt with respect to unsafe failure modes also be considered to be potentially unsafe.

Once all the components that can fail in a potentially unsafe manner are identified, a weighting distribution must be developed based on the failure rates of each of the components. This can typically be derived from MIL HDBK 217F reliability estimates, from manufacturer reliability data, or from field reliability data. The components which are external to the processor should be divided into logical groups called functional blocks, similar to the functional block specification approach used within the processor. At the end of this process, each of the external functional blocks as well as the processor itself should be assigned a specific weighting factor.

This process is obviously highly dependent on the overall system architecture and complexity, but a general rule-of-thumb is that typically 80 - 90% of the overall weight is assigned to components external to the processor, while 10 - 20% of the overall weight is assigned to the processor itself.

12.1.1. Comparison of Internal / External Fault Injection Methods

For the faults that are internal to the processor, the approach is to inject the faults into registers within the memory map. As such, these faults are fully supported by the register fault model described in Section 6.2.1 and the control unit/instruction decode fault model described in Section 6.2.3, as discussed in Section 6.2.5 and Section 6.2.6.

For faults which are external to the processor, the faults are injected on the data, address, control bus pins of processor. As such, these faults are fully supported by the bus fault model described in Section 6.2.4.

12.2. Derivation of Generic Fault Selection Weighting Factors

In general, the details regarding the internal architecture and complexity of commercial processors is extremely difficult to obtain due to its proprietary nature. In this case, the most conservative approach is to derive weighting factors for the internal processor functional blocks based on a uniform distribution of the overall weight across the number of functional blocks. This will be the most common situation, as internal details for modern processors are not readily available.

12.3. Derivation of Processor-Specific Fault Selection Weighting Factors

In general, the details regarding the internal architecture and complexity of commercial processors is extremely difficult to obtain due to its proprietary nature. When such details directly exist, it is relatively straightforward to derive weighting factors for the internal processor functional blocks based on block complexity. When such details do not directly exist, it may be possible to derive complexity information from other sources. Such as case is illustrated in this section using the MC68332 processor as an example. For this processor, the internal architecture and complexity details do not directly exist, but are derived from two test grading papers and a computer architecture textbook describing an earlier processor in the same family.

12.3.1. Derivation of Internal Processor Weighting Factors for the MC68332

Reference [5] is a test grading paper from Motorola which provided the number of all possible single stuck-at faults for the CPU, SIM, TPU, and QSM modules of the 332 MCU. The number of single stuck-at faults was used as a complexity factor to allocate the failure rate assigned to the 332 MCU to its components. The other information available from Motorola is the die surface area occupied by the different MCU components, derived from reference [20]. The components considered there were the four modules considered in [5], as well as the RAM, Chip Select (CS), and Inter-Module Bus (IMB).

The five major components of the 332 MCU are considered to be the CPU, SIM, TPU, QSM, and RAM. In order to consider all these components, an average number of single stuck-at faults per unit of die surface area was calculated from [5] and [20] to arrive at fault estimates for the RAM, CS, and IMB components. The CS component is then considered part of the SIM for purposes of complexity estimation. The IMB component is not inserted in the table as its faults are considered distributed among the other blocks.

Table 12.1 illustrates the weighting factor estimation process for the five major 332 MCU components (CPU, SIM, TPU, QSM, and RAM). Note that the average number of faults per unit area was 78.44 (based on CPU, SIM, TPU, and QSM), which was used to estimate the number of faults for the RAM, CS, and IMB.

Location	# Single Stuck-at Faults from [7]	Die Surface Area from [15]	Estimated # Single Stuck-at Faults	Weighting Factor
CPU	181,208	1,564	181,208	50.69%
SIM (including CS)	39,128	544	39,128	10.95%
TPU	49,679	1,190	49,679	13.90%
QSM	18,110	374	18,110	5.07%
RAM	--	884	69,341	19.39%
TOTALS:	288,025		357,466	100%

Table 12.1. Derivation of Weighting Factors for MC68332

Unfortunately, no information is available from Motorola regarding further breakdown of the internal modules of the 332 CPU component. In order to arrive at a more refined breakdown of this particular component, the structure of the Motorola MC68000 was applied to further sub-divide the CPU into the six sub-elements as illustrated in **Table 12.2.** This analysis is based on surface area data provided in Figure 11.1, page 217 of [60] which shows a floorplan of the Motorola MC68000 CPU.

Location	Blocks from 68000 Floorplan	Relative Surface Area	Percentage of Surface Area
CPU Components			
Register File	20/21 OPPAH, 12/13 OPPAL, 1/2 OPPD	272	23.05%
Program Counter	1/21 OPPAH, 1/13 OPPAL	14	1.19%
Control Unit	control interface, parameterization PLA	223	18.90%
Fetch and Decode Logic	PLA1, PLA2, PLA3, branching PLA, trap PLA, parameter register, ROM+decoder, selection, address determination	460	38.98%
External Bus Interface	bus control logic, FC0, FC1, FC2	84	7.12%
ALU	ALU control, 1/2 OPPD	127	10.76%
	Totals:	1180	100.00%

Table 12.2. MC68000 Surface Area Data

The following MC68000 blocks in the floorplan were ignored for this calculation: all data/address/ control pads, IPL, clock generation, and clock logic.

Combining **Table 12.1** and **Table 12.2** gives the final weighting table for the 332 MCU components shown in **Table 12.3.**

Location	Complexity from 68000	Weight Calculation	Final Weight Factor
CPU Components			
Register File	23.05%	50.69% * 23.05%	11.68%
Program Counter	1.19%	50.69% * 1.19%	0.60%
Control Unit	18.90%	50.69% * 18.90%	9.58%
Fetch and Decode Logic	38.98%	50.69% * 38.98%	19.76%
External Bus Interface	7.12%	50.69% * 7.12%	3.61%
ALU	10.76%	50.69% * 10.76%	5.46%
Other 332 MCU Components			
SIM	--	--	10.95%
TPU	--	--	13.90%
QSM	--	--	5.07%
RAM	--	--	19.39%
		Total:	100.00%

Table 12.3. Final Weighting Table for MC68332 Components

13. Adaptation of the Generic Processor Fault Model to the Fault Injection Environment

Because the observability and controllability available for a given fault injection experiment is limited, a given faulty behavior might require a sequence of actions and masks in order to be accurately simulated. This is especially true when trying to simulate permanent faults, where a sequence of faults might need to be injected in order to fully represent the effect of a permanent fault.

13.1. Augmented Fault Model Summary

In general, all of the faulty behaviors discussed in Section 6.2 can be represented in the fault injection environment, even for registers that are not directly accessible using the programmer's model of the processor, such as the PC (see Section 6.2.2). Regarding the results of the FIEs discussed in Section 7.3, the analysis revealed additional faulty behaviors that are not included in Section 6.2. The additional faulty behaviors are described in Section 7.4 and are repeated here:

1. The **Single Corruption/Different Functional Block** results imply that a fault within a certain functional block can affect locations within other functional blocks. For instance, a fault in the fetch/decode logic can eventually have an effect on registers in the register file that are not source or destination registers for the current instruction. All of the corruptions in **Table 7.5** except for the control unit corruptions are concluded to be propagation of the initial corruption. The control unit corruptions, however, represent additional faulty behavior that needs to be added to the fault model.
2. The **Multiple corruptions** case implies that multiple locations are corrupted simultaneously. The masks for the multiple corruptions may or may not be equal. Since the representation of these faulty behaviors is just an extension of how a single register is corrupted in the fault injection environment, these faulty behaviors can be represented in the fault injection environment as well.
3. Finally, the **Terminate Fetch and Execute Process** results imply that there are faults which will cause the processor system to cease performing any functions even though no data corruptions have occurred. For instance, this behavior could be emulated by holding the system clock to a fixed value.

Thus, there is no faulty behavior in the proposed fault model (discussed in Section 6.2) or the enhancements to the fault model based on the FIE results (discussed in Section 7.3) that cannot be represented in the fault injection environment.

13.2. Mask Implementation

With regard to the mask values discussed in Section 6.2 and Section 7.4, practically speaking there are no masks values that cannot be represented with a randomly generated mask according to a given distribution. Thus, all mask values will be produced as randomly generated m-bit masks, where m is equal to the appropriate width, generated according to a uniform distribution (for example, see Section 7.3.3). See [12] for additional details.

14. Summary

This document describes the behavioral-level generic processor fault model that will be used to generate the fault injection experiments for the UVa numerical safety evaluation process. The basis of the model is to consider any location that is accessible using the programmer's model of the processor and the memory map as a potential candidate for the corruption of the information. The model allows any register, any memory location, any address, or any instruction to be corrupted in any way. Each fault in the model is defined by a time of occurrence, a duration, a location, and a mask value. The time of occurrence can be any memory reference cycle. The duration can be either permanent (the length of the experiment) or one instruction cycle. The location is any register, any memory location, any address, or any instruction. The mask value is all possible corruptions of any single byte of address, data, or instruction.

The fault space as defined by the fault model will be apportioned appropriately according to a variety of factors such as percentage of transient and permanent faults, percentage of faults external and internal to the processor, and percentage of area occupied by the various modules within the processor. Details of this apportionment are presented in the document, except for the details of the transient/permanent apportionment which is presented in [10].

The rationale behind the choice of the fault model is that it provides the most comprehensive representation of the possible corruptions that can occur within the processor while requiring the least number of assumptions regarding the implementation. Thus, it should provide the most conservative estimate for the numerical safety specification for this effort. The crux to using such a model is the fault list construction procedure described in [12]. Essentially, the fault model produces a very unconstrained fault space, and then the fault list construction is used to constrain the fault space to those faults that actually produce an error and thus have an effect on the system under analysis.

15. References

[1] Abraham, Jacob A. and Kenneth P. Parker, "Practical Microprocessor Testing: Open and Closed Loop Approaches", *IEEE COMPCON*, Spring 1981, pp. 308-311.

[2] Annaratone, M.A. and M.G. Sami, "An Approach to Functional Testing of Microprocessors", *Proceedings of the 12th International Symposium on Fault-Tolerant Computing*, June 1982, pp. 158-164.

[3] Arlat, J.M., M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell, "Fault Injection for Dependability Validation: A Methodology and Some Applications", *IEEE Transactions on Software Engineering*, Vol. 16, No. 2, February, 1990, pp. 166-182.

[4] Brahme, Dhananjay and Jacob A. Abraham, "Functional Testing of Microprocessors", *IEEE Transactions on Computers*, Vol. C-33, No. 6, June 1984, pp. 475-485.

[5] Cheng, Tony, Eric Hoang, David Rivera, Alan Haedge, Jamie Fontenot, and Glenn Carson, "Test Grading the 68332", *Proceedings of the International Test Conference*, 1991, pp. 150-156.

[6] Choi, Charles Y., Barry W. Johnson, and Joseph A. Profeta III, "Safety Issues in the Comparative Analysis of Dependable Architectures," *IEEE Transactions on Reliability*, Vol. 46, No. 3, September 1997, pp 316-322.

[7] Choi, H., W. Wang, and K.S. Trivedi, "Conditional MTTF and its Computation in Markov Reliability Models," *Proc. 1993 Annual Reliability and Maintainability Symposium*, January 25-28, 1993, pp. 55-63.

[8] Choi, J.G. and P.H. Seong, "Dependability Estimation of Digital System by Operational Profile-Based Fault Injection", International Topical Meeting on Probabilistic Safety Assessment - Volume I, 1999, pp. 499-506.

[9] Cutright, E., DeLong, T., Johnson, B., *Numerical Safety Evaluation Process for Safety-Critical Systems*, UVA Technical Report UVA-CSCS-NSE-001.

[10] Cutright, E., DeLong, T., Johnson, B., *Analytical Model*, UVA Technical Report UVA-CSCS-NSE-002.

[11] Cutright, E., DeLong, T., Johnson, B., *Statistical Model*, UVA Technical Report UVA-CSCS-NSE-003.

[12] Cutright, E., DeLong, T., Johnson, B., *Fault List Construction Algorithm*, UVA Technical Report UVA-CSCS-NSE-005.

[13] Czeck, Edward W. and Daniel P. Siewiorek, "Effects of transient gate-level faults on program behavior" *Digest of Papers 20th International Symposium on Fault-Tolerant Computing*, 1990, pp. 236-243.

[14] Fedi, Xavier and Rene David, "Experimental Results From Random Testing of Microprocessors", *Proceedings of the 14th International Symposium on Fault-Tolerant Computing*, 1984, pp. 225-230.

[15] Fedi, Xavier and Rene David, "Some Experimental Results From Random Testing of Microprocessors", *IEEE Transactions on Instrumentation and Measurement*, Vol. IM-35, No. 1, March 1986, pp. 78-86.

[16] Frenzel, James F. and Peter N. Marinos, "Functional Testing of Microprocessors in a User Environment", *Proceedings of the 14th International Symposium on Fault-Tolerant Computing*, 1984, pp. 219-224.

[17] Geradin, J., "The DEF Injector Test Instrument, Assistance in the Design of Reliable and Safe Systems", *Computers in Industry*, 1989, pp. 311-319.

[18] Goble, W.M., *Control Systems Safety Evaluation and Reliability*, Instrument Society of America, 1998.

[19] Guthoff, J. and V. Sieh, "Combining Software-Implemented and Simulation-Based Fault Injection into a Single Fault Injection Method", *Proceedings of the 25th International Symposium on Fault-Tolerant Computing*, June 1995, pp. 196-206.

[20] Harwood, Wallace, and Mark McDermott, "Testability Features of the MC68332 Modular Microcontroller", *Proceedings of the International Test Conference*, 1989, pp. 615-623.

[21] Heuring, Vincent P. and Harry F. Jordan, *Computer Systems Design and Architecture*, Addison-Wesley, 1997.

[22] IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1993, IEEE, Inc., 1993.

[23] Johnson, Barry W., *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Wesley Publishing Company, 1989.

[24] Joshi, Bharat S. and Seyed H. Hosseini, "Efficient algorithms for microprocessor testing", *Proceedings Annual Reliability and Maintainability Symposium*, 19-22 January, 1998, pp. 100-104.

[25] Kanawati, G.A., N.A. Kanawati, and J.A. Abraham, "FERRARI: A Tool for the Validation of System Dependability Properties", *Proceedings of the 22nd International Symposium on Fault-Tolerant Computing*, July 1992, pp. 336-344.

[26] Kanawati, G.A., N.A. Kanawati, and J.A. Abraham, "EMAX: An Automatic Extractor of High-Level Error Models", *Proceedings of the 9th AIAA Computing in Aerospace Conference*, 1993, pp. 1297-1306.

[27] Kanawati, Ghani A., Nasser A. Kanawati, and Jacob A. Abraham, "FERRARI: A Flexible Software-Based Fault and Error Injection System", *IEEE Transactions on Computers*, Vol. 44, No. 2, February 1995, pp. 248-260.

[28] Kao, Wei-lun and Ravishankar K. Iyer, "DEFINE: A Distributed Fault Injection and Monitoring Environment", *Fault-Tolerant Parallel and Distributed Systems*, 1995, pp. 252-259.

[29] Kildiran, Gunhan and Peter N. Marinos, "Functional Testing of Microprocessor-like Architectures", *Proceedings of the International Test Conference*, 1986, pp. 913-920.

[30] Klenke, Robert, Associate Professor of Electrical Engineering, Virginia Commonwealth University.

[31] Kohavi, Zvi, *Switching and Finite Automata Theory*, McGraw-Hill, Inc., 1978.

[32] Koshevenko, A.V. and S.G. Sharshunov, "Functional Testing of RISC Microprocessors", *Automation and Remote Control*, Vol. 59, No. 10, October 1998, pp. 1469-1477.

[33] Lin, Tonysheng and Stephen Y.H. Su, "The S-Algorithm: A Promising Solution for Systematic Functional Test Generation", *IEEE Transactions on Computer-Aided Design*, Vol. CAD-4, No. 3, July 1985, pp. 250-263.

[34] Lovric, Tomislav and Klaus Echtle, "ProFI: Processor Fault Injection for Dependability Validation", *IEEE International Workshop on Fault and Error Injection for Dependability Validation of Computer Systems*, 1993.

[35] Lovric, Tomislav, "Processor Fault Simulation with PROFI", *Proceedings of the European Simulation Symposium ESS 95*, October 1995, pp. 353-357.

[36] Nemolochnov, O.F. and Yu. A. Shchupak, "Model of Functional Faults for the Automaton Representation of Primary Elements of a Logic Circuit", *Automation and Remote Control*, Vol. 54, No. 5, Pt. 2, May 1993, pp. 855-864.

[37] Patel, Janak H. and Leona Y. Fung, "Concurrent Error Detection in ALU's by Recomputing with Shifted Operands", *IEEE Transactions on Computers*, Vol. C-31, No. 7, July 1982, pp. 589-595.

[38] Patel, Janak H. and Leona Y. Fung, "Concurrent Error Detection in Multiply and Divide Arrays", *IEEE Transactions on Computers*, Vol. C-32, No. 4, April 1983, pp. 417-422.

[39] Rimen, M. and J. Ohlsson, "A Study of the Error Behavior in 32-bit RISC Subjected to Simulated Transient Fault Injection", *Proceedings of the International Test Conference*, 1992, pp. 697-704.

[40] Robach, C. and G. Saucier, "Diversified Test Methods for Local Control Units", *IEEE Transactions on Computers*, May 1975, pp. 562-567.

[41] Robach, Chantal and Gabriele Saucier, "Dynamic Testing of Control Units", *IEEE Transactions on Computers*, Vol. C-27, No. 7, July 1978, pp. 617-623.

[42] Robach, Chantal, and Gabriele Saucier, "Microprocessor Functional Testing", *Proceedings of the IEEE International Test Conference*, 1980, pp. 433-443.

[43] Rosenberg, H. and K. Shin, "Software Fault Injection and Its Application in Distributed Environment", *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing*, 1993, 208-217.

[44] Saluja, Kewal K., Li Shen, and Stephen Y.H. Su, "A Simplified Algorithm for Testing Microprocessors", *Proceedings of the IEEE International Test Conference*, 1983, pp. 668-675.

[45] Sharshunov, S.G. and V.P. Tchipulis, "On Generating Tests for Microprocessors", *Measurement*, January-March 1986, pp. 28-38.

[46] Shen, Li and Stephen Y.H. Su, "A Functional Testing Method for Microprocessors", *Proceedings of the 14th International Symposium on Fault-Tolerant Computing*, 1984, pp. 212-218.

[47] Smith, D. Todd, Allan White, Todd A. DeLong, Barry W. Johnson, and Ted C. Giras, *A Tutorial on Architectural Analysis Using Reliability and Safety*, Technical Report No. 990609, Center for Safety Critical Systems, University of Virginia, 1999.

[48] Smith, D. Todd, Barry W. Johnson, and Joseph A. Profeta III, "System Dependability Evaluation via a Fault List Generation Algorithm", *IEEE Transactions on Computers*, Vol. 45, No. 8, August 1996, pp. 974-979.

[49] Sridhar, T. and J.P. Hayes, "Testing Bit-Sliced Microprocessors", *Digest of the 9th Symposium on Fault Tolerant Computing*, Madison, WI, June 1979, pp. 211-218.

[50] Sridhar, Thirumalai and John P. Hayes, "A Functional Approach to Testing Bit-Sliced Microprocessors", *IEEE Transactions on Computers*, Vol. C-30, No. 8, August 1981, pp. 563-571.

[51] Su, Stephen Y.H. and Yu-I Hsieh, "Testing Functional Faults in Digital Systems Described by Register Transfer Language", *Journal of Digital Systems*, 1981, pp. 161-183.

[52] Thatte, S.M. and J.A. Abraham, "A Methodology for Functional Level Testing of Microprocessors", *Proceedings of the 8th International Symposium on Fault-Tolerant Computing*, June 1978, pp. 90-95.

[53] Thatte, Satish M. and Jacob A. Abraham, "User Testing of Microprocessors", *Digest of Papers IEEE COMPCON*, Spring 1979, pp. 108-114.

[54] Thatte, Satish M. and Jacob A. Abraham, "Test Generation for General Microprocessor Architectures", *Digest of the 9th Symposium on Fault Tolerant Computing*, Madison, WI, June 1979, pp. 203-210.

[55] Thatte, Satish M. and Jacob A. Abraham, "Test Generation for Microprocessors", *IEEE Transactions on Computers*, Vol. C-29, No. 6, June 1980, pp. 429-441.

[56] Ubar, R., "Combining Functional and Structural Approaches in Test Generation for Digital Systems", *Microelectronics & Reliability*, Vol. 38, No. 3, 1998, pp. 317-329.

[57] Yount, C.R., "The Automatic Generation of Instruction-Level Error Manifestations of Hardware Faults: A New Fault-Injection Model", Ph.D. dissertation, Carnegie Mellon University, May 1993.

[58] Yount, Charles R. and Daniel P. Siewiorek, "A Methodology for the Rapid Injection of Transient Hardware Errors", *IEEE Transactions on Computers*, Vol. 45, No. 8, August 1996, pp. 881-891.

[59] Zemva, Andrej and Baldomir Zajc, "Functionality Fault Model: A Basis For Technology-Specific Test Generation", *Microelectronics & Reliability*, Vol. 38, No. 4, April 1998, pp. 597-604.

[60] Anceau, Francois, *The Architecture of Microprocessors*, Addison-Wesley, 1986.