U.S.NRC
United States Nuclear Regulatory Commission
Protecting People and the Environment

NUREG/CR-7151, Vol. 1

Development of a Fault Injection-Based Dependability Assessment Methodology for Digital I&C Systems

Volume 1

Office of Nuclear Regulatory Research

AVAILABILITY OF REFERENCE MATERIALS IN NRC PUBLICATIONS

NRC Reference Material

As of November 1999, you may electronically access NUREG-series publications and other NRC records at NRC's Public Electronic Reading Room at http://www.nrc.gov/reading-rm.html. Publicly released records include, to name a few, NUREG-series publications; *Federal Register* notices; applicant, licensee, and vendor documents and correspondence; NRC correspondence and internal memoranda; bulletins and information notices; inspection and investigative reports; licensee event reports; and Commission papers and their attachments.

NRC publications in the NUREG series, NRC regulations, and Title 10, "Energy," in the *Code of Federal Regulations* may also be purchased from one of these two sources.

1. The Superintendent of Documents
   U.S. Government Printing Office Mail Stop SSOP
   Washington, DC 20402–0001
   Internet: bookstore.gpo.gov
   Telephone: 202-512-1800
   Fax: 202-512-2250
2. The National Technical Information Service
   Springfield, VA 22161–0002
   www.ntis.gov
   1–800–553–6847 or, locally, 703–605–6000

A single copy of each NRC draft report for comment is available free, to the extent of supply, upon written request as follows:

Address: U.S. Nuclear Regulatory Commission
Office of Administration
Publications Branch
Washington, DC 20555-0001
E-mail: DISTRIBUTION.RESOURCE@NRC.GOV
Facsimile: 301–415–2289

Some publications in the NUREG series that are posted at NRC's Web site address http://www.nrc.gov/reading-rm/doc-collections/nuregs are updated periodically and may differ from the last printed version. Although references to material found on a Web site bear the date the material was accessed, the material available on the date cited may subsequently be removed from the site.

Non-NRC Reference Material

Documents available from public and special technical libraries include all open literature items, such as books, journal articles, transactions, *Federal Register* notices, Federal and State legislation, and congressional reports. Such documents as theses, dissertations, foreign reports and translations, and non-NRC conference proceedings may be purchased from their sponsoring organization.

Copies of industry codes and standards used in a substantive manner in the NRC regulatory process are maintained at—

The NRC Technical Library
Two White Flint North
11545 Rockville Pike
Rockville, MD 20852–2738

These standards are available in the library for reference use by the public. Codes and standards are usually copyrighted and may be purchased from the originating organization or, if they are American National Standards, from—

American National Standards Institute
11 West 42nd Street
New York, NY 10036–8002
www.ansi.org
212–642–4900

Legally binding regulatory requirements are stated only in laws; NRC regulations; licenses, including technical specifications; or orders, not in NUREG-series publications. The views expressed in contractor-prepared publications in this series are not necessarily those of the NRC.

The NUREG series comprises (1) technical and administrative reports and books prepared by the staff (NUREG–XXXX) or agency contractors (NUREG/CR–XXXX), (2) proceedings of conferences (NUREG/CP–XXXX), (3) reports resulting from international agreements (NUREG/IA–XXXX), (4) brochures (NUREG/BR–XXXX), and (5) compilations of legal decisions and orders of the Commission and Atomic and Safety Licensing Boards and of Directors' decisions under Section 2.206 of NRC's regulations (NUREG–0750).

U.S.NRC
United States Nuclear Regulatory Commission
Protecting People and the Environment

NUREG/CR-7151, Vol. 1

Development of a Fault Injection-Based Dependability Assessment Methodology for Digital I&C Systems

Volume 1

Manuscript Completed: November 2011
Date Published: December 2012

Prepared by:
C. R. Elks, N. J. George, M. A. Reynolds, M. Miklo,
C. Berger, S. Bingham, M. Sekhar, B. W. Johnson

The Charles L. Brown Department of Electrical
and Computer Engineering
The University of Virginia
Charlottesville, Virginia

NRC Project Managers:
S. A. Arndt, J. A. Dion, R. A. Shaffer, M. E. Waterman

NRC Job Code N6214

Prepared for:
Division of Engineering
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001

Office of Nuclear Regulatory Research

NUREG/CR-7151, Vols. 1 to 4 have been reproduced from the best available copy.

ABSTRACT

Today's emergent computer technology has introduced the capability of integrating information from numerous plant systems and supplying needed information to operations personnel in a timely manner that could not be envisioned when previous generation plants were designed and built. For example, Small Modular Reactor (SMR) plant designs will make extensive use of computer based I&C systems for all manner of plant functions, including safety and non-safety functions. On the other hand, digital upgrades in existing light water reactor plants are becoming necessary in order to sustain and extend plant life while improving plant performance, reducing maintenance costs of aging and obsolete equipment, and promoting prognostic system monitoring and human machine interface (HMI) decision making.

The extensive use of digital instrumentation and control systems in new and existing plants raises issues that were not relevant to the previous generation of analog and rudimentary digital I&C systems used in the 1970's style plants. These issues include the occurrence of unknown failure modes in digital I&C systems and HMI issues. Therefore, digital system reliability/safety, classification of digital I&C system failures and failure modes, and software validation remain significant issues for the Light Water Sustainability and SMR initiatives and the digital I&C system community at large.

The purpose of the research described in volume 1 thru volume 4 is to help inform the development of regulatory guidance for digital I&C systems and potential improvement of the licensing of digital I&C systems in NPP operations. The work described herein presents; (1) the effectiveness of fault injection (as applied to a digital I&C system) for providing critical safety model parameters (e.g., coverage factor) and system response information required by the PRA and reliability assessment processes, (2) the development and refinement of the methodology to improve applicability to digital I&C systems, and (3) findings for establishing a basis for using fault injection as applied to a diverse set of digital I&C platforms. Some of the specific issues addressed in Volume 1 are:

- Fault Injection as a support activity for PRA activities.
- Development of the UVA fault injection based methodology.
- Fault models for contemporary and emerging IC technology in Digital I&C Systems.
- Requirements and challenges for realizing Fault Injection in Digital I&C systems.
- Solutions to challenges for realizing fault injection in digital I&C systems.

Volume 1 presents the findings of developing a fault injection based quantitative assessment methodology with respect to processor based digital I&C systems for the purpose of evaluating the capabilities of the method to support NRC probabilistic risk assessment (PRA) and review of digital I&C systems. Fault injection is defined as a dependability validation technique that is based on the realization of controlled validation experiments in which system behavior is observed when faults are explicitly induced by the deliberate introduction (injection) of faults into the system [Arlat 1990]. Fault injection is therefore a form of *accelerated testing* of fault tolerance attributes of the digital I&C system under test.

Volumes 2 and 3 of this research present the application of this methodology to two commercial-grade digital I&C system executing a reactor protection shutdown application.

In Volumes 2 and 3, the research identified significant results related to the operational behavior of the benchmark systems, and the value of the methodology with respect to providing data for the quantification of dependability attributes such as safety, reliability, and integrity. By applying a fault injection-based dependability assessment methodology to a commercial grade digital I&C, the research provided useful evidence toward the capabilities and limitations of fault

injection-based dependability assessment methods with respect to modern digital I&C systems. The results of this effort are intended to assist NRC staff determine where and how fault injection-based methodologies can best fit into the overall license review process.

The cumulative findings and recommendations of both applications of the methodology and application of the generalized results to broader classes of digital I&C systems are discussed in volume 4.

The digital I&C systems under test for this effort, herein defined as Benchmark System I and Benchmark System II, are fault tolerant multi-processor safety-critical digital I&C systems typical of what would be used in a nuclear power plant 1-e systems. The benchmark systems contain multiple processing modules to accurately represent 4 channel or division 2 out of 4 reactor protection systems. In addition, the systems contain a redundant discrete digital input and output modules, analog input and output modules, inter-channel communication network modules, other interface modules to fully represent and implement a Reactor Protection System. The application Reactor Protection System software was developed using the benchmark systems software development and programming environments.

To establish a proper operational context for the fault injection environment a prototype operational profile generator tool based on the US NRC systems analysis code TRACE [NRC 2011] was developed. This tool allowed generation of realistic system sensor inputs to the Reactor Protection System (RPS) application based on reactor and plant dynamics of the simulated model. In addition, the tool allowed creation of accident events such as large break LOCAs, turbine trips, etc., to stress the RPS application under the various design basis events.

Bibliography

[NRC 2001] Commission, U.S. Nuclear Regulatory. *Computer Codes.* April 2011. http://www.nrc.gov/about-nrc/regulatory/research/comp-codes.html (accessed 2011).

[Arlat 1990] J. Arlat, M. Aguera, et. al. "Fault Injection for Dependability Evaluation: A Methodology and Some Applications." *IEEE Transactions on Software Engineering*, February 2 , 1990.

FOREWORD

As discussed in the NRC Policy Statement on Probabilistic Risk Assessment (PRA), the NRC intends to increase its use of PRA methods in all regulatory matters to the extent supported by state-of-the-art PRA methods and data. Currently, I&C systems are not modeled in PRAs. As the NRC moves toward a more risk-informed regulatory environment, the staff will need data, methods, and tools related to the risk assessment of digital systems. Fault injection methods can provide a means to estimate quantitatively the behavior model parameters of the system. The quantification of these parameters (in a probabilistic sense) can be used to produce more accurate parameter estimates for PRA models, which in turn produces more a accurate risk assessment to inform the risk oversight process.

A challenge for evaluating system reliability relates to relatively undeveloped state of the art methods for assessing digital system reliability. Quantitative measures of digital system reliability are available for digital system hardware, but procedures for evaluating system level reliability (both hardware and software) are not well defined in current industry literature. However, comprehensive use of fault injection techniques for providing critical data toward evaluating digital system dependability may reduce software reliability uncertainties.

The conduct of fault injection campaigns often yields more information than just quantifying the fault tolerance aspects of a system; it also is a means to circumspect and comprehend the behaviors of complex fault tolerant I&C systems to support overall assessment activities for both the developer and the regulator. Fault injection experiments cannot be performed without gaining a deeper understanding of a system. The process itself is a learning experience, providing richer insights into how a system behaves in response to errors arising from system faults. The inclusion of fault injection information into review processes and PRA activities can enlighten the review processes of digital I&C systems. Finally, the process of conducting fault injection testing allows two very important pieces of information to come into direct connection with each other: what the system is supposed to do, and what it actually does. This information is essential for anticipating system behaviors, performing verification and validation (V&V) activities, and conducting methodical system evaluations.

This report describes an important step toward developing a systematic method of evaluating digital system dependability. Volume 1 presents a broad and in-depth development of a digital system dependability methodology, and the requirements and challenges of performing fault injections on digital I&C systems. The process developed in this research project was applied to two digital systems that modeled nuclear power plant safety functions. The results of this phase of the research are described in volume 2 and volume 3. The cumulative findings and recommendations of both applications of the methodology and application of the generalized results to broader classes of digital I&C systems are discussed in volume 4.

TABLE OF CONTENTS

TABLE OF CONTENTS
(continued)

TABLE OF CONTENTS
(continued)

TABLE OF CONTENTS
(continued)

LIST OF FIGURES

LIST OF TABLES

ACRONYMS AND ABBREVIATIONS

ABV	Assertion-based Verification
ABVFI	Assertion-based Verification Fault Injection
ANS	American Nuclear Society
ANSI	American National Standards Institute
BDM	Background Debug Mode
CCF	Common Cause Failure
CFR	Code of Federal Regulations
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CRDM	Control Rod Drive Mechanism
DFWCS	Digital Feedwater Control System
EDM	Error Detection Mechanism
EMI	Electromagnetic Interference
ESFAS	Engineered Safety Features Actuation System
FARM	Faults, Activations, Readouts, and Measures
FDIM	Fault Detection, Isolation, and Mitigation
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
HIRF	High Intensity Radiated Fields
HMI	Human Machine Interface
HW	Hardware
I&C	Instrumentation and Control
IBM	International Business Machines
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IC	Integrated Circuit
I/O	Input/Output
ISA	Instruction Set Architecture
JTAG	Joint Test Action Group
LANSCE	Los Alamos Neutron Science Center
LOCA	Loss of Coolant Accident
LWFI	Light Weight Fault Injector
MI	Monitor Interface
MP	Main Processor
MPU	Main Processor Unit
NPP	Nuclear Power Plant
NRC	Nuclear Regulatory Commission
PTLsim	Processor Technology Laboratory Simulator
PRA	Probabilistic Risk Assessment
PSD	Power Supply Disturbance
RAM	Random Access Memory
RC	Reverberation Chamber
RE	Runtime Executive
RF	Radio Frequency
RG	Regulatory Guide
RPS	Reactor Protection System
RTL	Register Transfer Level
SCADA	Supervisory Control and Data Acquisition
SEU	Single Event Upset
SMR	Small Modular Reactor
SoC	Systems on a Chip

SW	Software
SWIFI	Software Implemented Fault Injection
TC	Target Computer
TMR	Triple Modular Redundant
TRACE	TRAC/RELAP Advanced Computational Engine
UVA	University of Virginia
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLSI	Very Large Scale Integration

1. INTRODUCTION

1.1. Background

This report is volume 1 of a multi-volume set of reports that present the cumulative efforts, findings, and results of U.S. Nuclear Regulatory Commission (NRC) contract JCN N6124 – "Digital System Dependability Performance". The reports are organized as follows:

Volume 1 – Presents a broad and in-depth development of the methodology, the requirements and challenges of realizing fault injection on digital instrumentation and control (I&C) systems.

Volume 2 – Presents the application of the methodology to Benchmark System I.

Volume 3 – Presents the application of the methodology to Benchmark System2- employing the lessons learned from Benchmark System I.

Volume 4 – Presents the cumulative findings and recommendations of both applications of the methodology and generalizes the results to broader classes of digital I&C systems.

1.2. Purpose

This report (Volume 1) presents the findings of developing a fault injection-based quantitative assessment methodology with respect to processor-based digital I&C systems for the purpose of evaluating the capabilities of the method to support NRC probabilistic risk assessments (PRAs). Another purpose of this work is to help inform the development of regulatory guidance processes for digital I&C systems and potential improvements of the licensing process for digital I&C systems in nuclear power plant (NPP) safety systems. The work described herein broadly presents: (1) a theory and methodology for fault injection (as applied to a digital I&C system), (2) the usefulness of providing critical parameters and information required by the PRA and reliability assessment processes, (3) the challenges to applying a fault injection method to contemporary digital I&C systems, and (4) the findings for addressing these challenges and establishing a basis for implementing fault injection for digital I&C platforms.

1.3. Background and Motivation

Given the revitalization of the nuclear power industry in this country, there is near uniform agreement in the nuclear industry that significant technology and production challenges must be addressed to enable e3fficient construction of new plants and refurbishment of existing plants. These challenges are largely being driven by the need to extend the life of current operating NPPs by an additional 20 to 30 years (up to 60 years total plant life) to meet projected energy consumption demands while new plants are constructed and licensed to operate [Energy 2011].

Next generation NPPs and modernized plants will be fundamentally different from their predecessors. Today's emergent computer technology has introduced the capability of integrating information from numerous plant systems and supplying needed information to operations personnel in a timely manner that could not be envisioned when previous generation plants were designed and built. At present, numerous versions and different types of new advanced digital I&C systems are in the regulatory licensing application process. However, with the introduction of software-based and hardware description language-based I&C systems for NPP control and monitoring, new human-machine integration, potential digital failure mode issues have arisen that could adversely affect safety and security [Committee 1997]. However, the need for these digital I&C systems to be as dependable as their predecessors across a wide spectrum of threats, faults, and failures to ensure public safety is of the utmost importance.

In recent years, significant effort has gone into improving safety critical system design methodologies, assessment methods, and the updating of regulatory industry standards and NRC regulatory guidelines to ensure that digital I&C systems can be designed and assessed to the high safety requirement levels required for highly critical applications. Of particular interest recently are quantitative dependability assessment methodologies that employ fault injection methods to ensure proper compliance of digital I&C system fault handling mechanisms [Arlat 1990(a); Yu 2004; Smith D., 2000; Elks 2009(a); Aldemir 2007; Smidts 2004]. The goal of a dependability assessment methodology is to provide a systematic process for characterizing the safety and performance behavior of embedded systems (e.g., digital I&C systems) in the presence of faults.

Dependability evaluation involves the study of failures and errors and their potential impact on system attributes such as reliability, safety, and security. Very often the nature of failures or crashes and long error latency often make it difficult to identify the causes of failures in the operational environment. Thus, it is particularly hard to recreate a failure scenario for large, complex systems just from failure logs alone. To identify and understand potential failures, the use of an experiment-based or measurement based approach for studying the dependability of a system is gaining acceptance in the nuclear industry for better understanding the effects of errors and failures to promote an informed understanding of risk. Such an approach is useful not only during the concept and design phases, but also during licensing review activities.

From a practical point of view, most digital I&C systems are designed to be safety critical employing extensive fault detection/tolerance and design diversity features to ensure proper fail operational and fail safe behavior in the event of a system failure. For example, Fault Detection, Isolation, and Mitigation (FDIM) software or online diagnostic functions of the benchmark systems in this research effort account for as much as 40 to50 percent of the executable system software code [Barton 1990; Palumbo 1986; Young 1989]. This code is rarely exercised in the real world because faults and failures are an infrequent occurrence. This FDIM code is vital toward system dependability and safety compliance, and can only be effectively tested and validated by realistic fault injection campaigns.

1.4. Relevance of Research with Respect to Regulatory Guidance

The NRC has created (and continues to improve) a comprehensive set of regulatory guidelines for reviewing and assessing the safety and functionality of digital I&C systems. The NRC PRA technical community has not yet agreed on how to model digital system reliability in the context of a PRA and the level of detail that digital I&C systems require in reliability modeling. It is clear that PRA models must adequately represent the complex system interactions that can contribute to digital system failure modes. Nonetheless, the essential research aim of the PRA technical community is to accurately model digital I&C systems behaviors that take into account interactions of the system, fault handling behaviors, coverage of fault tolerance features, and the view of the system as a integrated software and hardware system.

A central part of this framework is Appendix B to Title 10 of the Code of Federal Regulations (CFR) Part 50 (10 CFR 50). The purpose of Appendix B is to establish broad quality assurance requirements for the design, development, and operation of systems, and components whose operation is deemed critical to the safe operation of nuclear power plants. The pertinent requirements of this appendix apply to all activities affecting the safety related functions of those systems, and components. In support of Appendix B 10 CFR 50 are Regulatory Guide (RG) 1.153, RG 1.152 and RG 1.168, which endorse the Institute of Electrical and Electronics Engineers (IEEE) standards IEEE Std 603, IEEE Std 7-4.3.2, and both IEEE Std 1012 and IEEE Std 1028, respectively, as a basis for compliance to regulatory requirements for software

based systems. NUREG-0800 Section 7 describes the standard review plan criteria for safety analysis reports with respect to digital I&C systems. Taken all together, these laws, guidelines and reports establish the criteria for review of the applicant/licensee's testing, analysis, and technical justification documents for installing a safety-related digital I&C system into a plant. Importantly, the applicant must show that the I&C system design, including the underlying design bases and performance requirements, can perform appropriate safety functions in the presence of normal and off normal operating conditions (including faulted and failure behavior of the I&C system). For example, IEEE Std 603-1991 contains the criteria for determining the continued functioning of safety systems in the presence of faults:

> *"The methods to be used to determine that the reliability of the safety system design are appropriate for each safety system design and any qualitative or quantitative reliability goals that may be imposed on the system design."*
>
> *Clause 5.1* ***Single-Failure Criterion.*** *The safety systems shall perform all safety functions required for a design basis event in the presence of:*
> *(1) any single detectable failure within the safety systems concurrent with all identifiable but non-detectable failures;*
> *(2) all failures caused by the single failure; and*
> *(3) all failures and spurious system actions that cause or are caused by the design basis event requiring the safety functions.*
>
> *The single-failure criterion applies to the safety systems whether control is by automatic or manual means. . . .*

Fault injection as part of a quantitative assessment process is a robust testing process that can support verification and validation and question and answer activities to gather evidence that the I&C system can perform its safety functions in the presence of faulted and failure conditions, which would be in direct compliance with IEEE Std 603-1991. In addition, those aspects of appendix B of 10CFR50, NUREG-0800 and other relevant guidelines that address requirements for testing processes, methods and evidence to support safety function operational effectiveness are clear candidates for the application of fault injection methods. Fault injection is *a formal-based process* to collect evidence to gauge the dependability of safety functions associated with I&C systems, that has an underlying mathematical theory (with explicitly stated assumptions), which allows one to place stronger justification or refutation to the claims of the overall design and safety of the I&C system.

1.4.1. Relationship to NRC Research Activities

The research conducted under this contract was performed with consideration of previous and on-going research efforts related to the safety and reliability assessment of digital I&C systems. Accordingly, the research effort was attentive of complementary research efforts and how those efforts could benefit from the work accomplished through this effort. Specifically, the researchers recognized that the products developed from this research had the potential to be used in other research efforts. Therefore researchers thus endeavored to catalog research findings in way that promoted broader relevance and helpful information for other research efforts.

1.4.2. Research Objectives

The overall objective of this work was to develop a body of evidence to inform the development of regulatory guidance processes for digital I&C systems and potentially improve the licensing process of digital I&C systems in NPP operations. In support of this objective the research investigated the effectiveness of fault injection (as applied to digital I&C systems) for providing

critical parameters and information required by PRA and reliability assessment processes. The results and findings of this effort are aimed at assisting NRC staff determine when, where and how fault injection based methodologies can best fit in the overall license review process. The major goals of the research effort are listed below:

Objective 1
Demonstrate the effectiveness of the University of Virginia quantitative safety assessment process on commercial safety grade I&C systems executing reactor protection applications with respect to a simulated NPP safety system design.

Objective 2
Identify, document, and develop improvements to the process that make it easier and more effective to apply to a wider spectrum of digital I&C systems. Document the limitations, sensitive assumptions, and implementation challenges that would encumber the application of fault injection processes for digital I&C systems.

Objective 3
Document the quantitative and qualitative results that can be obtained through application of the assessment process, and provide the technical basis upon which NRC can establish the regulatory requirements for safety-related digital systems, including the acceptance criteria and regulatory guidance documents.

Secondary Objective 1
Assess the level of effort and cost for implementing fault injection capability in a vendor or licensee environment.

Secondary Objective 2
Identify and develop innovative fault injection methods that would make fault injection more efficient and easier to adopt by NRC and the nuclear industry.

The scope of this work is targeted at safety critical digital I&C systems, but applies to non-safety related systems as well. The target benchmark systems were configured to be representative of a four-channel Reactor Protection System (RPS) system, but were limited in a scale due to budget constraints on equipment availability. Therefore, the systems lacked some redundant hardware modules that would normally be found in an actual RPS. The overall complexity and configuration of the system was sufficient to stress the methodology, which was the objective of the research effort. The specific benchmark system data results obtained from the study should be interpreted with respect to the benchmark system configuration described in this report unless otherwise stated.

In addition, the methodology that was developed and applied in this research effort is part of a larger comprehensive assessment and review process, and is not intended to be interpreted as a “replacement” for existing processes. Rather the methodology is viewed as a complementary method to support existing and emerging design assurance and license review processes in an effort to establish more efficient, repeatable and objective design assessment and review processes.

1.4.3. Scope of Study

Fault injection-based methodologies are but one part of a comprehensive process of estimating the reliability of digital systems (hardware and software) for the purpose of PRA applications. From the highest level perspective, the essential pieces of information needed for reliability estimation are (1), the knowledge of the likelihood of faults (software or hardware) and (2) the consequence of activating these faults in the system context. Fault injection methods are most

useful in characterizing system responses to activated faults - the second requirement. That is, providing empirical knowledge on the triggering, detection, tolerance, and propagation of errors due to software (SW) or hardware (HW) faults in the system. How a digital I&C system responds to faults and mitigates faults are essential elements for accurate system reliability modeling. As such, the methodology developed and presented here is aimed at providing empirical data in support of estimating system fault response data, such as fault detection, error propagation, fault latency, timing delays, etc.

1.5. Project Organization and Timeline

This project was carried out in three phases. The first phase, which is principally reported in this volume developed and refined the methodology so that it could be applied to the benchmark systems. The second phase of the work applied the methodology to the first benchmark system based on the recommendations and plan of action from the first phase of the work. The third and final phase of the work applied the methodology to a second benchmark system based on the lessons learned from the first and second phase of the work. Figure 1-1 shows the progression of this effort through the lifecycle of the project.

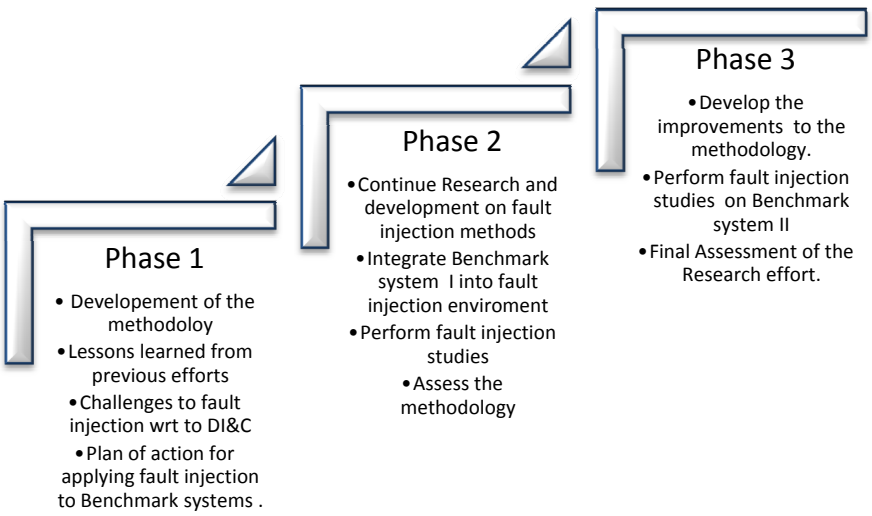

Figure 1-1 Phases and activities of the research effort

1.6. Organization of this Report

This report is intended to provide a contemporary and comprehensive perspective on fault injection for digital I&C systems. In addition, this report also provides a broad and deep perspective on fault injection with specific focus on digital I&C systems. This report is organized around three main themes: (1) concepts of dependable systems, (2) theory of fault injection with respect to digital I&C systems, and (3) fault injection-based assessment methodology issues and challenges. Each Section builds on and connects to previous Sections. Sections 1, 2 and 3 provide a broad and deep foundation for understanding the concepts of fault injection with respect to digital I&C systems. Section 4 presents a detailed overview of the proposed methodology prior to the outset of this research. Section 5 presents an in-depth survey and characterization of the state of the art in fault injection practices. Section 6 describes the digital I&C benchmark systems to be used in this study. Section 7 presents lessons learned from previous research efforts on fault injection. Section 8 identifies the challenges and issues with

implementing a fault injection process on digital I&C systems, and provides recommendations for resolving the challenges.

1.7. Concepts

This section introduces the basic concepts needed to understand the context of this research. These concepts include overview of modern digital I&C systems, definitions, safety metrics, and an overview of fault injection principles.

1.7.1. Digital and Computer Based I&C Systems: Overview

In order to provide relevance beyond the benchmark systems that were evaluated, the research developed a generic representation of an I&C system based on the evaluation of several current digital I&C systems being proposed for new reactor applications and the emerging technologies that may be used in new I&C systems. The characterization that seemed most suitable is illustrated in Figure 1-2.

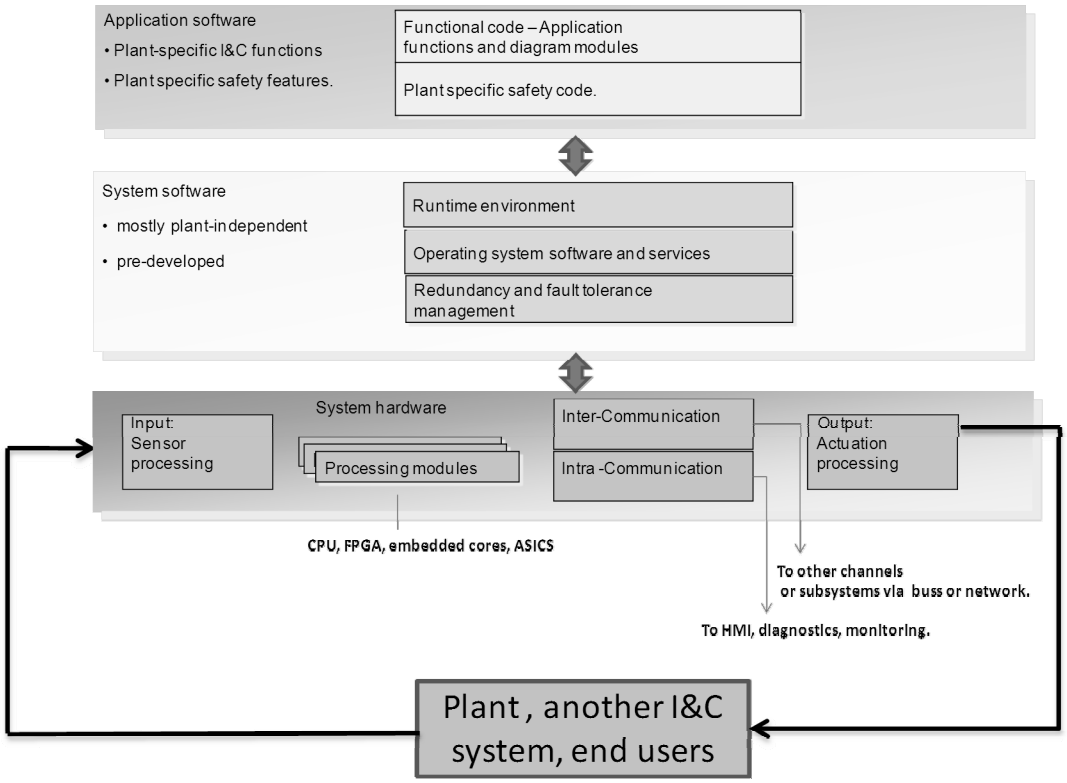

Figure 1-2 Generic digital I&C system architecture model

Modern digital I&C systems characteristic of the systems addressed in this research are neither strict embedded systems nor general purpose computing platforms. Rather, these systems fall into a special class of embedded computing platforms called *adaptive or configurable embedded computing*. That is, the hardware and software architectural elements of the platform allow the architecture to be tailored to specific constraints of the application domain. To achieve such flexibility the architecture may trade-off attributes such as optimal performance, simplicity, and cost with respect to a fully custom embedded system. Most I&C systems being considered for NPP applications fall into this class of systems. Figure 1-2 illustrates several important concepts of modern I&C systems, specifically:

(1) Application adaptive functionality: Configurable to different plant designs, ability to change parameters and programming to optimize performance and safety.

(2) Layered Architecture: Separation of Application independent functions and Application dependent functions.

(3) Intra-communication and inter-communication functionality to facilitate integration of I&C system operational information to other plant systems and personnel.

(4) Redundancy and diversity to support fail-safe operation and/or degraded operation in the presence of faults and failures.

(5) Interfaces and sub-systems to support health monitoring of I&C operations and system behavior.

(6) Interfaces to support operator monitoring and actions.

Digital I&C systems are used in NPP systems such as safety systems, plant process control systems, monitoring systems, data communication systems, and sensor processing systems. The digital I&C systems depicted in Figure 1-2 serve a variety of functions within NPP operations. The generic digital I&C architecture provides a mapping of function to form, and implementations to realize the functionality.

The wide range of uses illustrates that digital I&C systems are not just characterized by their internal form and function, but also by their interaction context with the environment in which they operate. Context is important. These types of systems may interact with other systems through communication systems and may indirectly interact with other systems through plant dynamics. Context establishes the basis for what a system is supposed to do, and what a system is not supposed to do. Context establishes the basis for what reasonable assumptions should be made concerning the assessment of a digital I&C system and what relevant conclusions can be made from the assessment process. The research described in this report strived to keep this principle in mind as the methodology was developed and tested.

Another overarching aspect of modern digital I&C systems (and embedded systems) is their use of programmable elements throughout the design and implementation. Traditionally, embedded systems were viewed from two perspectives: the software and the hardware. This perspective is largely driven by the development processes that realize the functionality of the system. Software enables the system to perform its intended functionality in the context of its environment. Hardware provides the necessary programmability to allow software to be flexible to different applications. Neither view by itself is representative of actual behavior.

Software does nothing without hardware to animate it. Hardware is just an organization of digital functions and signals. The digital I&C system should be viewed from a unified perspective, seeing software and hardware not as completely different domains but rather as an integrated system to achieve a purpose that is more representative of the functionality that the I&C system implements.

This unified view of I&C systems is often represented at the object code level or register transfer level (RTL) in digital I&C systems. It is here that the interactions of software and hardware take place. This is an important concept because the failure of a hardware function can adversely affect the functionality and reliability of the software relying upon that function. In like manner, a

design flaw in a software function or improper programming can produce different errors or failures depending on the hardware architecture and organization. Digital I&C assessment methodologies, therefore, should be flexible enough to allow this multi-level view of the application.

Recent trends in digital technology advances have strengthened this integrated view to the point where there is less distinction between software and hardware. Integrated circuit and microelectronic capacities have increased to the point that both software processors and custom hardware now commonly coexist on a single integrated circuit (IC) package – these Systems on a Chip (SoC) have become very common. This is particularly true of field programmable gate array (FPGA) technology in which embedded processor cores, network and bus protocol engines, analog-to-digital and digital-to-analog conversions, and memory management functions are often mapped into a single FPGA structure. FPGA and SoC technologies are hardware that act like software. Users can change the hardware organization at any time during the design, development, and field operation of FPGAs to meet the changing needs of the customers. This technology is already finding its place in digital I&C systems – as both digital I&C systems used in this research effort employed FPGA technology.

1.7.2. Concepts of Dependable Systems

This section provides informative background discussions on the concepts of dependable systems. Dependable systems concepts developed by [Avizienis 2004] are widely recognized as a standard for understanding the concepts of critical system attributes. This report adopts the concepts of the Avizienis 2004] model of dependability to provide consistent terminology when discussing faults, errors, and failures with respect to digital I&C systems. A review of the [Avizienis 2004] paper provides a more detailed appreciation of the topic. A *dependable real-time system* has the ability to provide its intended, expected and agreed upon functions, behavior, and operations in a correct timely manner [Avizienis 2004]. The alternate definition is the ability to avoid service failures that are more frequent and more severe than is acceptable.

1.7.2.1. System Function, Behavior, Structure, and Service

In dependability theory, a *system* is defined as an entity that interacts with other entities, i.e. other systems, including hardware, software, humans, and the physical world. Digital I&C systems in NPPs perform these interactions to control, monitor, and actuate plant systems. These interactions with other systems are *the environment* or *the context* of a given system. The border between the system and its environment is the *system boundary or common interface*.

Digital I&C and communication systems are often characterized by fundamental properties: functionality, performance, safety, reliability, security, and cost. The function of a system is what the system is intended to do and is described by the functional specification in terms of functionality and performance. The behavior of a system is what the system does to implement its function and is described by a sequence of states. The total state of a system is the following set of the states: computation, communication, stored information, interconnection (e.g., inputs and feedback), and physical condition. As discussed in Section 1.7.1, the structure or composition of a digital I&C system is what enables it to generate behaviors

The *service* delivered by a digital I&C system is its behavior as it is perceived by its user(s); a user is another system that receives service from the given digital I&C system. The delivered service is a typically a sequence of information in the form of commands, state status, and requested information (e.g., sensor data). A system may sequentially or simultaneously be

giving and receiving with respect to another system (i.e., deliver service to and receive service from the other system).

1.7.2.2. The Attributes of Dependable Systems

The attributes of dependable systems are the primary means by which the quantitative and qualitative requirements of a system are specified. Following are some basic terms and concepts related to dependable system attributes as used in this report:

- ***Definition 1: Reliability***, a conditional probability that the system will perform correctly throughout the interval [t_0, t], given the system was performing correctly at time t_0 [Johnson 1989], which is related to the continuity of service.
- ***Definition 2: Availability***, a probability that a system is operating correctly and is available to perform its functions at the instant time, t [Johnson 1989], which is related to readiness for usage.
- ***Definition 3: Safety*****,** a 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 [Johnson 1989], which is related to the non-occurrence of catastrophic consequences on the environment.
- ***Definition 4: Integrity***: absence of improper system alterations [Johnson 1989].
- ***Definition 5: Maintainability***: ability to undergo modifications and repairs [Johnson 1989].

1.7.2.3. Impairments to Dependability

Faults, errors, and failures affect the ability of a system to deliver its dependability attributes (e.g., safety, reliability, performance, etc.). Hence, they are called the *impairments* of dependability.

Correct service is delivered when the service accurately reflects the system function. A *service failure*, abbreviated here to *failure*, is an event that occurs when the delivered service deviates from correct service. A failure is either because the service does not comply with the functional specification, or because this specification did not adequately describe the system function. A failure is a transition from correct service to incorrect service (i.e., not implementing the system function). The deviation from correct service may assume different forms that are called *failure modes*.

Since delivered service is a sequence of the digital I&C system states, a service failure means that at least one (or more) external state of the system deviates from the correct service state. The deviation is called an *error*. *Error propagation* occurs when an error is successively transformed into other errors through execution of functions on the digital system, (e.g., errors from component A propagate to component B when it receives information from component A). At this time, service delivered by A to B becomes incorrect, and the ensuing service failure of A appears as an external fault to B and propagates the error into B via its use interface. A service failure occurs when an error is propagated to the system interface and causes the service delivered by the system to deviate from correct service.

The cause of an error is called a *fault*. Faults can be internal or external to a system. A fault is active when it produces an error; otherwise, it is dormant. An active fault is either 1) an internal fault that was previously dormant and that has been activated by the computation process and/or environmental conditions, or 2) an external fault. *Fault activation* is the application of an input pattern to a component that causes a dormant fault to become active.

Implicit in the definitions of the above terms is a cause-effect relationship. The well-known 3-universe model depicted in Figure 1-3 shows the relationship between faults, errors, and failures [Johnson 89]. Faults cause errors, and errors may propagate to the system interface to cause service failures.

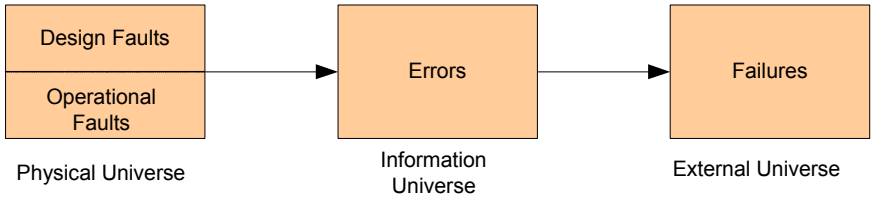

Figure 1-3 Cause-effect relationship among faults, errors, and failures using the 3-Universe Model

1.7.3. A Taxonomy of Faults for Digital I&C Systems

This section presents a taxonomy of threats that may affect a digital I&C system during its lifetime. The taxonomy is derived from [Avizienis 2004], and is very general in its domain of applicability. The purpose of this fault taxonomy is to present a complete and structured view of the domain of faults applicable to digital I&C systems. Since main purpose of this research effort was to assess the applicability and utility of fault injection, it is reasonable to start with a well-structured and complete view of the fault space. The taxonomy presented in this section is complete with respect to the types of fault classes in digital I&C systems, however, it may be the case that certain fault classes in the taxonomy may not occur due to the digital I&C system environment or the design and operational aspects of the digital I&C system. Farther research along these lines may be helpful for confirming the taxonomy with respect to digital I&C systems. This research performed a preliminary comparison of the fault classes in the taxonomy with those compiled in a recent survey effort by the NRC[1]. Knowledge of all possible fault classes allows the user to decide which classes should be included in the assessment process.

To begin, a digital I&C system *life cycle* consists of two phases: *development* and *operations*. **The development phase** includes all activities from presentation of the user's initial concept to the decision that the system has passed all acceptance tests and is ready to be deployed for use. **The operational phase** of a system life cycle begins when the system is accepted for use and placed in service. Operations consist of alternating periods of service delivery, service outage, and service shutdown. A *service outage* is caused by a service failure. It is the period when incorrect service (including no service at all) is delivered by the system. A *service shutdown* is an intentional halt of service by an authorized entity. Maintenance actions may take place during all three periods of the operational phase. During the operational phase the system interacts with its *operating environment* and may be adversely affected by faults

[1] As a cursory check on the above taxonomy, survey results from *Industry Survey of Digital I&C Failures ORNL report ORNL/TM-2006/626* are cross referenced with the above fault taxonomy. All fault reports in the ORNL reports were classifiable in one of the three fault classifications for the above fault taxonomy.

originating in it. According to [Avizienis 2004] taxonomy; all faults that may affect a system during its life are classified according to eight basic attributes:

(1) The **phase** of the system life during which the faults originate:

- *Development faults* that occur during (a) system development, (b) generation of procedures to operate or to maintain the system;
- *Operational faults* that occur during service delivery of the operational phase.

(2) The **location** of the faults with respect to the system boundary:

- *Internal faults* that originate inside the system boundary;
- *External faults* that originate outside the system boundary and propagate errors into the system by interaction or interference.

(3) The phenomenological cause of the faults:

- *Natural faults* that are caused by natural phenomena without human participation;
- *Human-made faults* that result from human actions.

(4) The **dimension** in which the faults originate:

- *Hardware (physical) faults* that originate in, or affect, hardware;
- *Software (information) faults* that affect software, i.e., programs or data.

(5) The **objective** of introducing the faults:

- *Malicious faults* that are introduced by a human with the malicious objective of causing harm to the system;
- *Non-malicious faults* that are introduced without a malicious objective.

(6) The **intent** of the human(s) who caused the faults:

- *Deliberate faults* that are the result of a harmful decision;
- *Non-deliberate faults* that are introduced without awareness.

(7) The **capacity** of the human(s) who introduced the faults:

- *Accidental faults* that are introduced inadvertently;
- *Incompetence faults* that result from lack of professional competence by the authorized human(s), or from inadequacy of the development organization.

(8) The temporal persistence of the faults:

- *Permanent faults* whose presence is assumed to be continuous in time;
- *Transient faults* whose presence is bounded in time.

If all combinations of the eight elementary fault classes were possible, there would be 256 different *combined fault classes.* In fact, according to [Avizienis, 2004] there are only 31 likely combinations; which are shown in Figure 1-4.

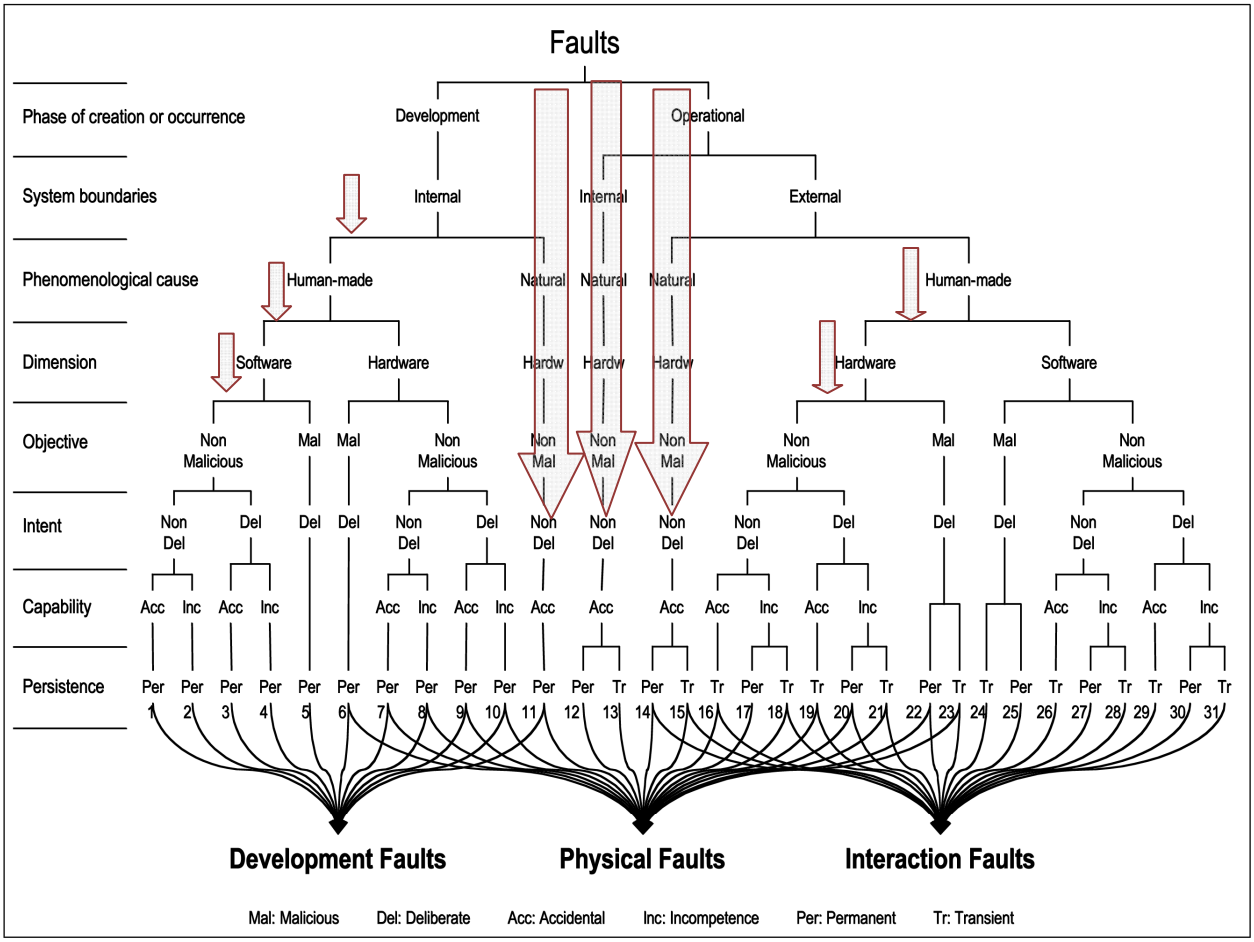

Figure 1-4 Taxonomy of fault classes for digital I&C systems [Avizienis 2004]

Referring to Figure 1-4, the combined faults are shown to belong to three major partially overlapping groupings:

- **Development faults** that include all fault classes occurring during system development;
- **Physical faults** that include all fault classes that affect hardware;
- **Interaction faults** that include all external faults.

The overlap between groupings of fault classes shows that origins of a fault along with its dimension (software or hardware) are important in describing the complete nature of a fault. As a case in point, from Figure 1-4 the development faults listed in columns 6-11 are both grouped in development and physical fault classes. This means that the origins of these faults are in the development processes of the hardware, and their manifestation occurs in the physical fault

domain - hardware. Examples of these types of faults are manufacturing defects and hardware design flaws. Accordingly, the physical fault class in Figure 1-4 is overlapped with the hardware dimension of the other two fault classes.

Important classes of faults in the above taxonomy are **interaction faults**, which occur during the operational phase; therefore they are operational faults. These faults are caused by the interaction of the digital I&C system with its operational environment.

Most classes of faults originate due to some human action in the operational environment; therefore, they are of the type *human machine interaction faults*. These are represented in fault classes 16-31 in Figure 1-4. An exception to human machine interaction faults are external natural faults (e.g.,14-15) caused by external disturbances such as cosmic particle strikes in the ICs of the digital system, and electromagnetic interference (EMI) disturbances. Another important class of faults in the taxonomy is configuration faults, i.e., wrong setting of parameters that can affect the correct operation of the system. These faults can occur in the control logic, display symbols, diagnostics, communication protocol of the digital I&C system etc. Such faults can occur during configuration changes performed during maintenance periods.

Finally, the arrows depicted in Figure 1-4 indicate the types of significant fault classes known to occur in digital I&C systems, which were investigated in the development of the fault injection methodology described in this report. The broad arrows indicate the principle aim of the research, which was to find representative fault models for these physical-based fault classes and determine the feasibility of using physical-based faults injected into the benchmark systems.

The smaller arrows indicate to a lesser extent the research objective of emulating software and interaction-based fault classes through fault injection techniques. Due to the limited scope and time of this research effort, not every fault class in the above taxonomy was investigated with respect to fault injection. However, the above fault classes are representative and provide a vehicle to investigate the capabilities of applying fault injection to digital I&C systems. Section 4 describes the fault models for the above fault classes intended to be implement based on the above taxonomy.

1.8. References

[Avizienis 2004] A. Avizienis, J.C. Laprie. "Basic Concepts and Taxonomy of Dependable and Secure Computing." *IEEE Transactions on Dependable and Secure Computing Archive*, 2004: vol. 1.

[Arlat 1990(a)] Arlat, J., M. Aguera, Y. Crouzet, J.-C. Fabre, E. Martins, and D. Powell. "Experimental evaluation of the fault tolerance of an atomic multicast system." *#IEEE_J_R#* 39, no. 4 (1990(a)): 455-467.

[Elks 2009(a)] C. Elks, B.W. Johnson, M. Reynolds. "A Perspective on Fault Injection Methods for Nuclear Safety Related Digital I&C Systems." *6th International Topical Meeting on Nuclear Plant Instrumentation Control and Human Machine Interface Technology.* Knoxville, TN: NPIC&HMIT, 2009(a).

[Smidts 2004] C. Smidts, M. Li. *Validation of a Methodology for Assessing Software Quality.* NUREG/CR-6848, Washington, D.C.: NRC, Office of Nuclear Regulatory Research, 2004.

[Committee 1997] Committee, CETS. *Digital Instrumentation and Control Systems in Nuclear Power Plants: Safety and Reliability Issues.* National Academic Press, 1997.

[Smith 2000] D. Smith, T. DeLong, B.W. Johnson. "A Safety Assessment Methodology for Comples Safety Critical Hardware/Software Systems." *International Topical Meeting on Nuclear Plant Instrumentation, Controls, and Human-Machine Interface Technology.* Washington, D.C., 2000.

[Energy 2011] Energy, Department of. *U.S. Department of Energy.* 2011. http://www.ne.doe.gov/LWRSP/overview.html (accessed 2011).

[Barton 1990] J.H. Barton, E.W. Czeck, Z.Z. Segall, D.P. Siewiorek. "Fault Injection Experiments Using Fiat." *IEEE Transactions on Computers*, 1990: 575-582.

[Palumbo 1986] Palumbo, D.L., Butler, R.W. "Performance Evaluation of a Software Implemented Fault-Tolerant Processor." *AIAA Journal of Guidance and Control,* vol. 3, no. 6, 1986: 175-185.

[Young 1989] Young, S.D., Elks, C.R., "Performance Evaluation of a Fault Tolerant Processor." *Proceedings of AIAA Computers in Aerospace Conference*, 1989: 625-635.

[Johnson 1989] Johnson, Barry W. "Design and Analysis of Fault Tolerant Digital Systems." *Design and Analysis of Fault Tolerant Digital Systems.* 1989.

[Aldemir 2007] T. Aldemir, M.P. Stovsky, J. Kirschenbaum, D. Mandelli, P. Bucci, L.A. Mangan, D.W. Miller, A. W. Fentiman, E. Ekici, S. Guarro, B.W. Johnson, C.R. Elks, S.A. Arndt. *Reliability Modeling of Digital Instrumentation and Control Systems for Nuclear Reactor Probabilistic Risk Assessment.* Regulatory Guide NUREG/CR-6942, NRC, 2007.

[Yu 2004] Y. Yu, B.W. Johnson. "Coverage Oriented Dependability Analysis for Safety-Critical Computer Systems." *The International System Safety Conference (ISSC).* System Safety Society, 2004.

2. RESEARCH METHODOLOGY

2.1. Overview

The research methodology for this report consists of the following six steps:

(1) Developing of a fault injection model.

(2) Selecting and describing the benchmark systems and application (RPS).

(3) Identifying candidate fault injection methods for the benchmark digital I&C systems.

(4) Reviewing previous results and lessons learned from applying the methodology to a Digital Feedwater Control System (DFWCS) (NUREG/CR 6985) [Aldemir 2007].

(5) Identifying key challenges for fault injection with respect to digital I&C systems,

(6) Establishing an implementation plan for applying fault injection to the benchmark systems for phase 1 and phase 2 part of this research.

Each of these steps are discussed in detail in the remaining Sections of this report.

2.2. Development of a Fault Injection Model

It is important to have a formal model to characterize the applicability and understanding of the fault injection process to ultimately guide its use and facilitate understanding of the results with respect to assumptions. The importance of the formal model is to provide a reference for fault injection-based methodologies with respect to the necessary requirements for fault injection. Before assessing the capabilities and limitations of the fault injection-based methodology a formal model of what the fault injection methodology is trying to achieve must be defined. Section 3 describes a formal fault injection-based model in terms of the Faults, Activations, Readouts, and Measures (FARM) model [Arlat 1993]. This work was extended to better represent the application of fault injection to digital I&C systems [Elks 2005].

2.3. Selection of the Benchmark Systems

The benchmark systems used in this study represent the state of the art in safety grade digital I&C systems. Both of the benchmark systems developed for this research are commercial grade I&C systems representative of systems deployed in NPP applications. These digital I&C systems belong to a class of high-integrity, safety-critical, real-time systems typically used in nuclear applications such as Reactor Protection/Shutdown Systems. Therefore, the benchmark systems selected for this research represent, to the best extent possible, the complexity and behavior of the contemporary digital I&C systems depicted in Figure 1-2. A description of the systems is presented in Section 6.

2.3.1. Application

The research developed RPS multi-dimensional trip functions that used a number of reactor variables. The RPS functions are similar to the functions used in [Smidt 2004], in that the functions represent a reduced model of a typical RPS. The RPS functions developed for this

research used 3 process variable measurements: reactor coolant system flow, hot leg pressure, and hot leg temperature. These reactor process variables are monitored to prevent power operation in an off-nominal basis as would be in an event such a loss of coolant accident (LOCA).

The application for both systems was modeled on a typical nuclear power industry protection system trip function. The RPS function was developed using the software development tools and environment for the respective benchmark systems. The RPS functions were developed using the function block oriented auto code generation tools from the vendors of the benchmark systems. The documents used to help develop the RPS functions are:

- NRC Regulatory Guide 1.152: Criteria for Programmable Digital Computer System Software in Safety-Related Systems of Nuclear Power Plants
- ANSI/IEEE/ANS Standard 7-4.3.2 (1982): Application Criteria for Programmable Digital Computer Systems in Safety Systems of Nuclear Power Generating Stations
- ANSI/IEEE Std 279-1971. “Criteria for Protection Systems for Nuclear Power Generating Stations”
- U.S. Standard IEEE 323

It should be noted that the purpose of this work was aimed at developing a fault injection methodology for digital I&C systems, and not to produce high quality, high assurance software for the RPS functions or to assess the development platforms used to produce the RPS functions, as would be typically done for licensed digital I&C system.

The software development environment for both benchmark I&C systems began with the specification of an I&C system comprising function diagrams and hardware diagrams using *function block editors*. These tools perform a series of consistency and plausibility checks on the diagrams created. This type of software compositional process typically reduces the possibilities of error in the plant-specific I&C specification. The concept behind the engineering of I&C functions with these function block code generator systems is based on the graphical "interconnection" of function blocks to produce I&C functions in the form of function diagrams.

2.4. Lessons Learned and Review of Previous Fault Injection Efforts

In order to provide the highest degree of success, previous efforts applying fault injection to several digital processor based systems were reviewed. Most notably was the application of a fault injection process to a benchmark DFWCS used in an NPP. The results of the application were part of NUREG/CR-6985 [Aldemir 2009] and fully detailed in the University of Virginia (UVA) technical report CSCS 2007-003, “*Quantitative Dependability Assessment of the Benchmark Digital Feed-Water Control System: Final Report*” [Elks 2007]. In addition to the above, the literature for applications of fault injection to commercial/industrial applications was reviewed. The results of this step helped inform the research on promising methods for fault injection for digital I&C systems.

2.5. Candidate Fault Injection Methods for Benchmark Digital I&C systems

Realization and application of Fault Injection for digital I&C systems is a complex process of determining the types of faults to inject into the system, how to inject the faults into the system, establishing the context of the fault injection process, and analyzing the data to extract

measures for the risk informed assessment process. There are many different types of fault injection that may or may not be suitable for physical-based fault injection of digital I&C systems. This step in the research process helped narrow down the best candidates for fault injection for not only the benchmark systems, but also other classes of digital I&C systems. The result of this step is a candidate fault injection technique for each benchmark system.

2.6. Identifying Key Challenges with Respect to Digital I&C Systems

After selecting a candidate fault injection technique and reviewing previous efforts, the research set about identifying key challenges and open problems with respect to fault injection in processor-based I&C systems and FPGA-based I&C systems. These challenges and open problems were identified, and are discussed in Sections 7 and 8 of this report. These are considered to be areas of candidate improvements to the methodology. A list of challenges and open problems are provided in this report.

Unobtrusive in-situ fault injection. A key problem identified for *physical* digital I&C platforms is realizing fault injection methods that can corrupt information inside IC technology (e.g., processors and FPGAs) in a manner that does not upset the real time operation of the digital I&C systems, nor requires modification of the SW/HW of the target digital I&C system.

- ***Automation***. This is one of the most important design goals that a fault injection tool developer must provide for the tool to be effective. For highly reliable or safety critical digital I&C applications where the probability of system failure is very low (e.g., on the order of 10^{-5} failures/year), statistical fault injection requires that a large number of experiments be performed in order to obtain a commensurate degree of confidence in the data used for model parameters. Thus, automation of experiments is an essential feature that enables collecting large volumes of data with little need to manually intervene in the experimentation process.

Experiment Management. To ensure controllable, repeatable and credible fault injections the user must have the ability to manage the types of faults to be injected into the system, where they are injected, how they are injected, and when they are injected. Additionally, the responses to the fault injections must be acquired in a manner that allows the responses to traced back to the faults so that a fault injection trial can be repeated as needed.

Operational and Environmental Profiles. Context is important in fault injection. The operational profiles must be representative of the different system configurations and workloads that would be experienced in actual field operations. The ability to place the inputs and feedback paths into an actual context is critical to establishing the validity of the results of the fault injection process. Having means to simulate or generate realistic profiles is important to the overall utility of the fault injection process.

2.7. Research and Implementation Plan for Applying Fault Injection to the Benchmark Systems

The final step in the research methodology for this report was to create a research and implementation plan to realize the UVA fault injection-based safety assessment methodology on the benchmark systems. This plan had two categories of tasks. The first task category includes items that required additional research and development to determine their potential for implementation. These items were identified and additional efforts were assigned to them. An example of this category is fault list generation methods. Identifying and generating a list of faults for a target digital I&C system is a non-trivial task. As such this step was an on-going work for the project. The second categories were tasks that needed little or no additional

research effort to implement to determine their overall effectiveness. The second category may be viewed as items that needed to be accomplished to support the overall research objectives.

2.8. References

[Arlat 1993] Arlat, J. A., Crouzet, Y., Laprie, J.-C., Powell, D. "*Fault Injection and Dependability Evaluation of Fault-Tolerant Systems*." IEEE Transactions on Computers, no. 42, 1993.

[Elks 2005] Elks, C. "*A Theory of Run-Time Verification*." Charlottesville, VA: University of Virginia, Ph.D. Thesis, 2005.

[Smidts 2004] C. Smidts, M. Li. *Validation of a Methodology for Assessing Software Quality.* NUREG/CR-6848, Washington, D.C.: NRC, Office of Nuclear Regulatory Research, 2004.

[Elks 2007] Elks, C.R. *Quantitative Dependability Assessment of the Benchmark Digital Feed-Water Control System: Final Report.* UVA-CSCS-2007-003, Charlottesville: University of Virginia, 2007.

[Aldemir 2007] T. Aldemir, M.P. Stovsky, J. Kirschenbaum, D. Mandelli, P. Bucci, L.A. Mangan, D.W. Miller, A. W. Fentiman, E. Ekici, S. Guarro, B.W. Johnson, C.R. Elks, S.A. Arndt. *Reliability Modeling of Digital Instrumentation and Control Systems for Nuclear Reactor Probabilistic Risk Assessment.* Regulatory Guide NUREG/CR-6942, NRC, 2007.

[Aldemir 2009] T. Aldemir, S. Guarro, J. Kirschenbaum, D. Mandelli, L.A. Mangan, P. Bucci, M. Yau, B.W. Johnson, C. Elks, E.Ekici, M.P. Stovsky, D.W. Miller, X. Sun, S.A. Arndt, Q. Nguyen, J. Dion. *A Benchmark Implementation of Two Dynamic Methodologies for the Reliability Modeling of Digital Instrumentation and Control Systems.* Regulatory Guide NUREG/CR-6985, NRC, 2009.

3. FAULT INJECTION CONCEPTS AND THEORY

This Section presents a detailed overview of fault injection as a dependability assessment practice. A basic introduction to the concept is presented, and then a more detailed discourse on the methodology to establish a base of understanding for the research, results, and contributions that is described later in this report.

3.1. Motivation for Fault Injection Methods

Dependability evaluation involves the study of failures and errors and their potential impact on system attributes such as reliability, safety, and security. The destructive nature of a failure or crash and long error latency often make it difficult to identify the causes of failures in the operational environment. Thus, it is particularly difficult to recreate a failure scenario for large complex systems just from failure logs and observations alone.

Fault Injection is defined as a dependability validation technique based on the realization of formal controlled validation experiments in which system behavior is observed while faults are explicitly induced in the system by the deliberate introduction (injection) of faults [Arlat 1992]. That is, faults are injected into the system and the resulting behavior is observed. This technique speeds up the occurrence and the propagation of faults in a system for the purpose of observing the effects of faults on the system performance and behavior. Depending on the objectives of fault injection, the measures or the information from fault injection often serve different purposes.

To identify and understand potential failures, the use of experiment-based or measurement-based approaches for studying digital I&C system dependability is gaining acceptance in the nuclear industry to better understand the effects of digital I&C system faults, errors, and failures on overall plant operations [Smith 2000; Kaufman 1998; Elks 2009(a); Elks 2010(a); Reynolds 2009; Aldemir 2009; Aldemir 2007; Huang 2005; Huang 2011; Chu 2008]. From a broader perspective, fault injection has been used extensively in many industries to aid in the assessment of fault tolerant systems and safety critical systems over the past 30 years, and is widely used in the software development and testing community for improving software quality and protection against cyber threats. In addition, the International Electrotechnical Commission (IEC) standard IEC 61508 "*Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems*," recommends the use of fault injection to determine the effects of faults and their mitigation on safety critical systems. Thus, fault injection as technique to aid in the dependability evaluation of safety critical systems is not a novel concept, but one of continued maturation and acceptance.

Further, fault injection-based methods are aimed at providing information to support an informed understanding of risk with respect to digital I&C systems as they are deployed in new and existing plants. The overall safety and reliability of critical digital I&C systems strongly depends on the fault tolerance and mitigation strategies employed, and the correct implementation of these strategies. Therefore, methods to credibly quantify and characterize the safety and reliability of digital I&C systems is of great interest to the NRC.

3.2. Common Misconceptions about Fault Injection

Section 2.5 defined fault injection. This section describes some common misconceptions about fault injection in order to define its purpose more clearly.

Misconception 1: Fault injection estimates failure rates.
One misconception about fault Injection is that it is a technique or approach for estimating failure rates or failure mode failure rates of systems, sub-systems, or components. While fault injection can, under certain limiting assumptions, accelerate the occurrence of observed failures in a system, it does not produce or approximate field failure rate data. Rates of failures are driven by natural processes such as thermal degradation, electro-migration, cosmic particle strikes for hardware origin faults, etc. This failure data is largely estimated from field data sets on the failure of ICs. Fault injection emulates the effects of these naturally occurring events and provides a means for estimating how these faults impact digital I&C system operations.

Misconception 2: Fault injection emulates only hardware faults.
There is some concern that fault injection is mainly relegated to emulating hardware faults. While fault models for fault injection are most mature and accepted with respect to hardware and environmental-based faults, this does not infer that fault injection is restricted to emulating only hardware faults.

Injecting software faults into systems is not only possible, but is practiced by the software test engineering community as a means to test commercial software robustness [Chillerage 2002; Voas 1998]. The challenge or issue with regard to injecting software faults into digital systems lies not with the fault injection process, but with the acceptance of a software fault/failure model for safety critical software.

At present, research in software reliability and appropriate fault models for safety critical software is still evolving. One focus of the research described in this report was to briefly investigate how fault injection of software-based faults could be achieved on the benchmark I&C systems.

3.3. Goals of Fault Injection

This section discusses the major goals for the application of fault injection. From a general perspective, fault injection has two complementary major goals: (1) as an aid to validation and (2) as an aid to design assurance.

3.3.1. Validation Support

In the validation process, the role of fault injection is related to the concept of coverage, (i.e., how can one obtain confidence in the dependability enhancement methods and mechanisms used in the digital I&C system?). Thus, fault injection can be viewed as a means for testing the fault tolerance and safety enhancement features of the system with respect to the inputs they have been designed to cope with, namely the faults.

In practice, there are two means by which fault injection aids in the validation of digital I&C systems. The first is to aid in the confidence of the development processes of the digital I&C system. It is assumed the design of safety critical digital I&C systems incorporate rigorous design and development assurance processes required by governing bodies or law. Often very stringent methods are used to model and analyze the fault tolerance capabilities of the system during the design and development phase. These methods include fault simulation, formal verification of fault tolerance algorithms, structured software development, etc. In this case, the goal of the fault injection process is essentially qualitative and is aimed at checking the adequacy of the verification procedures and of the fault tolerance mechanisms with respect to the considered fault assumptions. For instance, if fault injection revealed a number of faults that were predicted to be detected by the development process and system models, but went undetected, then there would be cause to investigate the adequacy of the development process.

This is type of validation support is sometimes called *model-based testing*. That is, the testing of the system supports the confirmation or refutation of the system models built to comprehend and predict the behavior of the system.

In the second case, fault injection is used to determine the efficiency and effectiveness of the system fault detection and fault tolerance mechanisms with respect to an operational context (i.e., the coverage or coverage factor and execution parameters of the fault tolerance mechanisms with respect to an operational profile or environmental context). In this case, the fault injection is carried out as a statistical-based experiment to determine how effective the fault tolerance mechanisms are in their operational setting.

Both of these cases are complementary to one another, and are often performed in same fault injection experimental setting.

For model based testing, the fault set is usually made up of specific faults that are identified during the development process as "stressors". That is, corner case conditions for which the fault handling behavior of the system may be imperfect or may have side-effects that are not completely characterized. Furthermore, it is practically required that the experiments carried out be reproducible and repeatable (i.e., the same fault produces the same readouts). This enables the possibility of design changes to be thoroughly checked.

In the case of coverage-based testing, the main concern is that the fault set corresponds to a representative distribution of faults among the possible faults and that a large number of faults be injected to guarantee a sound statistical confidence level in the measure.

3.3.2. Design Assurance Support

Fault injection can be applied at various steps in the design and development assurance process and in particular, the results of fault injection can be used to improve the test procedures and the effectiveness and efficiency of fault tolerance mechanisms. As such, fault injection can be considered a design aid in their development, provided that it is applied in the early phases (i.e., on axiomatic or empirical models). Another significant goal of fault injection as a design aid concerns the establishment of fault dictionaries that can be used to develop diagnosis procedures, self-test routines, and improve on-line monitoring.

3.4. A Formal Model of Fault Injection: the Modified FARM Model

As with all well-formed methodologies, it is important to have a formal model to characterize the applicability and understanding of the fault injection process to ultimately guide its use and facilitate understanding of the results with respect to assumptions. A well established and practical model for characterizing fault injection is the FARM model [Arlat 1990(b); Arlat 1992].

The FARM model presented in this report is modified from the original FARM model to be more representative of typical digital I&C systems of the type presented in Section 1.7.1. In addition, the original FARM model concepts have been extended by providing a more formal development of the conditions for fault injection based on [Elks 2005].

A digital I&C system (as depicted in Figure 1-2 and Figure 3-1) is represented as a *reactive* state-based system with sets of well-defined attributes. The state-based assumption is reasonable in that digital processor-based I&C systems, whether realized in processor technology or FPGA-based technology, are state-based. Digital I&C systems in NPP applications almost always belong to one of three classes of systems: control systems, actuation systems, or monitoring systems. All of these systems are *reactive* systems. Reactive

systems continuously accept inputs, compute actions on the basis of these inputs, and produce outputs on an ongoing basis [Wieringa 2003].

Consider the digital I&C system shown in Figure 3-1, which will be refer to as the target system. When fault injection is applied to the target system, the input domain corresponds to the following sets: a set of faults ***F*** taken from a class of faults "F_{class}" a set of activations ***A*** that specifies the domain used to functionally exercise the system; an output domain corresponding to a set of readouts ***R***, and a set of derived measures ***M***. Together, the "FARM" model sets constitute the major attributes that can be used to fully characterize fault injection. The following subsections define these sets more clearly.

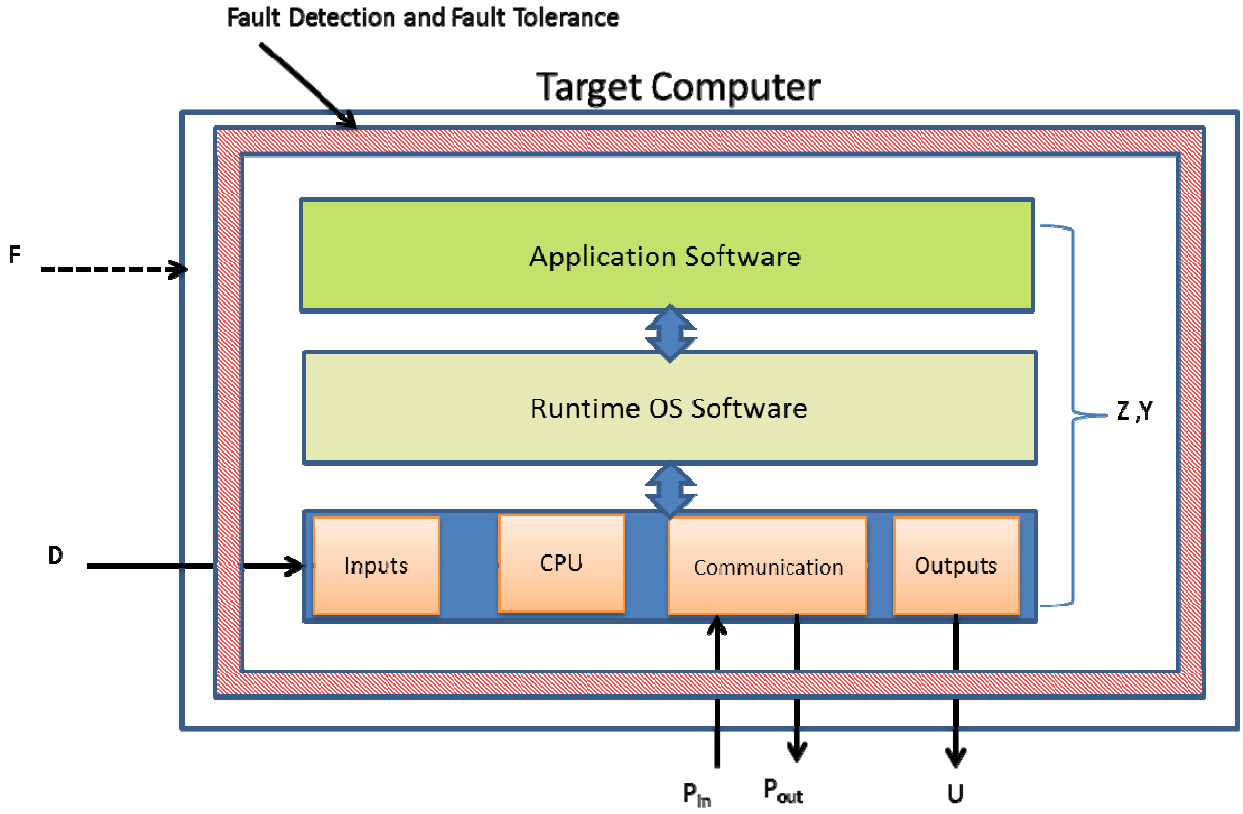

Figure 3-1 FARM model for digital I&C

3.4.1. Attributes of the FARM Model

The attributes of the model are defined as follows:

- The set ***F*** is the set of all anticipated or perceived faults to which the target system could be exposed over its operational phase. The set ***F*** may contain several types of fault classes (F_{class}) from the fault taxonomy in Figure 1.4. The *fault hypothesis* of the system describes the classes of faults that the system is expected to defend against. The *maximum fault assumption* sets the limit on the number of faults that the target system can reliably detect during a given interval. These three attributes help guide the fault model selection process and the fault injection experiments.

- The set ***A*** is the set of inputs to the system aimed at exercising the injected faults. These include the sensor inputs, feedback, and inbound communication messages. The

important aspect of this attribute is that it places the system in the context of the operational behavior, including nominal, off-nominal, and unexpected system conditions.

- The set ***Z*** defines the internal state space of the target I&C system. This attribute expresses the notion that a digital-based or processor-based system is state-based. That is, its behavior is discrete time-based with transitions from one state to a set of other states. A *state* describes a behavioral mode of the system in which it is waiting for a trigger to execute a transition. In some finite-state machine representations, it is also possible to associate actions to a state:

  - Entry action: performed *when entering* the state,
  - Exit action: performed *when exiting* the state.

  A *transition* is a set of actions to be executed when a condition is fulfilled or when an event is received. In the FARM model, the application runtime OS and the hardware have state space.

- The set ***Y*** defines the set of **current** internal states. The set ***Y*** describes the current state of the machine or system at any particular discrete time reference (e.g., an instruction boundary, input event changes, etc.) ***Y*** is a subset of ***Z.***

- The set ***U*** characterizes the outputs provided by the system to its connected environment (e.g., plant controls). The set ***U*** represents both pure digital outputs and digital outputs that are converted to the analog domain.

- The set $\boldsymbol{P_{in}}$ designates the inward bound communications to the system from other systems, other channels, or users.

- The set $\boldsymbol{P_{out}}$ defines the set of outward bound communications and services to other I&C systems, channels or users. $\boldsymbol{P_{out}}$ can represent outbound information from one channel to other redundant channels in a system, or outbound information to monitoring systems or to users.

- The set ***D*** designates the external input data from sensors.

- The set, ***R***, comprises the readouts collected for each fault injection experiment to characterize the behavior of the system in the presence of faults.

- The measure set ***M*** defines the experimental measures, coverage, fault latency estimate, fault dictionary entries, etc., obtained by combining and processing the elements of the ***FAR*** sets.

3.4.2. Fault Injection Defined by the FARM Model

Fault injection is a formal, experiment-based approach. For each experiment, a fault ***f*** is selected in ***F*** and an activation trajectory ***a*** is described in ***A.*** The reactions of the system are observed and form a readout ***r*** that fully characterizes the outcome of the experiment. An experiment is thus characterized by the triple ordinate ***< f, a, r >***, where the readouts, ***r,*** for each experiment form a global set of readouts ***R*** for the test sequence and can be used to elaborate a measure in ***M.*** A *campaign* is a collection of experiments to achieve the quantification of a measure ***M***.

Consider a test sequence of ***n*** independent fault injection experiments. In each experiment a point in the ***{F x A}*** space is randomly selected according to the distribution of occurrences in ***{F x A}*** and the corresponding readouts collected. Expanding the ***F*** to include the fault space dimensionality of time, location, fault type, and fault value yields six parameters that define a fault injection experiment.

a = the set of external inputs
Δ = is the duration of the injected fault
t = fault occurrence time, or when the fault is injected into the system
l = fault location
f_m = a specific fault type as sampled from fault classes
***v*=** Fault value

Figure 3-2 illustrates the basic concept of a fault injection experiment. Specifically, Figure 3-2 shows that faults from ***F*** are sampled from the fault space (discussed in section 3.7). These faults are elaborated by their fault type f_m, the fault duration Δ, the fault location ***l***, time of occurrence ***t***, along with the set of inputs ***a*** to characterize a set of experiments. The fault experiments are applied to the target computer, and a set of readouts (the ***R*** set) is used to derive the ***M*** set (coverage estimation) by statistical estimation. More importantly, from a practical perspective, the parameters of the coverage equation serve as the essential requirements in the development of any fault injection methodology or tools to support fault injection. Fault injection frameworks of any type must address the control of these parameters and the observable responses of a system to these parameters as they are sampled. The following sections describe the statistical theory behind the coverage estimation and the dependent parameters of coverage.

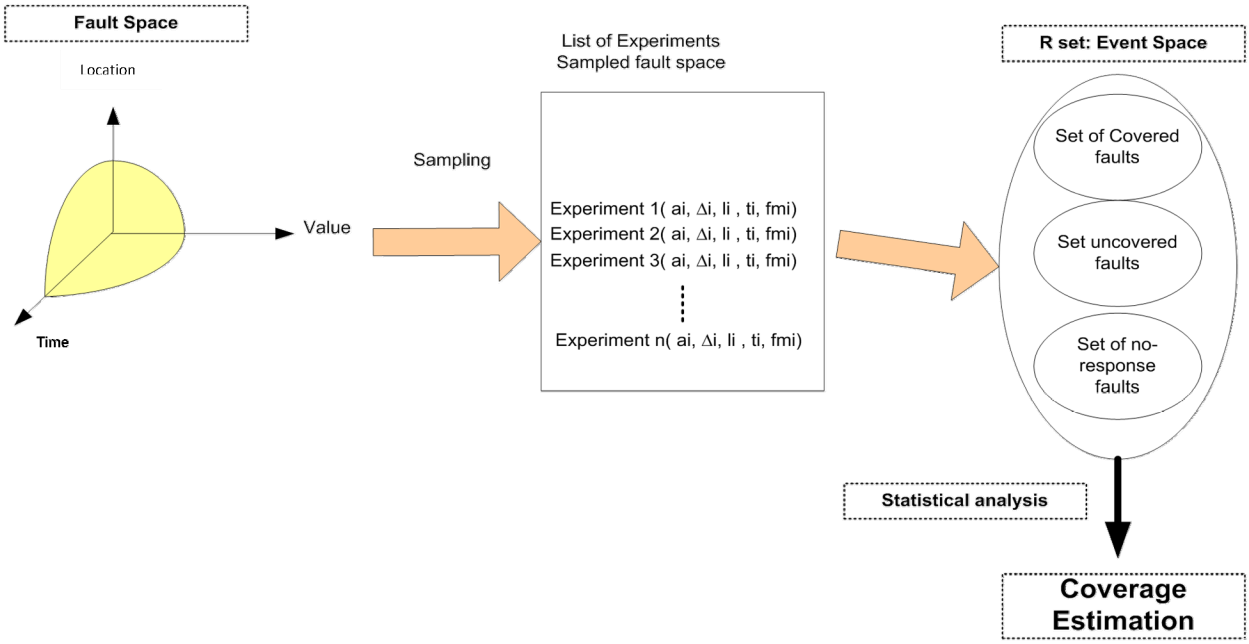

Figure 3-2 Fault injection experiment

3.4.3. Fault and Error Behavior Characterization of the Target Computer

In accordance with general classification, the attributes of the target computer (TC) are distinguished from the ***F***, ***R***, and ***M*** sets to characterize its behavior under the influence of faults. The reason for this distinction is to fully describe how injected faults affecting the target computer can be observed to form a basis for the ***R*** set.

Let the target computer be characterized as a *deterministic state machine* [Kohavi 1978] defined by the tuple (***A, Z, z_0, δ, ω, U***) where:

A = the input space (a finite, non-empty set of symbols)
U = the output space (a finite, non-empty set of symbols)
Z = a finite, non-empty set of states.
$\mathbf{z_0}$ = an initial state, an element of *Z*.
δ = the state-transition function: $\delta : Z \times A \rightarrow Z$
ω = the output function $\omega: Z \times A \rightarrow U$

The state transition relation ***δ*** maps each input and state pair (A_i, Z_j) to the set of next state sets in ***Z***. The notation of $\delta: Z \times A \rightarrow Z$ follows from automata theory, which describes the behavior of sequential digital systems [Kohavi 1978]. The state transition relation symbolizes the notion of program states that evolve in response to inputs (***A***) and clock inputs to the digital I&C system (e.g., the central processing unit (CPU) clock and digital clocking circuits). In addition, the value of a state ***z*** may be forwarded to the output by the output function $\omega: Z \times A \rightarrow U$ when a transition from one state to another occurs.

Note the model of Figure 3-1 symbolizes fault injection as being part of the input domain, *I*, where ***I* = {*A* x *F*} = {*D, Y, F,* P_{in}}.** For elements of the input space, for some state of the machine, for some message of the input communication, and for some fault selected from the fault span, the cross product of ***D, Y, F,* P_{in}** defines the total input domain of the target system for a specific fault campaign. The set ***A*** is often called the *operational profile* of the system. The functional inputs (***D,* P_{in}**) must be distinguished from non-functional inputs ***F*** (e.g., faults and disturbances). This is done by defining a *fault application function* ***ψ.*** The fault application function ***ψ*** introduces the concept that when a fault is applied to the system, the various finite state machine attributes of the target computer defined above can be affected by the fault. The fault application function is formally defined below.

Let ***f*** be a fault action taken from some fault class F_{class}.

The generalization is then

$$[f_i]_0^n = [f_0, f_1, \cdots f_n \in \mathcal{F}_{class}] \tag{3.1}$$

where $[f_i]_0^n$ is a selection of fault actions taken from various fault classes.

The fault activation function takes a fault from the fault space ***F*** and applies it to the target computer, denoted by:

$$\psi([f_i]_0^n) \rightarrow \psi(f_0, f_1, f_2 \ldots\ldots f_n). \tag{3.2}$$

Essentially, the fault activation function is the application of the *fault model* to target computer. Accordingly, the impact of a fault vector $\mathbf{f_i}$ can be defined precisely when the fault is activated and applied to the elements of the state machine:

$$\forall t, \exists \{a(t), z(t), \mathrm{u}(\mathrm{t}), \delta\} \ni \psi(a, z, \mathrm{u}, \delta, f_i; t) \neq \psi(a, z, u, \delta, f_0; t) \tag{3.3}$$

where $\mathbf{f_0}(\boldsymbol{t})$ is the vector "absence of fault".

Equation (3.3) expresses the fact that when a fault is applied to the target computer, its internal execution behavior can be altered. The state (***z***), the control flow (δ), the output (***u***) may be corrupted by the fault. This corresponds to the deviation from the expected or correct execution with respect to the following observations:

Either as an internal error when only the state vector ***Z*** is altered:

$$f(a, z, f; t) = (z', u, t) \neq (z, u; t) \tag{3.4}$$

where ***z'***(***t***) denotes an internal state distinct from the nominal state ***z***(***t***),

or, as an error affecting the service when, as a result of a failure, the vector from ***U*** also deviates from the specified service:

$$f(a, z, f; t) = \begin{Bmatrix} (z', u'; t) \neq (z, u; t) \\ (z, u'; t) \neq (z, u; t) \end{Bmatrix} \tag{3.5}$$

where ***u'***(***t***) denotes an output distinct from the nominal one ***u***(***t***).

Control flow errors are faults affecting the program behavior. The target computer has *next-state error* on receiving input word A_i in state ***z*** if $\delta(A,Z) \neq \delta^*(A,Z)$, where δ is the correct state transition relation and δ^* is the *erroneous state transition* relation. An *erroneous transition* in the presence of some fault is a sequence of program states $\beta = z_0, z_1, z_2 \ldots z_{j-1}, z_j$ such that for each ***j, j>0,*** z_j is obtained from z_{j-1} by executing a fault action from $\Psi(f)$ that is enabled in z_{j-1}. The general notion here is that from each program state in ***Z*** there exists the possibility of taking an erroneous transition or a correct transition to the next state when ***f*** is present and active.

The target computer has an *output error* on receiving input word A_i in state ***z*** if $\omega(Z, U) \neq \omega^*(Z, U)$, where ω is the correct output function and ω^* is the faulty output function of the TC. An *erroneous output* is obtained by imposing fault action $\Psi(f)$ onto a set of the observable states ***Z*** and ***U*** such that correct output state variables are not equal to the erroneous output state variables of $\omega^*(Z, U)$.

The target computer has *input error* in state ***Z*** if some input word A_i^* is not a valid word from A and the TC accepts this word.

Lastly, the notion of a latent error is introduced. The evolution of a system state with respect to error manifestation does not depend on time for all internal states ***z(t).*** This leads to a partition of the state vector ***z(t)*** that distinguishes the state vector $z_s(t)$, which characterizes the state variables that are sensitized at time ***t***. That is, the internal variables that actually impact the evolution of the system at time ***t*** from the state vector $z_d(t)$, which characterizes the variables that are not sensitized at time ***t***. Such a distinction is useful in practice to account for the latent error or dormant error problem. A latent error is an error that has not been recognized by the system error detection mechanisms as an error.

$$\psi(a, y_d, y_s, f; t) = (z_d', z_s, u; t) \rightarrow \psi(a, y_d', y_s, f; t) = (z_d', z_s, \boldsymbol{u}; t) \tag{3.6}$$

Equation (3.6) illustrates the condition when a state error is latent, and the result of this state error being dormant is that the output state remains uncorrupted. The concern with latent faults is that the faults may remain undetected for long periods of time until an unusual event or another fault occurs, at which time the faults become active. In such cases, the system now has two active faults with which to deal; the system may not be designed to handle these coincident fault conditions. This distinction also reinforces the fault-error-failure pathology since

dormant faults may not create erroneous behaviors and all erroneous states do not necessarily cause a failure [Arlat 1992]. Fault injection campaigns should address the latent or dormant fault problem when appropriate.

3.4.3.1. Controllability and Observability

The concepts of controllability and observability play an important role in fault injection. They define what elements of the target computer can be influenced by fault injection. Different types of fault injection methods will have various degrees of controllability and observability, so it is important to discuss them in the context of a fault injection model.

The target computer with internal state *Z* is called **controllable** if and only if the system states can be changed by the fault application function. The target computer has *full fault controllability* in this case. When only some of the states can be controlled by the fault application function, the target computer has *partial fault controllability*.

The target computer with an initial state, $z(t_0)$ is **observable** if and only if the value of the initial state can be determined from the system output u(t) that has been observed through the time interval $t_0 < t < t_f$. If the initial state cannot be so determined, the system is **unobservable**.

The concept of controllability determines the *extent or reachability* of a fault injection as applied to the target computer. *Observability* determines what readouts "R set" are detectable or observable from the outcomes of a fault injection experiment. Both of these attributes play significant role in application of physical fault injection as discussed in Section 5.

3.4.4. Characterization of Error Detection Mechanisms for the Target Computer

This section defines and describes the characteristics and attributes of the Error Detection Mechanisms (EDMs) of the Target Computer. This section builds on the theory presented in [Elks 2005]. Digital I&C systems that are used in safety critical applications should have fault detection and fault tolerance functions to prevent fault-induced errors and failures from propagating to critical plant systems and processes. EDMs are the basic building blocks of fault tolerance. The importance of characterizing EDMs in digital I&C systems is that the outputs of EDMs are part of the basic ***R*** set of the FARM model. EDMs produce error messages after a fault injection to indicate the response to the injected fault.

EDMs are defined from an *abstract perspective*, as opposed to giving specific implementation details. This approach allows characterization of a wide range of EDMs for digital I&C systems relying primarily on the properties that the systems *should* have, irrespective of their hardware and software composition [Elks 2005].

Informally, an EDM is a distinct *module or component* either internal or external to the target computer that observes a given set of conditions or states from a system. These states can be viewed at different levels of abstraction, such as operating system states, application level variables, input sensors, outputs, or even hardware signals. The role of an EDM is to determine or recognize when a set of conditions or states in the target computer are deviated from a specified level of normal operation. Examples of EDMs are comparators, voters, threshold checks, set point checks, and error detecting codes.

In section 3.3.3, the finite state machine representation of the target computer was introduced and showed how the fault application function could alter the form or attributes of the state machine representation. These attributes are the inputs to the EDM, and can be characterized as the *event language or alphabet* of the EDM. In cases where these events are faulty, they

may not be contained in the event language of the EDM. The EDM asserts an error condition or predicate (***P***) when it suspects the monitored events are faulty. These concepts are refined more clearly in the following discussion.

Definition 3.1 An *EDM* is a distinct module that observes a set of conditions or executions from a target computer and determines if these conditions or executions are *consistent with its notion of validity.*

Validity means that the EDM is in agreement with its *reference of correctness*. A reference of correctness for an EDM is the means for determining acceptance or rejection an input. As examples, a comparator accepts inputs that are identical with each other, and rejects those that are not. The reference of correctness is an identical agreement between the two inputs. A threshold comparison compares an input to a reference value and determines if the input is out of range. Here the reference of correctness is agreement to a bound limit.

The definition of EDM is formalized as follows:

Let β be a sequence of state conditions taken from {*Z, A, U*}. This is denoted as

$$\beta \in \{Z, A, U] = s_o, s_1, s_2, s_3 ..... s_{j-1}, s_j . \tag{3.7}$$

Let α be the acceptance condition of the EDM or the reference of correctness condition, and let ***P*** be predicate for the state sequences of ***β***. ***P*** is a predicate that indicates an acceptance condition. Then $\boldsymbol{P_\alpha}$ is the predicate for a detection condition of the EDM.

Let ***Z*** be a *check process* of ***β***.

$$Z(\beta) \rightarrow P_\alpha \tag{3.8}$$

"Z *detects P"* is a *detector* in the EDM if

$$"Z \text{ accepts } \beta" \in \mathrm{P}_\alpha(\beta) \rightarrow \text{True and "Z rejects } \beta" \in \neg \mathrm{P}_\alpha(\beta) \rightarrow \text{False} \tag{3.9}$$

With these formal definitions it is possible to fully characterize the EDM based on desirable *properties* an EDM should have. Conceptually, an EDM is *complete* with respect to an acceptance condition α if for any β such that $\boldsymbol{P_\alpha \rightarrow}$ ***false,*** the *EDM* is guaranteed to reject β and raise an alarm. Furthermore, an EDM is *sound* with respect to α if whenever EDM signals an alarm, then always β is assuredly erroneous (e.g., no false alarms). Also, an EDM is *accurate or reliable* if for any correct run β, the EDM never accuses any run of β of being erroneous. These properties are important to fault injection processes because in reality, EDMs and fault tolerance mechanisms are not perfect.

As an example, an EDM may have *weak completeness*. Weak completeness says that for **some** set of runs in β where that $\boldsymbol{P_\alpha \rightarrow}$ ***false***; the EDM will detect/reject these runs as non-compliant. In doing so, it may also accept **some** runs that are erroneous, but not detectable as such by $\boldsymbol{P_a}$. In these instances, the EDM is *incomplete*. This is the basis for the *imperfect coverage problem.*

From an assessment perspective, the relationship between incompleteness of EDMs and system non-coverage is important. The measurement/estimation of non-coverage fault injection

indicates that some of the error detection mechanisms in a target computer are *incomplete* with respect to the fault injected into them.

The modified FARM model described in this section is useful toward establishing a basis not only for conducting experiments, but also serves as a reference model for gauging the adequacy and sufficiency of fault injection methodologies. Moreover, it also aids as a guide in designing and developing fault injection environments. This is an important issue, because the results or outcomes of fault injection experiments can be assessed in accordance with the governing modified FARM model.

3.5. Characterization of a Fault Injection Experiment

Referring back to the FARM model, a fault injection experiment is characterized by the ***F, A,*** and ***R*** sets. For a given fault ***f*** and set of activations ***a*** (workload and operational profile) a readout ***r*** is observed. The readout from the experiment usually consists of monitored states (***Y***), and outputs ***(U***) of the digital I&C system. The monitored states are often process variables of the application, error messages from the error detection mechanisms, etc. The set of such observed states can be defined in terms of combinations of a set of detection predicates ***P*** that were defined in section 3.4.4 on the experiment readouts of the ***R*** set, e.g., {fault_injected}, {fault_activated and error_signaled}, etc. A detection predicate is either true or false. That is, an error is signaled or not, a fault is active or not.

A fault injection experiment is carried out with respect to time. Specifically, there is an observation time interval ***T*** in which the response of the system to gather the readouts ***R*** from the target system is observed. This time interval ***T*** is called the *censoring time* of the experiment. The expected behavior with respect to a single predicate in relation to ***T*** is observed. For example, Figure 3-3 presents two common test cases for most fault tolerant digital I&C systems. The first is when the target system is supposed to detect and correct the error attributed to a fault (e.g., fault masking), the second is when the system is supposed to signal an error whenever a fault is present (e.g., just fault detection). Figure 3-3 illustrates the behaviors over the observation interval ***T***.

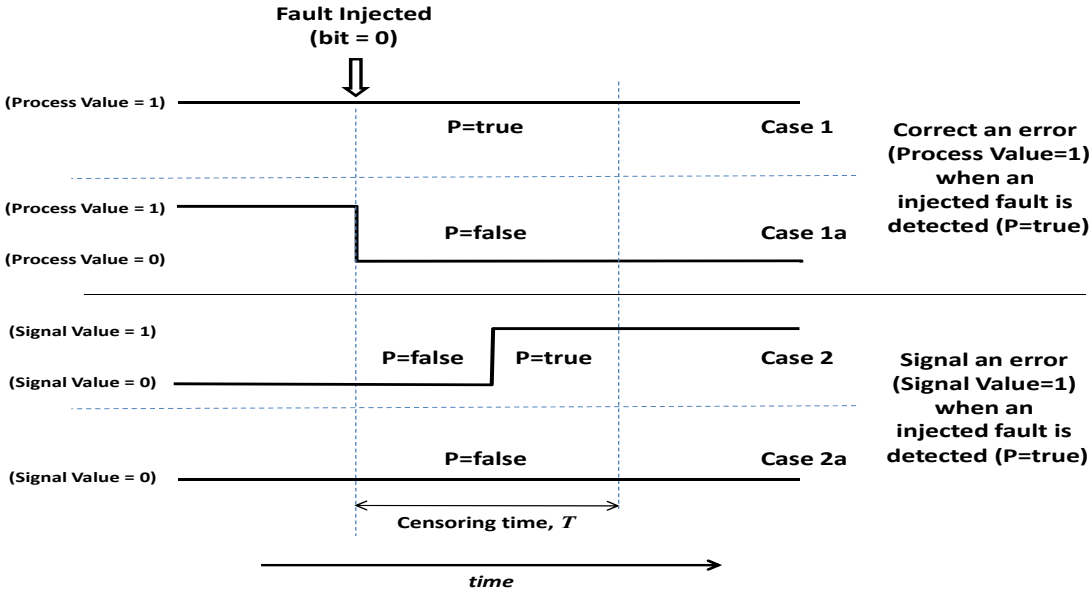

Figure 3-3 Characterization of time censored fault injection experiment

Case 1 indicates that the fault was tolerated corrected and acceptable service was delivered. Case 1a indicates that the error was not detected, and erroneous results were delivered. Case 2 shows that the fault was latent for some time, before being detected and signaled. This occurs when ***P*** asserts to true. Case 2a shows that the fault did not signal an error predicate. This could be due to a long fault or error latency, or if the fault is undetectable.

Also, the observation domain ***T*** is bounded and the readouts obtained from the experiments form a set of so-called Type ***I*** (or time) censored data**;** the observed times are known only up to the upper bound ***T (censoring time)*** of the observation domain [Sharma 1994]. The characteristics of the considered target system and especially the temporal parameters of the fault detection (e.g., error and fault latency) to be evaluated have a direct impact on the determination of ***T.***

The choice of ***T*** relies on a careful analysis of the *a priori* (partial) information available concerning the fault detection and fault tolerance features of the digital I&C system, the dynamics of the environment that are being controlled by the digital I&C system, and may necessitate a set of preliminary experiments for its proper adjustment. In the following discussion the concept of an experiment graph to characterize the outcomes of fault injection experiments is introduced.

3.5.1. Characterization of Fault Injection Outcomes

The second important use of the FARM model is to characterize the possible readouts ***R*** collected during a fault injection campaign of a target system. The objective is to formulate or observe the possible outcomes of the campaign. This is achieved by way of an assertion graph that abstracts the specification of the behavior of the target system under test. An assertion graph can be established *a priori* to describe anticipated behaviors or can be obtained *a posteriori* from the analysis of the ***R*** set.

Figure 3-4 provides an example of the assertion graph used in this research [Arlat 1992]. Transition 1 corresponds to the activation of an injected fault as an error; the associated time defines the *fault dormancy.* Transition 2 represents the situation where an injected fault is not activated; such an experiment is not significant when error detection coverage is evaluated with respect to error patterns (resulting from activated faults) rather than with respect to the faults injected. Transition 3 depicts the case of a detected error; the associated time characterizes the *latency of error detection.* Transition 4 represents the case where an error is apparently tolerated although it was not detected; whereas transition 6 depicts the (normal) situation where the error is tolerated after having been detected. Transitions *5* and 7 distinguish the cases of failure of the detection and tolerance mechanisms. In particular, transition 4 characterizes a singular behavior that is not always easy to diagnose in practice since it may result from either 1) an activated fault that remains hidden (latent) or 2) a propagated error that is tolerated or that is eliminated by some other-unobserved-mechanism.

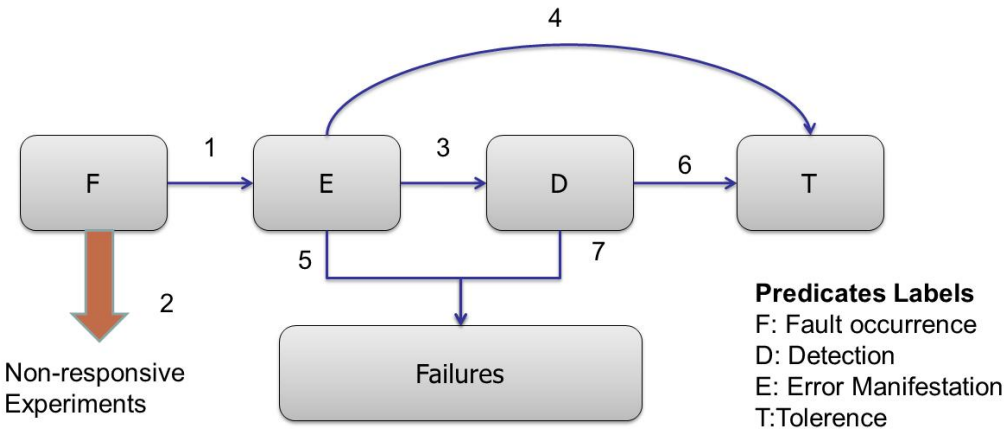

Figure 3-4 Fault injection experiment predicate diagram

3.6. Development of the Coverage Function

As stated in Section 3.1, one of the purposes of fault injection is to evaluate the error and fault handling mechanisms of a fault tolerant digital system. An extremely important parameter in the design and assessment of fault tolerant systems is *fault coverage, C*. The fault coverage available in a system can have a significant impact on the reliability and safety of a system [Smith 1997; Choi 1997; Yu 2004]. The intuitive definition of fault coverage is that it is a measure of a system's ability to perform fault detection, fault isolation, and fault recovery given the existence of a fault. Thus, coverage is a specific measure from the ***M*** set in the FARM model.

It is generally assumed that if perfect and exhaustive testing of the system could be carried out, the estimate of *coverage* obtained would be exactly equal to the system's fault coverage. Obviously, the ideal case cannot be achieved because of the limitations of the testing process. As a result, to obtain meaningful data and good estimation accuracy, fault injection campaigns must consist of a large number of experiments because of the high reliability or safety requirements of digital I&C in safety related NPP applications. The following sections describe the formal and mathematical description of concepts that are fundamental to statistical estimation of fault coverage.

3.6.1. The Fault Space

The development of a statistical model for fault-coverage estimation requires definition of the set of possible faults that are to be considered in the experimentation process. The statistical model can only be developed when this set is completely defined. The fault taxonomy presented in Section 1.7.3 is a starting point for considering what faults the system may be subject to during its lifetime. The set of all faults is called the *fault space*, and is denoted in this document by F. It is worth noting the distinction of the fault space from *fault universe*, denoted by U, which is the set of all possible faulty states of the system. The notion of the fault universe is derived from the simple consideration that the occurrence of a fault is a transition to a system state that should not be reached, ideally, according to the allowed transitions defined by the system's design [Pescosolido 2002]. Hence a fault can be considered as the faulty system state itself, and an experimental fault injection can be considered a forced transition to that state.

It is clear that a complete description of the fault universe cannot be made because a *complete* definition of the system state is never available. Moreover, the fault universe is considered a

continuous multi-dimensional space, and hence, is uncountable and infinite. Theoretically, a distributed parameter model of some physical quantity is needed to completely define the state of a system and hence also to define a faulty state. This limited ability to completely describe a system is most often solved by using concentrated parameter models.

This transition from distributed parameter models to concentrated parameter models involves reduction of the fault universe to the fault space. The fault space is a more simplified description of the fault universe and is usually discrete and finite, yet in general is multidimensional. It is not possible to define an exhaustive set of faults and test the system with each to possibly obtain the best estimate of coverage, but provides a statistical population from which the faults can be sampled and injected into the system.

A number of characteristic attributes, such as location, value, time, or workload, can be used to identify a fault in the fault space. In general, a fault space is regarded as a multi-dimensional space with *d* dimensions, where each fault is identified as a point in the *d*-dimensional space by a set of *d* coordinates. Hence the fault space can be considered as the Cartesian product of the sets of values that can possibly be assumed by each attribute or axes in the multi-dimensional fault space.

If α_i represents the axis corresponding to attribute A_i, and a_i represents the value assumed by A_i, then the fault space can be defined as,

$$\mathsf{F} = \alpha_1 \times \alpha_2 \times \ldots \times \alpha_d \qquad and \qquad f(a_1, \ldots, a_d) \in \mathsf{F} \tag{3.10}$$

where

$f(a_1, \ldots, a_d) \in \mathsf{F}$ is the fault for which $A_i = a_i$.

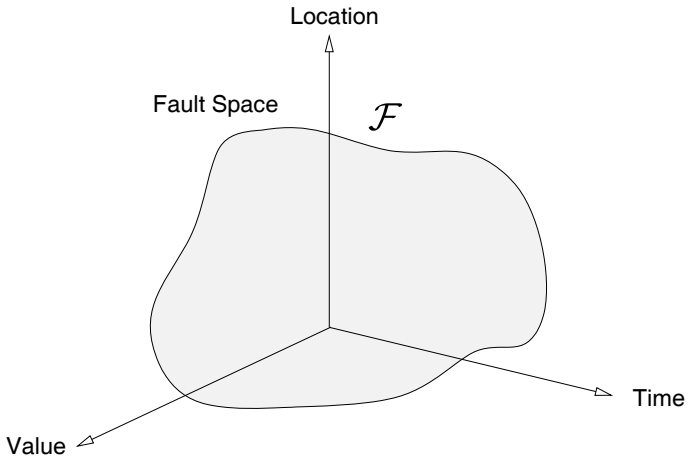

Figure 3-5 Three dimensional fault space

3.6.2. Fault Distribution

The occurrence of a fault in a system can be regarded as a random event and hence can be described in a probabilistic framework. The fault distribution deserves careful consideration in

the development of a statistical model, as the quality of the results can be significantly affected by the assumptions made about it. In the Bernoulli model, a simple approximation is provided by the uniform distribution, which assigns equal relative probabilities of occurrence to every fault in the fault space. The probability function is thus defined on a random variable *F* as,

$$P(F = f) = \frac{1}{|\mathsf{F}|} \quad \forall (f \in \mathsf{F}) \tag{3.11}$$

where $|\mathsf{F}|$ denotes cardinality of the fault space. The uniform distribution is a commonly used model for coverage estimation because it provides a conservative approximation on the fault distribution in the absence of any known information about the underlying fault distribution.

3.6.3. Coverage Definition

For a given fault injection experiment, let t_{pi} denotes the instant of assertion of predicate *p* for experiment *i,i*=1, . . . , *n,* let $Y_I(t)$ denote the random variable defined by,

$$Y_i(t) = I(t_{pi}) = \begin{Bmatrix} 1;\ if\ the\ assertion\ of\ p\ is\ observed\ [0,t] \\ 0;\ otherwise \end{Bmatrix} \tag{3.12}$$

The random event described by the predicate *p* can be associated to a binary random variable *Y*, which assumes the value 1 when the predicate is true and 0 when the predicate is false during the observation interval *T*. The variable *Y* is then distributed like a Bernoulli distribution with parameter *C*. From the definition of the Bernoulli variable the parameter of the distribution equals the mean of the variable. In this case,

$$\begin{aligned} C = E[Y] \quad &= 1 \cdot Prob(Y = 1) + 0 \cdot Prob(Y = 0) = Prob(Y = 1) \\ &= \sum_{f \in \mathsf{F}} Prob(Y = 1 \mid F = f) Prob(F = f) \end{aligned} \tag{3.13}$$

The last expression is obtained by applying the theorem of total probability. It is generally assumed that the fault tolerance mechanism will always either cover or not cover a certain fault *f*, the probability $Prob(Y = 1 \mid F = f)$ is either 0 or 1, and can be expressed as a function of the fault,

$$y(f) = Prob(Y = 1 \mid F = f) = \begin{cases} 1 & \text{if } f \text{ is covered} \\ 0 & \text{if } f \text{ is not covered} \end{cases} \tag{3.14}$$

From Equation (3.14), the following expression for fault coverage is obtained,

$$C = \sum_{f \in \mathsf{F}} y(f) Prob(F = f) \tag{3.15}$$

3.6.3.1. Forced Coverage

If a certain distribution of the fault space is imposed, then the event of occurrence of a fault $f \in F$ must be associated with another random variable $\underline{F}$, which is different from F, and 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 given by,

$$\underline{C} = E[\underline{Y}] = \sum_{f \in F} P(\underline{Y} = 1 \mid \underline{F} = f) P(\underline{F} = f) \tag{3.16}$$

Although the two variables Y and $\underline{Y}$ may 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 occurs with the fault distribution or with the forced distribution. Therefore the values of C and $\underline{C}$ must also be related. To determine the relationship between the two distribution parameters, a new random variable Q is introduced, which is also a function of the variable $\underline{F}$ that is given by,

$$Q = q(f) = \frac{P(F = f)}{P(\underline{F} = f)} \tag{3.17}$$

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

$$E[P] = \sum_{f \in F} q(f) P(\underline{F} = f) = \sum_{f \in F} P(F = f) = 1 \tag{3.18}$$

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 average, to unity.

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

$$\begin{aligned} \rho &= E[(\underline{Y} - E[\underline{Y}])(P - E[P])] = E[(\underline{Y} - \underline{C})(P - 1)] \\ &= E[\underline{Y}P] \;\; E[\underline{Y}] - \underline{C}E[P] + \underline{C} \\ &= \sum_{f \in F} y(f) q(f) P(\underline{F} = f) - \underline{C} - \underline{C} + \underline{C} \\ &= \sum_{f \in F} y(f) P(F = f) - \underline{C} = C - \underline{C} \end{aligned} \tag{3.19}$$

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

$$C = \underline{C} + \rho \tag{3.20}$$

For most statistical models, *C* is *optimistic* for $\rho < 0$, and is *pessimistic* for $\rho > 0$. For values of the covariance close to 0, the estimator can be considered unbiased.

3.6.4. The Simple Bernoulli Model

The simple Bernoulli model is the simplest model for fault coverage estimation and is based on the standard statistical approach of estimation of the mean of a Bernoulli population, which makes use of simple random sampling. It provides a very generic approach to the problem of fault coverage estimation. The simplicity of the model drastically minimizes the cost and time for experimental setup and data analysis, which makes the model widely accepted for coverage estimation by designers of dependable systems.

With the Bernoulli model, a test sequence can be considered as a series of Bernouilli trials. Let ***n*** denote the number of independent fault injection experiments and ***N (t)*** the total number of assertions of ***P*** (coverages) observed in a time interval **[0, t]**. Let the random event described

by the predicate ***p*** be associated to a binary random variable ***Y***, which assumes the value 1 when the predicate is true and 0 when the predicate is false during the observation interval ***T***. The number of assertions of the predicate ***p*** cumulated within the time interval **[0,t]** can thus be expressed as,

$$N(t) = \sum_{i=1}^{n} Y_i\,(t) \tag{3.21}$$

It follows that ***N (t)*** is distributed according to the binomial distribution,

$$P\{N(t) = k; n\} = \binom{n}{k} [C(t)]^k [1 - C(t)]^{n-k} \tag{3.22}$$

Accordingly, the mathematical expectation of the numer fo coverages is written as

$$C(t) = \frac{E[N(t)]}{n} \tag{3.23}$$

The statistical estimator of ***C(t)*** is thus,

$$\hat{C}(t) = \frac{N(t)}{n} \tag{3.24}$$

As $t \to \infty$, the asymptotic value of ***C*** is given by the simple average, that is the arithmetic average of n observations of the random variable ***Y***.

$$\hat{C} = \frac{1}{n} \cdot \sum_{k=1}^{n} \underline{Y}_k \tag{3.25}$$

The mean of the estimator is calculated as,

$$E[\hat{C}] = E\left[\frac{1}{n} \cdot \sum_{k=1}^{n} \underline{Y}_k\right] = \frac{1}{n} \cdot \sum_{k=1}^{n} E[\underline{Y}_k] = \dot{C} \tag{3.26}$$

This value of fault coverage given by the mean of $\underline{Y}$ is the total fraction of faults that are covered by the fault tolerance mechanism. It is called *coverage proportion* and is denoted by $\dot{C}$. The relationship between coverage proportion and coverage factor was defined in equation (3.20) as $C = \dot{C} + \rho$. The estimation provided by the sample average in this model is *optimistic* for $\rho < 0$, and is *pessimistic* for $\rho > 0$. For values of the covariance close to 0, the estimator can be considered unbiased.

The accuracy of the estimator can easily be calculated as the variance as,

$$\begin{aligned} V[\hat{C}] &= V\left[\frac{1}{n} \cdot \sum_{k=1}^{n} \underline{Y}_k\right] = \frac{1}{n^2} \cdot \sum_{k=1}^{n} V[\underline{Y}] \\ &= \frac{\dot{C}(1-\dot{C})}{n} = \frac{(C-\rho)(1-C+\rho)}{n} \\ &= \frac{C(1-C)}{n} + \frac{\rho(2C-1-\rho)}{n} \end{aligned} \tag{3.27}$$

and its unbiased estimator $\hat{S}$ is given by,

$$\hat{S}[\hat{C}] = \frac{\hat{C}(1-\hat{C})}{n-1} \tag{3.28}$$

3.6.4.1. Confidence Intervals

Confidence intervals for the fault coverage are usually defined using the technique of the pivotal transformation applied to the point estimator. From the definition of confidence intervals,

$$P(C \in [\hat{C}_1 : \hat{C}_2]) = \gamma \tag{3.29}$$

and using the pivotal quantity,

$$\frac{\hat{C} - E[\hat{C}]}{V[\hat{C}]} \tag{3.30}$$

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 (3.30) has the standard normal distribution. The symmetric double-sided 100γ% confidence interval is then given by,

$$(\hat{C} - K_\gamma \sqrt{\hat{S}[\hat{C}]}) < \dot{C} < (\hat{C} + K_\gamma \sqrt{\hat{S}[\hat{C}]}) \tag{3.31}$$

where K_γ indicates the $\left[\frac{100(1+\gamma)}{2}\right]^{th}$ standard normal percentile. 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 $n > 100$, the two distributions are equivalent [Arlat 1993]. The single sided 100γ% confidence interval is instead given by,

$$\dot{C} > \hat{C} - z_\gamma \sqrt{\hat{S}[\hat{C}]} \tag{3.32}$$

where z_γ indicates the 100γ% 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 $\hat{C}$, which is different from the real coverage factor C if the fault coverage mechanism is *unfair*. The difference between the coverage proportion and the system coverage is given by the covariance, and is a measure of how fairly the fault tolerance mechanism behaves with 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 less likely faults leads to a positive covariance, while an *unfair* fault tolerance mechanism that covers least likely faults better than the most likely faults will result in a negative covariance. Therefore the confidence interval defined above is valid only when it can be assessed that $\rho \approx 0$.

3.6.4.2. The Number of Fault Injection Experiments

In conducting a campaign of fault injections to estimate coverage values, the number of fault injections needed to provide a statistically sound estimate of coverage for each component must be estimated. For practical purposes, the single sided or one-side confidence interval model to estimate the number of fault injections may be used, although there are several other statistical models that can be used to derive the number of fault injection estimates [Pescosolido 2002].

As indicated in previous sections, the estimation of the fault coverage can be modeled as a binomial random variable Y_i, where,

$$Y_i = \begin{cases} 1\forall \\ 0\forall \end{cases} \tag{3.33}$$

The fault injection experiments are performed to generate Y_1, ……Y_n, outcomes or responses where each Y_i is assumed to be independent and identically distributed (e.g independent trials). The expected value of the random variable is $E(X) = C$, and the variance of the random variable is $Var(Y) = C(1-C)$.

Given the statistic,

$$S_n = \sum_{i=1}^{n} Y_i, \tag{3.34}$$

the probability of $S_n = j$ is,

$$P(S_n = j) = \begin{bmatrix} n \\ j \end{bmatrix} C^j (1-C)^{n-j}, 0 \le j \le n \tag{3.35}$$

If S_n out of n faults are observed to be covered, then C, the lower side of the confidence interval, satisfies the following equation,

$$P(S_n > s_n) = 1-\gamma \tag{3.36}$$

where $P(S_n > s_n)$ is the probability S_n is greater than or equal to s_n, given that the fault coverage value equals C_l, and γ is the confidence coefficient. The single-sided confidence interval is calculated as,

$$1-\gamma = \sum_{j=s_n}^{n} P(S_n = j) = \sum_{j=s_n}^{n} \begin{bmatrix} n \\ j \end{bmatrix} C^j (1-C)^{n-j}, 0 \le j \le n \tag{3.34}$$

Now, consider the case where all the faults are covered. In the case,

$$1-\gamma = \sum_{j=s_n}^{n} \begin{bmatrix} n \\ j \end{bmatrix} C^j (1-C)^{n-j} = C_l^n \tag{3.37}$$

Rearranging the above equation and solving for ***n*** yields,

$$n = \frac{\ln(1-\gamma)}{\ln C_l} \tag{3.38}$$

where *n* is the number of faults to be injected for a given confidence bound.

One can see from equation that as the coverage parameter approaches unity the number of fault injections grows significantly. Table 3-1 shows the number fault injections as a function of coverage and confidence intervals.

Table 3-1 Coverage estimation as function of experiment runs.

n	C_I $\gamma = .80$	C_I $\gamma = .90$
100	0.983	0.977
1000	0.9983	0.9977
10,000	0.99983	0.99977
100,000	0.999983	0.999977
10^k	$0.9_{k-1}83$	$0.9_{k-1}77$

As rule of thumb, the number of experiments needed for a given coverage level is approximately 10^x where x is the number of "9's" in the coverage factor. For example, for ***C*** = .999 at least 1000 fault injection experiments are needed.

3.6.4.3. Assumptions and Discussions about the Simple Bernoulli Model

The model discussed in this section has advantages and disadvantages. It is usually chosen by fault injection assessors as the most appealing approach to fault coverage estimation because of the simplicity and familiarity of the theoretical and practical development of a fault injection campaign resulting from the application of this model.

One of the shortcomings of the estimation process described in this section is that it assumes a uniform fault distribution, which may or may not be valid. It is usually used when the information about the underlying fault distribution is completely unknown. It was shown in Section 3.6.4 that this kind of assumption can lead 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 fair (i.e. $\rho \approx 0$) can this assumption lead to precise estimates 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 assume a uniform fault distribution provide equally precise estimates if only the condition of fairness is verified.

Whether the fairness condition is verified in real systems or not is an open research question, 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 a 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 historically have been more often observed and studied, thus the fault detection/tolerance mechanisms are designed to mitigate those faults.

From such considerations it could be concluded that in highly realiable fault tolerant 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 [Powell 1995] in statistical estimation of the distribution parameter of a Bernoulli random variable to assume that the pivotal quantity of Equation (3.29) is distributed as a standard Gaussian distribution. 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 (3.30) 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 an 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 [Smith 1997; Cukier 1997]. This is usaully the case for most fault injection studies.

3.7. Fault Models

Faults from either hardware failures or software failures can have significant impact on the reliability and safety operation of digital I&C systems. Hardware faults can cause software to malfunction beyond its intended operation. For instance, a hardware fault in a register can misdirect a pointer data structure to the wrong location in memory. Traditionally, the main purpose of a fault model is to define the fault conditions that may occur on a semiconductor device, integrated circuit, or software. In a sense, a fault model indirectly describes the types of failure mechanisms a digital device can experience. Thus, fault models are abstract representations of real faults that occur in digital systems. Applying these fault models to digital I&C systems and observing the responses is a key component of fault injection based assessment processes. It is important to note that fault models must be both representative and supportable in the physical fault injection context.

Among the various attributes (such as workload, fault set, measurements and measures) that have been introduced within the FARM model to precisely characterize a dependability benchmark, the determination of a *representative fault set* for digital I&C is identified as one of the key challenges [Arlat 2002]. A representative fault set is a set of faults that accurately represent the types of failures that can occur in an operational context in digital I&C systems. The determination of fault representativeness attempts to show that a fault model is representative with respect to actual faults.

Fault models for digita- based systems are often classified by their time dimension,

- Permanent, which remain in existence indefinitely,
-
- Transient, with a usually short temporal duration,

- Intermittent, similar to transient as they have a temporal duration, but that appear and disappear repeatedly in time, without a periodic behavior.

3.7.1. Permanent Faults

The permanent fault model represents a physical defect in the semi-conductor IC or supporting hardware that is activated when certain input and state conditions place demands on the hardware where the physical fault exists. Examples include electro-migration, hot carrier effects, and time dependent dielectric (oxide) breakdown. These failure mechanisms usually result in permanent faults/failures in devices. Recently, there has been some concern that the FIT caused by permanent failure mechanisms in semiconductors is increasing due to aggressive scaling and process technologies [George 2010(a); Baumann 2005(a)]. Shrinking geometries, lower power voltages and higher frequencies have a negative impact on reliability of digital technology by increasing the rates of occurrence of transient, intermittent, and permanent faults.

3.7.2. Transient Faults

It is estimated that 80% of all faults that occur in computer based systems are transient in nature [Yount 1996; Baumann 2005(a); Baumann 2005(b)]. Also known as soft errors, transient faults generally do not cause permanent damage to silicon-based digital circuits and disappear with reset of the system, or get overwritten by dynamic data. While a number of sources of transient faults exist, such as electrostatic discharge, power fluctuations, High Intensity Radiated Fields (HIRF); those that arise from high-energy particle strikes) have recently gained more significance as a result of aggressive scaling of device dimensions, lower voltage operations, with a consequent increase in the corresponding soft error rates observed [Baumann 2005(a)].

The initiating failure mechanism of single event upset (SEU) transient faults is the high energy particle strike. High energy cosmic particles (e.g., Hadrons, protons, and neutrons) interacting with silicon can cause ionization reactions in the silicon cells of an IC causing the cell to absorb a charge and thus flip the state of a memory cell or propagate a pulse charge. Historically, particle strikes would cause only one bit to flip because the cell geometry size was large compared to the area affected by the particle strike. However, with contemporary semiconductors featuring geometries around 25nm, the occurrence of multiple event upsets are now possible and have been observed in the laboratory [Constantinescu 2002]. In addition, logic cells inside CPUs are now vulnerable to particle upsets for the first time [Baumann 2003]. Taken all together, the transient fault phenomena in digital I&C systems is one that should be considered when selecting appropriate fault models for fault injection.

3.7.3. Intermittent Faults

Intermittent fault manifestation can come from a number of sources. The most widely reported cause of intermittent faults inside an IC is defects that occur during the manufacturing process [Constantinescu 2006; Gracia-Moran 2010; Constantinescu 2003]. At the board and system level, intermittent faults often are associated with degraded connections, poor contact points, unbalanced ground current loops, and power fluctuations. Intermittent faults are often the most difficult to isolate because of their non-stationary behavior.

3.7.4. Software Faults

Software faults are basically an artifact of design. It is a potential, unintended functionality that must be assumed to be *always* present in the system. Software is more a reflection of the *design* of the system than it is a component of the system. As such, it is more appropriate,

when discussing safety or reliability, to talk about failures of the system that are caused by software than it is to talk about failures of the software alone. This places the argument into the context of a system and its interacting environment.

Software failures, especially those associated with common mode or common cause origin, are a potential serious problem with digital I&C systems. The software testing and software reliability research community has been using variants of fault injection to test the robustness of software for some years with reasonable success [Lyu 1996]. The purpose of software testing in the commercial realm is mainly geared toward defect removal and product quality enhancement, not safety enhancement. Typical techniques include mutation testing, error or fault seeding, sandboxing, and fuzzing.

When speaking of injecting software faults into a system, there should be some clarification on what the purpose of such an exercise would be. One purpose of injecting software faults would be to test the resilience or robustness of the digital I&C system to detect, isolate and recover from such faults. This presumes that the injected software faults have the potential to get introduced into the system, either by system or application software changes, or by malicious intent (e.g., malware).

In the context of digital I&C systems, the system would have to possess the capability to detect and mitigate such faults. If the system has no detection or tolerance mechanisms to deal with the software faults then it will most likely fail. However if the system is developed in a manner that provides some resilience to software faults, then fault injection is an appropriate process to test the resilience of the system to these threats. This is analogous to anti-virus software that runs on desktop computers, its purpose is to detect and isolate viral code that is not intended to be running on the machine.

At present, there is a need for guidance to determine what would be an adequate software fault model that is appropriate for digital I&C systems and under what circumstance would it be useful. In summary, the ability to inject software faults into a system is not a significant technical barrier; the methods developed in this research can achieve this with moderate success. The issue at hand is determining the necessary guidance so that the results have some meaning to the overall assessment process.

3.8. References

[Arlat 2002] Arlat, J, and Y Crouzet. "Faultload representativeness for dependability benchmarking." 2002. 29-30.

[Arlat 2003] Arlat, J, Y Crouzet, J Karlsson, P Folkesson, E Fuchs, and G H Leber. "Comparison of Physical and Software-Implemented Fault Injection Techniques." *IEEE Transactions on Computers*, no. 52 (2003): 1115-1133.

[Arlat 1992] Arlat, J. "Fault Injection for the Experimental Validation of Fault-Tolerant Systems." *Proceedings of Workshop Fault-Tolerant Systems.* Kyoto, Japan: IEICE, 1992. 33-40.

[Arlat 1990(b)] Arlat, J., et al. "Fault injection for dependability validation: a methodology and some applications." #*IEEE_J_SE*# 16, no. 2 (1990(b)): 166-182.

[Baumann 2005(a)] Baumann, R. C. "Radiation-induced soft errors in advanced semiconductor technologies." #*IEEE_J_DMR*# 5, no. 3 (2005(a)): 305-316.

[Baumann 2005(b)] Baumann, R. "Soft Errors in Advanced Computer Systems." *IEEE Design and Test of Computers*, 2005(a): 258-266.

[Baumann 2003] Baumann, R. "Technology scaling trends and accelerated testing for soft errors in commercial silicon devices." 9th IEEE On-Line Testing Symposium. Dallas TX: 2003.

[Elks 2009(a)] C. Elks, B.W. Johnson, M. Reynolds. "A Perspective on Fault Injection Methods for Nuclear Safety Related Digital I&C Systems." *6th International Topical Meeting on Nuclear Plant Instrumentation Control and Human Machine Interface Technology.* Knoxville, TN: NPIC&HMIT, 2009(a).

[Elks 2010(a)] C. Elks, M. Reynolds, B. Johnson, N. George, M. Waterman, J. Dion. "Application of a Fault Injection Based Dependability Assessment Process to a Commercial Safety Critical Nuclear Reactor Protection System." *Dependable Systems and Networks Symposium.* Chicago, IL, 2010(a).

[Yount 1996] C.R. Yount, D.P. Siewiorek. "A Methodology for the Rapid Injection of Transient Hardware Errors." *IEEE Transactions on Computers*, 1996: 881-891.

[Choi 1997] C.Y. Choi, B.W. Johnson, J.A. Profeta III. "Safety Issues in the Comparative Analysis of Dependable Architectures." *IEEE Transactions on Reliability*, September 1997: 316-322.

[Chillarege 2002] Chillarege, R., Goswami, K., Devarakonda, M. "Experiement Illustrating Failure Acceleration and Error Propagation in Fault-Injection." *IEEE International Symposium on Software Reliability Engineering*, 2002.

[Constantinescu 2003] Constantinescu, C. "Trends and Challenges in VLSI Circuit Reliability." *IEEE Micro Magazine*, 2003: 14-19.

[Constantinescu 2002] C. Constantinescu. "Impact of Deep Submicron Technology on Dependability of VLSI Circuits." *Dependable Systems and Networks.* International Conference, 2002. 205-209.

[Constantinescu 2006] Constantinescu, C. "Intermittent Faults in VLSI Circuits." *IEEE Workshop on System Effects of Logic Soft Errors.* IEEE, 2006.

[Smith 2000] D. Smith, T. DeLong, B.W. Johnson. "A Safety Assessment Methodology for Comples Safety Critical Hardware/Software Systems." *International Topical Meeting on Nuclear Plant Instrumentation, Controls, and Human-Machine Interface Technology.* Washington, D.C., 2000.

[Smith 1997] D.T. Smith, B.W. Johnson, N. Andrianos, J.A. Profeta III. "A Variance Reduction Technique Using Fault Expansion for Fault Coverage Estimation." *IEEE Transactions on Reliability*, September 1997: 366-374.

[Elks 2005] Elks, C. *A Theory of Run-Time Verification.* Charlottesville, VA: University of Virginia, Ph.D. Thesis, 2005.

[Huang 2011] Huang, Bing, M. Rodriguez, Ming Li, J. B. Bernstein, and C. S. Smidts. "Hardware Error Likelihood Induced by the Operation of Software." *#IEEE_J_R#* 60, no. 3 (2011): 622-639.

[Huang 2005] Huang, Bing, Xiaojun Li, Ming Li, J. Bernstein, and C. Smidts. "Study of the impact of hardware fault on software reliability." 2005.

[Gracia-Moran 2010] J. Gracia-Moran, D. Gil-Tomas. "Searching Representative and Low Cost Fault Models for Intermittent Faults in Microcontrollers: A Case Study." *IEEE 16th Pacific Rim International Symposium.* Tokyo, Japan: IEEE, 2010. 11-18.

[Voas 1998] J. Voas, G. McGraw. *Software Fault Injection: Implementing Programs Against Errors.* Wiley, 1998.

[Kohavi 1978] Kohavi, Z. *Switching and Finite Automate Theory.* McGraw-Hill, 1978.

[Chu 2008] L. Chu, G. Martinez-Guridi, M. Yue, J. Lehner, P. Samanta. *Traditional Probablistic Risk Assessment Methods for Digital Systems.* NUREG/CR-6962, NRC, 2008.

[Kaufman 1998] L. Kaufman, B.W. Johnson. *The Importance of Fault Detection Coverage in Safety Critical Systems.* NUREG/CP-0166, Proceedings of the 26th Water Reactor Safety Information Meeting: U.S. NRC, 1998.

[Lyu 1996] Lyu, M. *Handbook on Software Reliability Engineering.* McGraw Hill, 1996.

[Cukier 1997] M. Cukier, J. Arlat, D. Powell. *Frequentist and Bayesian Coverage Estimations for Stratified Fault-Injection.* Research Report 96336, Bologna, Italy: LAAS-CNRS, 1997.

[Reynolds 2009] M.Reynolds, C.R. Elks, N. George, M.Sekhar, T. Delong, B.W. Johnson. "A Quantitative Safety Assessment Methodology for Safety-Critical Programmable Electronic Systems Using Fault Injection." *SAE World Congress.* Detroit, MI, 2009.

[George 2010(a)] N. George, C. Elks, B. Johnson, J. Lach. "Transient Fault Models and Architecture Vulnerability Factor (AVF) Revisited." *Dependable Systems and Networks Symposium.* Chicago, IL, 2010(a).

[Pescosolido 2002] Pescosolido, M. "Statistical Models for Coverage Estimation." *School of Engineering and Applied Science Masters Thesis.* University of Virginia, May 2002.

[Powell 1995] Powell, D., E. Martins, J. Arlat, and Y. Crouzet. "Estimators for fault tolerance coverage evaluation." *#IEEE_J_C#* 44, no. 2 (1995): 261-274.

[Sharma 1994] Sharma, K.K. "On the Estimation of Sample Size and Censoring Time and Life Testing Experiments" *Microelectronics Reliability, vol 34*, 1994: 125-134.

[Aldemir 2007] T. Aldemir, M.P. Stovsky, J. Kirschenbaum, D. Mandelli, P. Bucci, L.A. Mangan, D.W. Miller, A. W. Fentiman, E. Ekici, S. Guarro, B.W. Johnson, C.R. Elks, S.A. Arndt. *Reliability Modeling of Digital Instrumentation and Control Systems for Nuclear Reactor Probabilistic Risk Assessment.* Regulatory Guide NUREG/CR-6942, NRC, 2007.

[Aldemir 2009] T. Aldemir, S.Guarro, J. Kirschenbaum, D. Mandelli, L.A. Mangan, P. Bucci, M. Yau, B.W. Johnson, C. Elks, E. Ekici, M.P. Stovsky, D.W. Miller, X.Sun, S.A. Arndt. "Dynamic Reliability Modeling of Digital Instrumentation and Control Systems in Nuclear Power Plants." *6th International Topical Meeting on Nuclear Power Plant Instrumentation Control and Human Machine Interface Technology.* Knoxville, TN: NPIC&HMIT, 2009.

[Wieringa 2003] Wieringa, R.J. *Design Methods for Reactive Systems, 1st ed.* Morgan Kaufman, 2003.

[Yu 2004] Y. Yu, B.W. Johnson. "Coverage Oriented Dependability Analysis for Safety-Critical Computer Systems." *The International System Safety Conference (ISSC).* System Safety Society, 2004.

4. OVERVIEW OF THE FAULT INJECTION BASED DEPENDABILITY ASSESSMENT METHODOLOGY

This section provides an overview of the UVA fault injection-based dependability assessment methodology. The original methodology is described in more detail in NUREG/CR reports [Kaufman 1999(a); Kaufman 1998; Cutright 2003(a); Cutright 2004; Bastien 2004; Cutright 2003(b); Cutright 2003(c); Cutright 2003(d); Kaufman 1999(b); Kannan 2000; Yu 2003]. The purpose of this section is to provide an overview of the original methodology as it relates to the research effort describe in this report.

4.1. Introduction

The prevailing conventional approach to reliability and availability prediction is usually accomplished by the use probabilistic models that use component level failure rates published in handbooks or supplied by the manufacturers. This approach provides an early indication of system dependability, but the model as well as the underlying data later need to be validated by actual measurements. In addition, these models typically do not account for recovery provisions and fault coverage considerations which are critically important to reliability and safety assessments.

Probabilistic modeling focuses mainly on system hardware reliability estimation. However, accounting for software reliability is essential. Typically, software reliability growth models are based upon fault rates [Lyu 1996], and at present it is difficult to obtain creditable predictions when there are few observed failure data, as is the case for safety critical systems.

A complement to the probabilistic modeling approach is offered by measurement-based dependability evaluations [Smith 2000, Iyer 2002, Tang 1996]. The measurement-based approach is appropriate for the operational environment and during testing. The primary advantage of this approach lies in the use of measurements and models for interpretation of the measurements. Measurements can be performed on commercial grade components without the need to reveal proprietary information. Based on the measurements, the approach can provide assessments of various dependability measures (e.g., failure rate, safety, reliability, availability) with stated confidence levels. Measurement-based dependability assessment was developed on the basis of innovative techniques of fault injection, failure measurement and probabilistic analysis. The methodology consists of methods for failure data collection and processing, statistical analysis, and dependability modeling. Figure 4-1 shows the basic steps in a measurement-based dependability methodology.

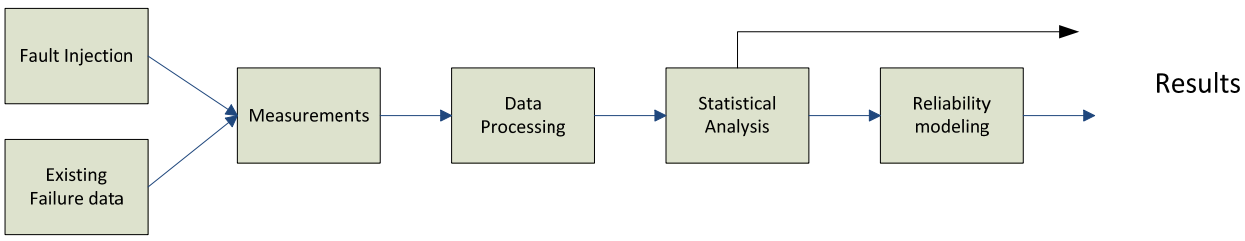

Figure 4-1 Measurement-based dependability methodology

The UVA fault injection-based dependability assessment methodology was developed to complement measurement-based dependability assessments, realizing that fault injection may

serve different goals and purposes [Smith 2000, Elks 2010, Reynolds 2009]. Thus, the methodology was intended to be as flexible as possible to the needs of the assessor and designer. The goal of the dependability assessment methodology described in this section is to provide a generic, systematic means of characterizing the dependability behavior of digital I&C systems and their input/output interactions in the presence of anomalous behaviors, faults, and failures.

A central part of Phase 1 of this research was to thoroughly examine the methodology before the application of the methodology to the benchmark systems to determine *a priori* the challenges, issues, and deficiencies that need to be addressed in the latter phases of this effort. As such, it was expected that the methodology as described in this section would evolve to meet those challenges. As a reminder, the methodology presented here is the original unmodified version of the methodology.

The methodology described in this Section builds on the theory and concepts presented in the last Section (e.g., the FARM model of fault injection) to provide a practical means for implementing fault injection in the measurement-based dependability assessment process. An overview of the process is shown in Figure 4-2. In this depiction, the process is driven by the needs of PRA modeling efforts to estimate more accurately parameters for PRA modeling activities. Statistical sampling principles are used to guide the parameter estimation process. Then, representative fault models are selected with respect to the target I&C system. After the faults are injected into the system, the data is post-processed to produce new estimates of model parameters, which are then instantiated back into the PRA models to enhance better predictions by the PRA models.

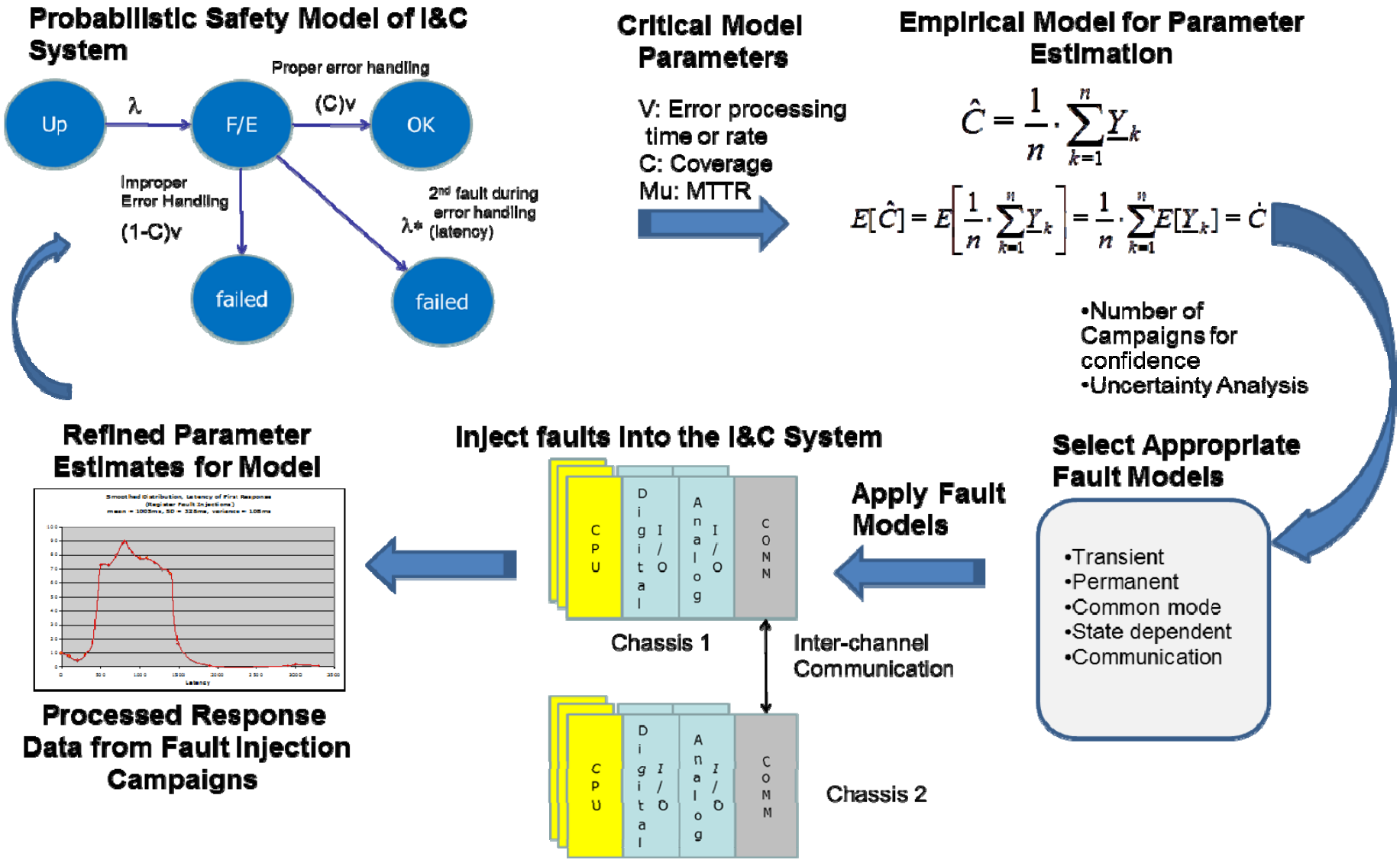

Figure 4-2 Conceptual view of the UVA fault injection-based safety assessment methodology

4.2. Step 1: Support PRA Modeling Needs

The process steps of the methodology are illustrated in Figure 4-3. Each step in the methodology illustrated in Figure 4-3 is described in this Section. Referring to Figure 4-3, the starting point in the methodology is to understand what is needed from the PRA process.

The purpose of the reliability and safety assessment process is to ensure a system will meet its reliability and safety requirements, show that risk mitigation measures produce reliability and safety improvements, and the unreliability risk is controlled to an acceptable level. A probabilistic safety and reliability safety assessment process usually begins with asking three basic questions: (1) what can go wrong, (2) what is the likelihood, and (3) what are the consequences? Fault injection-based methodologies are most useful in characterizing questions 1 and 3 – determining failure modes and the consequences of failure.

The essential process of a PRA [Kumamoto 1996] is as follows:

- What are the end states or adverse consequence states of concern? These would include core damage in a reactor, loss of flight control in an aircraft, unattended release of a hazardous material. This activity attempts to answer the 3rd question above.

- A set of initiating events for each consequence state is developed. An initiating event is often related to a disturbance or a component failure deviating from normal operation.

- Modeling methods such as fault trees, event trees, and Markov models are used to characterize a sequence of events that lead from the initiating event to the end consequence states. These sequence scenarios often include hardware system failures, human errors, human-machine errors, common mode failures, etc. The process tries to enumerate to the best extent possible all of the possibilities that may occur. This step answers question 1.

- Probabilities of these scenarios are determined to the best extent possible from past historical data, expert engineering judgment, and experimental analysis. These probabilities are the answer to question 2.

- The accident scenarios are ranked according to their severity and their frequency of occurrence to ascertain the overall risk, and what is an acceptable level of risk. What is acceptable is determined and set by the managing or oversight organization (e.g., the licensee and the NRC). Determining the risk of a system involves placing the risk in the context of its operating environment.

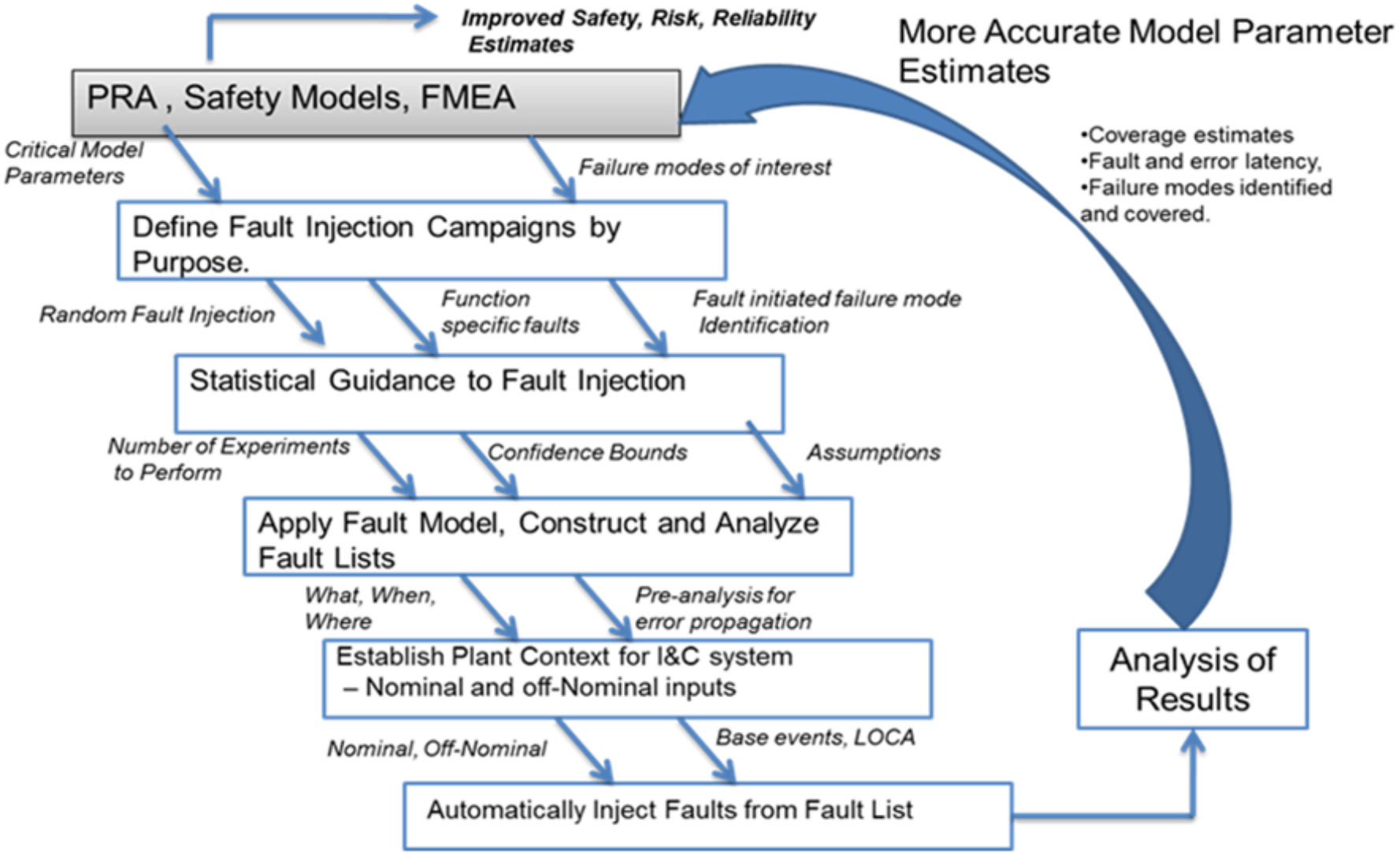

Figure 4-3 Step by step view of the UVA fault injection-based safety assessment methodology

From the research perspective, the concerned was with how fault injection can support PRA processes, including the attributes of reliability and safety. With respect to the PRA process described above, faults injected into the digital I&C system can provide the following important information:

- Is the response of the system to the fault consistent with what was expected? Were there side-effects? This information informs step 2 and step 3.

- Was the fault detected? Was the detection handled properly? Did the system mitigate the fault as intended? This information informs step 2, step 3, step 4 and step 5.

- Did the system respond consistently to the same set of faults with similar operating conditions? This information informs step 2 and step 3.

- Does a fault produce a failure mode not postulated? This information informs step 1, step 2, step 3 and step 4.

The PRA modeling process described above typically begins with defining or selecting a set of measurement-based attributes that are appropriate toward informing the risk assessment process. These attributes include reliability, unreliability, safety, etc. In a typical risk informed PRA process, there may be several dependability attributes that are used to characterize the system risk. In digital I&C system reliability assessments, these measures include probability of system failure, probability of coincident failure, probability of failure on demand, mean time to system failure, mean time to unsafe system failure, and steady state unsafe system failure.

The important point is that PRA activities employ modeling methods such as fault trees, event trees, and Markov models to assist in the determination of risk. These models have parameters that represent attributes of the system, such as physical failure rates, detection capability, capability to tolerate faults, fail-safe capability, repair capability, etc. Fault injection methods provide a means to quantitatively estimate the behavior model parameters of the system.

A behavioral model parameter is a measure of how a system behaves or responds with respect to a stimulus (e.g., a fault or set of inputs or a disturbance). The important coverage factor parameter presented in Section 3.6 of this report is a behavioral parameter in a PRA model. Equally important is stating the assumptions the models or model parameters in light of incomplete knowledge of the systems. Since fault injection provides response information that can be used to statistically estimate these parameters, the quantification of these parameters (in a probabilistic sense) can be used to produce more accurate parameter estimates for the PRA models, which in turn produces more accurate risk assessments to inform the oversight process.

4.3. Step 2: Fault Injection by Purpose and Type

It is not uncommon to use fault injection for different purposes in order to obtain a more complete picture of a system's behavior response. As indicated in Section 3.4, fault injection is used in the design and validation processes of digital I&C systems. As discussed in step 1, the accurate estimation of the parameters used in the reliability and safety models of a system is a primary use of fault injection. Model parameters such as failure mode coverage, fault coverage, fault latency, and real-time responses are difficult to estimate *a priori* in digital-based systems. Fault injection methods (either physical or simulation based) are often used to provide data to estimate these parameters so the predictions of the models are more accurate and credible.

From a broader stance, fault injection is a measurement-based process that provides important experimental techniques for assessment and verification of fault-handling mechanisms. The process allows researchers and system designers to study how computer-systems react and behave in the presence of faults. Fault injection is used in many contexts and serves purposes, such as:

- Supporting on-line monitoring so that systems can react in an appropriate way.
- Assessment of the effectiveness, i.e., fault coverage, of software and hardware implemented fault-handling mechanisms.
- Studies of error propagation and error latency in order to guide the design of fault-handling mechanisms.
- Providing evidence to support the resilience of the system to unexpected faults and failures.
- Measuring the time required for a system to detect or to recover from errors.
- Testing the security detection capabilities of fault-handling protocols in distributed systems.
- Verifying failure mode assumptions for components or subsystems.

Since fault injection can be used for many purposes, it is necessary to identify as early as possible the type of fault injection and the measurements that will be used and whether fault injections should be applied to a physical system or a model of the physical system.

Fault injection can, in principle, be carried out in two ways: faults can be injected in an actual system (i.e., a physical computer system) or in a model of a system (i.e., a prototype of a commercial product). System models for fault injection experiments can be built using two basic techniques: *software simulation* and *hardware emulation*.

The main advantage of performing fault injections on an actual system is that the implementation of the fault-handling mechanisms is assessed and verified realistically. Thus, system representativeness is usually higher when using a physical system compared to using software simulation or hardware emulation. On the other hand, fault models used in simulation-based and emulation-based fault injections can usually emulate a larger set of representative faults more accurately than in physical-based fault injection system and the reachability of these faults into the system is much greater. However, there is an issue of model fidelity with simulation-based fault injections; that is, is the model an accurate representation of the physical system.

Fault injection techniques have specific drawbacks and advantages. A comprehensive survey of fault injection methods and techniques to serve as a guide toward selecting fault injection for a target digital I&C system is presented in the next Section.

4.4. Step 3: Statistical Modeling Guidance for Fault Injection Experiments

The purpose of statistical modeling is to provide a formal basis for (1) conducting fault injection experiments and (2) providing a statistical model for a estimating the measures of a fault

injection experiment, such as coverage. As developed in Section 3, the statistical model supports four specific needs of the methodology:

(1) Characterize the fault injection experiment in formal statistical framework.

(2) Quantify and characterize the uncertainty of model parameter.

(3) Characterize and define the assumptions of the estimation process.

(4) Statistically estimate based on the assumptions of the model and model parameters the numbers of observations are required to estimate a parameter to a known confidence level.

Section 3 presented an overview of the statistical development of fault injection. In practice, there are many statistical models that can be adopted to guide a fault injection experiment. The methodology described in this report is not tied to one specific model; rather it allows users to decide which statistical models are best suited for a particular operational context. In reference [Pescosolido 2002] a detailed survey and analysis of statistical models for the estimation of fault coverage is provided to better guide the user in this area. Appendix A summarizes the characteristics of several popular statistical models used in fault injection.

4.5. Step 4: Fault Model Selection

Digital I&C systems are subject to faults and failures from a variety of sources, and can manifest these faults and failures in many ways as was discussed in the fault taxonomy discussion in Section 1.7.3 [Avizienis 2004]. Fault models are abstract representations of real faults. For example, a single event upset caused by a power surge or a cosmic particle strike can be modeled by the bit-flip fault model. Fault models allow assessors to evaluate the effectiveness of fault detection, diagnostic testing, and fault tolerance mechanisms with respect to the faults that are anticipated to arise in the operation of a digital I&C system. Applying these fault models to I&C systems and observing the responses is a key component of fault injection-based assessment processes. Selecting the appropriate fault model for a fault injection campaign is a crucial decision.

Research of fault models for digital I&C systems has been ongoing for at least 20 years [Ubar 1998; Yount 1993; Yount 1996; Zemva 1998; Anceau 1986; Rashid 2010(a); Rashid 2010(b); Wangqi 2004]. Recent research of fault models has been focused on new device technologies, software flaws, and common mode faults [Baumann 2005(a); Bernardi 2004; Sierwaski 2009]. New device technologies such as the deep sub-micron complementary metal oxide silicon (CMOS) process provides unparalleled performance scaling, however it can negatively impact reliability.

Faults arising from hardware failures or software failures can have significant impact on the reliability and safety operation of digital I&C systems. Hardware faults can cause software to malfunction beyond its intended operation. For instance, a hardware fault in a register can misdirect a pointer data structure to the wrong location in memory. It is important to note that fault models must be both representative and supportable in the physical fault injection context.

As part of the research described in this report, fault models for processor-based I&C systems were reviewed, and a means to use these fault models that would be supportable across a wide variety digital I&C platforms was investigated. Verifying the representativeness of fault models is generally a difficult challenge. The approach for addressing this problem was to:

- Model and simulate processors at a highly detailed level to see how low level faults (at the gate level) manifest at higher levels (RTL) [Delong 2005; George 2010(a); George 2010(b)].

- Conduct literature reviews of digital I&C systems and other embedded systems to identify fault models adopted by the scientific community.

Figure 4-4 shows the taxonomy of supported fault models and the methods used to generate fault lists from the fault model types. Based on these investigations, the fault models shown in Figure 4-4 represent and extend the capabilities of previous fault injection methods. The research along these lines was a best effort in support of the overall goals of developing a fault injection-based system dependability evaluation methodology. Building a fault injection environment to support the entire fault models listed in the fault taxonomy of section 1.5.3 was beyond the scope of the research effort described in this report.

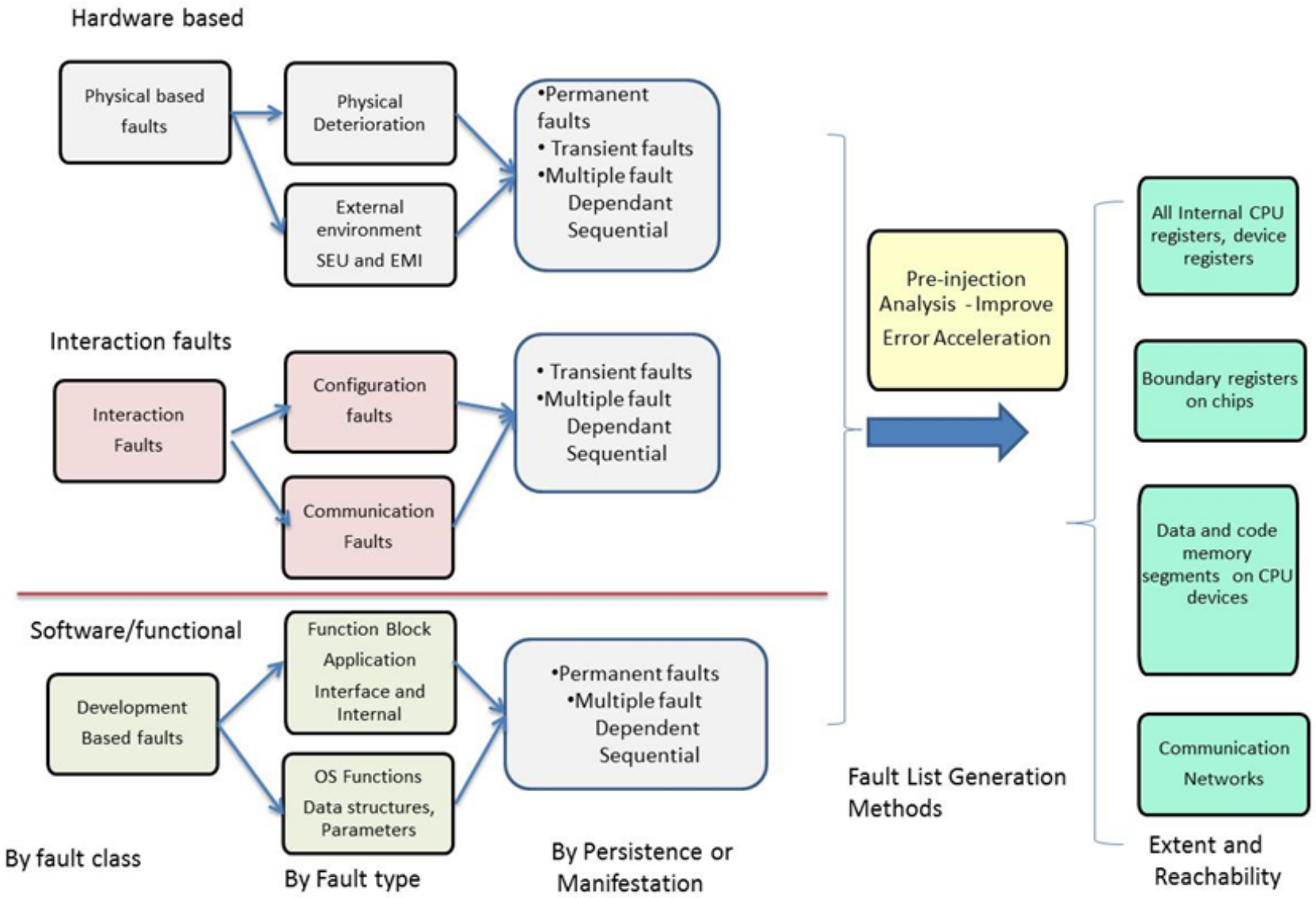

Figure 4-4 Proposed fault model support for the methodology

Referring to Figure 4-4, in the physical fault domain there are two types of faults that are of relevance: (1) deterioration or degradation faults, and (2) external interference faults. The first type of faults are due to the natural wear-out and degradation of semiconductor devices, ICs and board level interconnections. Typically, these faults are modeled as permanent faults. The second type of faults arise from the natural interference in the system from external sources. The error manifestation of both fault classes is either a permanent or transient error.

It is estimated that 80% of all faults that occur in computer based systems are transient in nature [Yount 1996; Constantinescu 2002; Baumann 2005(b)]. The effects may last for milliseconds or may persist until the system is re-booted or refreshed. In recent years, the impact of SEUs caused by neutron particles and high intensity radiation fields (e.g., EMI) affecting digital systems has become more significant due to the aggressive scaling of semiconductor manufacturing processes. Thus, the choice of the transient fault model on the basis of these facts is justified. The transient fault is represented by the modified single and multi-bit flip model proposed in [George 2010(a)] as a result of this work.

The other fault model that is of importance is the permanent fault model. The permanent fault model represents a physical defect in the semi-conductor IC or supporting hardware that is activated when certain input and state conditions place demands on the hardware where the physical fault exists. Examples include electro-migration, hot carrier effects, and time dependent dielectric (oxide) breakdown. These failure mechanisms usually result in permanent faults or failures to devices. As with increased SEU susceptibility with new IC technologies, the rates of failure caused by permanent failure mechanisms in semiconductors is increasing due to aggressive scaling and process technologies [Srinivasan 2004]. Specially, the increase in FIT rate from the 150 nm technology to the 35 nm technology is approximately 500%.

Emulation of the permanent fault model is based on a special case of the bit flip fault model. In this special case bits are flipped in memory or register locations on every occurrence of the use of the affected register or memory location through the use of a trigger register. Use of this model requires extremely high performance fault injection methods so as to not adversely impact the performance of the target I&C system when injecting a permanent fault. Until recently permanent fault models were not practical due to their negative performance impact on system performance. As part of the research described in this report a high performance fault injector to support the demanding needs of emulating permanent faults (and other types of faults) on digital I&C systems was developed. This new fault injector is described in later sections.

The next type of hardware domain fault is a multiple fault type. Multiple faults are the occurrence of two or more faults that are nearly coincident in time (i.e., concurrent relative to the period of interest). These faults can be dependent (i.e., one fault is triggered by another fault) or sequential (i.e., one fault is followed by another fault). Multiple faults can be either transient or permanent in persistence. Single faults also can be dependent faults, such as a fault triggered when a specific operational mode is activated or deactivated in the digital I&C system.

The above fault classes are mainly attributable to hardware domain faults. Developmental based faults, or software errors also are important contributors to digital I&C system failures. As part of this work the research efort used a very conservative approach to support software domain fault emulation. The approach used is function block-oriented fault injection.

Most digital I&C systems of the type found in NPP systems are programmed by function block programming because the applications development engineer understands the operation of the I&C system through the function block representation. Function block-oriented fault injection is aimed at characterizing the function block at its lowest level – data structures, assembly code,

parameters, and internal and interface variables – so that both hardware and proposed software fault models can be applied to the function blocks.

The main goal of function block-oriented fault injection is to promote traceability from the function block representation (high level) to the low level implementation (assembly code, data structures, and variables that are realized on the target I&C machine). While strictly function block-oriented fault injection is not a fault model, the process allows fault models to be placed into the context of the application functionality of the I&C system. By allowing the manipulation of data structures at the interface of function blocks and in the internal space of function blocks, the impact of various software faults (e.g., assignment, checking, reference parameters, etc.) can be assessed, which can provide useful data to the I&C community on how to model software faults and errors (this modeling issue is still an open question).

The important point is that emulating a fault in a function block provides information on the *impact* a particular software fault will have on a system. That is, identifying the failure modes or errors that result, and under what conditions those failure modes or errors arose. The impact of a fault, however, does not reveal the likelihood of the fault being present in the system. This is outside the domain of fault injection-based methods, and is part of a larger research effort related to software quality assurance and metrics.

Lastly*, interaction fault models* that occur in the operational environment and the communication network are significant with respect to highly integrated control rooms incorporating digital I&C systems. There are many human-machine interaction faults that are possible in such an environment; however this research effort focused on three types of interaction faults. The interaction fault models proposed by this research are principally: sensor faults, communication protocol and data faults, and configuration faults. Sensor faults are modeled as permanent or transient faults arising from hardware degradation, noise, and EMI interference. Communication protocol and data faults are modeled as transient faults for protocols and data. Configuration faults are modeled as permanent faults since they represent a configuration change in the system by accidental intent.

4.6. Step 5: Establishing the Operational Profile and Workload

4.6.1. Introduction

An *operational profile* (OP) is a quantitative representation of how a system will be used within its use environment [Musa 1998]. The OP models how users interact and use the system, specifically the occurrence probabilities of system and user modes over a range of operations. Traditionally, it is used to generate test cases and to direct testing of the most used functions; thus, the potential for improved reliability with respect to the use environment is achieved. It associates a set of probabilities or weighting factors to the program input space and therefore assists in the characterization of possible behaviors of the program or collection of programs that represent a system. Determining the OP of a non-trivial system is a challenging part of dependability assessment in general [Shukla 2004].

The OP is traditionally evaluated by enumerating field inputs to the various software functions that comprise a system and then determining their occurrence frequencies from customer and user databases. Musa pioneered a five-step approach to develop the OP, which is based on collecting information on customers and users, identifying the system modes, determining the functional profile, and recording the input states and their associated occurrence probabilities experienced in field operation.

Another term that is often used interchangeably with OP is the *Workload* of a system. While OP and *workload* are similar, they are not the same. A *workload* is a set of tasks or functions and their respective activation input space that reflects the *processing capacity and demand* on an embedded digital system. These tasks are typically application-specific and real-time in nature. The workload on a system can vary depending on the configuration of the system, or its operating state. Thus, a workload is sub-set of an OP.

4.6.2. Operational Profiles for Real-Time Systems

Most digital I&C systems such as Benchmark System I and Benchmark System II are *reactive real-time systems*. A *reactive system* is characterized by its ongoing interaction with its environment, continuously accepting requests from the environment and continuously producing reactions [Wieringa 2003]. In reactive systems, correctness or safeness of the reactive system is related to its behavior over time as it interacts with its environment. Unlike, functional computations, which compute a value upon termination, reactive programs usually do not terminate. A reactive system has *states* which describe the current and past conditions of the plant (the thing that is being controlled or monitored), (2) *environmental stimulus* (inputs) cause *transitions* from one state to another state in the control or processing algorithm, and (3) transitions cause *reactions* (output commands) that dictate control to the plant.

Digital I&C systems that are real-time and reactive operate on a deterministic, time-triggered basis. The software executing in the target computer consists of a set of concurrent tasks governed by a real-time operating system kernel. Each task is represented as finite sequence of events with respect to the operating system task scheduling. These tasks are scheduled on cyclical basis. More precisely, let a *task, J,* running on the target computer be a finite sequence of states defined as s_i, $i = 1,2…n$. This view is typical of a control system that computes a control output command in response to a sensor input. Let a *cycle* of task *J* be the complete scheduled repeated execution of *J* that occurs at frequency f_c. The length of the cycle in time is called the rate of the cycle, denoted as T_s, which is the *sample time* in control system terminology. Therefore, the executions of task *J* occur one cycle at time for every cycle time and are infinitely repeating. A collection these tasks and their respective inputs are the *real-time OP* of the target system.

The difference between an OP for general purpose computing and a real-time OP is that general purpose OPs typically represent many customer or user domains, while real-time OPs are specific to a particular application (user) and its environment. In this research effort, the OP is defined in the context of its application specific nature (i.e., the RPS).

Real time OPs to be used in the fault injection experiments must be selected to be representative of the system under various modes of operation and configuration. Digital I&C system configurations may invoke different hardware and software modules in response to real time demands, and it is important that the fault injection assessment includes sufficient combinations of these configurations to ensure a thorough evaluation of their behavior in the presence of faults.

4.6.3. Characterization of Real-time Operational Profile for Fault Injection

The first step in characterizing an OP is to establish a use profile the digital I&C system uses according to its various operational modes. As an example, the process with respect to a generic digital I&C system is described in this section.

A typical digital I&C system used in a safety critical plant application has at least four defined operating modes (see Table 4-1). These are operating modes (1) boot up mode, (2) normal

mode, (3) test mode, and (4) parameter change mode. In the boot-up mode, diagnostics, self-tests, and health checks are executed before engagement to the normal operating mode. During this phase, the systems does not receive inputs from the NPP system. The average boot-up time duration is around 2-3 minutes. A “boot-up” of the system would most likely occur after plant outages or after reactor shutdown events.

Table 4-1 Example composition of an operational profile for digital I&C system.

Operational Mode	Sub-System Activity					Time Interval
	Processors	Fault Tolerance	Input/Output (I/O)	Communi-cations	Service Unit Interface	
Boot up	Diagnostic test patterns, config checks, com checks.	May be temporarily disabled or diminished during testing	Diagnostic checks, Connectivity. I/O disengaged	Protocol initialization, diagnostics, and connectivity.	Diagnostics, connectivity checks, com checks.	2-3 minutes
Normal	Safety Function or control law operational. Various of modes of operation depending on plant configuration	Full system error and fault detection.	Acquisition of plant specific Inputs for the safety functions or control laws. Outputs signals are sent to the plant interfaces.	Data and health status traffic is passed between operational units.	System health and performance messages are sent to the operator service and monitoring station	10 -18 months
Test mode	Special Diagnostic routines, and run-time monitors are available to run concurrently with safety and control law functions	May invoke special diagnostics to enhance the detection, and isolation of a fault	Plant specific Inputs, output(s) may be disengaged.	Data and health status traffic is passed between operational units	Special diagnostic messages are sent to the operator and monitoring station	As needed for testing (~24 hours -48 hours)
Parameter Change	Ability to re-calibrate plant parameters in the safety function and control laws.	Should have full system error and fault detection.	Plant specific Inputs and outputs, possibly special test inputs to validate the parameter change	Data and health status traffic is passed between operational units	System health and performance messages are sent to the operator service and monitoring station	Plant dependent (~8 hours)

During normal operation, the digital I&C system monitors or controls the plant system to ensure safe and reliable operation for the prescribed safety envelope. The normal operating mode is the mode where the safety functions or control algorithms would be allowed to execute. The functional modes for normal mode operation are application dependent, but they always relate to the operating state of the plant. For instance, in an NPP the reactor could be operating at rated power, a transition mode, start-up mode, low-power mode, or manual mode. The

functional modes within the safety functions or control algorithm would be a part of the operational profile make-up.

Another mode is the test mode. The test mode allows for a part (or all) of a digital I&C system to be placed in a special mode in which operational aspects of the system can be measured for surveillance monitoring purposes. In this mode, the safety functions may be executing, but the actuation outputs of the digital I&C may be disengaged. Special monitoring diagnostics are usually invoked to monitor various sub-systems performance, error reports, and trends. Test mode operation usually occurs as part of a planned surveillance interval, a planned outage, or an unplanned outage event.

The parameter change mode allows specific parameter changes to the control or safety function software. This mode of operation may occur during normal operating conditions where the controller needs to tune or calibrate the control system to compensate for slow dynamic changes occurring within the plant. Like the test mode, this mode of operation is fairly infrequent as compared to the normal operating mode. A parameter change is usually a planned action by the operating staff.

After the operational modes have been identified and characterized, the next step is to define how the operational will be used in the testing environment. Since workload and the input stimulus to the system in various modes of operation can have significant impact on the estimation of parameters such as coverage [Folkesson 1999], it is important to represent the operations of the system accurately. Since digital I&C systems are usually reactive real time systems responding to their environment their interaction to the input environment will be discussed first.

If there are no inputs or a reduced set of inputs from the external environment, only a reduced set of software and hardware resources on the digital I&C system may be used (cyclic idle tasks, diagnostics, and so forth). This portion of the OP is referred to as a system ***light*** workload. A transient fault that occurs during a system light workload has a higher probability of not being activated as an error. If activated as an error, the transient fault may not produce a failure until an input/output activity starts. Also a permanent fault that occurs during a system light workload has a higher probability of not being activated as an error by the idle tasks, but it has a higher probability of being activated as an error and detected by the system diagnostics since those tasks are running without preemption. This assumes that the fault activation interval is longer than the time in which diagnostics are completed.

If there are many inputs and events from the external environment, a majority of the software and hardware resources of the digital I&C system are used. This portion of the operational profile is a system ***heavy*** workload. In general, when testing the functionality of a given system, the OPs should be selected to exercise as much of the system as possible. 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 [Choi 1999]. A transient fault that occurs during a system heavy workload has a higher probability of being activated as an error, and 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 a diagnostic function. The exception to this case is when the fault occurs in a location not being used by the workload. In this case, the diagnostic functions will take a longer time to detect the fault or error assuming that the fault activation 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 (i.e., an incorrect output must 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), should 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.

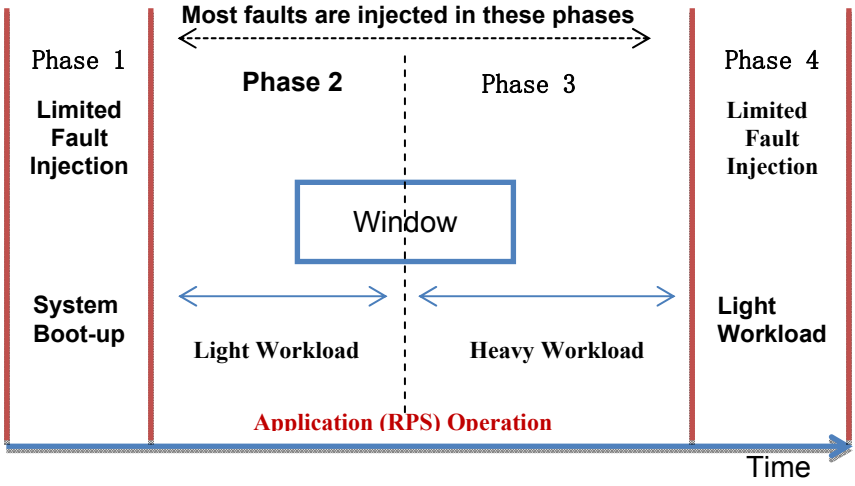

Figure 4-5 Representative operational profile for fault injection experiments

In summary, the operational profile shown in Figure 4-5 is considered to be representative of the types of OPss that must be used for fault injection experiments in order to properly exercise complex hardware/software systems. As such, OPs like the one shown in Figure 4-5 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 4-5, OPs consist of:

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

Most faults are injected during **Phase 2** and **Phase 3**, and on a limited basis in **Phase 1** and **Phase 4** due to the justifications presented above. It is worth noting that the window shown in Figure 4-5 can be seen as the OP to be applied for a given fault injection experiments. Sliding the window left and right produces a set of OPs with many stress conditions.

4.7. Step 6: Injecting Faults into the Target System

Figure 4-6 shows the essential components of a fault injection environment. This has been adapted from reference [Hsueh 1997]. There are a number of fault injection techniques and tools that are available to the designer for dependability validation. Section 5 provides a detailed survey and classification of the various fault injection techniques that are applicable to digital I&C system.

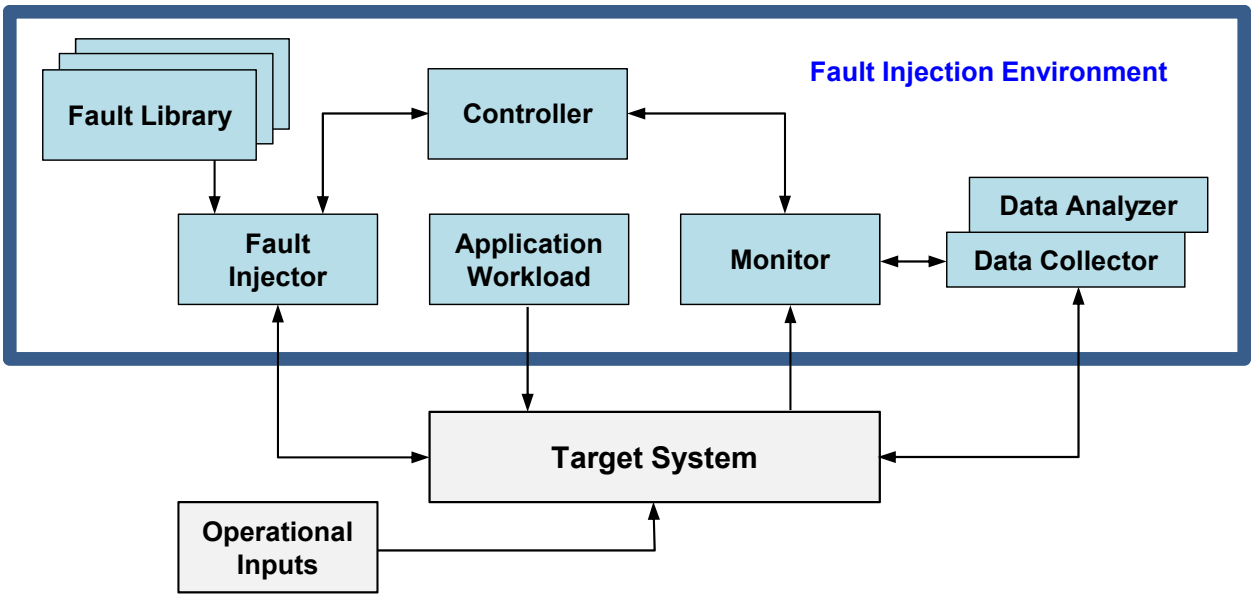

Figure 4-6 Basic architecture of a fault injection environment

Referring to Figure 4-6, the generic representation of the fault injection environment combines the key elements of a fault injection methodology into a setting where the theory of fault injection can be realized. In Figure 4-6, the *Target System* is the system to which the fault injection is applied. The Target System executes tasks assigned from the application workload environment. The *Application Workload(s)* of the system are representative programs the target system typically executes in its application domain. The *Operational Inputs* define the input domain for the Target System with respect to the various workloads that the system executes. The *Fault Library(s)* contains the list of faults to be injected into the target machine by the *Fault Injector*. The list includes faults that are representative of fault classes that are expected to be encountered. The *Monitor* globally keeps track of execution on the target and initiates data collection when necessary. The *Data Collector* captures the effects of faults as they propagate through the target. The effectiveness of the data collector determines the quality of the results obtained upon analysis of the collected data by the *Data Analyzer*.

4.7.1. Required Attributes of a Fault Injection Environment

This section presents attributes that a fault injection environment should have to satisfy the requirements of the FARM model of fault injection. Some requirements pertain to providing the capability for automating fault injections, and some requirements are concerned with the ability to represent different fault models so that a wide variety of failure modes can be tested.

4.7.1.1. Support for Fault Models

An effective fault injector must be able to emulate various fault models so that the assessor of a system can test the fault tolerance mechanisms under the effects of various types of the faults indicated in the fault taxonomy of Section 1.7.3 and in the previous section. Hence, the ability to model various fault models using the same fault injection environment is a valuable feature. Furthermore, the ability to use several different fault injection techniques from a single environment aids in the overall usability of the fault injection environment from one platform to another.

4.7.1.2. Support for Precise Fault Activation

Support for precise fault activation is an essential feature related to the requirements of the FARM model. Recall, the fault space F has three basic dimensions; (1) location of fault activation, (2) time of fault activation, and (3) duration of fault activation. In fault injection, the ability to inject a fault based on these dimensions that might be applied to emulate a certain fault is a concern. For example, a fault may need to be injected at a random point in time to emulate a transient fault. Or, a fault may be injected when a certain mode or input condition or certain event occurs on a given variable. The ability to set up composite timing and triggering constraints is an essential feature that ensures various operational modes of the target system can be exercised effectively. Being able to precisely control the time, location, and duration of when a fault is to be injected improves the controllability of the fault injection process, and thus improves the repeatability and efficacy of the fault injection experiments.

4.7.1.3. Support for Experiment Control

The design of experiments consists of deciding a number of controllable parameters such as fault location, fault value, fault dependence, fault timing, persistence of faults, and so on. For example, there are a number of available choices as to what a fault value might be. These might include:

- Min/Max: The value of a variable that must be corrupted can be made a minimum or a maximum of what it is capable of representing.
- Random bit-flip: A randomly selected bit in the binary representation of the variable can be toggled.
- Random value: A random value can be chosen from a valid set of values a variable might be allowed to represent.
- Invert: All bits in the binary representation of the variable could be toggled.

Similarly, persistence of the fault might be permanent or transient depending on whether the value is corrupted each time it is used or just once at a random point in time.

A list of fault locations corresponds to what is known as a *fault list*. A collection of similar lists like these is called a fault library. The fault injector must be able to load a number of fault lists or libraries and should be able to swap between fault lists giving the ability to set up complex fault patterns. The fault injection environment that is built must provide the capability to easily decide these parameters and thereby allow easy set-up of experiments.

4.7.1.4. Support for Automation

Being able to automate fault injection is essential to being able to collect large amounts of data so that adequate confidence can be placed in the parameters that are determined during the statistical estimation process. A number of capabilities must come together to enable automation of fault injection.

When integrated into a complex microprocessor based system, the fault injector must be able to communicate with the system in order to inject a fault, or to convey status messages back to the data collection subsystem or any other part of the fault injection environment. The fault injector must have output signals that it can issue and also input signals that it can detect. Being able to issue and detect these signals is crucial for communication between the Target System and the Fault Injector at different levels. For example, the fault injector must be coordinated with system

resets, when the system has completed booting up, and so on. Exchanges of this kind of information is needed at a level that is different from the main connection between the fault injector and the target system, which is typically at a physical interface level (e.g., the CPU, a test port, the pins of a device, etc.).

The fault injector also should have commands that can be issued from a remote host computer to perform various tasks, such as the ability to compile and execute command scripts to enable automatic set up of tasks to support fault injection. These various tasks include detection of events, setting up, enabling or disabling software or hardware breakpoints, performing memory/register corruptions, and halting and resuming the CPU, through low level commands.

4.8. References

[Zemva 1998] A. Zemva, Z. Baldomir. "Functionality Fault Model: A Basis For Technology-Specific Test Generation." *Microelectronics & Reliability*, 1998: 297-604.

[Smith 2000] Smith, D. T., DeLong, T. A., Johnson, B. W., and Giras, T. C., "A Safety Assessment Methodology for Complex Safety-Critical Hardware/Software Systems", Proceedings of the International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies, Washington, D.C., November 13-17, 2000, 13 pages, (Invited Paper).

[Reynolds 2009] M. Reynolds, C.R. Elks, N. George, M. Sekhar, Todd Delong, and Barry Johnson. "A Quantitative Safety Assessment Methodology for Safety-Critical Programmable Electronic Systems Using Fault Injection"*,* 2009 SAE World Congress April 20-23 2009, Cobo Center, Detroit, Michigan

[Elks 2010] C. Elks, M. Reynolds, N. George, M. Miklo, M. Sekhar, B. W. Johnson. *"*New Fault Injection Methods For Safety Critical Digital I&C Systems: Application to Commercial Safety Grade I&C Platforms*"*. Seventh American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Control and Human-Machine Interface Technologies NPIC&HMIT 2010, Las Vegas, Nevada, November 7-11, 2010.

[Anceau 2004] Anceau, F. *The Architecture of Microprocessors.* Addison-Wesley, 1986.

[Avizienis 2004] Avizienis, A, J-C Laprie, B Randell, and C Landwehr. "Basic concepts and taxonomy of dependable and secure computing." *IEEE Transactions on Dependable and Secure Computing*, no. 1 (2004): 11-33.

[Baumann 2005(b)] Baumann, R. C. "Radiation-induced soft errors in advanced semiconductor technologies." *#IEEE_J_DMR#* 5, no. 3 (2005(b)): 305-316.

[Baumann 2005(a)] Baumann, R. "Soft Errors in Advanced Computer Systems." *IEEE Design and Test of Computers*, 2005(a): 258-266.

[Bernardi 2004] Bernardi, P, M Reorda, L Sterpone, and M Violante. "On the evaluation of seu sensitiveness in sram-based fpgas." 2004. 115-120.

[Yount 1996] C.R. Yount, D.P. Siewiorek. "A Methodology for the Rapid Injection of Transient Hardware Errors." *IEEE Transactions on Computers*, 1996: 881-891.

[Constantinescu 2002] Constantinescu, C. "Impact of Deep Submicron Technology on Dependability of VLSI Circuits." *Dependable Systems and Networks.* International Conference, 2002. 205-209.

[DeLong 2005] DeLong, T. A., D. T. Smith, and B. W. Johnson. "Dependability metrics to assess safety-critical systems." *#IEEE_J_R#* 54, no. 3 (2005): 498-505.

[Cutright 2003(a)] E. Cutright, T. DeLong, B.W. Johnson. *Analytical Safety Model.* UVA-CSCS-NSE-002, Charlottesville: University of Virginia, 2003(a).

[Cutright 2003(b)] E. Cutright, T. DeLong, B.W. Johnson. *Generic Processor Fault Model.* UVA-CSCS-NSE-004, Charlottesville: University of Virginia, 2003(b).

[Cutright 2003©] E. Cutright, T. DeLong, B.W. Johnson. *Statistical Model.* UVA-CSCS-NSE-003, Charlottesville: University of Virginia, 2003(c).

[Cutright 2003(d)] E.D. Cutright, T.A. Delong, B.W. Johnson. *A Numerical Safety Evaluation Process for Safety-Critical Systems.* UVA-CSCS-NSE-001, Charlottesville: University of Virginia, 2003(d).

[Folkesson 1999] Folkesson, P, and J Karlsson. "Considering Workload Input Variations in Error Coverage Estimation." 1999. 171-188.

[Kumamoto 1996] H. Kumamoto, E.J. Henley. *Probabilistic Risk Assessment and Management for Engineers and Scientists, 2nd ed.* IEEE, 1996.

[Choi 1999] J. G. Choi, P.H. Seong. "Dependability Estimation of Digital System by Operational Profile-Based Fault Injection." *Proceedings of the Probabilistic Safety Assessment.* Washington D.C. , 1999. 499-506.

[Kannan 2000] K. Kannan, L. Kaufman, B.W. Johnson. *Network Architectures for Safety-Critical Control Applications.* Technical Report 000111.0, Charltotesville: University of Virginia, 2000.

[Kaufman 1999(a)] L. Kaufman, B.W. Johnson. *Embedded Digital System Reliability and Safety Analysis.* NUREG/GR-0020, NRC, 1999(a).

[Kaufman 1998] L. Kaufman, B.W. Johnson. *The Importance of Fault Detection Coverage in Safety Critical Systems.* NUREG/CP-0166, Proceedings of the 26th Water Reactor Safety Information Meeting: U.S. NRC, 1998.

[Kaufman 1999(b)] L. Kaufman, S. Bhide, B.W. Johnson. *Analysis of Common Mode and Common Cause Failures in Digital Embedded Systems.* Technical Report 991222.0, Charlottesville: University of Virginia, 1999(b).

[Rashid 2010(b)] L. Rashid, K. Pattabiraman, S. Gopalakrishnan. "Towards Understanding the Effects of Intermittent Hardware Faults on Programs." *2010 International Conference on Dependable Systems and Networks Workshops.* DSN-W, 2010(b). 101-106.

[Rashid 2010(a)] L. Rashid, Pattabiraman, K., Gopalakrishnan, S. "Modeling the Propagation of Intermittent Hardware Faults in Programs." *IEEE 16th Pacific Rim International Symposium.* Tokyo, Japan: IEEE, 2010(a). 19-26.

[Hsueh 1997] M.-C. Hsueh, T.K. Tsai, R.K. Iyer. "Fault Injection Techniques and Tools." *Computer*, 1997: 75-82.

[Musa 1998] Musa, J. *Software Reliability Engineering.* McGraw Hill, 1998.

[George 2010(b)] N. George, C. Elks, B. Johnson, J. Lach. "Spatial Multi-Bit Soft-Error Tolerance in Logic." *Dependable Systems and Networks Symposium.* Chicago, IL, 2010(b).

[George 2010(a)] N. George, C. Elks, B. Johnson, J. Lach. "Transient Fault Models and Architecture Vulnerability Factor (AVF) Revisited." *Dependable Systems and Networks Symposium.* Chicago, IL, 2010(a).

[Pescosolido 2002] Pescosolido, M. "Statistical Models for Coverage Estimation." *School of Engineering and Applied Science Masters Thesis.* University of Virginia, May 2002.

[Wangqi 2004] Q. Wangqi, L. Xiang, W. Jing, D.M.H. Walker. "A Statistical Fault Coverage Metric for Realistic Path Delay Faults." *VLSI Test Symposium Proceedings.* IEEE, 2004. 37-42.

[Shukla 2009] R. Shukla, D. Carrington, P. Strooper. "Systematic Operational Profile Development for Software Components." *Software Engineering Conference, 11th Asia-Pacific.* 2004. 528-537.

[Sierawski 2009] Sierawski, B. D., et al. "Impact of Low-Energy Proton Induced Upsets on Test Methods and Rate Predictions." *#IEEE_J_NS#* 56, no. 6 (2009): 3085-3092.

[Srinivasan 2004] Srinivasan, J., S. V. Adve, P. Bose, and J. A. Rivers. "The impact of technology scaling on lifetime reliability." 2004. 177-186.

[Ubar 1998] Ubar, R. "Combining Functional and Structural Approaches in Test Generation for Digital Systems." *Microelectronics & Reliability*, 1998: 317-329.

[Wieringa 2003] Wieringa, R.J. *Design Methods for Reactive Systems, 1st ed.* Morgan Kaufman, 2003.

[Yu 2003] Y.Yu, B.W. Johnson. *Bayesian Belief Network and Its Applications.* UVA-CSCS-BBN-001, Charlottesville: University of Virginia, 2003.

[Yount 1993] Yount, C.R. *The Automatic Generation of Instruction-Level Error Manifestation of Harware Faults: A New Fault-Injection Model.* Carnegie Mellon University: Ph.D. Dissertation, 1993.

5. FAULT INJECTION METHODS FOR DIGITAL I&C SYSTEMS: A SURVEY AND CHARACTERIZATION

This Section presents a comprehensive description of contemporary and emerging techniques for fault injection. The first three sections address fault injection into physical systems. This Section is significant because the advantages and disadvantages of fault injection methods and techniques with respect to digital I&C systems must be weighed during the selection process. The principle aim of this research was to identify and develop methods for injecting faults into digital I&C systems of the type found in Benchmark System I and Benchmark System II systems, consequently the following sections provide essential information on this research. Four additional sections address software simulation-based fault injection, hardware emulation-based fault injection, hybrid fault injection, and novel methods to provide a comprehensive state-of-the-art view of fault injection for digital I&C systems. Hardware faults can be injected or emulated by all the techniques described in this Section; some techniques can accommodate software fault injection. The methods that can accommodate software faults are identified in this Section.

5.1. Introduction

As discussed earlier in this report, fault injection is a dependability validation technique based on the realization of formal controlled validation experiments in which system behavior is observed while faults are explicitly induced in the system by the deliberate introduction (injection) of faults. That is, faults are injected into the system and the resulting behavior is observed. This technique accelerates the occurrence and the propagation of faults in a system for the purpose of observing the effects of faults on the system performance and behavior. Moreover, it cannot be emphasized strongly enough that the selection of a fault injection technique must directed by what what is to be measured or observed in the fault injection process.

For the practitioner or user of fault injection, the variety of fault injection techniques and tools are many. The claimed capabilities (often stated without assumptions), and tradeoff space make decisions about fault injection difficult. This report provides a structured survey in order to organize fault injection methods into classes. Additionally, the benefits, assumptions, and disadvantages of various techniques are summarized so that decision-making regarding the selection of fault injection methods can be done in a systematic manner.

5.1.1. Preliminaries

Before discussing and comparing different fault injection techniques in detail, the terminology use in this Section must be defined. As introduced in Section 4, a *Target System* is a generic term for the digital I&C system under test or assessment. The target system executes a *workload*, which is determined by the program executed by the target system and the data processed by the program. The faults injected during the experiments constitute the *fault-load*.

The difference between *a fault injection experiment* and *a fault injection campaign* is that a fault injection experiment corresponds to injecting one fault and observing, or recording, how the target system behaves in presence of that fault. To gain statistical confidence in the assessment or the verification of a target system, data must be collected from many fault injection experiments. A *fault injection campaign* comprises a series of fault injection experiments conducted on a target system.

Fault injection techniques can be compared and characterized on the basis of several different properties. The following properties are used to characterize the fault injection techniques described in this Section:

- *Controllability* – the ability to control the injection of faults in time and space.
- *Observability* – the ability to observe and record the effects of an injected fault.
- *Repeatability* – the ability to repeat a fault injection experiment and obtain the same result.
- *Reproducibility* – the ability to reproduce the results of a fault injection campaign.
- *Reachability* – the ability to reach possible fault locations inside an integrated circuit, or within a program.
- *Fault representativeness* – how accurately the fault load represents real faults.
- *Workload or Operational Profile representativeness* – how accurately the workload represents real system usage.
- *System representativeness* – how accurately the target system represents the real system

5.2. Classification of Fault Injection Techniques

As shown in Figure 5-1, the application and realization of fault injection is broadly classified into two predominant categories; (1) *physical-based fault injection,* and (2) *simulation-based fault injection.* In physical-based fault injection faults are injected into the hardware and software of the operational digital I&C system. The target system usually represents what will be deployed in the field of operations. The level of representation of the system under test is very high, and in many cases the tested system and the actual system are identical.

On the other hand, simulation-based fault injection involves the construction of a simulation model of the target system, which typically includes varying levels of detail of the actual digital I&C system structural and behavioral information. Often the models are developed using a modeling description language, such as SystemC [Bhasker 2004], Architecture Description Language [Association 2011], SIMICS [River 2011], or very high speed integrated circuit hardware description language (VHDL) [Ashenden 1995]. This type of fault injection is generally used during the design stages for verifying the intended implementation of a system before it is released to manufacturing. The simulation-based fault injection is non-intrusive and can support various system abstraction layers such as application levels, operating system levels, and hardware levels. Hence, a wide variety of fault types can be simulated using this technique. However, because the simulation-based fault injection uses models to represent an actual system under test, system representation fidelity concerning system interactions can be an issue.

Figure 5-1 illustrates the classification of fault injection based on these classes. The remainder of this Section discusses each of these classes.

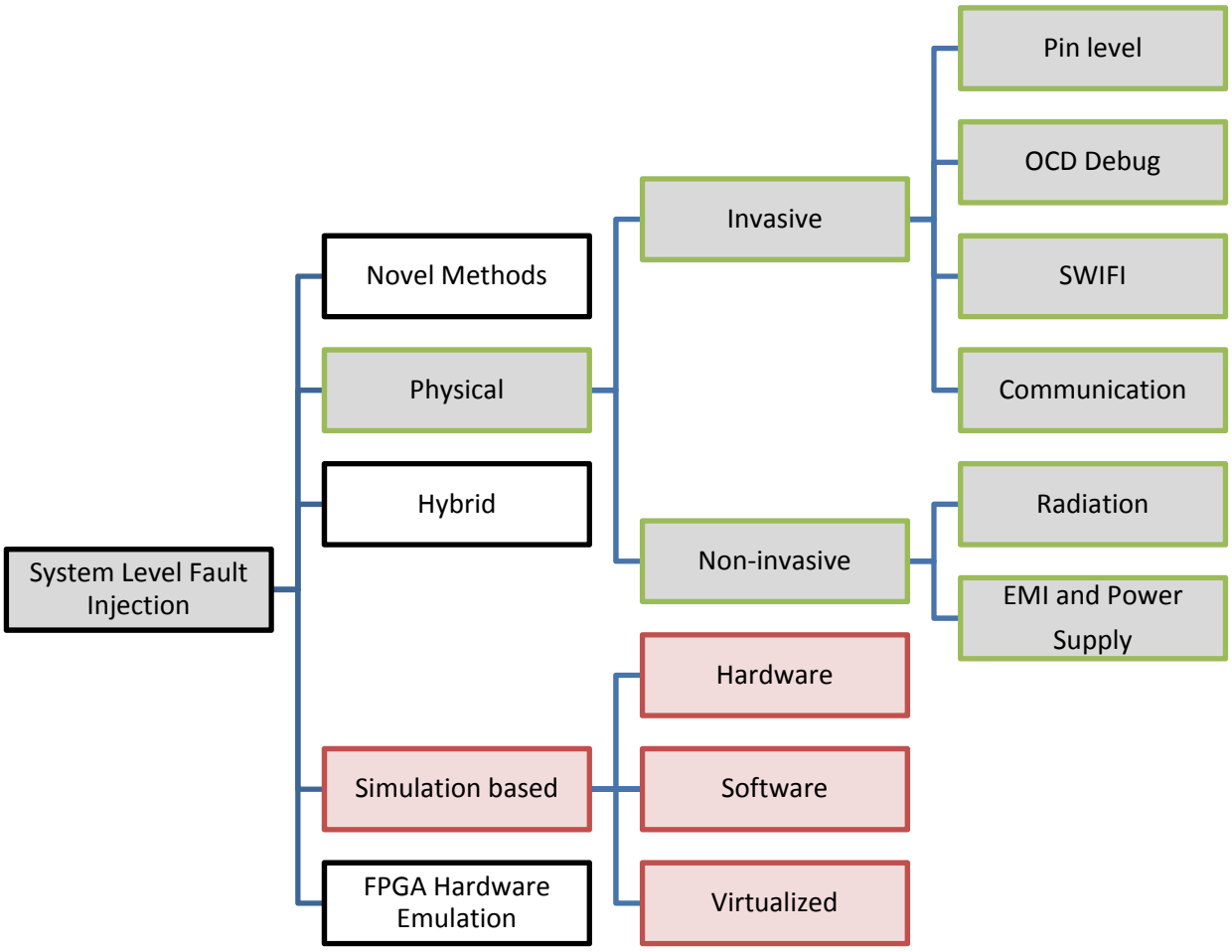

Figure 5-1 Taxonomy of fault injection techniques

5.3. Physical Implemented Fault Injection Methods

Physical-based fault injection includes four techniques: pin-level fault injection, power supply disturbances, radiation-based fault injection, and test port-based fault injection. Physical based fault injection involves exercising a system under test or analysis with specially designed test hardware to allow the injection of faults into the target system and to examine the fault effects. These four techniques are discussed in the following sections.

5.3.1. Pin-Level Fault Injection

In pin-level fault injections, faults are injected via probes connected to electrical contacts of ICs or discrete hardware components. This method was used in the 1970's for generating fault dictionaries for system diagnosis. Many experiments and studies using pin-level fault injection were carried out during the 1980's and early 1990's.

Pin level faults are injected at the IC pin level or at the boundary of the IC. The fault models that are typically used in pin level fault injections are the stuck-at, bridging of two or more connections, noise, reduced voltage, and transient fault models. There are two types of pin level fault injection techniques normally used:

Forcing technique: In this technique the fault is injected directly into an IC pin terminal or connector without disconnecting the IC or part from the board. The fault injector probe forces a

low or high logical level at the selected points. Since, the fault injector is "fighting" the output voltage of the IC pin, there is always a chance the IC could be damaged if the probe's current exceeds the limit of the IC pin driver circuits.

Insertion technique: A special IC socket carrier replaces the original chip socket of the IC. The special IC socket device is instrumented so that the voltage levels on the pins can be altered by the fault injection apparatus. This allows the connection between the IC and the chip socket to be cut off before injecting the fault. Thus, the injection is performed on the side that remains at high impedance. Because there is not any signal forcing, there is not any danger of damage to the IC pin driver circuits.

Several pin-level fault injection tools have been developed (e.g., MESSALINE [Arlat 1990(a)] and RIFLE [Madeira 1994]). A key feature of these tools is that they support fully automated fault injection campaigns. However, the increasing level of integration of electronic circuits and advanced IC packaging methods has rendered the pin-level technique somewhat problematic (and in some cases infeasible) as a general method for evaluating fault-handling mechanisms in computer systems. The method is, however, still valid for assessing systems where faults in electrical connectors pose a major problem, such as automotive and industrial embedded systems. As such, the pin-level fault injection technique is a special case, and not so much a general method to be considered for digital I&C systems.

Advantages

- Best suited for faults that are associated with board faults and output driver circuits of ICs.
- Experiments can be conducted in an expedient manner.
- Experiments can be run in real-time, allowing for the possibility of running a large number of fault injection experiments.
- Fault injection experiments are performed using the same software that will run in the field.
- Running the fault injection experiments on hardware that is executing the system software has the advantage of including design faults that might be present in the actual hardware and software design.
- Commercial tools are readily available.

Disadvantages

- Hardware fault injection can introduce the risk of damage to the IC.
- The technique is relatively obsolete with respect to contemporary IC technology.
- There is a high level of device integration, multiple-chip hybrid circuits, and dense packaging technologies that limit accessibility to pin level fault injections. Many current IC package technologies completely eliminate the possibility of pin level fault injection (e.g., ball grid arrays and pin grid arrays).
- The technique cannot represent faults that occur inside the IC.

- There is a limited set of injection points and a limited set of injectable faults.

- The technique requires special-purpose hardware in order to perform the fault injection experiments, although the hardware can be purchased commercially.

5.3.2. Power Supply Disturbance Fault Injection

Power supply disturbances (PSDs) are rarely used as primary methods for fault injection because of low repeatability and controllability of the experiments. The principle reason for using PSD fault injection is to investigate common cause failures (CCFs) due to power supply disturbances in the digital system. As such, the PSD technique has been used mainly as a complement to other fault injection techniques in the assessment of error detection mechanisms for processor-based safety critical systems [Karlsson 1991; Miremadi 1995; Rajabzadeh 2004; Tummeltshammer 2009]. The impact of PSD fault injections is usually much more severe than the impact of other commonly used injection techniques (e.g., those that inject single bit-flips) since PSDs tend to affect many bits and thereby a larger part of the system state. Interestingly, some error detection mechanisms show lower fault coverage for PSDs than for single or multi bit-flip errors [Rajabzadeh, 2004; Tummeltshammer, 2009]. For this reason, PSD fault injection should be a special type of fault injection to consider if a CCF due to power disturbances is a concern. Since CCFs are a concern in digital I&C systems, this type of fault injection should be explored as a possible complementary technique to more general fault injection methods.

Advantages

- Fault models for PSD are well accepted. These include power level droop, noise augmentation, delay, and modulated power fluctuations.

- The technique is relevant to real world faults, and is becoming increasingly important with lower voltage, higher speed IC technologies.

Disadvantages

- There is low repeatability and controllability.

- The technique can be difficult to isolate a power supply feed due to the different DC voltages used in digital ICs. For example, core voltage for a processor is different from the input/output (I/O) voltage.

- Implementation requires special circuits to disturb the power supplies at the board and chip level. May require some modification to the system.

5.3.3. Radiation Based Fault Injection

Modern electronic integrated circuits and systems are sensitive to various forms of external disturbances such as EMI and particle radiation. Traditionally, digital systems that were exposed to various forms of radiation were mainly confined to aerospace and space-borne systems. However, due to aggressive scaling in IC technology, the vulnerability factor for ground-based digital I&C systems is on the increase. Thus, one means of validating or investigating a fault tolerant system's susceptibility to this type of disturbance is to expose the system to disturbances such as high energy cosmic particles and EMI.

A growing reliability concern for computer systems is the increasing susceptibility of ICs to soft errors (i.e., bit-flips caused when highly ionizing cosmic particles affect sensitive regions within

in a circuit). Soft errors have been a concern for electronics used in space applications since the 1970's. In space, soft errors are caused by cosmic rays, i.e., highly energetic heavy-ion particles. Historically, heavy-ions are not a direct threat to electronics at ground-level and airplane flight altitudes because they are mostly absorbed when they interact with the Earth's atmosphere. However, more current circuit technology generations have become increasingly sensitive to high energy neutrons, which are generated in the upper atmosphere when cosmic rays interact with the atmospheric gases. Such high energy neutrons are a major source of soft errors in ground-based and aviation applications using modern ICs. To underscore this vulnerability, nearly all modern microprocessors, FPGAs manufactured in technologies with feature sizes below 90 nm are equipped with low level error detection and error correcting code mechanisms to cope with soft errors [George 2010(b)]. While these low level error detection and correction mechanisms improve the reliability of the IC operations, they do not eliminate all error manifestations [George 2011].

To assess the efficiency of such fault tolerance mechanisms, semiconductor manufacturers are now regularly testing their circuits by exposing them to ionizing particles. The neutron beam facility at the Los Alamos Neutron Science Center (LANSCE) in the United States is often used for such tests as its energy spectrum is very similar to that of natural high energy neutrons at sea-level. Similar neutron beam facilities are the Osaka University RCNP and the ANITA Neutron Source at The Svedberg Laboratory in Uppsala, Sweden. Results of neutron beam testing are reported for the Intel Itanium microprocessor in [Constantinescu 2005(a)], for the SPARC64 V microprocessor in [Ando 2008], and for several generations of microprocessors from Sun Microsystems in [Dixit 2009]. Sometimes proton radiation is used as a slightly less expensive alternative to neutron beam testing. Results of proton beam testing of the IBM POWER 6 processor can be found in [Kellington, 2007].

The sensitivity of ICs to heavy-ion radiation can be exploited for assessing the efficiency of fault-handling mechanisms. In [Gunneflo, 1989] and [Karlsson 1991], results from fault injection experiments conducted by exposing circuits to heavy-ion radiation from a Californium-252 source is reported. This method was also used in the previously mentioned study [Arlat 2003], in which the impact of four different fault injection techniques was compared. In this study, the main processor as well as the communication processor of a node in a distributed system was exposed to heavy-ion radiation. The results showed that the impact of the soft errors injected by the heavy-ions varied extensively and that they activated many different error detection mechanisms in the target system.

5.3.3.1. Electromagnetic and High Intensity Radiated Electromagnetic Fields (HIRF)

With respect to EMI disturbances, the principle threat of concern is the HIRF. In particular, HIRF environments have the potential to cause random fault manifestations in individual digital system components and to generate simultaneous system-wide faults that overwhelm fault detection and management mechanisms. This is typically called *the massive transient fault* phenomena. Research and reports of these phenomena are well known in the aviation and aerospace community. Examples of serious accidents strongly suspected to be caused by HIRF are the TWA Flight 800 and the Swissair Flight 111 accidents [Evans 2000]. These phenomena typically occur when a digital I&C system comes in close proximity to a source EMI radiator, such as civilian or military radio frequency (RF) transmitters, or even a hand held RF transmitter such as a police/emergency communication radio.

In [Arlat 2003], EMI fault injection was used along with three other fault injection techniques to evaluate error detection mechanisms in a computer node in a distributed real time system. A primary goal of this study was to compare the impact of pin-level fault injection, EMI, heavy-ion radiation and software-implemented fault injection. The study showed that the EMI injections tended to "favor" one particular error detection mechanism. For some of the fault injection

campaigns almost all faults were detected by one specific CPU-implemented error detection mechanism, namely spurious interrupt detection. This illustrates the difficulty in using EMI as a fault injection method. Another use of EMI for fault injection is reported in [Vargas 2005].

EMI or HIRF fault injection is typically conducted in a specialized test facility using a reverberation chamber (RC). An RC is an electrically conductive shielded enclosure used for generating an electromagnetic environment for radiated susceptibility and emissions testing. The operational concept is similar to a very large microwave oven. When a radiated field at a resonant frequency enters the cavity, it is reflected back and forth between the walls with low energy loss, and additional energy entering the cavity reinforces the standing wave and increases its intensity. This resonance phenomenon allows the generation of high intensity electromagnetic fields with relatively low input power. However, it has the disadvantage that the spatial distribution of the field is not uniformly distributed. In practice, a transmit antenna is used to emit RF energy inside the chamber setting up a complex field structure within the chamber. Rotating mechanical stirrers then "mix" the energy, effectively changing the boundary conditions and creating new complex field structures. When sampled over time, this stirring results in a statistically uniform filed environment.

The principle facility in the US for conducting HIRF and EMI susceptibility testing is at the NASA Langley Research Center High Intensity Radiated Fields Laboratory located in Hampton, Virginia [Meyer 2009].

Radiation-based fault injection and EMI/HIRF injection has low repeatability. Due to low controllability, it is difficult to precisely synchronize the activity of the target system with the time and the location of an injection in a radiation-based fault injection experiment. Thus, it is difficult to repeat an individual experiment to reproduce exact results. However, the ability to statistically reproduce results over many fault injection campaigns is usually high in both types of fault injection environments.

Advantages

- Radiation-based fault injection can reach regions inside ICs that are difficult to reach with other physical fault injection techniques.
- Radiation-based testing is the primary means to verify the effects of single upset events and multiple event upsets due to cosmic particle strikes on semi-conductor components.
- Radiation-based fault injection and HIRF disturbance injection have very low intrusiveness, that is, the target system behavior is not altered to accommodate fault injection.
- Both radiation-based fault injection and HIRF testing are required when a digital I&C system is expected to operate in environments where these threats are known to exist.

Disadvantages

- Radiation-based fault injection has poor repeatability and reproducibility of experiments due to poor controllability. Repeatability and reproducibility are required for attaining statistical significance of results.
- Both radiation-based fault injection and HIRF testing requires highly specialized testing facilities to conduct tests. However, such facilities are operational at national labs and other government agencies (NASA).

5.3.4. Test Port and On Chip Debugging (OCD) Based Fault Injection

Test port-based fault injection encompasses techniques that use test ports to inject faults in microprocessors. Many modern microprocessors (since ~1998) are equipped with built-in debugging and testing features, which can be accessed through special I/O-ports known as test access ports (TAPs), or just test ports. Test ports are defined by standards such as the IEEE-ISTO 5001-2003 (Nexus) standard [Organization 2003] for real-time debugging, the IEEE Std 1149.1, "Standard Test Access Port and Boundary-Scan Architecture (JTAG)" [IEEE Std 1149.1 01], and Intel's OnCE and Motorola's Background Debug Mode (BDM) facility. Nexus and Joint Test Action Group (JTAG) are standardized solutions used by several semiconductor manufacturers, while BDM and OnCE are proprietary solutions for debugging developed by Freescale and Intel. The important point is that nearly all modern processors used in embedded computing are equipped with some form of OCD capability. Volume 3 describes a customized high performance OCD based fault injector was developed for digital I&C systems [Miklo 2011].

Tools for test port-based fault injection are usually implemented on top of an existing commercial microprocessor debug tool, since such tools contain all functions and drivers that are needed to access a test port. The type of fault that can be injected via a test port depends on the debugging and testing features supported by the target microprocessor. Normally, faults can be injected in all registers in the instruction set architecture (ISA) of the microprocessor and certain I/O test registers of the main signals of processor. BDM and Nexus also allows injection of faults in main memory. BDM and Nexus allow access to hardware structures in the micro-architecture that are invisible to the programmer. However, information on how to access such hardware structures is usually not normally disclosed by manufacturers of microprocessors, but manufacturers have been cooperative with the fault injection community along these lines [Communications 2011]. Since OCD methods can manipulate or alter the instructions on the target processor, OCD methods can, in principle, be used to emulate software faults.

Tools that support test port-based fault injection include GOOFI (Generic Object Oriented Fault Injection) [Aidemark 2001], INERTE (Integrated NExus-based Real-Time fault injection tool for Embedded systems) [Yuste, 2003], Xception [Maia 2005] and HiPeFI (High Performance Fault Injection) [Miklo 2011]. GOOFI supports both JTAG-based and Nexus-based fault injection, while INERTE is specifically designed for Nexus-based fault injection. An environment for BDM-based fault injection is described in [Rebaudengo, 1999]. Most of these efforts, with the exception of [Miklo 2011] use commercial debugging tools to implement the fault injection capability. HiPeFi is based on high performance implementation of BDM on FPGA, thus allowing very fast modifications of memory and registers.

Injecting a fault via a test port involves four major steps:

(1) Setting a breakpoint via the test port and waiting for the program to reach the breakpoint;
(2) Reading the value of the target location (a register or memory word) via the test port;
(3) Manipulating this value and then writing the new, faulty value back to the target location;
(4) Resuming the program execution via a command sent to the test port.

The time overhead for injecting a fault depends on the speed of the test port. JTAG and BDM are low-speed ports, whereas Nexus ports can be of four different classes with different speeds. The simplest Nexus port (Class 1) is a JTAG port, which uses serial communication and therefore only needs 4 pins. Ports compliant with Nexus Class 2, Class 3 or Class 4 use separate input and output ports, known as auxiliary ports. These are parallel ports that use several pins for data transfer. The actual number of data pins is not fixed by the Nexus standard, but for Class 3 and Class 4 ports the standard recommends 4 data pins to 16 data pins for the auxiliary output port and 1 data pin to 4 data pins for the auxiliary input port.

The main advantage of test port-based fault injection is that faults can be injected internally in microprocessors without making any alterations to the system's hardware or software. This is crucial in digital I&C system testing because the alteration of the system HW and SW is not generally looked upon as acceptable in Factory Acceptance Testing or Site Acceptance Testing. Compared to software-implemented fault injection, it provides better capabilities of emulating real hardware faults. Finally, advanced Nexus ports (Class 3 and Class 4) provide outstanding possibilities for data collection and observing the impact of injected faults within a microprocessor. Existing tools have not fully exploited these possibilities. Hence, microprocessors with high-speed Nexus ports and BDM ports constitute interesting targets for the development of new fault injection tools, which potentially can achieve much better observability than existing tools. As such, as part of this research effort, this area was identified as a high value target for research to advance the state of the practice in fault injection for digital I&C systems (see Section 8.6).

Advantages

- Excellent controllability, which leads to high repeatability and reproducibility of fault injection experiments.
- Very low intrusiveness. No alteration to target system is required.
- Can inject faults into hidden registers of the target processor that are not visible to the programmer (e.g., pipeline status register).
- Can emulate a variety of fault types including permanent faults with certain caveats[2].
- Can provide a means to profile program behavior before fault injection to locate vulnerabilities. That is, which registers, memory locations, and variables are being used most frequently.
- Typically low cost implementation when using commercial debugging tools.

Disadvantages

- Target system must have an OCD standard test port such as JTAG, BDM, or Nexus. Not all digital I&C systems have these test ports.
- Using commercial debugging tools to implement OCD-based fault injection requires optimization of the debugger tool to implement fault injection. This usually involves programming the low APIs to improve the debugger tool performance.
- Efficiency of the fault injection can be low if faults are injected into processor locations where resources (e.g., registers, memory locations) are not being used. Pre-fault analysis of the target program addresses this problem.

[2] Permanent faults can be emulated with OCD methods, however, the repeated "triggering" on the memory or register usage can cause execution time delays in the processor under test. Significant time delay usually occurs after about 20 trigger events per scan cycle. It is rare to have one resource exercised 20 times in one execution cycle.

5.3.4.1. Software-implemented fault Injection

Software-implemented fault injection (SWIFI) encompasses techniques that inject faults through software executed on the target system. There are two approaches that can be used to emulate hardware faults by software:

- run-time injection
- pre run-time injection

In run-time injection, faults are injected while the target system executes its application or workload. This requires a mechanism that (1) stops the execution of the workload, (2) invokes a fault injection routine, and (3) restarts the workload. Thus, run-time injection can incur a significant run-time overhead if the implementation of the fault injection method is not optimized. In pre run-time injection, faults are introduced by manipulating either the source code or the binary image of the workload before it is loaded into memory. Pre run-time injection usually incurs less run-time overhead than run-time injection, but the total time for conducting a fault injection campaign is usually longer for pre run-time injection since it requires more time for preparing each fault injection experiment.

There are several fault injection tools that can emulate the effects of hardware faults by software, but they use different techniques for injecting faults and support different fault models. Most of these tools use run-time injection, since it provides better opportunities for emulating hardware faults than pre run-time injection. Pre-fault injection methods have been mainly adopted by the software testing community as a means to investigate the effects of various software defects or flaws on the reliability of the system [Smidts 2011; Voas 2000].

Software-implemented fault injection relies on the assumption that the effects of real hardware faults can be emulated either by manipulating the state of the target system registers and memory via run-time injection, or by modifying the target workload through pre run-time injection.

The validity of this approach varies depending of the fault type and where the fault occurs. Consider for example emulation of a soft error (i.e., a bit-flip error induced by a strike of a high energy particle). Flipping bits in main memory or processor registers can be done easily by software. On the other hand, the effect of a bit-flip in a processor's internal control logic can be difficult to emulate accurately by software manipulations of registers in the processor. This aspect also applies to OCD and test port fault injection methods as well. This is not a shortcoming of either fault injection method, but rather a *fault modeling issue* related to developing a fault model that can accurately represent the fault occurrences at the internal micro-controller and internal logic level of the processor where fault injection is applied. Accordingly, this was identified as another high value target area for investigation for research that was investigated in the latter stages of this research effort. This topic is discussed in more detail in Section 9 (fault model representation).

Emulating a permanent hardware fault requires a more elaborate set of manipulations than emulating a transient fault. For example, the emulation of a stuck-at fault in a memory word or a processor register would require a sequence of manipulations performed every time the designated word or register is read by a machine instruction. On the other hand, a transient fault requires only a single manipulation. The time overhead imposed by fault emulation thus varies for different fault types. Emulating the permanent fault model can impose significant time overheads by either SWIFI-based methods or OCD-based methods. For this reason, SWIFI or OCD based fault injection tools developed to date recommending the use of a fault model in their tools have not been identified. This as another justification for developing a portable

customized fault injector that can implement permanent fault models with minimal impact to system performance or timing.

There are eight tools capable of emulating hardware faults through software. These tools represent important steps in the development of software-implemented fault injection for emulation of hardware faults. The tools are FIAT [Barton, 1990], FERRARI [Kanawati 1992], FINE [Kao 1993], DEFINE [Kao, 1995], FTAPE [Tsai 1996], DOCTOR [Han 1995], Xception [Carreira 1998], MAFALDA [Arlat 2002] and Exhaustif [Dasilva 2007]. These tools use different approaches to emulate hardware faults and implement partly different fault models. Some also provide support for emulating software faults.

Researchers started to investigate software-implemented fault injection in the late 1980's. In the beginning, the focus was on developing techniques for emulating the effects of hardware faults. Work on emulation of software faults started a few years later.

One of the first tools that used software to emulate hardware faults was FIAT [Barton 1990], developed at Carnegie Mellon University. FIAT injected faults by corrupting either the code or the data area of a program's memory image during run-time. Three fault types were supported: zero-a-byte, set-a-byte and two-bit compensation. The last fault type involved complementing any 2 bits in a 32 bit word. Injection of single-bit errors was not considered, because the memory of the target system was protected by parity.

More advanced techniques for emulation of hardware faults were included in FERRARI [Kanawati 1992], developed at the University of Texas, and in FINE [Kao 1993], developed at the University of Illinois. Both these tools supported emulation of transient and permanent hardware faults in systems based on SPARC processors from Sun Microsystems. FERRARI could emulate three types of faults: address line, data line, and condition code faults; while FINE emulated faults in main memory, CPU-registers and the memory bus. DEFINE [Kao 1995], which was an extension of FINE, supported fault injection in distributed systems and introduced two new fault models for intermittent faults and communication faults.

DOCTOR [Han, 1995] is a fault injection tool developed at the University of Michigan targeting distributed real-time systems. It supports three fault types: memory faults, CPU faults and communication faults. The memory faults can affect a single-bit, two bits, one byte, and multiple bytes. The target bit(s)/byte(s) can be set, reset and toggled. The CPU faults emulate faults in processor registers, the op-code decoding unit, and the arithmetic logic unit. The communication faults can cause messages to be lost, altered, duplicated or delayed. DOCTOR can inject transient, intermittent and permanent faults, and uses run-time injection for the transient and intermittent faults. Permanent faults are emulated using pre run-time injection.

FTAPE [Tsai, 1996] is a fault injector aimed at benchmarking fault tolerant commercial systems. It was used to assess and test several prototypes of fault tolerant computers for on-line transaction processing. FTAPE emulates the effects of hardware faults in the CPU, main memory and I/O units. The CPU faults include single and multiple bit-flips and zero/set registers in CPU registers. The memory faults include single and multiple bit and zero/set faults in main memory. The I/O faults include SCSI and disk faults. FTAPE was developed at the University of Illinois in cooperation with Tandem Computers.

Xception [Carreira, 1998] is a fault injection tool developed at the University of Coimbra, Portugal. This tool also uses the test ports for OCD to inject faults. Thus it injects faults in a way that is similar to test port-based fault injection. The difference is that Xception controls the setting of breakpoints and performs the fault injections via software executed on the target processor rather than sending commands to a test port.

Xception injects faults through exception handlers executing in kernel mode, which can be triggered by the following events: op-code fetch from a specified address, operand load from a specified address, operand store to a specified address, and a specified time passed since start-up. These triggers can be used to inject both permanent and transient faults. Xception can emulate hardware faults in various functional units of the processors such as the integer unit, floating point unit, and the address bus. It can also emulate memory faults, including stuck-at-zero, stuck-at-one and bit-flip faults. Xception is unique because it is one of very few academic tools that has been commercialized. Xception is sold by Critical Software, Coimbra, which released the first commercial version of the tool in 1999.

MAFALDA [Arlat 2002] is a tool for assessing commercial off-the-shelf microkernels. It uses software-implemented fault injection to inject single or multiple bit-flips in the code and data segments of the microkernel under assessment. In addition, MAFALDA also allows corruption of input parameters during invocation of kernel system calls, and thus supports robustness testing of microkernels.

A more recent commercial tool, similar in functionality to Xception, is Exhaustif [Dasilva, 2007]. It instruments the workload with a software component that injects faults at run-time. This component is configured through a communication interface (e.g.,, serial port, Ethernet) using a graphical user interface. It supports several fault models based on corruption of processor registers and memory, and interception of function calls. The software component that injects faults on the target system requires several kilobytes for code and data, which may be significant in terms of intrusiveness.

Advantages

- Lowest cost for all physical fault injection implementation methods.
- Excellent controllability, which leads to high repeatability and reproducibility of experiments.
- Several commercial versions are available, for example, Xception.

Disadvantages

- Requires the target software to be modified to incorporate exception handlers for fault injection. This may be unacceptable in certain applications especially with regards to digital I&C systems.
- Requires a “free” port on the target system so the host computer can communicate with the exception handler modules in the target system software.
- Can only reach resources that are visible to the programmer. These are typically processor registers, memory registers, and special purpose I/O mapped registers.

5.4. Simulation-Based Fault Injection

As mentioned in the introduction, simulation-based fault injection can be performed at different levels of abstraction, such as the device level, logical level, function block level, ISA level, and system level. Simulation models at different abstraction layers are often combined in so called mix-mode simulations to overcome limitations imposed by the time overhead incurred by detailed simulations. FOCUS [Choi 1992] is an example of a simulation environment that

combines device-level and gate-level simulation for fault sensitivity analysis of circuit designs with respect to soft errors.

At the logic level and the function block level, circuits are usually described in a hardware description language (HDL) such as Verilog or VHDL. Several tools have been developed that support automated fault injection experiments with HDL models (e.g., MEFISTO [Jenn 1994] and the tool described in [Delong 1996]). There are several different methods for implementing the fault injection process, such as modifying the HDL code [Assaf 2004], modifying the HDL simulator, commanding the simulator through scripts or, in a more recent example [Das 2006], using the force and release constructs in Verilog to emulate stuck-at faults.

Recently, several studies assessing the soft error vulnerability of complex high-performance processors have been conducted using simulation-based fault injection. In [Wang 2006] a novel low-cost approach for tolerating soft errors in the execution core of a high-performance processor is evaluated by combining simulations in a detailed Verilog model with an ISA-level simulator. This approach allowed the authors to study the impact soft errors for seven SPEC2000 integer benchmarks through simulation.

DEPEND [Goswami 1997] is a tool for simulation-based fault injection at the functional level aimed at evaluating architectures of fault-tolerant computers. A simulation model in DEPEND consists of a number of interconnected modules, or components, such as CPUs, communication channels, disks, software systems, and memory. DEPEND is intended for validating system architectures in early design phases and serves as a complement to probabilistic modeling techniques such as Markov and Petri net models. DEPEND provides the user with predefined components and fault models, but also allows the user to create new components and new fault models (e.g., the user can use any probability distribution for the time to failure for a component).

Virtualized System Fault Injection is new method that is gaining acceptance. This is based on a new simulation and modeling technique called *full system simulation or virtualization* [Bastien 2004]. Full-system simulation combines fast instruction-set simulators of the target digital I&C system with accurate models of all components in the physical hardware to form a virtual machine of the target I&C system. A *virtual machine* is a software implementation of a machine (i.e. a computer) that executes programs identical to a physical machine. Thus, the software experiences a virtual computer system that is functionally identical to the physical system, capable of running the same unmodified binaries, including device drivers, operating systems, protocol stacks, and applications.

Full system simulation or virtualized simulation is a relatively new simulation technology, becoming feasible on desktop workstations only in the past six or seven years. This technology has benefited significantly from virtual machine technology software (e.g., VMware), and as such new opportunities to exploit virtual machine breakthroughs for accurately modeling I&C systems are becoming available. Tools such as SIMICS [River 2011], Simplescaler [Austin 2002], and Processor Technology Laboratory Simulator (PTLsim) [Yourst 2007] are a few of the virtualized simulators currently being used. SIMICS is by far the most widely used and powerful. SIMICs is a commercial tool, Simplescaler and PTLsim are open-source. Basic fault injection is built into the SIMICs toolset. The research effort described in this report built upon the basic capabilities of the SIMICs fault injection by providing saboteur modules that allow a tester to insert fault injection modules into hardware, software, and firmware of a target system [Bastien 2004; Sekhar 2008]. A description of SIMICS can be found at http://www.virtutech.com/.

Advantages

- Controllability and observability of the target system are very high because the target system is being simulated in a controlled environment.
- Repeatability and reproducibility are both high due to high controllability.
- Intrusiveness is low because the target system is in a simulation environment.
- Typically supports a larger set of fault models than physical fault injection.
- Complements physical fault injection by enabling fault injection into specific sub-systems or components of the target I&C system that would be difficult to reach with physical-based fault injection.

Disadvantages

- Model fidelity or system representativeness is the most significant issue with simulation based methods. High fidelity models of the system require a significant amount of time to develop and can be a significant cost factor.
- Simulating complete digital I&C systems to a high degree of fidelity is very challenging and in some cases infeasible. The exception to this case is the commercial SIMICs tool set, which enables the possibility of full system model development. Even then it could be a significant effort to model a digital I&C system in SIMICs with the complexity of the benchmark systems used in this research.

5.4.1. Hardware Emulated Fault Injection

The advent of large FPGA circuits has provided new opportunities for conducting model-based fault injection with hardware circuits. One of the main reasons for the increasing popularity of FPGAs is that they provide many of the advantages of both hardware and software components. Being a hardware component, an FPGA is efficient compared to software, and the rapid development of the technology results in growing capacities. At the same time, an FPGA-based application can be designed, developed, and configured by the end user, that is to say, by the developers themselves, and this provides for flexibility that could earlier only be achieved by using a software component. This is a new phenomenon, a hardware device that has development properties and attributes like software.

Circuits designed in a HDL are usually tested and verified using software simulation. Even if a powerful computer is used in such simulations, it may take considerable time to verify and test a complex circuit adequately. To speed up the test and verification process, techniques have been developed where HDL-designs are tested by hardware emulation in a large FPGA circuit. This technique also provides excellent opportunities for conducting fault injection experiments. Hardware emulation-based fault injection has all the advantages of simulation-based fault injection such as high controllability and high repeatability, but requires less time for conducting a fault injection experiment compared to using software simulation.

The use of hardware emulation for studying the impact of faults was first proposed in [Kwang-Ting 1999]. The authors of that paper used the method for fault simulation, i.e., for assessing the fault coverage of test patterns used in production testing.

Fault injection can be performed in hardware emulation models through compile time reconfiguration and run-time reconfiguration. Reconfiguration refers to the process of adding hardware structures to a model that are necessary to perform the experiments. In compile-time reconfiguration, these hardware structures are added by instrumentation of the HDL models. An approach for compile-time instrumentation for injection of single event upsets (soft errors) is described in [Civera 2003]. This work presents different instrumentation techniques that allow injection of transient faults in sequential memory elements as well as in microprocessor-based systems.

One disadvantage of compile-time reconfiguration is that the circuit must be re-synthesized for each reconfiguration, which can impose a severe overhead on the time it takes to conduct a fault injection campaign. In order to avoid re-synthesizing the target circuit, a technique for run-time reconfiguration is proposed in [Antoni 2003]. This technique relies on directly modifying the bit-stream that is used to program the FPGA-circuit. By exploiting partial reconfiguration capabilities available in some FPGA circuits, this technique achieved substantial time-savings compared to other emulation-based approaches to fault injection.

A tool for conducting hardware emulation-based fault injection called FADES is presented in [Andrés 2006; Andrés 2008]. This tool uses run-time configuration and can inject several different types of transient faults, including bit-flips, pulse, and delay faults, as wells as faults that cause digital signals to assume voltage levels between “1” and “0”.

Even though FPGA-based techniques overcome the performance issues present in software simulation, it is often difficult to obtain ideal observability due to the communication required for observing the behavior of emulated circuits. A recently proposed method [Ejlali 2007] reduces this overhead by having a single combinational circuit for both the faulty circuit and the fault-free circuit. The complete circuit repeatedly executes one clock cycle with the fault-free flip-flops followed by one clock cycle with the faulty flip-flops, and the output is multiplexed to a comparator, in order to identify which faults cause errors on the target. This method provides good observability under the bit-flip fault model and avoids duplicating the entire combinational circuit, which is assumed to be unaffected by faults.

There is a concern with representativeness when using hardware emulated circuits, rather than the actual hardware. In [Ramachandran 2008], the results of fault injection on a hardware-emulated IBM POWER6 processor (the authors call it “hardware accelerated simulation”) are compared to radiation-based fault injection. The results show a close match between the two techniques, thereby providing evidence in favor of hardware emulation-based fault injection.

Advantages

- Speeds up the fault injection process by emulating the hardware on FPGA devices.
- All the advantages of simulation based fault injection, but speeds up the fault injection process by several orders of magnitude.
- Excellent controllability and repeatability of experiments.
- Most suitable for sub-system or components of a target digital I&C system that are difficult to reach with other physical based fault injection methods, specifically, FPGA type devices.

Disadvantages

- If compile-time fault injection is used the FPGA must be re-synthesized for each fault injection campaign, which can impose a severe overhead on the time it takes to conduct a fault injection campaign. A typical re-compile time for a FPGA can be from several minutes to hours.

- System or model representativeness can be an issue when using FPGAs. The processor model structure that is downloaded into a FPGA is not structurally equivalent to the real processor structure. While they are supposed to be functionally equivalent, in some cases this is not guaranteed [Lenhart 2007].

5.5. Hybrid Fault Injection

Hybrid approaches to fault injection combine several fault injection techniques to improve the accuracy and scope of the verification or the assessmentof a target system. An approach for combining software-implemented emulation of hardware faults and simulation-based fault injection is presented in [Guthoff 1995]. In this approach, the physical target is run until the program execution hits a fault injection trigger, which causes the physical system to halt. The architected state of the physical system is then transferred to the simulation model, in which a fault is injected (e.g., in the non-visible parts of the micro-architecture). The simulator is run until the effects of the fault have stabilized in the architected state of the simulated processor. This state is then transferred back to the physical system, which subsequently is restarted so that the system-level effects of the fault can be determined.

An extension of the FERRARI tool which allows it to control a hardware fault injector is described in [Kanawati 1995]. The hardware fault injector can inject logic-0/logic-1 faults into the memory bus lines of a SPARC 1 workstation. The authors used the hardware fault injector to study the sensitivity of the computer in different operational modes. The results showed that the system was more likely to crash from bus faults when the processor operated in kernel mode, compared to when it operated in user mode. This study showed that it is feasible to extend a tool for software-implemented fault injection with other techniques at reasonable cost, since many of the central functions of a tool are independent of the injection technique.

A more recent tool that supports the use of different fault injection techniques is NFTAPE [Stott 2000], developed at the University of Illinois. This tool is aimed at injecting faults in distributed systems using a technique called Light Weight Fault Injectors (LWFIs). The purpose of the LWFI is to separate the implementation of the fault injector from the rest of the tool. NFTAPE provides a standardized interface for the LWFIs, which simplifies the integration and use of different types of fault injectors. NFTAPE has been used with several types of fault injectors using hardware-implemented, software-implemented, and simulation-based fault injection.

Advantages

- Can support different techniques of fault injection to provide better reachability and scope of the fault injection process.

- Provides an environment where the advantages of one fault injection technique can offset the disadvantages of another fault injection technique.

Disadvantage

- Integration of the different methods into a single fault injection framework can be challenging.

5.6. Novel Methods

Novel Methods refer to fault injection methods that don't fit cleanly into a specific category. Recently (and the most notable) are fault injection methods that incorporate fault injection capability with formal verification techniques, which combines the advantages of both methods into a single method or technique. These new methods are often called *symbolic fault injection*.

This section briefly describes formal methods since it may not be well known to the reader. *Formal methods* are a particular kind of mathematically-based techniques for the specification, development, and verification of software and hardware systems [Gupta 2004]. In practice, two types of formal methods are used today: automated theorem proving and model checking. Automated theorem proving deals with the development of computer programs that show that some statement (the *conjecture*) is a *logical consequence* of a set of statements (the *axioms* and *hypotheses*). It is primarily used to create provable correct programs and specifications of systems. On the other hand, *Model checking* is a technique for automatically verifying the correctness properties of *finite-state* system models [Baier 2008]. It tests automatically whether a model of a system satisfies a given specification. Typically, the systems one has in mind are hardware or software systems, and the specification contains safety requirements expressed in *temporal logic*. Model checking is by far the most widely used of the two techniques.

In [Pattabiraman 2008], a symbolic fault injection framework (called SymPLIFIED) for verifying error detecting mechanisms in programs and systems using symbolic execution and model checking is described. The goal of the framework is to expose error cases that could potentially escape detection and cause program failure. The focus is on transient hardware errors. The framework consists of three models to perform symbolic fault injection: (1) a formal *execution model* to represent programs expressed in a generic assembly language, and reasons about the effects of errors originating in hardware and propagating to the application without assuming specific detection mechanisms; (2) an *error detection model* that specifies the semantics of general error detectors; and (3) an *error model* that represents errors using a single symbol, thereby merging multiple error values into a single symbolic value in the program. This includes single and multi-bit errors in the register file, main memory, cache, as well as errors in computation. The advantage of this framework is that one can perform exhaustive fault injection over the entire state space of a program/system. The power to perform such exhaustive fault injection comes from the model checking capabilities of the tool. However, because the framework uses a model of the program or system, it is subject to model fidelity issues. Nonetheless, this type of fault injection is very promising.

Another recent development is the development of Assertion-based Verification Fault Injection (ABVFI). ABVFI is an emerging technique that involves the use of formal methods in the analysis of fault coverage for a system or components of a system. ABVFI extends an emerging technique called assertion-based verification (ABV), a variant of model checking, to mathematically analyze design behavior in the presence of faults. Given a defined fault model, this method considers all possible combinations of faults across both time (when the fault occurs) and space (where the fault occurs), for a complete analysis of fault coverage.

ABVFI replaces simulation-based fault injection with formal verification engines, such as model checking, in a fault injection campaign. Like simulation-based fault injection, formal verification-based fault injection provides high controllability and observability and can model many types of

faults. However, the advantages of using formal methods are that the analysis performs an exhaustive search of the input and fault space, ensuring that no corner case fault goes uncovered, and in less time than an exhaustive analysis with simulation-based fault injection.

One benefit of model checking is that when a property is violated, a counter example is produced describing a scenario that led to the property's failure. The counter example can then be used as an aid in determining the vulnerabilities of a design. ABVFI counter-example results can also be helpful at other stages of system verification. A compilation of uncovered faults and their error behaviors can be used to feed a fault injection analysis at a higher system level. The benefit is that the system level fault list generation can be limited to the known failure scenarios of the low-level components.

In [Bingham 2009] an ABVFI methodology was developed that allows fault injection to be performed at the register transfer level of a design.

The novel fault injection methods presented in this section are on the cutting edge of system verification and validation research. Specifically, techniques like those described in this section are a means to guide physical and simulation based fault injections to investigate high impact faults and corner cases where a system is vulnerable.

5.7. Characterization of Fault Injection Techniques for Digital I&C Systems

This section briefly summarizes important issues that must be taken into account when selecting a fault injection technique. This section takes the information learned from the survey and the experiential knowledge gained from applying fault injections to previous systems. The purpose is to characterize fault injection techniques to better inform the reader on the applicability of specific fault injection techniques for digital I&C systems. In addition to fault representativeness (i.e., the plausibility of the supported fault model with respect to actual faults), which is one concern that is often raised in conjunction with fault injection experiments and for which some objective insights were provided in the previous Section, a wide range of criteria can be considered to assess the merits of the fault injection techniques.

Without any claim of a comprehensive analysis, the following basic properties mentioned in Section 5.1 are a means to characterize the fault injection techniques discussed in this Section:

- Reachability and observability
- Controllability, with respect to space and time,
- Repeatability (with respect to experiments),
- Reproducibility (with respect to results),
- Non-intrusiveness,
- Time measurement (e.g., error detection latency),
- Efficiency - generating significant experiments (minimizing no-response experiments).
- System representativeness

A characterization of the considered fault injection techniques based on these basic properties is shown in Table 5-1 and is explained in this section. For each property, the techniques are graded on a scale of *none, low, medium, and high*. It is worth noting that, although it is quite generic in scope, this analysis also builds upon insights gained during the previous fault injection experiments.

The observability property is the *reachability property* attached to a fault injection technique, which is defined as the ability to reach possible fault locations in an IC that implements the target system.

Controllability is defined with respect to both the space and time domains. The space domain relates to the ability to control which of the reachable fault locations are actually injected. The time domain corresponds to controlling the instant when faults are injected and the duration for which the faults are injected. Thus time has two attributes: (1) instance, and (2) persistence.

Repeatability refers to the ability to repeat experiments exactly or with a very high degree of accuracy. This property is highly desirable, particularly when the aim of the experiments is to remove potential design or implementation faults in the fault tolerance mechanisms. Repeatability requires a high degree of controllability in both the space and the time domains.

Reproducibility refers to the ability to reproduce results statistically for a given set-up. Reproducibility of results is an absolute requirement to ensure the credibility of fault injection experiments. Repeatability normally implies reproducibility; if an experiment can be controlled exactly, then it is always possible to reproduce the same results. However, reproducibility can be achieved without repeatability.

Intrusiveness relates to the property of avoiding or minimizing any undesired impact of fault injection on the behavior of the target system in both time and space. Time intrusiveness refers to altering the timing behavior of the system. Space intrusiveness refers to altering the system functionality of the target system by augmenting the target system with additional hardware and/or software to accommodate the fault injection technique.
Time measurement refers to the acquisition of timing information associated with the monitored events (e.g., measurement of error detection latency), which is an important outcome of fault injection experiments. Time measurement allows one to determine the effects of a fault in relation to the target system real-time operation. For instance, a fault can be detected and mitigated by the system, however the error detection and mitigation response may be too late with respect to the controlled plant dynamics. When this occurs the system has a *dynamic system failure*.

Efficiency refers to the ability to produce a high proportion of meaningful results from a fault injection campaign. The type of efficiency considered here concerns the testing power offered by the techniques (i.e., their ability to produce a limited number of non-significant experiments). A non-significant experiment occurs, for example, when a fault is injected into a hardware or software component which is not accessed or used by the workload executed during the experiment. Efficiency is also closely associated with fault list generation methods for some fault injection techniques (e.g., SWIFI and OCD methods).

System representativeness refers to how accurately a target system represents the real system. In actuality, system representativeness is more a property of the target system and not fault injection per se. However, since fault injection is largely divided along physical-based techniques and simulation-based techniques, it is important to include this attribute to reinforce the fact that simulation-based approaches always use models of the system that are subject to fidelity issues.

Implementation complexity indicates the extent of skills, resources, and special equipment needed to realize the fault injection technique. A low implementation capability is something within the reach of a small team of mid-level, experienced software and hardware engineers. The level of effort is one to two staff years over a six month timeframe. A high level implementation capability requires a more experienced team of specialized hardware and

software engineers with significant resources. The Level of effort is on the order of two to four staff years over a six month time frame.

Table 5-1 Characterization of basic fault injection techniques.

Properties	Physical-based Fault Injection				Simulation-based Fault Injection				Hybrid methods	Novel Methods
	Pin Level	Power Supply	Radiation	SWIFI	OCD-based FI	Emulated Hardware	SIMICs	VHDL	All	Symbolic Fault Injection
Reachability	Low to Med	Low	Med to High	Med	Med to High	Med	High	High	High	Very High
Controllability (Space)	High	Med	Low	High	High	Med to High	High	High	High	High
Controllability (Time)	Med	None to Low	None	Med	High	Med	High	High	High	High
Repeatability	High	None to Low	None to Low	High	High	High	High	High	Med	High
Reproducibility	High	Low	None to Low	High	High	High	High	High	High	High
Intrusiveness (Space)	None	None	None	Low to Med	None	Low to Med	Low	Low	Low to Med	Low to Med
Intrusiveness (Time)	Low	None	None	Low to Med	Very Low to Med	Low to Med	None	None	Low to Med	None
Time Measurement	Med to High	Low	Low	Med	Med	Med	High	High	Med	High
Efficiency	Med to High	Med to High	Med to High	Low to Med	Med to High	Med	Med to High	Med to High	Med to High	Very High
System Representa-tiveness	Real	Real	Real	Real	Real	Med to High	High	Med	High	Med
Implementation Complexity	Med to High	Med	Very High	Low to Med	Med	Med to High	Low	Low to Med	Very High	Probably Med to High

5.8. Discussion of the Comparisons

The discussion of the comparison between the fault injection methods is restricted to physical based fault injection methods since the aim of the research was to apply physical-based fault injection on target benchmark systems.

Pin level fault injection (see Table 5-2) has several advantages worth noting. First, it has high degree of repeatability and reproducibility, meaning that statistically significant results can be achieved from the method. Secondly, it has very low intrusiveness to the system operation, which is desirable. However, the reachability aspects of the technique are fairly limited, especially when trying to emulate faults that occur within packaged ICs. In addition, the complexity of implementing pin level fault injection can be challenging when confronted by today's highly dense integrated IC packages and small pin pitch sizing. Pin level fault injection is more of specialized technique that tends to complement other fault injection techniques.

Power supply fault disturbance injection is another specialized technique. It can produce fault effects that are unique to it alone, and for this reason it cannot be entirely dismissed. However, the low repeatability and reproducibility of the technique is a significant shortcoming as a general fault injection technique.

Radiation based fault injection has the ability to reach deep inside ICs to perturb locations that are hard to reach (e.g., pipeline micro-control units) by any other physical fault injection methods. The technique usually requires placing the target system in a special ionizing radiation test chamber, which can be very expensive and requires a special license to operate. Repeatability is low as is reproducibility.

Software implemented fault injection is one the more popular methods used in recent years. It has high repeatability and reproducibility attributes, good controllability, and is probably the easiest fault injection method to implement. However, it requires modifying the target system software so that low level exception handlers can be added to the target system. In some cases this not an issue, in other cases it is difficult to implement these exception handlers with the OS and system level source code of the target, such as in commercial digital I&C systems. In addition, the ability to emulate permanent faults is somewhat limited.

The OCD-based fault injection method addresses some of the shortcomings of SWIFI-based methods. The method does not require modifying the system software or the system hardware. Since the OCD fault injection method interfaces to the processor debug port, it only requires a commercial debugger tool or a specially designed fault injector built around the debug port protocol. Of all the physical fault injection techniques surveyed and compared, OCD methods appear to offer the best opportunity to achieve repeatable, credible fault injection results from digital I&C systems. However, taking advantage of OCD fault injection methods requires the onboard microprocessors and other ICs of the digital I&C system to have OCD capability. Older microprocessor technology (prior to 1998) does not have OCD capability. However, almost all contemporary microprocessors built in last 10 years have some if not full OCD capability.

Hybrid techniques that are integrated into a single fault injection framework offer the promise of having a "tool box" of techniques from which to choose. The only significant challenge with this approach is the development of such an integrated environment for general purpose fault injection. Several fault injection tools have already started moving in this direction, namely GOOFI and Xception. In the research effort described in this report the "tool box" approach was adopted for the fault injection environment for the benchmark systems.

In summary, while some techniques stand out (e.g., OCD , SWIFI and hybrid techniques), it should be noted that some techniques complement others to produce a set of fault injection methods that broaden the scope of fault injection capabilities for digital I&C systems. For example, pin level and power supply disturbance fault injections tend to emulate faults that are difficult to produce with OCD and SWIFI methods. The tool box approach to fault injection is advocated in this report, recognizing that not one specific technique will provide the scope needed for full system fault injections of digital I&C systems.

5.9. References

[Dasilva 2007] A. Dasilva, J.-F. Martinez, L. Lopez, A.-B. Garcia, L. Redondo. "Exhaustif: A Fault Injection Tool for Distributed Heterogeneous Embedded Systems." *Proceedings of the 2007 Euro American Conference on Telematics and Inforamtion Systems.* 2007.

[Dixit 2009] A. Dixit, R. Heald, A. Wood. "Trends from Ten Years of Soft Error Experimentation." *5th IEEE Workshop on Silicon Errors in Logic-Systems Effects (SELSE-5).* Stanford, CA: IEEE, 2009.

[Ejlali 2008] A. Ejlali, G. Miremadi. "Error Propagation Analysis Using FPGA-Based SEU-Fault Injection." *Microelectronics Reliability*, 2008: 319-328.

[Rajabzadeh 2004] A. Rajabzadeh, S.G. Miremadi, M. Mohandespour. "Experimental Evaluation of Master/Checker Architecture Using Power Supply and Software-Based Fault Injection." *10th IEEE International On-Line Testing Symposium.* Funchal, Madeira Island, Portugal: IEEE, 2004. 239-244.

[Ando 2008] Ando, H., R. Kan, and K. Takahis, K. Hatanaka Y. Tosaka. "Validation of Hardware Error Recovery Mechanisms for the SPARC64 V Microprocessor." *IEEE International Conference on Dependable Systems and Networks*, 2008: 62-69.

[Arlat 2002] Arlat, J, and Y Crouzet. "Faultload representativeness for dependability benchmarking." 2002. 29-30.

[Arlat 2003] Arlat, J, Y Crouzet, J Karlsson, P Folkesson, E Fuchs, and G H Leber. "Comparison of Physical and Software-Implemented Fault Injection Techniques." *IEEE Transactions on Computers*, no. 52 (2003): 1115-1133.

[Arlat 1990(a)] Arlat, J., M. Aguera, Y. Crouzet, J.-C. Fabre, E. Martins, and D. Powell. "Experimental evaluation of the fault tolerance of an atomic multicast system." *#IEEE_J_R#* 39, no. 4 (1990(a)): 455-467.

[Ashenden 1995] Ashenden, P. *The Designer's Guide to VHDL.* Morgan Kaufman, 1995.

[Association 2011] Association, IEEE Standard. *Systems and Software Engineering-Architecture Description.* March 1, 2011. http://www.iso-architecture.org/ieee-1471/docs/ISO-IEC-IEEE-latest-draft-42010.pdf (accessed 2011).

[Bastien 2004] B. Bastien, B.W. Johnson. *A Technique for Performing Fault Injection Using Simics.* UVA-CSCS-SFI-001, Charlottesville: University of Virginia, 2004.

[Baier 2008] Baier, C. *Principles of Model Checking.* MIT Press, 2008.

[Bhasker 1999] Bhasker, J. *A SytemC Primer, 2nd edition.* Star Galaxing Publishing, 2004.

[Kwang-Ting 1999] C. Kwang-Ting, H. Shi-Yu, d. Wei-Jin. "Fault Emulation: A New Methodology for Fault Grading." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1999: 1487-1495.

[Smidts 2011] C. Smidts, Y. Shi, M. Li, W. Kong, J. Dai. *A Large Scale Validation of a Methodology for Assessing Software Reliability.* NUREG/CR-7042, U.S. NRC, 2011.

[Communications 2011] Communications, Personal. "iSystems Embedded Development Tools." *www.isystem.com/.* 2011.

[Constantinescu 2005(a)] Constantinescu, C. "Neutron SER Characterization of Microprocessors." *Proceedings of IEEE Dependable Systems and Networks Conference.* IEEE, 2005(a). 754-759.

[Andres 2008] D. D. Andres, J.C. Ruiz, D. Gil, P. Gil. "Fault Emulation for Dependability Evaluation of VLSI Systems." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2008.

[Andres 2006] D.D. Andres, J.C. Ruiz, D. Gil, P. Gil. "Run-Time Reconfiguration for Emulating Transient Faults in VLSI Systems." *International Conference on Dependable Systems and Networks.* 2006. 291-300.

[Stott 2000] D.T. Stott, B. Floering, D. Burke, Z. Kalbarczyk, R.K. Iyer. "Nftape: A Framework for Assessing Dependability in Distributed Systems with Lightweight Fault Injectors." IEEE, 2000. IEEE International Computer Performance and Dependability Symposium (IPDS 2000).

[DeLong 1996] DeLong, T A, B W Johnson, and J A Profeta. "A Fault Injection Technique for VHDL Behavioral-Level Models." 1996. 24-33.

[Jenn 1994] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, J. Karlsson. "Fault Injection into VHDL Models: The Mefisto Tool." *24th International Symposium on Fault-Tolerant Computing.* Pasadena, CA, 1994. 66-75.

[Evans 2000] Evans, D. "Safety in Avionics: EMI: A Serial Killer." *Avionics Magazine*, November 1, 2000.

[Vargas 2005] F. Vargas, D.L. Cavalcante, E. Gatti, D. Prestes, D. Lupi. "On the Proposition of an EMI-Based Fault Injection Approach." *11th IEEE International On-Line Testing Symposium (IOLTS 2005).* Stevenson, Washington: IEEE, 2005. 207-208.

[Kanawati 1995(b)] G. Kanawati, N. Kanawati, J. Abraham. "FERRARI: A Flexible Software-Based Fault and Error Injection System." *IEEE Transactions Computers*, 1995(b).

[Miremadi 1995] G. Miremadi, J. Torin. "Evaluating Processor-Behavior and Three Error-Detection Mechanisms Using Physical Fault-Injection." *IEEE Transactions on Reliability*, 1995: 441-454.

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

[Choi 1992] G.S. Choi, R.K. Iyer. "Focus: An Experimental Environment for Fault Sensitivity Analysis." *IEEE Transactions on Computers*, 1992: 1515-1526.

[George 2011] George, N. *Modeling and Mitigating Transient Faults in Nanoscale Digital Systems.* Charlottesville, VA: University of Virginia, Ph.D. Thesis, 2011.

[Goswami 1997] Goswami, K. "Depend: A Simulation-Based Environment for System Level Dependability Analysis." *IEEE Transactions on Computers*, 1997: 60-74.

[Gupta 2004] Gupta, R. *Formal Methods and Models for System Design .* Klewer, 2004.

[Aidemark 2001] J. Aidemark, J. Vinter, P. Folkesson, J. Karlsson. "GOOFI: Generic Object-Oriented Fault Injection Tool." *IEEE/IFIP International Conference on Dependable Systems and Networks.* Goteborg, Sweden: IEEE, 2001. 83-88.

[Carreira 1998] J. Carreira, H. Madeira, J.G. Silva. "Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers." *IEEE Transactions on Software Engineering*, 1998: 125-136.

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

[Karlsson 1991] J. Karlsson, U. Gunneflo, P. Liden, J. Torin. "Two Fault Injection Techniques for Test of Fault Handling Mechanisms." *International Test Conference.* Nashville, TN, 1991. 140-149.

[Voas 1998] J. M. Voas, G. McGraw. *Software Fault Injection: Inoculating Programs Against Errors.* New York, NY: Wiley, 1998.

[Barton 1990] J.H. Barton, E.W. Czeck, Z.Z. Segall, D.P. Siewiorek. "Fault Injection Experiments Using Fiat." *IEEE Transactions on Computers*, 1990: 575-582.

[Kellington 2007] J.W. Kellington, R. McBeth, P. Sanda, R.N. Kalla. "IBM Power6 Processor Soft Error Tolerance Analysis Using Proton Irradiation." *3rd IEEE Workshop on Silicon Errors in Logic-Systems Effects (SELSE-3).* Austin, TX: IEEE, 2007.

[Pattabiraman 2008] K. Pattabiraman, N. Nakka, Z. Kalbarczyk, R. Iyer. "Simplified: Symbolic Program-Level Fault Injection and Error Detection Framework." *IEEE International Conference on Dependable Systems and Networks with FTCS and DCC.* DSN 2008, 2008. 472-481.

[Antoni 2003] L. Antoni, R. Leveugle, B. Feher. "Using Run-Time Reconfiguration for Fault Injection Applications." *IEEE Transactions on Instrumentation and Measurement*, 2003: 1468-1473.

[Lenhart 2007] Lenhart, T. *A Formalized Verification Methodology for Soft IP Cores Used in Safety-Critical Hardware Applications.* Charlottesville, VA: University of Virginia, M.S. Thesis, 2007.

[Assaf 2004] M. Assaf, S. Das, E. Petriu, L. Lin, C. Jin, D. Biswas, V. Groza, M. Sahinoglu. "Hardware and Software Co-Design in space Compaction of Digital Circuits." *IEEE Instrumentation Measurement Techniques Conference.* 2004. 1472-1477.

[Miklo 2011] M. Miko, C. Elks, R. Williams. "Design of a High Performance FPGA Based Fault Injector for Real-Time Safety-Critical." *22nd IEEE International Conference on Application Specific Systems, Architectures and Processors.* Santa Monica, CA, 2011.

[Rebaudengo 1999] M. Rebaudengo, M. Sonza Reorda. "Evaluating the Fault Tolerance Capabilities of Embedded Systems via Bdm." *17th IEEE VLSI Test Symposium.* IEEE, 1999. 452-457.

[Madeira 2000] Madeira, H, D Costa, and M Vieira. "On the emulation of software faults by software fault injection." *Proceedings International Conference on Dependable Systems and Networks.* New York, NY: IEEE, 2000. 417-426.

[Maia 2005] Maia, R, L Henriques, R Barbosa, D Costa, and H Madeira. "Xception fault injection and robustness testing framework: a case-study of testing." 2005.

[Meyer 2009] Meyer, Ray. *Electromagnetics and Sensors Branch Facilities.* 2009. electromagnetics.larc.nasa.gov/facilities/hirf.htm (accessed 2011).

[George 2010(b)] N. George, C. Elks, B. Johnson, J. Lach. "Spatial Multi-Bit Soft-Error Tolerance in Logic." *Dependable Systems and Networks Symposium.* Chicago, IL, 2010(b).

[Wang 2006] N.J. Wang, S.J. Patel. "Restore: Symptom-Based Soft Error Detection in Microprocessors." *IEEE Transactions on Dependable and Secure Computing*, 2006: 188-201.

[Organization 2011] Organization, IEEE Industry Standards and Technology. *Nexus 5001 Forum Standard.* 2003. http://www.nexus5001.org/standard (accessed 2011).

[Civera 2003] P. Civera, L. Macchiarulo, M. Rebaudengo, m. Sonza Reorda, M. Violante. "New Techniques for Efficiently Assessing Reliability of SOCS." *Microelectronics Journal*, 2003: 53-61.

[Ramachandran 2008] P. Ramachandran, P. Kudva, J. Kellington, J. Schumann, P. Sanda. "Statistical Fault Injection." *38th IEEE/IFIP Internation Conference on Dependable Systems Networks (DSN 2008).* IEEE, 2008. 122-127.

[Tummeltshammer 2009] P. Tummeltshammer, An Steininger. "On the Role of the Power Supply as an Entry for Common Cause Faults." *12th International Symposium on Design and Diagnostics of Electronic Circuits & Systems.* Liberec, Czech Republic: IEEE, 2009. 152-157.

[Yuste 2003] P. Yuste, D. deAndres, L. Lemus, J. Serrano, P. Gil. "Inerte: Integrated Nexus-Based Real-Time Fault Injection Tool for Embedded Systems." *International Conference on Dependable Systems and Networks.* Sanf Francisco, CA, 2003. 669-669.

[River 2011] River, Wind. *Wind River Simics.* 2011., http://www.windriver.com/products/simics/ (accessed 2011).

[Bingham 2009] S. Bingham, J. Lach. "Enhanced Fault Coverage Analysis Using ABVFI." *Workshop on Dependable and Secure Nanocomputing.* 2009.

[Han 1995] S. Han, K.G. Shin, H.A. Rosenberg. "Doctor: An Integrated Software Fault Injection Environment for Distributed Real-Time Systems." *International Computer Performance and Dependability Symposium.* Erlangen, Germany, 1995. 204-113.

[Das 2006] S.R. Das, S. Mukherjee, E.M. Petriu, M.H. Assaf, M. Sahinoglu, W.-B. Jone. "An Improved Fault Simulation Approach Based on Verilog with Application to ISCAS Benchmark Circuits." *Instrumentation and Measurement Technology Conference (IMTC 2006).* 2006. 24-27.

[Sekhar 2008] Sekhar, M. *Generating Fault Lists for Efficient Fault Injection into Processor Based I&C Systems.* Charlottesville, VA: University of Virginia, 2008.

[Austin 2002] T. Austin, E. Larson, D. Ernst. "SimpleScaler: An Infrastructure for Computer System Modeling." *IEEE Computer* , 2002.

[Tsai 1996] T.K. Tsai, R.K. Iyer, D. Jewitt. "An Approach Towards Benchmarking of Fault-Tolerant Commercial Systems." *26th International Symposium on Fault Tolerant Computing.* 1996. 314-323.

[Gunneflo 1989] U. Gunneflo, J. Karlsson, J. Torin. "Evalusation of Error Detection Schemes Using Fault Injection by Heavy-Ion Radiation." *19th International Symposium on Fault-Tolerant Computnig (FTCS-19).* Chicago, IL, 1989. 340-347.

[Kao 1995] W.-l. Kao, R.K. Iyer. "Define: A Distributed Fault Injection and Monitoring Environment." In *Fault-Tolerant Parallel and Distributed Systems*, by D. Avresky, eds. D. Pradhan, 252-259. IEEE Computer Society, 1995.

[Kao 1993] W.-l. Kao, R.K. Iyer, D. Tang. "Fine: A Fault Injection and Monitoring Environment for Tracing the Unix System Behavior Under Faults." *IEEE Transactions on Software Engineering*, 1993: 1105-1118.

[Yourst 2007] Yourst, M. T. "PTLSim: A Cycle Accurate Full System X86-64 Microarchitectural Simulator." *IEEE International Symposium on Performance and Analysis of Systems and Software.* 2007.

6. REACTOR PROTECTION SYSTEMS AND DESCRIPTION OF THE BENCHMARK SYSTEMS

6.1. Introduction

The RPS is one the primary defense in depth systems of a commercial nuclear power plant. The function of the RPS is to monitor measured process parameters associated with the safe operation of the reactor, to shut-down the reactor to prevent damage to the reactor system or the fuel cladding, and to prevent the reactor coolant system from exceeding its design limits. The RPS is the most important safety system in the nuclear power plant. It is designed to meet very high reliability and safety requirements without operator intervention.

6.2. Reactor Protection Overview

A typical RPS is a fairly complex system, monitoring many reactor and plant process variables. The following discussion is a simplified explanation [Westinghouse Electric Co. 2003] of a RPS. The RPS shuts down the nuclear fission process in a nuclear reactor core by removing electrical power from the Control Rod Drive Mechanisms (CRDMs), thereby releasing reactor control rods and allowing them to fall into the reactor core. The control rods contain materials that absorb thermal neutrons, which prevents the neutrons from causing the uranium fuel to fission, thereby stopping the fission process in the reactor core.

A RPS is typically comprised of 4 independent channels or divisions. The sensor data from the reactor is sent to the RPS trip logic which is designed to trip the reactor if any measured sensor value is outside of the safe operating limit of the reactor. Each RPS channel monitors several critical parameters of the reactor and plant. Upon a condition in the reactor system requiring a reactor trip to be initiated, the RPS sends four redundant trip signals to the CRDM circuit breakers. A combination of any two trip signals from the four redundant channels to the trip breakers will cause the CRDMs to release the control rods. All of the monitored sensor parameters are normally energized when the plant or reactor is operating within normal limits. If the terminating sensor relay de-energizes for any reason, it sends a trip signal to the RPS trip logic. Any two channels or divisions sending a trip signal will cause the RPS trip logic to send a trip signal to the CRDM.

Historically, the RPS logic and processing divisions have been entirely based on 1970's analog processing technology. The trend to replace obsolete analog-based RPS processing divisions has gained traction in recent years with the advent of highly reliable processor-based digital I&C systems [Hashemian 2011]. The benchmark systems in this research effort are simulation processor-based digital I&C safety systems.

Referring to Figure 6-1, a digital I&C-based RPS takes the monitored analog sensors values from the reactor and converts them to digital representations through an analog to digital conversion module. The digital sensor representations are then provide as input data to the onboard processing functions, which forward the data to the RPS trip logic, which is a real time task executing on the processor of the digital I&C system. The RPS trip logic performs logic calculations and comparisons to determine whether the reactor is within its safe operating limits. If it is not, the RPS trip Logic issue a trip command to be sent to the output modules of the digital I&C system. The output modules convert the trip command to a trip signal (either digital or analog) to be sent to the CRDMs. Listed in Table 6-1 are typical signals or sensor values monitored by the RPS.

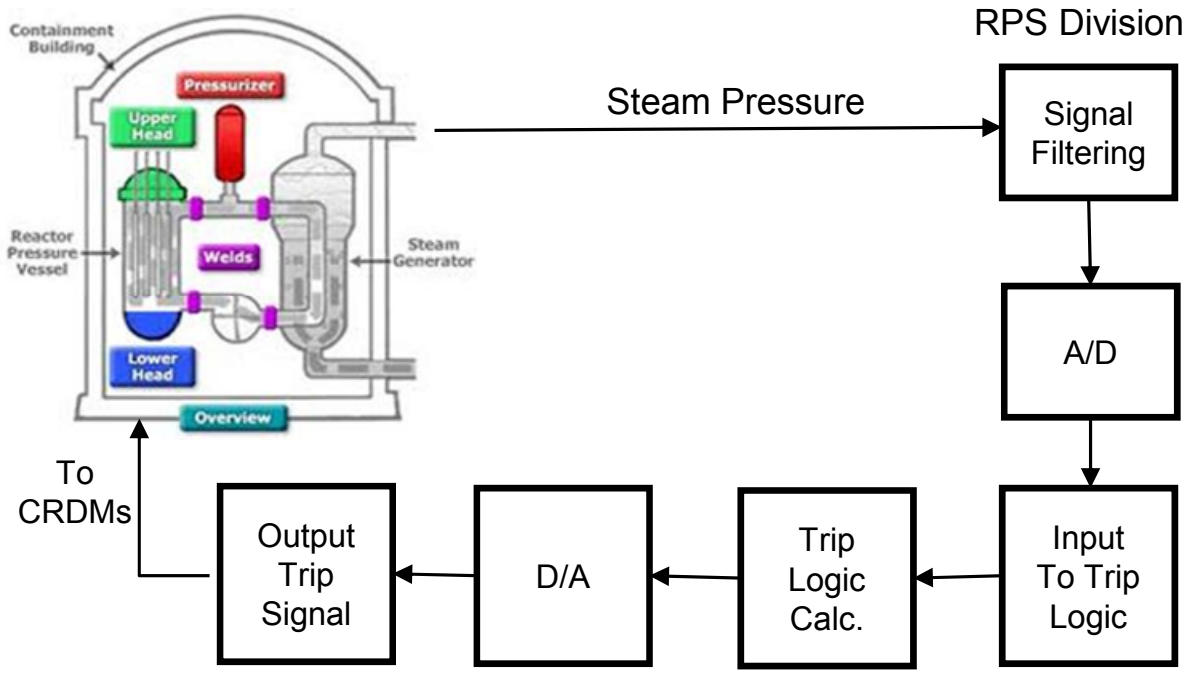

Figure 6-1 Typical RPS operation

Table 6-1 Typical signals or sensor values monitored by RPS.

Trip Signal	Description	Condition
Variable Overpower	The Variable Overpower Trip provides a reactor trip when the indicated neutron flux increases at a rate greater than a predetermined value for a sufficient time period requiring over power protection or a high preset value is reached.	The purpose of this trip is to prevent fuel damage caused by an over power event.
High Logarithmic Power Level	The high logarithmic power level trip initiates a reactor trip when indicated neutron flux power reaches a preset high value.	The purpose of this trip is to ensure the integrity of the fuel cladding and coolant system boundary in the event of unplanned criticality from a shutdown condition.
High Local Power Density	The high Local Power Density (LPD) trip initiates a reactor trip when the calculated core local power density in the fuel assembly reaches a predetermined value (setpoint).	The purpose of this trip is to prevent the linear heat rate in the core from exceeding the fuel design limit.
High Pressurizer Pressure	The high pressurizer pressure trip initiates a reactor trip when measured pressurizer pressure reaches a high preset value.	The purpose of this trip is to help assure the integrity of the Reactor Coolant Pressure Boundary for design basis events (e.g.,, feedwater line breaks. etc...) that can lead to an over-pressurization of the Reactor Coolant System.

Table 6-1 Typical signals or sensor values monitored by RPS.

Trip Signal	Description	Condition
Low Pressurizer Pressure	The low pressurizer pressure trip initiates a reactor trip when measured pressurizer pressure falls to a low preset value.	The purpose of this trip is to assist the Engineered Safety Features System in the event of a coolant accident and to provide a reactor trip in the event of reduction in pressurizer pressure.
Low Steam Generator Water Level	The low steam generator water level trip initiates a reactor trip when the measured water level in a steam generator region falls to a low preset value.	The purpose of this trip is to assist the Engineered Safety Features System by assuring that there is sufficient time for actuating the auxiliary feedwater pumps to remove decay heat from the reactor in the event of a reduction of steam generator water inventory.
High Steam Generator Water level	The high steam generator water level trip initiates a reactor trip when the measured water level in a steam generator region reaches a high preset value.	The purpose of this trip is to assist the Engineered Safety Features System by assuring that there is sufficient time for actuating the auxiliary feedwater pumps to remove decay heat from the reactor in the event of a reduction of steam generator water inventory.
Low Steam Generator Pressure	The low steam generator pressure trip initiates a reactor trip when measured steam generator secondary pressure falls to a low preset value.	The purpose of this trip is to provide protection against excess secondary heat removal events assumed in the Plant Safety to assist the Engineered Safety Features System in limiting the consequences of a feedwater or steam line rupture accident
Low Reactor Coolant Flow	The low reactor coolant flow trip initiates a reactor trip when the measured steam generator DELTA P across the primary side of either steam generator decreases at a rate great enough to require loss of flow protection or reaches a low preset value.	The purpose of this trip is to limit the consequences of a sheared reactor coolant pump shaft and steam line break.
Manual Trip	A manual reactor trip is provided permit the operator to trip the reactor.	

6.3. Description of the Benchmark Systems

6.3.1. Introduction

The following sections describe the hardware, software, and dependability features of the benchmark systems. The following documents were provided by the system developer and were used to describe the systems in this section.

Benchmark System I:

- Training Manuals
- Demo/Test System User Manual

- Demo/Test System Drawing Set
- Digital Input Modules manual
- Analog Input Modules manual
- Digital Output Modules manual
- Analog Output Modules manual
- Software Specification and Coding Environment Manual.
- Service Unit manual
- X-Bus Communications manual
- Processing Module manual
- Communication module manual
- NRC Demo/Test System Application Software Code and Hardware Parameters Listing

Benchmark System II:

- Overview of the System Operations
- Training Manuals

Benchmark System II Hardware User Manuals

- Planning & Installation Guide
- User's Manual for Field Terminations

Benchmark System II Manuals for Communication

- Network Planning & Installation
- Intelligent Communication Modules
- Sequence of Events User's Manual

Benchmark System II Programmer Manuals plus CD

- Getting Started Guide
- Users Guide
- Benchmark System II Libraries

6.3.2. Benchmark System I

Benchmark System I is a safety grade pre-qualified digital I&C system specifically developed for safety and high reliability functions in nuclear facilities. The benchmark system provided by the NRC for this study is a scaled version of a typical four- division RPS. Due to non-disclosure and proprietary agreements, the make and model of the target system are not disclosed in this report. The salient features of the target system are its ability to be adaptable to plant-specific requirements with varying degrees of redundancy. Its scalability permits development of solutions for a spectrum of safety-related tasks within the NPP system. Typical applications include RPS and Engineered Safety Features Actuation System (ESFAS) functions.

It should be noted that the benchmark systems used in this effort were testing platforms to exercise the fault injection methodology and software dependability methodology developed by this research. In that regard, the benchmark systems represent the complexity of RPS processing and fault tolerance from both a hardware and software perspective. Typical in-plant RPS digital I&C systems are considerably more enhanced in their fault tolerance and diversity attributes than the representative benchmark systems used in this study. Therefore, results of this study are intended to be a reflection on the ability of the methodology to accommodate fault

injection on digital I&C systems, and not be construed as representative of the performance and suitability of the benchmark systems for RPS applications.

6.3.3. Architecture and System Description of Benchmark System I

6.3.3.1. Overview

Figure 6-2 shows the architecture of the Benchmark System I. The system consists of four separate processors, each acting as a processing channel or division for the RPS application. Data exchange via fiber-optic bus systems distributes information such as sensor values, fault messages, and process parameter messages to each processing channel. The bus protocol for the data exchange network is an IEC standard Supervisory Control and Data Acquisition (SCADA) protocol, which will be referred to as X-bus.

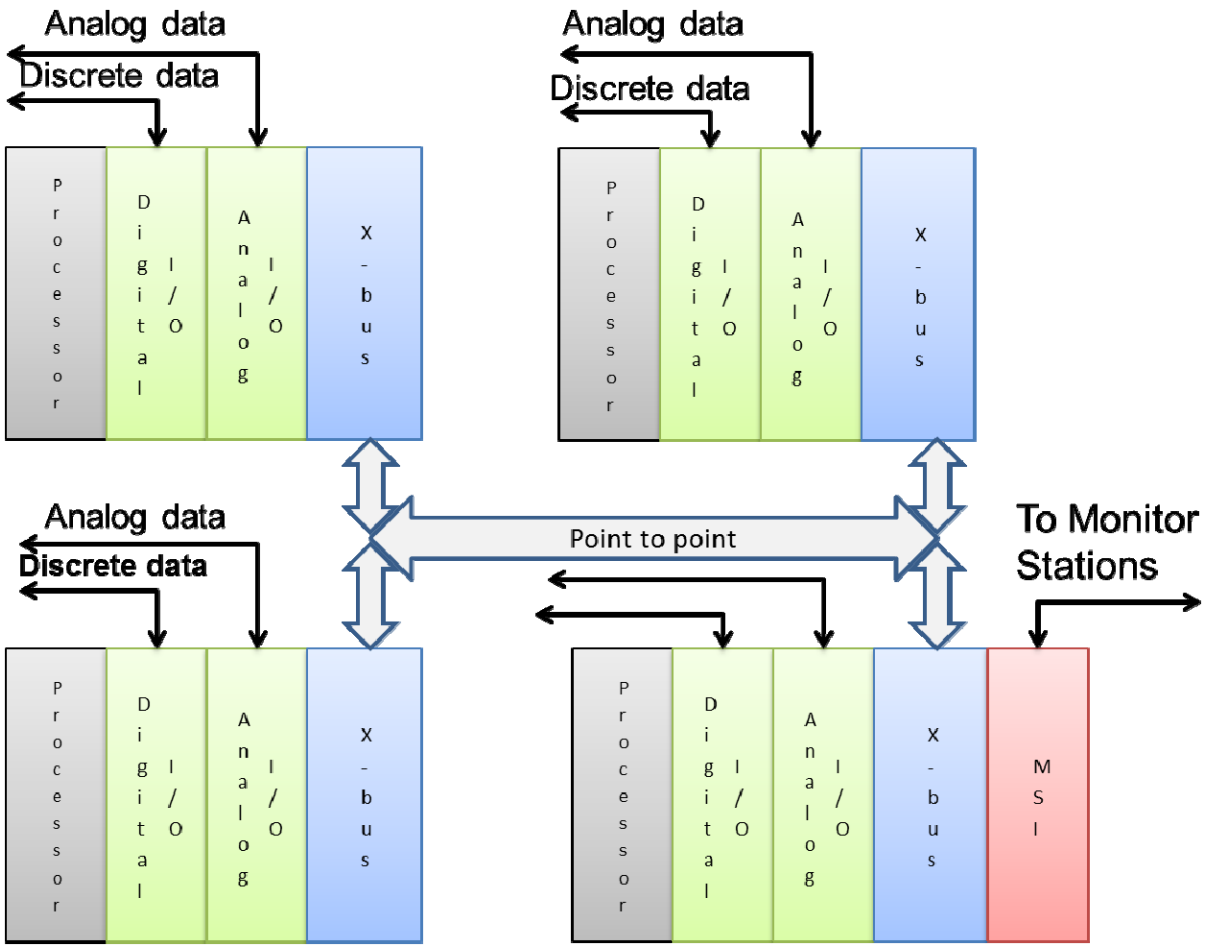

Figure 6-2 Benchmark System I detailed architecture

The inter-channel X-bus network is usually configured as a point-to-point topology, but can be configured as a linear bus or a ring topology. In the configuration for this research effort, it was configured as a point-to-point network. Communication between channels on X-bus is deterministically upper bounded by the synchronous circulating token nature of the X-bus protocol, meaning there is an upper bound for message delivery between processing channels. However, the Benchmark System I as a whole was not clock synchronized among the processing channels. That is, the processing channels operated independently and asynchronously from each other in their execution of a task. The Runtime Executive (RE)

operating system environment operates as a deterministic static scheduler for application tasks with several prioritized rate groups. All processing within a rate group is bounded by the cycle time of the rate group. Because of the repetitive cyclic nature of the processing, the execution time skew between processing channels is bounded.

Each channel typically has its own I/O processing. This includes multiple modules of digital input, output, analog input, and analog output. Fault masking features for the I/O sub-systems include detection of invalid signals due to known failure modes to improve the availability of the safety I&C functions. These fault masking features of the system include majority voting schemes (typically 2-out-of-4 for binary signals and 2nd minimum/2nd maximum selection for analog signals).

6.3.3.2. Fault Tolerance Features

The fault tolerance features of Benchmark System I are both application independent and application dependent. Depending on the degree of redundancy needed for an application, the user can configure a system as an *n* out of *m* voting scheme, where *n* is the number of channels that are in agreement with all other channels, and *m* are the total number of channels in the system. In addition to these application dependent fault tolerance features, the system executes a number of application independent fault detection mechanisms, such as runtime diagnostics and self-tests in the background of the RE to detect latent faults in the system.

6.3.3.3. Monitoring Interface

External communications to non-safety monitoring stations for purposes of monitoring the operation of the application (e.g., RPS) is facilitated by a special interface called the Monitor Interface (MI). The MI is responsible for gathering system level diagnostic health messages and application level messages from the system and forwarding this information to plant and operator monitoring stations. The MI serves as boundary between the safety functions and the non-safety functions. The MI and the messaging protocol is designed to be non-interfering with respect to the safety functions operating on the benchmark system.

6.3.3.4. RPS Configuration

In a RPS configuration, the system is configured as a two out of four voting system. This means that if any two channels indicate that any of the measured sensor variables from the reactor system are out of their safety range, the reactor trip logic in the RPS will initiate a shutdown command and signal to the reactor CRDM assembly. If a channel becomes faulty and it is detected as so, the RPS gracefully degrades to a two out of three voting scheme to allow continued operation in a limited capacity while maintenance and service can perform off-line diagnostics and repair of the failed channel.

Figure 6-3 shows the basic processing flow for one channel or division of a RPS. Each channel gathers a number of reactor sensor variables (see Table 6-1 to monitor the safe operation of the reactor system. Each of these sensor values is supplied to the RPS by four independent sensors. These values are acquired by the analog input modules, and are distributed to all other channels (e.g., channels A, B, C, and D). The analog sensor values are preprocessed by a 2nd min/max selection function to bound the influence of any outlier sensor values. The 2nd max function uses the second highest value recorded from the group of sensor readings. The 2nd min function uses the second lowest reading from a group of sensors. The output of the 2nd min/max function is then provided to a set point comparison function, which compares the conditioned sensor values to maximum and minimum set points for safe operation of the reactor system. The output of the set point function is a Boolean – Reactor trip or no-trip. The outputs of the set point functions are then sent to a set of reactor trip breaker relays configured in a two

out of four voter configuration. If two or more trip indications have been noted, the relay breakers will interrupt power to the CRDMs, thereby causing the reactor control rods to drop into the reactor core.

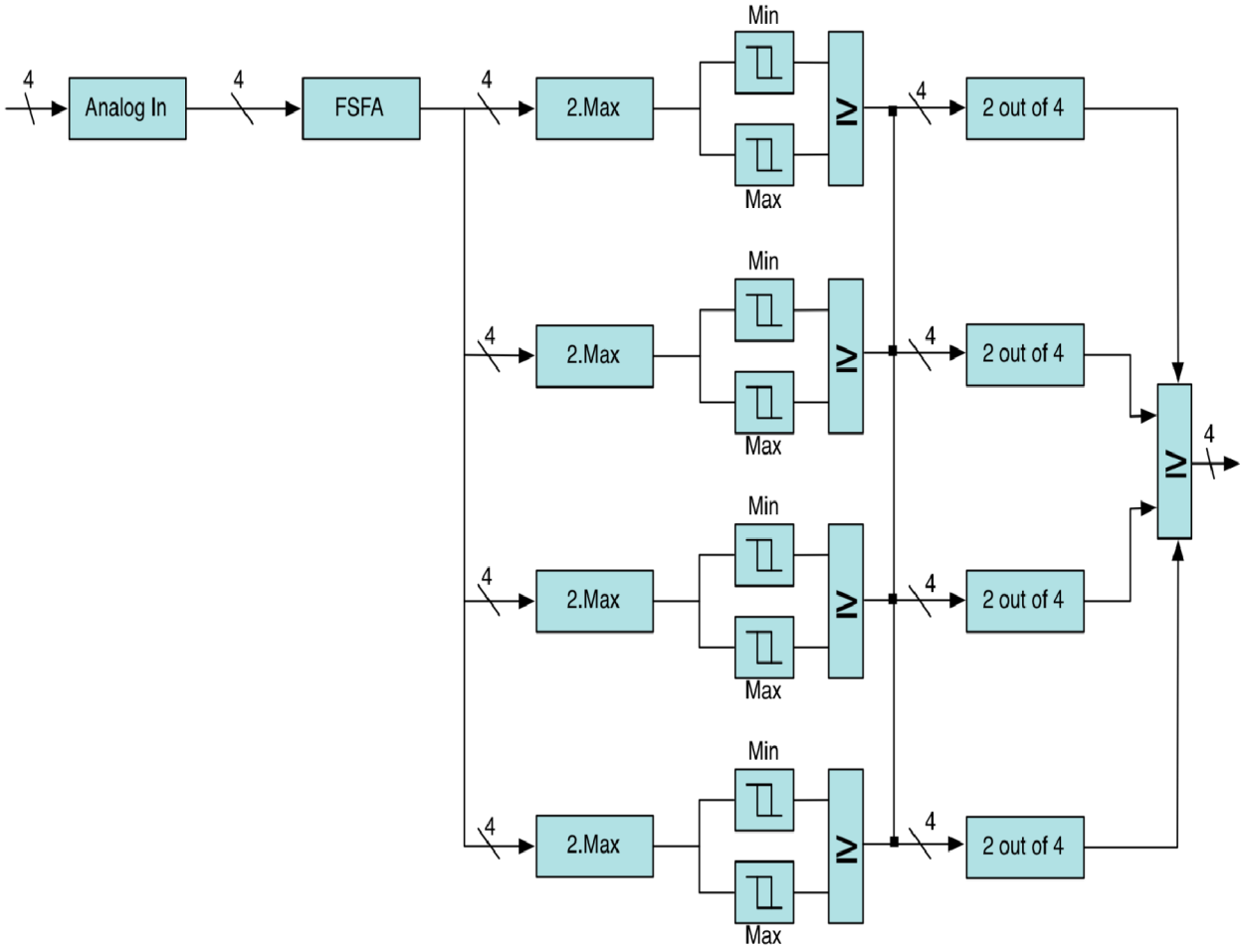

Figure 6-3 RPS diagram for Benchmark System I

In some safety systems, the output trip signals of the safety I&C functions are distributed to an additional voting process, which consists of two computers each running as master/checker pair. The benchmark system used in this research did not have this additional fault tolerance capability.

6.3.3.5. Software

The Benchmark System I software consists of (1) off-line software development and (2) on-line software to support task reliable execution and a fault tolerance capability. The off-line code development environment designs applications from a function diagram editor. Function diagrams are developed by selecting and connecting appropriate function block modules available from a function block library. For each processing module the application software code is compiled and auto generated from this specification (function diagram modules) and then linked to the RE system software resulting in a set of real-time tasks for the application.

The interface between application functions (function diagram modules) and the system software is generated by the software development code generator tool. It automatically

creates the call and data interface to the function diagram modules and describes the I/O and communication activities that are to be performed by the processing module.

The design of the processing cycle is one of the key preconditions for ensuring deterministic system behavior by maintaining strictly cyclic operation of each processor in a distributed I&C system independent of the status of the plant process. Each processing unit runs three tasks, which are scheduled by the RE on the basis of task priorities:

- Priority 1 (highest) – Cycle task: The cycle task operates with a predefined, constant cycle time. It handles all communication messages, the input and output signals, and the cyclic processing of the application safety functions. Having the highest priority of all three tasks, it ensures that the cyclic operation of the application safety functions are completed within the specified cycle time.

- Priority 2 – Service task: The service task processes service commands received from the Service Unit. When no service commands are pending it is suspended. It is reactivated by the cycle task each time a new service message has been received and the cycle tasks have been completed for that cycle. After processing of the service message, the service task is suspended again. There are two types of service requests that can be received from the service unit:

    - Type 1: Service requests are fulfilled without interrupting cyclic processing, such as reading and acknowledging system messages, tracing signal data, and on-line modification of operation parameters.

    - Type 2: Requests for diagnostic data for fault diagnosis of the CPU or the performance of tests that require the processor to be in the special operation modes TEST or DIAGNOSIS.

- Priority 3 (lowest) – Self-test task: The self-test task has the lowest priority of all tasks and is only processed when the service task and the cycle task are not active. The self-test task continuously performs tests of all relevant hardware components on the processor board (random access memory (RAM)-test, read only memory checksums, watchdog tests, etc.). This endless loop of tests consumes all “idle” time of the processor.

Through the above separation of functionality approach, the interactions between the application, RE, and system services level is more structured compared to combining all of the functionality within a single task.

6.4. Benchmark System II

6.4.1. Introduction

Benchmark System II is also a safety grade pre-qualified digital I&C system specifically developed for safety or high reliability functions in nuclear facilities. Like Benchmark System I, Benchmark System II is a scaled version of a typical four-division RPS. Due to non-disclosure and proprietary agreements the make and model of the target system cannot be disclosed. The salient features of the target system are its ability to be adaptable to plant-specific requirements, with varying degrees of redundancy. Typical applications include RPS and ESFAS functions.

6.4.2. Architecture and System Description

The Benchmark System II configuration consists of two chassis assemblies. As shown in Figure 6-4, each chassis includes termination panels, power supply modules, three main processor modules, redundant I/O modules, and communication modules. Each chassis is powered by two independent, redundant power supplies, each capable of providing the full power requirements of the chassis.

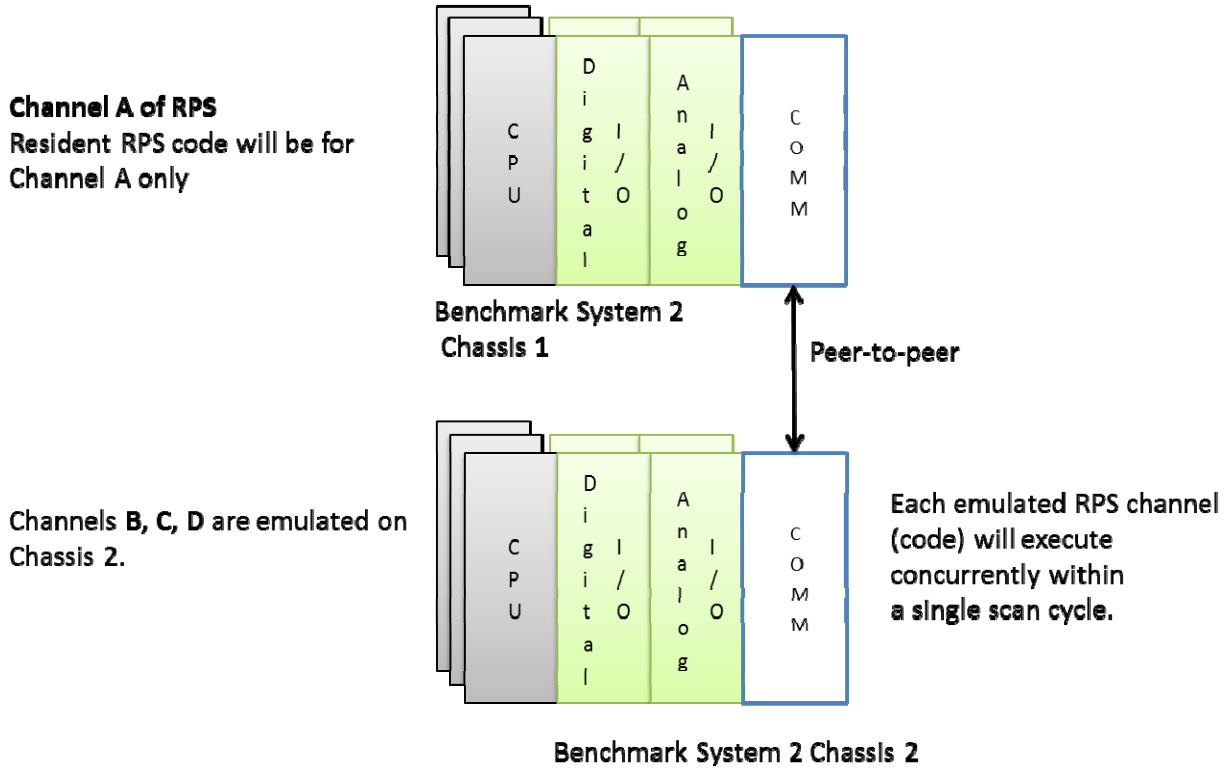

Figure 6-4 Benchmark System II architecture

In an in-plant RPS configuration there would be four full chassis assemblies for each channel or division. In Benchmark System II only two chassis assemblies were used. Therefore, one channel or division of the RPS was hosted on Chassis 1 (i.e., channel A), and the other three channels (i.e., B, C, and D) were hosted on chassis 2. Each emulated RPS division on chassis 2 was run as a separate task in the RE with appropriate independent I/O and inter-channel communication.

Each channel in the configuration is triple modular redundant (TMR) from input terminal to output terminal, as shown in Figure 6-5. The TMR architecture allows continued system operation in the presence of any single point of failure within the system. The TMR architecture also allows the Benchmark System II to detect and correct individual faults on-line, without interruption of monitoring, control, and protection capabilities. In the presence of a fault, the Benchmark System II will alarm the condition, remove the affected portion of the faulted module from operation, and continue to function normally in a dual redundant mode. The system returns to the triple redundant mode of operation when the affected module is replaced. Thus, the system acts as fault masking architecture with no active reconfiguration.

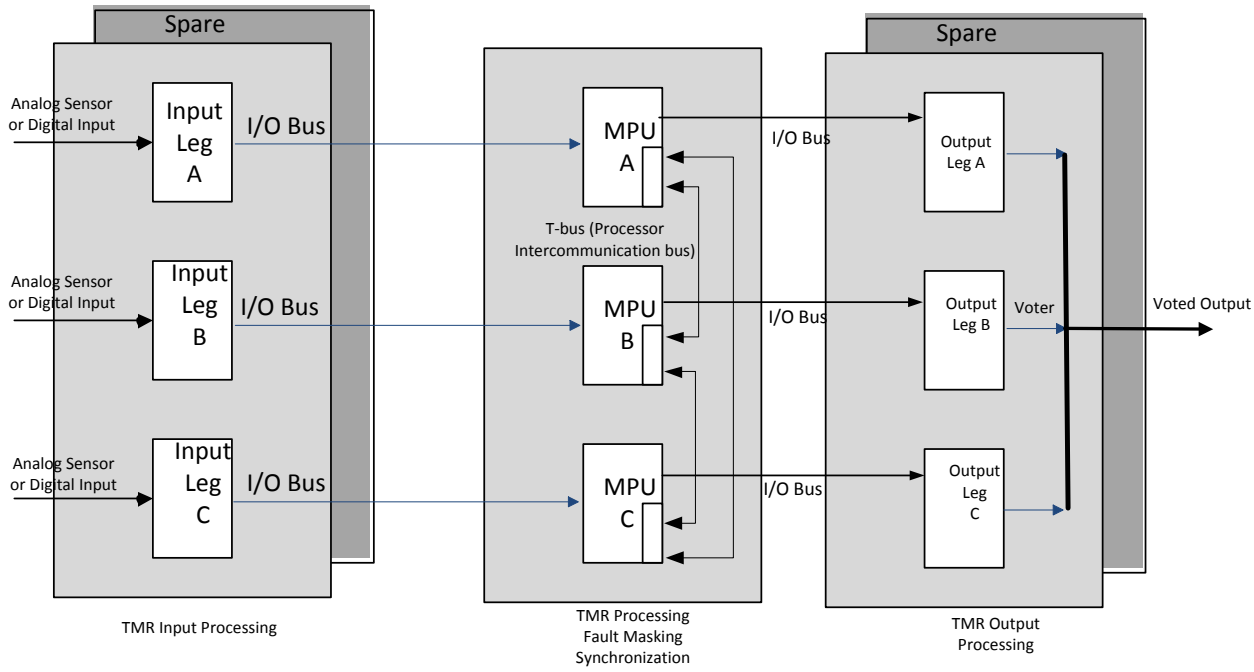

Figure 6-5 Benchmark System II detailed architecture

To facilitate module replacement, the Benchmark System II chassis includes provisions for a spare module, logically paired with a single input or output module. This design allows on-line, hot replacement of any module under power while the system is running, with no impact on the operation of the application.

Figure 6-5 also shows the arrangement of the input, main processor (MP), main processor units (MPUs), and output modules. As shown, each input and output module includes three separate and independent input or output paths or legs. These legs communicate independently with the three main processor modules. The legs receive signals from common field input termination points. The microprocessor in each leg continually polls the input points, and constantly updates a private input data table in each leg's local memory. E ach leg performs signal conditioning, isolation, and processing independently. The input modules possess sufficient leg-to-leg isolation and independence so that a component failure in one leg will not affect the signal processing in the other two legs.

Each MPU operates independently with no shared clocks, power regulators, or circuitry. Each MPU owns and controls one of the three signal processing legs in the system, and each leg contains two 32-bit processors. One of the 32-bit processors is a dedicated, leg-specific I/O and communication microprocessor that processes all communication with the system I/O modules and communication module. The second 32-bit primary processor manages execution of the application program (RPS) and all system diagnostics at the main processor module level.

6.4.3. Fault Tolerance Operation

Benchmark System II is a TMR fault tolerant system with a fully distributed voting scheme. That is, the voting for all inputs and outputs per chassis is conducted by three voters; each voter is associated with each leg of operation (e.g., MPU A, MPU B, and MPU C). In addition, all operations between the legs are synchronized by a distributed fault tolerant clock that is *1-f* fault tolerant. That is, it is capable of tolerating one arbitrary single fault per leg.

For digital inputs, the voted input table is formed by a two out of three majority vote on respective inputs across the three data tables. The voting scheme is designed for de-energize-to-trip applications, always defaulting to the de-energized state unless voted otherwise. Any single leg failure or corrupted signal feeding a main processor module is corrected or compensated at the main processor module level when the voted data table is formed.

For analog inputs, a mid-value selection algorithm chooses an analog input signal representation in the voted input table. The algorithm selects the median of the three signal values representing a particular input point for representation in the voted input tables. Any single leg failure or corrupted signal feeding a main processor is compensated at the main processor module level when the voted data table is formed. Significant errors are alarmed. All input and output modules include self-diagnostic features designed to detect single failures within the module.

Like the I/O modules, the communication modules have three separate communication busses and three separate communication bus interfaces, one for each of the three main processors. Unlike the I/O modules, however, the three communication bus interfaces are merged into a single microprocessor. That microprocessor votes on the communications messages from the three main processors and transfers only one of the messages to an attached device or external system. If two-way communications are enabled, messages received from the attached device are triplicated and provided to the three main processors.

The communication paths to external systems have appropriate levels of Cyclic Redundancy Checks (CRCs), handshaking, and other protocol-based features. These features are supported in hardware and firmware. Firmware provides core functionality common to all the communication modules with additional coding to support the specific communication protocol.

The main processor diagnostics monitor the health of each main processor as well as each I/O module and communication channel. The main processor modules process diagnostic data recorded locally and data received from the input module level diagnostics in order to make decisions about the health of the input modules in the system.

If a standby module is installed in the paired slot with a faulty module, and that module is itself deemed healthy by the main processors, the system automatically switches over to the standby unit and takes the faulty module off line. If no standby unit is in place, the faulty module continues to operate on two of the three legs and protection and control is unaffected.

6.4.4. Runtime Software Operation

The operating system, run-time library, and fault analysis for the main processor is fully contained in the flash memory on each module. The main processors communicate with one another through a high speed, voting, bi-directional serial channel called T-BUS. Each main processor has an I/O channel for communicating with one of the three legs of each I/O module. Each main processor has an independent clock circuit and selection mechanism that enables all three main processors to synchronize their operations each scan cycle to allow voting of data and exchange of diagnostic information.

The principle part of the OS is the RE. The RE is a static deterministic scheduler that schedules and executes the real-time tasks on a pre-determined cycle time (called a scan) to ensure deterministic operation of the tasks. At the beginning of each scan, each primary processor takes a snapshot of the input data table in dual port RAM and transmits the snap shots to the other main modules over the T-BUS. This transfer is synchronized using the T-Clock. Each module independently forms a voted input table based on respective input data points across the three snapshot data tables. If a main processor module receives corrupted data or loses

communication with a neighbor, the local table representing that respective leg data will default to the de-energized state.

The primary processors for each channel then execute the application program in parallel on the voted input table data and produce an output table of values in dual port RAM. The voting schemes explained above for analog and digital input data ensure the process control programs are executed on the same input data value representations. After the main processors complete the control algorithm, data is sent out to the output modules. Outputs from the main processors are provided to the I/O bus microprocessors through dual port RAM. The I/O bus microprocessors then transfer that data to the triplicated microprocessors on the output modules. The output modules then set the output hardware appropriately on each of the triplicated sections and vote on the appropriate state and/or verify correct operation.

The transmission of data between the main processor modules and the output modules is performed over the I/O data bus using a master/slave communication protocol. The system uses a CRC to ensure the health of data transmitted up to three times. Watchdog timers on each output module leg ensure communication has been maintained with its respective main processor module with a certain timeout period.

6.4.5. RPS Configuration for Benchmark System II

The RPS design for Benchmark System II is slightly different from that of Benchmark System I. The system is still configured as a teo out of four voting system. The principle difference is that RPS input sensor signals do not get a 2nd min/max treatment as in Benchmark System I because the 2nd min/max functionality is specific to Benchmark System I. Benchmark System II uses a mid-point select algorithm to bound sensor input values. Otherwise, the designs are functionally similar as described above.

Figure 6-6 shows the basic function of the Benchmark System II RPS for channel A of the system. In this diagram, the coolant flow trip signals are indicated. The bolded signals in the diagram indicate voted signals from the three legs (MPU A, MPU B, and MPU C). The legs are indicated in the diagram as the depth dimension. The coolant flow trip logic on channel B, channel C and channel D are logically and functionally the same as shown below. The coolant flow trip signals from channel B, channel C and channel D are transferred to channel A by the inter-channel communication network. Thus, all channels exchange their trip signals wirth each other. Not shown in the diagram is the last “OR” block that takes all trip signals (steam flow, hot leg pressure, etc…) and ORs them for the final trip signal.

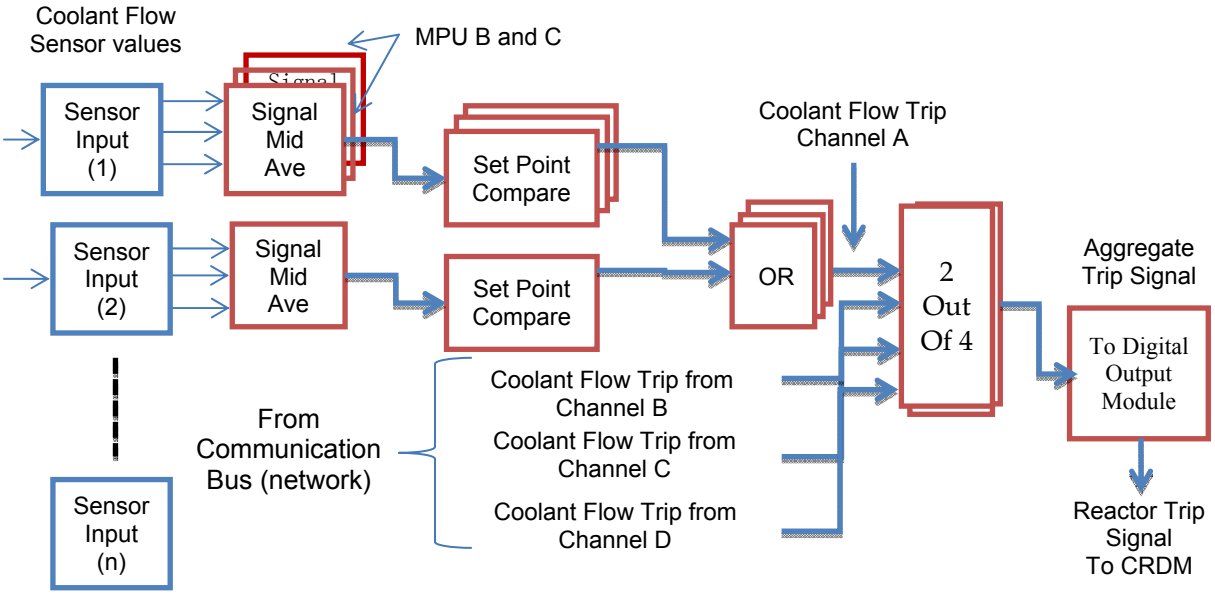

Figure 6-6 Channel A RPS for Benchmark System II

6.5. References

[Hashemian 2011] Hashemian, H.M. "USA's First Fully Digital Station." *Nuclear Engineering International*, January 2011: 37-41.

[Westinghouse 2003] Westinghouse Electric Co, LLC. *The Westinghouse AP1000 Advanced Nuclear Plant.* http://www.ne.doe.gov/pdfFiles/AP1000_Plant_Description.pdf, Westinghouse Electric Co, LLC, 2003.

7. LESSONS LEARNED FROM PREVIOUS EFFORTS

This section describes lessons learned from previous efforts at UVA, namely the application of fault injection to an authentic DFWCS. This section, along with Section 5, serves as a springboard to (1) identify candidate fault injection methods for the benchmark systems, (2) identify areas where the methodology is underprovided or lacking in support, and (3) provide a basis for implementation and research efforts for phase II and phase III of this research. The DFWCS presents the same scale and complexity as expected on the benchmark systems, so reviewing the effort, challenges, and results is significant to the overall research effort.

7.1. Introduction

The UVA safety assessment methodology was developed and matured primarily through *simulation-based fault injection techniques* starting in the early 1990's. While the capabilities developed along this path were considerable, the transition to physical-based fault injection at UVA was not without challenges. This is an important point because simulation-based fault injection techniques are considerably different than physical-based fault injection techniques. To better illustrate these differences, the attributes of simulation-based fault injection are listed here: [Arlat 2003].

- *Enhanced Observability and Controllability*. The simulation environment along with its model building tools allows full observability and controllability of the target system model.

- *Built in Automation*. The simulation environment along with programming support allows for total automation of the fault injection process inside the simulator.

- *Built in Time Measurement*. The simulation environment, due to its intrinsic timing capabilities, allows for precise measurement of all fault injection results.

- *Built in data Collection*. The simulation environment permits easy collection of program execution trace information to determine differences between non-faulted operation and faulted operations.

- *Fault Injection "everywhere".* The reachability of the fault injection process is only limited by the fidelity and the scale of the model for the target system.

In a physical fault injection environment none of the above qualities are intrinsic or given; and more often than not it is a significant challenge to modestly instrument some of the above attributes in physical-based fault injection environments. This is the principal issue of physical-based fault injection experiments; many of the above attributes must be developed, integrated, and implemented to support the fault injection process in a physical system. Moreover, the basic components of the fault injection environment must work together in a seamless automatic fashion [Hsueh 1997].

The next section discusses the challenges related to physical-based fault injection on the Calvert Cliffs NPP DFWCS prototype system at UVA and at the Calvert Cliffs NPP station labs. This was the first substantial application of physical-based fault injection to a digital I&C system at UVA.

7.2. Overview of the Digital Feedwater Control System

The function of the DFWCS is to control the input water level in its associated steam generator from approximately 1% reactor output power up to 100% reactor output power [Aldemir 2009]. As depicted Figure 7-1, the steam generator level is controlled by two computer-based controllers. The two controllers are designated as the main controller and the backup controller. The main and backup controllers operate as a redundant pair. These controllers are implemented using commercial-off-the-shelf (COTS) components that include two Azonix uMAC7000 Industrial PCs with 486DX4 33 Mhz processors running a scaled down version of MS Windows 3.1. A real-time kernel application was developed to schedule and execute the control laws on a 100 ms cycle time. The real-time kernel was executed as a supervisor level within Windows OS.

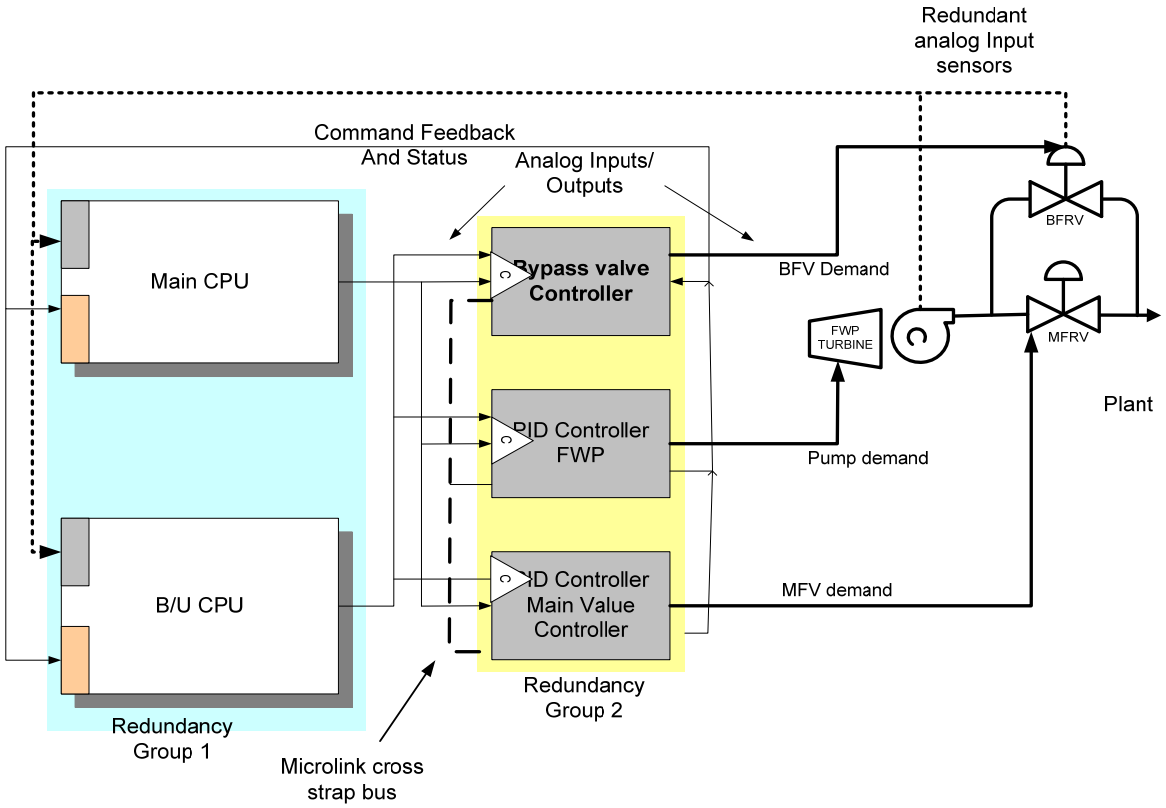

Figure 7-1 Overview of benchmark Digital Feedwater Control System (DFWCS)

Normally the main controller reads various inputs and provides control outputs while the backup controller tracks the main controller. The main and backup controllers act on duel analog sensor readings of reactor power level, steam generator (SG) level, feedwater flow rate, feedwater temperature, and steam flow rate. The DFWCS outputs set the operating positions of the main feedwater flow control valve and the bypass feedwater flow control valve, and speed change commands for the main feedwater pump. In the NPP, the feedwater pump commands are further shaped by a redundant turbine speed regulation controller.

In the event that the main controller fails, the backup controller can take over 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. These four PID controllers are implemented using Bailey Fischer & Porter (BFP) 53MC5000 series process control stations. Three of these controllers are dedicated to the main feedwater flow control valve, the bypass

feedwater flow control valve, and the feedwater pump speed regulator respectively, with the fourth controller acting as a spare for either the main feedwater flow control valve PID controller or the bypass feedwater flow control valve PID controller. The PID associated with each of the controlled devices performs a comparison of the desired setting from the main and backup controllers and delivers the final outputs to the controlled device.

In summary, the DFWCS presents an unusual architecture for real-time safety related applications. It uses Windows 3.1 as its underlying OS, which uses dynamic task allocation and memory management. Dynamic task allocation and memory management is almost never used in real time safety related systems due to its non-deterministic behavior. The real-time kernel compensated for this by scheduling tasks on a timer interrupt every 100 ms. However, the memory allocation management was still controlled by Windows, which meant it was dynamically reallocated if the processor was reset or another task needed more space in Windows.

7.3. Background on the DFWCS Fault Injection Efforts at UVA

Research on the DFWCS was sponsored by the NRC under 2 contracts, which are referred to in this report as Phase 1 and Phase 2, respectively. The first phase was "*Embedded Real-Time Safety-Critical Hardware/Software Systems,*" which defined and developed the fundamentals of the methodology described in Section 3. Phase 2 was a reassessment of the Phase 1 efforts, and the application of fault injection to the DFWCS to support a dynamic reliability modeling effort led by the Ohio State University (tOSU).

In the Phase 1 effort (2000 to 2003), a SWIFI method of was chosen after an unsuccessful attempt at another technique[3] [Maia 2005]. In run-time, SWIFI faults are injected while the target system executes its application or workload. This requires a mechanism (called an exception handler) that (1) stops the execution of the processor, (2) invokes a fault injection routine to corrupt the memory or register locations of the target system, and (3) restarts the processor and workload where it was interrupted. Thus, SWIFI can incur a significant run-time overhead if the implementation of the fault injection method is not optimized.

The effort to develop the exception handlers needed for SWIFI fault injection was challenging because the exception handlers had to be integrated into the Windows OS. In this case, the run-time overhead for SWIFI was not the prevailing issue. The dominant issue was the low controllability of the fault injection process, mainly due to implementation issues rather than technology issues. First, the exception handlers did not have the ability to trigger and break on specific program variables because of the difficulty of deriving or obtaining a memory map of the program residing in data memory. Secondly, because there were no breakpoint triggers for the fault injection process, the fault injector could not control two dimensions of the fault injection process space: time of the fault injection, and location of the fault injection. Lack of these two factors led to poor reproducibility and repeatability of experiments.

In Phase 2 (2004 to 2006) a new team of investigators (Carl Elks, Michael Reynolds, and Nishant George) were brought in to reassess the Phase I efforts in support of a NRC sponsored project led by tOSU called *Dynamic Reliability Assessment of Digital I&C Systems.* To fully support the needs of the tOSU investigators, UVA had to develop a method of fault injection for the DFWCS that allowed (1) injecting faults into memory and register locations that were being used by specific process variables of the control law (e.g., inputs, state variables, outputs, etc....) and (2) doing so in a manner that could be fully automated. At this time, the SWIFI method was abandoned in favor of In-Circuit Emulator (ICE)-based fault injection to enhance

[3] The first attempt at fault injection was with an in-circuit emulator (ICE)-based fault injection. It was unsuccessful due to incompatibility between the ICE machine and the processors on the DFWCS.

controllability of the fault injection process (see the sections below). A brief description of ICE-based fault injection is described below along with how it was used on the DFWCS. For a detailed description of the design and development of the ICE based Fault Injection Environment used in the DFWCS experiments see reference [George 2007].

7.4. Selecting a Fault Injection Method for the DFWCS

In Section 5 a number of methods for injection are reviewed and critiqued according to their advantages and disadvantages. In summary, software-based methods of fault injection are better suited to test fault tolerance mechanisms at the data flow and control flow level, when compared to hardware-based methods. In most cases it can be seen that the target digital I&C system OS needs to be instrumented with traps and exception handlers to enable triggering of fault injection. However, as noted above, SWIFI can be a difficult fault injection technique to implement due to these modifications and controllability of the technique can be an issue if not done correctly. At the time UVA was working with the DFWCS, and due to the previous issues the UVA research staff had with SWIFI, a method of fault injection closely related to OCD[4] fault injection called *In-Circuit Emulation* based fault injection or simply ICE-based fault injection [George 2007] was chosen.

An ICE machine is a tool used by designers of embedded systems to debug embedded software. Debugging embedded system software is particularly challenging because embedded systems usually lack suitable user-interface devices such as keyboards and displays. Under such circumstances, ICE machines provide a `window' into the system through which the designer can exercise a good deal of control of the embedded system at a very low level (e.g., Assembly code, and signals). In-circuit emulators usually have a pod (shown in Figure 7-2) that plugs directly into the socket where a CPU chip is inserted. There is interface circuitry that provides a connection between an ICE machine and a terminal PC. This terminal can be used to run an interactive user interface application by which the designer can monitor the embedded system.

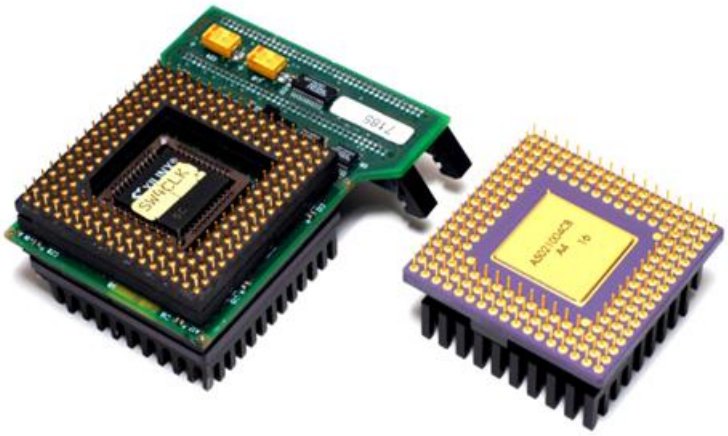

Figure 7-2 In-circuit Emulator (ICE) pod

At the time of this effort, ICE-based fault injection appeared to be a feasible and attractive option because fault injection was possible without any change to the software running on the target system, which was something the researchers wanted to avoid. In theory, ICE-based fault

[4] OCD-based fault injection was not an option on the DFWCS platform because there were no OCD features associated with the Intel-based x486 processor.

injection should provide better controllability and observability as compared to other alternative techniques (such as SWIFI and pin-level fault injection).

7.4.1. ICE-based Fault Injection

A discussed earlier, ICE machines are used primarily for debugging and not fault injection. The UVA effort to use an ICE machine for fault injection was a novel approach, so UVA had to develop much of the "know how" and technology to achieve the ICE-based fault injection capability.

A prototype implementation of an ICE-based fault injector was designed using the PowerPack EA/SW In-Circuit Emulator, manufactured by Microtek International Development Systems. Referring to Figure 7-3, the Microtek Powerpack EA486 ICE hardware consists of a pod that plugs into the socket of an Intel x486 CPU-based target system and an interface circuitry box that interfaces the pod to the host machine through the serial port. The interface box has a dedicated power supply, which powers the ICE-pod. The CPU of the target system is removed, and the ICE-pod is plugged directly into the CPU socket of the DFWCS.

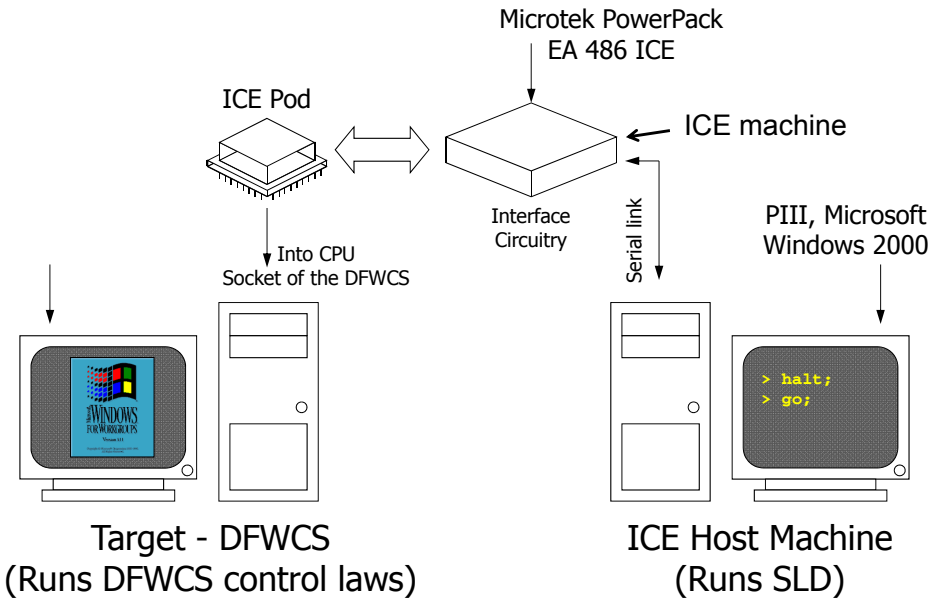

Figure 7-3 ICE–based fault injection on the DFWCS

The in-circuit emulator software is a source-level debugger (SLD) application that must be installed on a separate computer known as the *host*. The target system is the DFWCS. The interface box is connected to the host computer through the serial port. All communication between the host and the target system takes place through this interface. Once the ICE interface and the target system have been powered up, the SLD application can be initiated on the host. The ICE and SLD give a disassembled view of the memory around the location that the program counter register points to on the target system. When halted, the disassembled view of the code running on the target machine can be viewed on the host machine.

As can be seen in the Figure 7-4, the memory locations immediately following the location pointed to by the instruction pointer are displayed in the window, together with the binary code of the instruction at that location and its corresponding disassembled mnemonic assembly code.

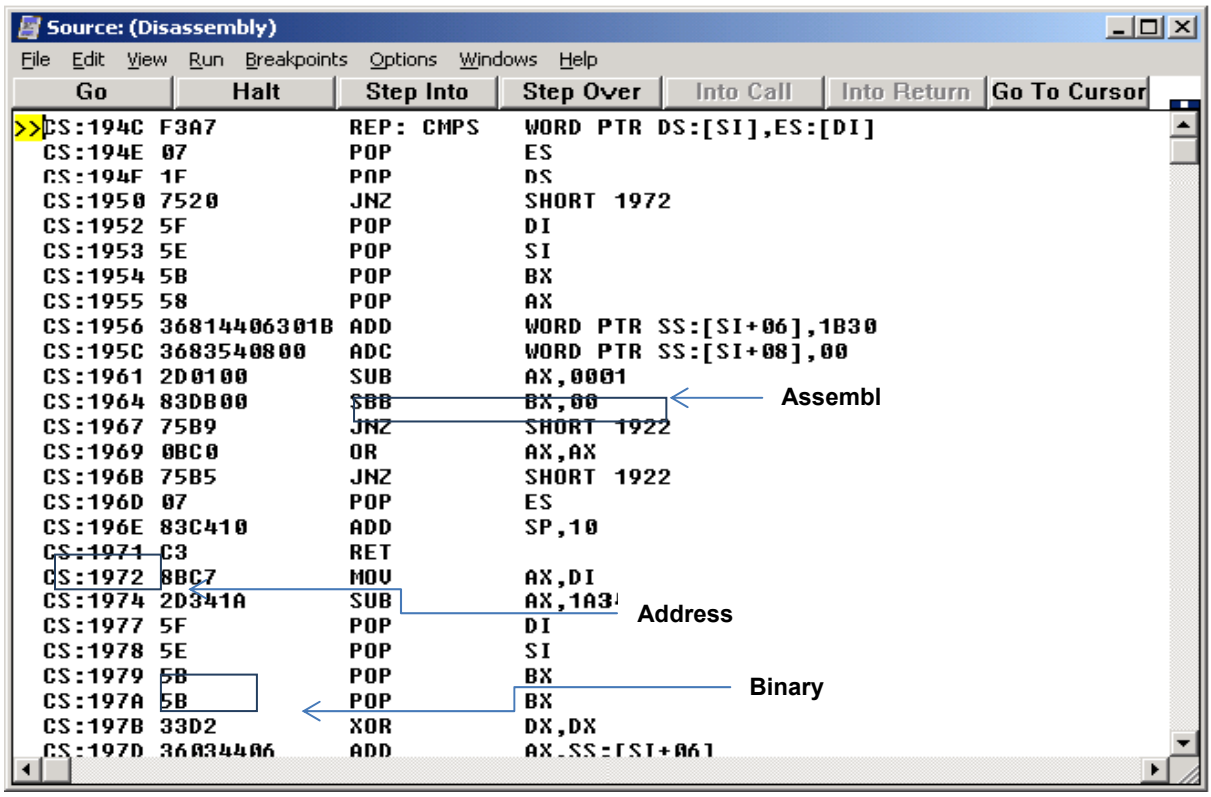

Figure 7-4 Screenshot of disassembly view on the host SLD application

7.5. Automation of Fault Injection Process

The SLD command tool for the ICE machine is equipped with the capability to execute script files, which contain a sequence of shell commands that, when used in sequence, allow the basic steps of fault injection to be executed in a serial manner. A script file is a sequence of commands and command arguments that the ICE machine uses to perform its debugging functions. This is a key capability in the realization of fault injection using the ICE machine. The normal SLD Graphics User Interfaces (GUIs) are not capable of being invoked in an automated fashion, which is required for conducting extensive fault injection campaigns. These script files enable automatic fault injection.

Figure 7-5 shows an example of the various shell commands that are executed at the shell command prompt of the SLD application to emulate a single fault injection. The basic steps that comprise a fault injection from the SLD host application are explained below with respect to Figure 7-5.

```
1: while($EMULATING){};          // Spin Loop -- Wait for halt
2: dump 153f:509a dword;         // Load contents of specified address
3: $val = $MEM_VALUE;            // Save value in a variable
4: $val = $val & 0x00000000;     // Apply fault mask
5: write 153f:509a $val dword;   // Write back corrupted value to memory
6: $FICount = $FICount + 1;      // Increment counter
7: $FICount;                     // Display counter
8: go;                           // Resume execution
```

Figure 7-5 Sequence of script command steps for automating fault injection

Step 1 is a spin loop that waits for the CPU to halt. This halt may be induced by a number of means depending on when the fault needs to be injected into the system. For example, the fault may be injected on an event defined as ``when a write occurs on a predefined memory location that corresponds to a location in the data segment". An event can be defined by setting various flags in the SLD application that correspond to signals within the CPU such as MEM_WRITE, MEM_READ, and CODE/DATA, can be set to suitable values, and its action may be specified as a halt. When the system halts, the variable becomes false and the script falls through to execute the remaining commands in the script.

Besides using events, software and hardware break points can be defined on memory events for code/data accesses to halt the CPU at desired points in time. In addition to these, external signals can be used to trigger CPU halts asynchronous to the target and host machines. This was the scheme that was used to inject faults at random points in time, and was initiated using trigger pulses generated external to the host and target machines.

Step 2 dumps the contents of the specified location into a predefined SLD system variable called MEM_VALUE. Step 3 copies this value from the system variable to a local variable so that it may be manipulated. Step 4 applies the mask that is specified in the fault list onto the value of the variable. Any logical operation may be performed in this step and each different logical operation or a different fault mask would correspond to a distinct fault. The value of the variable corrupted in step 4 is copied back into the location from which it was read in step 5. This can be considered as the actual injection step because it is in this step that a corrupted value gets updated in the memory. In steps 6 and 7 a local counter variable is incremented and displayed to keep track of the number of fault injections that have been completed. Step 8 resumes execution of the target.

The fault list contains a number of pieces of information that are used by scriptgen.cpp. These include a serial number, address, number of bytes to corrupt, a fault mask (defines which bit(s) to corrupt) and a mask operation (AND/OR/XOR and so on).

This example shows how the capabilities of the ICE machine were exploited to realize fault injections. If needed, more complicated events and breakpoints can be set up to meet arbitrary timing requirements. A sequence of sets of such statements can be grouped together in a single script to achieve consecutive automatic fault injections. For example, if after step 8, step 1 is repeated, then the SLD will execute a spin-loop-wait until the next halt event occurs. This time can be used for monitoring responses to the fault injection, such as for collecting data dumps. The target may also be restarted at this point to refresh the workload to rid of the target

any effects of the fault previously injected so that the next fault injection behaves independent of the previous fault injection.

7.6. Realization of Fault Models

The principle fault model used in this effort was the transient fault model [Cutright 2003(b)]. Using the ICE machine allowed changing the number of bits in a register or memory location. For this project, single bit corruption and multiple bit corruptions on a specific memory or register location per control cycle was chosen, including all “zeros’ or all “ones” on a 16 bit or 32-bit word.

For realizing a permanent fault model, the fault must be persistent. To increase the duration of the presence of a fault, the fault script was set up so that the value stored in the location could be corrupted every time it was overwritten with a fresh value. It should be noted that emulation of permanent faults incurs much more overhead in implementation using this fault injection technique since each value corruption occurs every time a values is written in the specified memory location, which may be many times during a single control cycle. Thus, the permanent fault model was used only during specific instances.

7.7. Fault List Generation

A review of various documents including the DFWCS hazard analysis document revealed the source code drove the construction of fault lists for the DFWCS effort. Locations such as analog inputs and outputs, digital inputs and outputs, set-points, internal variables, and test-points were identified as key locations for injecting faults, along with random locations in memory. A memory and link map of the application was generated when the DFWCS was compiled. These map files were automatically searched by a special parser program developed for obtaining the address components (segment:offset) of each variable in the DFWCS program. Because the DFWCS used the Windows 3.1 OS, which is a dynamic memory management operating system, the segment address is not static because it would depend on where the application was loaded into the memory, which is allocated by the operating system of the target. To overcome this difficulty, special probes were inserted into the boot code of the DFWCS application before each fault injection campaign to determine where the code was being relocated. Details of this procedure can be found in reference [George 2007]. As stated earlier, dynamic memory management is something of an anomaly in real-time safety critical system since almost all safety critical systems use static memory allocation.

7.8. Integration of ICE-based Fault Injector into DFWCS

The initial design of the ICE-based fault injector was implemented at UVA using a replica version of the DFWCS. After the initial testing at UVA, the fault injector was deployed at the NPP where integration of the fault injector into the full-scale DFWCS was begun in the NPP I&C laboratory. Integration of a fault injection environment into an operational environment is necessary because it addresses a number of engineering challenges when compared to a stand-alone fault injector. Fault injectors are never standalone and discussing them in the context of a system gives a more complete picture. A number of issues were faced during the design of the ICE-based fault injector at UVa. While integrating the fault injector into the operational environment, a new set of challenges were faced. This section discusses the challenges associated with the integration effort.

7.8.1. Integration of the ICE-based Fault Injector into the DFWCS

The development of the fault injection environment for the DFWCS application involved integrating the ICE-based Fault Injector into the DFWCS in the laboratory at the NPP. This integration posed a number of challenges and led to the development of solutions to overcome these challenges. This section lists and explains these challenges and solutions. A schematic showing the ICE-based fault injector environment integrated into the DFWCS is shown in Figure 7-6. The discussion in the remainder of this Section will refer mostly to this figure.

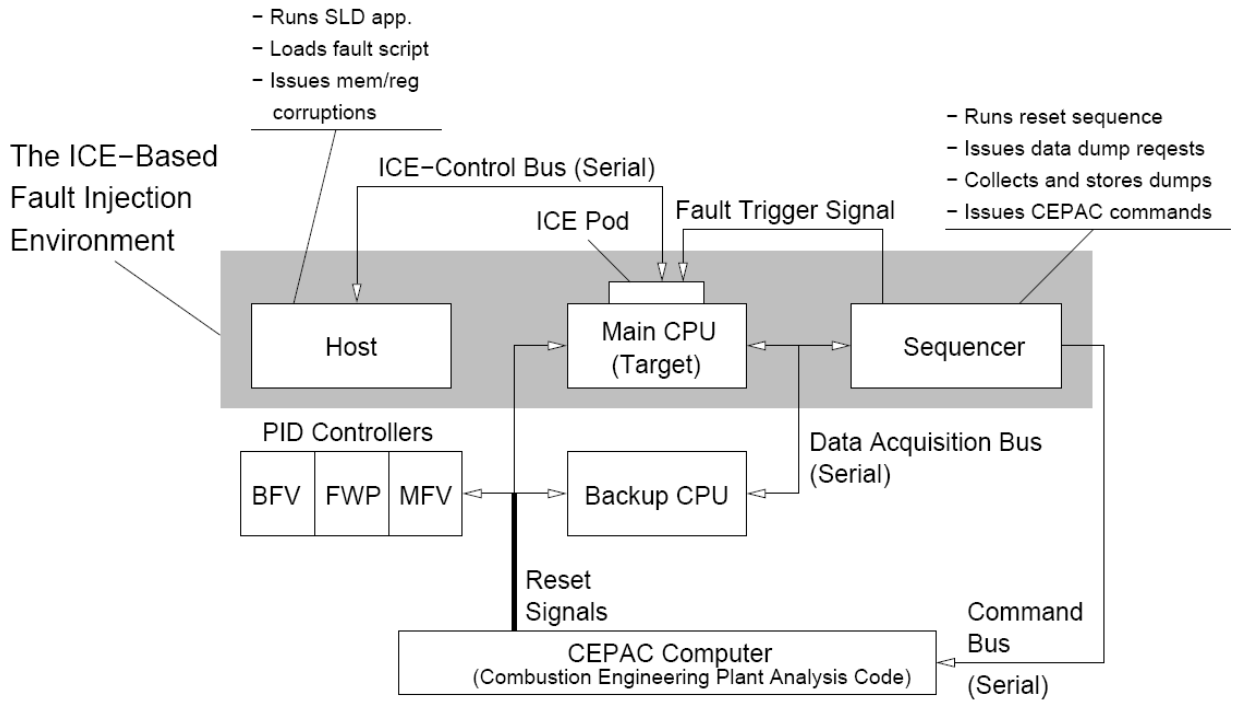

Figure 7-6 Integration of the ICE based fault injector into DFWCS environment

Challenge 1: Reset Sequence of the DFWCS

In order to support full automation of the fault injection process after each fault injection and data collection activity, the DFWCS should be reset automatically to clear the effects of the injected fault from the system state. This was not a straightforward process with the DFWCS. The DFWCS has an elaborate reset sequence that consists of eight to ten steps (some of which are dependent on each other) taking about 100 s. The complexities of the reset sequence are mainly due the use of diverse COTS components from different vendors in the DFWCS architecture.

The reset sequence was emulated using a sequencer computer running a LabView application, which sent out reset signals to various components of the DFWCS in a timely manner. The NPP I&C engineer who had detailed knowledge of the DFWCS designed and coded the reset sequencer for the test setup. The timing diagram of the signals the sequencer application generated is shown in Figure 7-7. Integration of the fault injector into this system involved modifying the sequencer application to provide an indication to the fault injection environment to inject a fault, only when the whole system was operating after the elaborate reset sequence. This indication was in the form of a trigger signal sent from the sequencer to the ICE machine. The ICE machine would recognize this incoming trigger pulse as an indication for fault injection and would initiate injection of the next fault from the fault list.

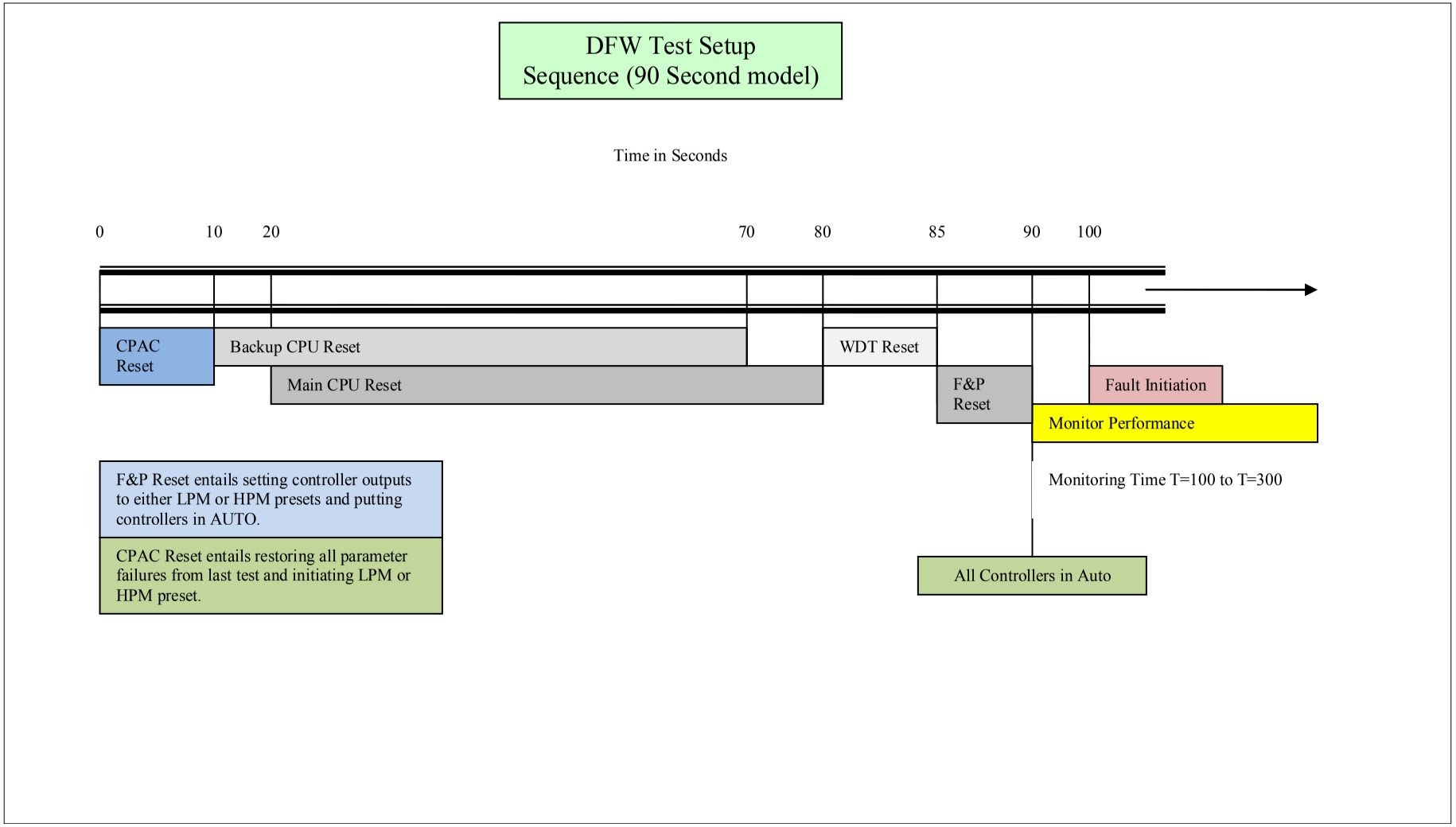

Figure 7-7 Timing diagram of the DFWCS-reset sequence

Challenge 2: Power-Fail Signal Masking

A fault injection involves three steps: halting the CPU, changing bits, and restarting the CPU. The DFWCS-self monitoring system was designed with the main and backup CPUs monitoring each other such that the instant either CPU was halted, the other CPU would identify that event as a *power fail* on the failed CPU. However, during fault injection, this was a false alarm because the main CPU was intentionally halted for fault injection. This was conveyed from the main to backup CPU as a power-fail signal line, which would drop to a voltage level of 0 V when the power was halted.

To overcome this false alarm problem, a power-fail-mask circuit was developed to pull up the signal on the power-fail line while the fault was injected. The ICE machine is equipped with a reset-out signal which can be pulsed using a shell command in the host application. This signal was fed into the clock input of a toggle flip flop, which would toggle its state if there was a pulse on the clock. This functionality was used with proper placement of the shell commands in the fault scripts to effectively mask the power fail signal for the time the fault was injected.

The power-fail line was multiplexed with two inputs. One input was the actual signal from the main CPU, and the other signal was a constant 1 V signal. The control signal for the multiplexer relay was the output of the toggle flip flop. The shell command for toggling the flip flop was first given immediately after the halt of the main CPU and was reset immediately after the execution was resumed after the fault injection. Figure 7-8 shows a schematic of the power-fail mask solution.

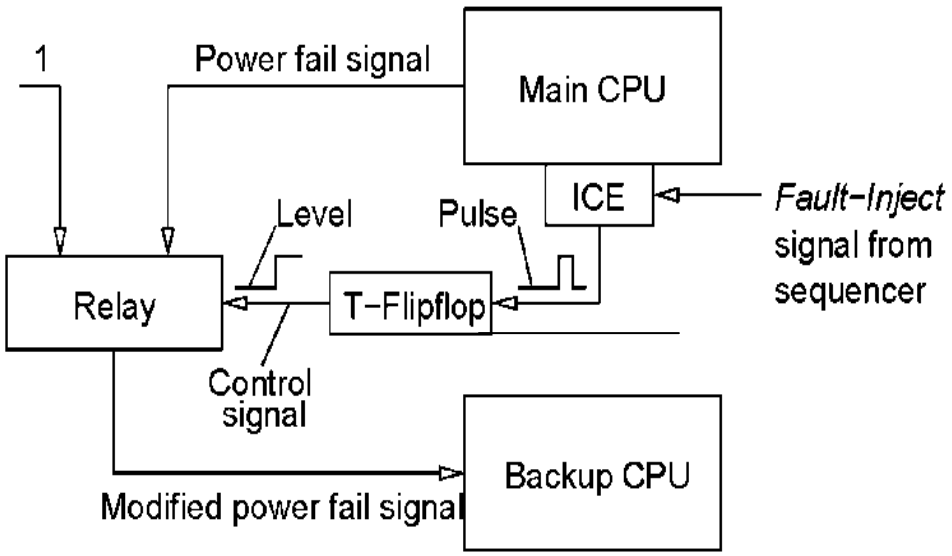

Figure 7-8 Power Fail mask circuit

In addition to the power fail mask signal, each processor had a watch dog timer (WDT) signal that was derived from an independent and external WDT module. Due to long delays in the fault injection process (primarily due to ICE machine inefficiency) the WDT signals were disabled during fault injection. In normal operation, the WDT signal is used to detect timing failures such as crashed or hung processors that would impede real-time operation of the main or backup processor.

Challenge 3: Static Electricity and Noise

The subsystems described in the previous sections operate at a wide range of voltages. For example, all of the CPUs operate at 5VDC, but all of the signals between CPUs, and inputs/outputs from sensors operate as 24V analog signals. The ICE machine operates at 5V DC and the signals coming from the ICE machine are also 5VDC. After the preliminary attempts at integration of the subsystems into a fault injection environment, it was observed that the fault injector was not stable. Random signals would trigger fault injections at wrong times, during the reset sequence. This was a major impediment which delayed the deployment and experimentation process. Until the random fault injections that were being observed were attributed to effects from static electricity and electrical noise, each fault injection campaign was very short, comprising not more than 10 to 15 experiments.

This warranted electrical isolation across all connections between the ICE machine signals and the DFWCS signals. Hence, among the connections shown in the diagram shown in Figure 7.7, the signals connecting the sequencer and the ICE, the ICE and the T-flip flop, and the T-flip flop to relay had to be coupled through opto-couplers. This ensured electrical isolation between various components of the fault injector integrated into the DFWCS. With this circuit in place, most of the random noise signals that were affecting the determinism of fault injection sequences were addressed. This resulted in the ability to run long fault injection campaigns that were capable of handling hundreds of faults per campaign.

Challenge 4: Data Acquisition

The data acquisition capability is an important component of the fault injection environment because responses to injected faults must be gathered for later analysis. Unless the results of fault injections can be captured effectively, the effort of building the fault injection environment can be negated. The DFWCS application has a built in data dump capability which logs events that occur in the system. These logs are generally available to the user when the operator queries the system for data dumps. The logs are also visible to the operator on the plasma display I/O units (PDUs). The PDUs are connected to the CPUs through the serial port COM1. A modification of this existing data dump capability was used to collect data in the sequencer computer. The sequencer computer, upon reception of a data dump was made to append some fault information and save it locally with a suitable filename. Data dumps were collected once before a fault injection and once after a period of monitoring after a fault injection. Dumps were collected from each of the main and backup CPUs. Thus a total of four dump files were obtained for each fault injection experiment that was conducted.

The data dump files contain three sections. Figure 7-9 shows the first section of data dump files. The first section consists of a listing all of the 142 set point values. The second section contains the minimum, maximum, and current values of 19 parameters. Figure 7-10 shows a portion of this section of the data dump. The third section contains event log messages with time stamps for each event. This section provides insight into detected faults and the nature of responses that were initiated by the DFWCS as a result of a fault injection. Figure 7-11 shows a portion of the third section of the data dump.

```
02/27/2007 15:58:19
SETPOINTS
  1    0.60   2   40.00   3  999.00   4 -999.00   5  150.00   6  100.00
  7  600.00   8   56.30   9    0.00  10    1.00  11   30.00  12    3.00
 13   15.00  14   10.00  15   15.00  16   25.00  17   12.00  18   20.00
 19    2.00  20    1.00  21    3.00  22  100.00  23  500.00  24   20.00
...
```

Figure 7-9 First part of data dumps contain setpoints

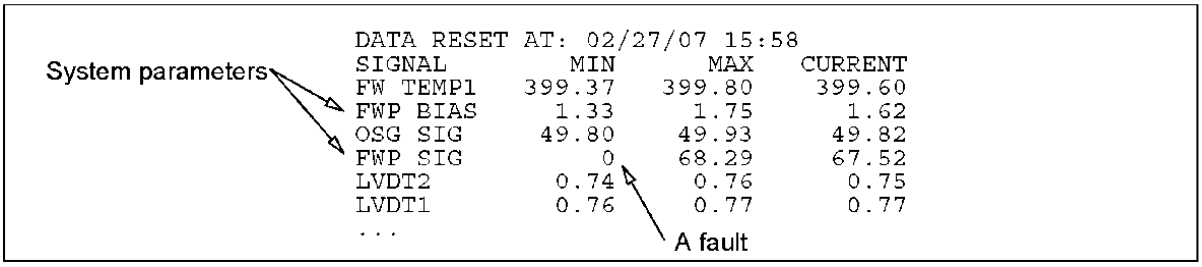

Figure 7-10 A portion of the second section of data dumps containing Min-Max

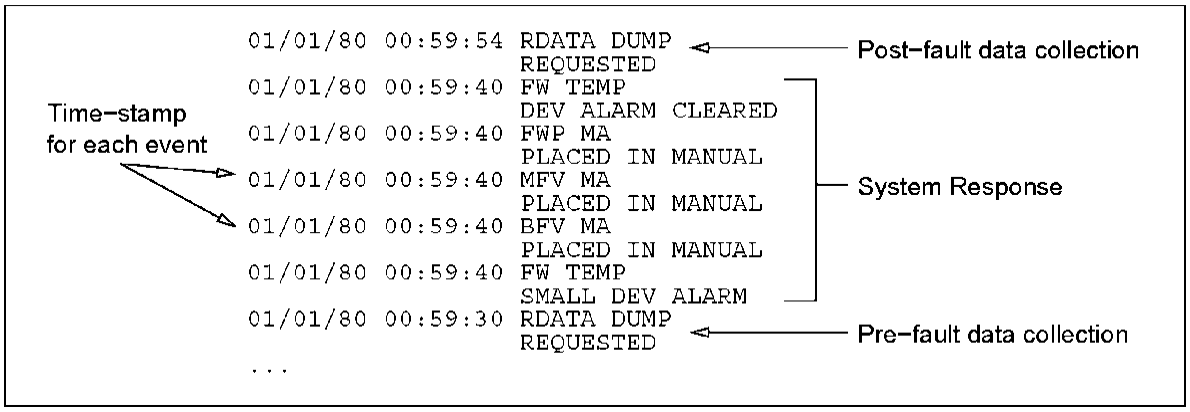

Figure 7-11 A portion of the data dump that contains event logs

Challenge 5: Data Analysis

Normally an assessor would have a custom-tailored data acquisition system, which can be customized to provide structured and directed data dumps. The data dumps that were obtained were, however, highly customized for the NPP application so that information required for extracting details about fault coverage were scattered within each file. As described in the previous section, each fault injection gives a total of four data dump files that contain pre-fault injection and post-fault injection information.

Because manual examination of each of 2400 dump files was impractical, parsers were written to extract specific information from these files and compile the extracted information into a more suitable format for analysis. For example, a parser was written to extract set point values from data dump files into comma-separated-value (CSV) files that could be read easily using a

spreadsheet application. Similarly, another parser compiled into CSV files all minimum, maximum, and current values of 19 of the system parameters available in the dumps. A PERL[5] parser was written to extract the most recent log messages that appeared between the two data dump requests. These files were analyzed for various modes of responses and were classified accordingly. More details about this classification are presented in the next section.

7.9. Experiments and Analysis of the Fault Injection Data.

The experimental setup was designed and tested on the prototype of the DFWCS at UVa. Once sufficient confidence was obtained in the functions of the fault injector it was deployed on the DFWCS at the NPP laboratory. This section describes details about the fault injection campaigns, such as fault locations and types, fault values, and timing. In addition, the results of the data were analyzed according to several needs.

7.9.1. Experiment Attributes

This section describes the various attributes of the fault space that defined a fault in the fault space that was used in the fault injection campaigns. The three attributes that defined a fault were location, timing and value. All of the experiments were run using the same workload - the DWFCS application.

7.9.1.1. Fault Locations

By analysis of the DFWCS hazards document, the compiler generated map file, and the source code, a total of 240 locations were identified as points for fault injection to stimulate the various error detection mechanisms in DFWCS. Variables in the application source code modules that corresponded to input processing, output processing, internal variables, and those used for acceptance tests were chosen for corruption. These variables included analog inputs, analog outputs, digital inputs, digital outputs, control law variables, set-points, and test-points. Each variable group is an array of structure variables, whose addresses were obtained from the compiler generated map file. The size (in number of bytes) of each variable group was calculated from the individual structure elements. Array offsets were thus computed to find the locations of each array element.

Table 7-1 shows details about the data structures that represent various variable groups that were chosen for fault injection. For example, there are 20 analog inputs. The data structure used to represent one analog input is 104 (68H) bytes long. Within that data structure, the actual value of the analog input itself is 8 bytes long and is of the float data type. The base address of the data structure that stores information about the first analog input is at 4FF6H, and hence the data structure containing information about the second analog input is located at 4FF6H + 68H.

Table 7-1 Data structures of the locations for fault injection.

Var. group	Size of structure	No. of instances	Base address	Size of value
Analog inputs	104 bytes	20	4FF6H	8 bytes
Digital inputs	30 bytes	16	5816H	1 byte

[5] **PERL** is a high-level, general-purpose, interpreted, dynamic programming language

Analog outputs	80 bytes	5	59F6H	8 bytes
Digital outputs	80 bytes	13	5B86H	1 byte
Test points	24 bytes	45	5D0CH	8 bytes
Set points	14 bytes	143	4824H	4 bytes

7.9.1.2. Fault Models

Both transient and permanent fault models were attempted in the experiments. However, when permanent faults were injected into memory locations, after long fault injection campaigns, the main computer would crash in a manner that was not restorable using a simple reset. The reason for this failure mode was that the main CPU flash memory became corrupted, and its image had to be rebuilt to restore operational status. This behavior was attributed to data and code memory segments of the main CPU being swapped in and out of flash during execution. If an OS code segment was corrupted through a fault and was then swapped out to the flash, it would result in damage to the OS image. Consequently, not many permanent fault injection experiments were performed and the research effort resorted to using the transient fault model. This type of fault is typical and recoverable in most computing systems, however, the flash disk drives in the DFWCS are not easily re-imaged with the Windows 3.1 OS, so the plant engineers decided it was best to limit the permanent fault campaign.

7.9.1.3 Fault Timing

Fault timing defines at what point in time the fault is to be injected into the system. It was mentioned earlier that the DFWCS requires approximately 100 seconds to boot up through the entire reset sequence. A fault can be injected at any time after the reset sequence. After the boot up sequence of the DFWCS, 10 seconds are needed for pre-fault data collection. The fault was injected at a random point in time *after* this event. A uniform distribution was used to determine a random point in time between 1 and 10 seconds and was added to the reset sequence as the amount of time to wait before injecting the fault. After the fault was injected, the system was allowed to run for 60 seconds before the post-fault data were collected.

7.9.1.3. Fault Value

There were a number of choices for the value to which a corrupted variable, register or memory location could be set, as shown below. The contents of the memory location or register chosen for corruption could be forced to any of the following:

- All zero state (minimum): 0000H
- All one state (maximum): FFFFH
- Random assignment: a randomly generated bit vector
- Inverted state: all bits inverted
- Single bit flips in a 32-bit word
- Multiple bit flips in a 32-bit word

7.10. Experiments

In effect, each unique fault value can be considered to be a different fault. However, the list of faults that were compiled mainly contained faults in which the values of analog inputs, outputs, set points and test points were set to the all-zero state, all-one state, and digital inputs and outputs were set to the all-zero state and all-one state. This was mainly attributed to the limited

number of fault injection experiments that were able to be conducted during the time access to the DFWCS was available at the NPP laboratory.

Automation limitations dictated that no more than 100 fault injection experiments could comprise a fault injection campaign. This was because it was observed that on certain occasions, though rarely, the fault injector failed to complete a campaign of 100 fault injections due to stability issues that were encountered as a result of static electricity and noise in the environment. The limited size of the fault injection campaigns required that a new campaign be started approximately once every five hours.

Moreover, due to schedule restrictions of the engineers at the power plant, up to three fault injection campaigns could be performed each day. With a maximum of 300 faults per day, some missed days, and some incomplete injection campaigns (which had to be rerun), 2400 fault injections were completed in a period of three to four weeks. Each of the 240 process variables that were identified for corruption was injected 10 times with different fault value corruptions. A very large percentage of this inefficiency is due to the fact that the experiments were executed remotely with little or no feedback on the status of the fault injection process from the NPP. Typically this is not the case in fault injection methods and experiments, most often the engineers and scientists who design, develop, and implement the fault injection environment are present in the laboratory to conduct the experiments, review the data, and fine tune the injection process as needed for maximum efficiency.

Each fault injection campaign resulted in four data dump files, as follows:

- Pre-fault injection dump from the main CPU - mcpua__.txt
- Pre-fault injection dump from the backup CPU - bcpua__.txt
- Post-fault injection dump from the main CPU - mcpub__.txt
- Post-fault injection dump from the backup CPU - bcpub__.txt

Each filename was suffixed with the experiment number in a given campaign (for example mcpub56.txt). Each data dump file was prefixed with fault information obtained from the fault list that indicated the fault location and fault value. Each fault list was numbered 100.lst, 200.lst, ..., 2400.lst.

7.10.1. Raw Data Results

A number of response modes were observed, including:

- **Fault covered due to masking or no-response faults**: This is the case when the effect of a fault does not propagate as an error because after the fault was injected, the variable would have been overwritten with a fresh, valid value. These are also called *no-response faults*. The response of this type of fault would correspond to those cases when the data dump files do not indicate any failure or any detected event between the pre- and post-fault injection data dump requests.

- **Fault covered by detection and recovery**: In this case, the effect of the injected fault is detected and reported as a message logged in the data dump files (e.g., DEVIATION ALARM, VALUE OOR (out of range), MAIN CPU FAILED, etc.). These are usually accompanied by a following message that might indicate restoration to a normal state, such as OOR CLEARED, or DEVIATION ALARM CLEARED, or MAIN CPU NORMAL. If the last message logged in the data dump is one that indicates a *normal* system, then it is a covered fault *with recovery*.

- **Main CPU failure**: If the last message that was logged in the data dumps indicated MAIN CPU FAILED, then it is a fault that is covered by a switch to the backup controller. Another indication that the main CPU failed is that if the data dump file from the main CPU was empty, then it implies that the main CPU was not functioning at the time the data dump was collected. If the failure of the main CPU was logged in the data dump of the backup CPU, then it is considered to be a covered fault.

- **Local uncovered fault**: If the data dump from the main CPU before data collection is *not* empty, and the dump from the main CPU after fault injection is empty, this corresponds to the case when the fault injection caused the main CPU to fail. If the backup CPU data dump failed to indicate a main CPU failure in this case, then it is considered a fault for which the main and backup CPU error detection mechanisms did not detect the fault. It should be noted that the WDT module was disabled for these fault injection experiments, so its detection power was engaged for this experiment. Thus, the fault is not uncovered for the system as whole, but only for the diagnostic and self-tests routines for the main and backup processors. In the 2400 fault injection experiments that were conducted, only one fault of this type was found.

- **Invalid cases**: Some fault injection campaigns resulted in some invalid states that did not bring the main CPU up after a certain fault injection experiment. These experiments are invalid because the main CPU had not booted up successfully. These are indicated by those data dumps where the main CPU data dump collected before the fault injection is also empty. This means that the main CPU was not running when the pre fault-injection data dump was collected. Some campaigns resulted in a large number of such cases(30 to 40), and were hence rerun. If only a small number of such cases occurred in a given campaign those experiments were discarded.

Figure 7-12 shows a distribution of the relative percentages of each of the response modes that were observed from the experiments. Table 7-2 provides the actual numbers obtained for each response mode.

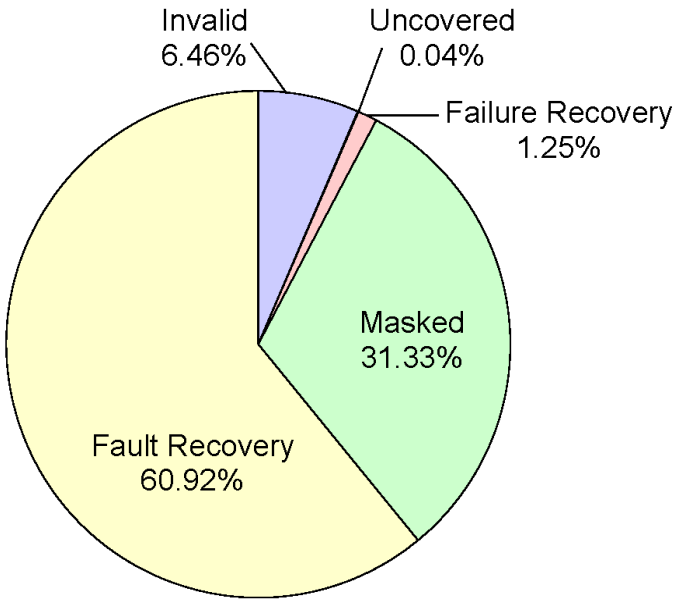

Figure 7-12 Relative percentages of observed responses

Table 7-2 Breakdown of observed response modes.

Response mode	Count
No-response faults	752
Recovered from Faults	1462
Recovered from Failure	30
Local Uncovered	1
Invalid Experiments	155
Total	2400

7.10.2. No-Response Faults

From Table 7-2 it is clear that a significant number of experiments did not produce a system response within the time the system was monitored after the fault injection. These experiments did not produce a system response mainly because the duration of the fault was not sufficient for the propagation of an error or the fault was latent for a period longer than system response monitoring time. Normally, no-response experiments are re-run to determine if they are due to injecting a fault into an unused portion of the memory or CPU register files, or if the location where the fault was injected was overwritten with a valid value immediately after the fault injection thereby undoing the effect of the injected fault. In this case, due to time constraints these experiments could not be re-run. As shown in Table 7-2, no-response faults (i.e., masked faults) comprised 31.33 per cent portion of the observed responses of the fault injection campaigns.

No-response faults are detrimental to the quality of the parameters estimated because statistical confidence is determined by the number of experiments performed. Since these cases must be re-run or discarded, all of such experiments correspond to wasted experimentation time and resources. To reduce the occurrence of no-response fault injections, one must use methods ofthat increase the likelihood of error propagation from an injected fault. Due to this result, a pre-fault injection analysis method was developed. This method is discussed briefly in the next Section, and is more fully described in Volume 2.

7.10.3. Coverage Estimation: A System Level View

This statistical view interprets the probability of system failure or an uncovered failure without regards to any specific failure modes. The simple Bernouli model was used in this effort [Pescodido 2002]. This statistic gives a system view of the fault tolerances capabilities of the entire DFWCS. From Table 7.2 it can be seen that 155 experiments were invalid and hence will be discarded. Also it must be noted that the injected fault was masked in 752 experiments. One might interpret these situations as covered faults because the effect of the fault did not propagate as an error. However, they do not convey any new information about the fault tolerance mechanisms of the system. Hence, adopting the conservative approach, the no response fault injections were discarded for the analysis.

The value for the number of experiments, n, in the expression for the point estimator of coverage defined in equation 7.1 is therefore, 2400-155-752 = 1493 experiments. The point estimate of the coverage is thus given by,

$$\begin{aligned}\hat{C} &= \frac{1}{n}\cdot\sum_{k=1}^{n}\underline{Y}_k \\ &= \frac{1}{1493}\times(y_1+y_2+y_3+\ldots+y_{1493})\end{aligned} \tag{7.1}$$

Since *one* uncovered fault was observed, the value of $\underline{Y}_k$ for that instance will be equal to 0, and equal to 1 for every other case. Hence,

$$\begin{aligned}\hat{C} &= \frac{1}{1493}\times(1+1+\ldots+0+\ldots+1+1) \\ &= \frac{1492}{1493} = 0.99933\end{aligned} \tag{7.2}$$

which corresponds to a non-coverage of $1-\hat{C} = 6.69792\times10^{-4}$. The accuracy of the estimate can be calculated using the expression for the unbiased estimator of variance defined in equation (3.23) as,

$$\begin{aligned}\hat{S}[\hat{C}] &= \frac{\hat{C}(1-\hat{C})}{n-1} \\ &= \frac{0.99933(1-0.99933)}{1493-1} \\ &= 4.4862\times10^{-7}\end{aligned} \tag{7.3}$$

The symmetric 100^{γ}% confidence interval was defined in equation 4.8.7 as

$$(\hat{C}-K_{\gamma}\sqrt{\hat{S}[\hat{C}]}) < \dot{C} < (\hat{C}+K_{\gamma}\sqrt{\hat{S}[\hat{C}]}) \tag{7.4}$$

The 95% confidence interval of the point estimate of coverage that was calculated is thus given by,

$$\begin{aligned}(0.9993-1.96\sqrt{4.4862\times10^{-7}}) &< \dot{C} < (0.9993+1.96\sqrt{4.4862\times10^{-7}}) \\ 0.9979 &< \dot{C} < 1.0\end{aligned} \tag{7.5}$$

This means it can be said with 95% confidence that the coverage proportion $\dot{C}$ lies between 1.0 and 0.9979. Furthermore, the value of actual system coverage parameter C and the coverage proportion $\dot{C}$ are very close in values if the fault tolerance mechanism is considered fair, that is, it handles all faults similarly and there is no difference in the way it handles fault that occur frequently as compared to those that occur less frequently.

Some discussion should be centered on the nature of the uncovered fault in the system wide fault tolerance context of the DFWCS. The DFWCS has three basic fault detection levels. The first is the main CPU validation tests – these are considered self-tests. They detect anomalies and errors in the data flow of the control processing algorithms. The second level of fault detection is the BU CPU. If the main CPU fails, then the BU CPU takes over control. The PID controllers determine the "health status" of the main and backup CPU's by monitoring three signals from each CPU as shown if figure 7-13. The first is the power fail signal which changes state on a power failure. The second is CPU self-test signal when the main or backup CPU detects a problem with the processing algorithm. The third signal is the WDT signal from the

external watch dog timer. If the WDT signal times out (500ms), the then signal changes state commanding the PID controllers not to use the outputs from the failed CPU. If both main and backup CPU's fail, then the PID's switch to manual mode.

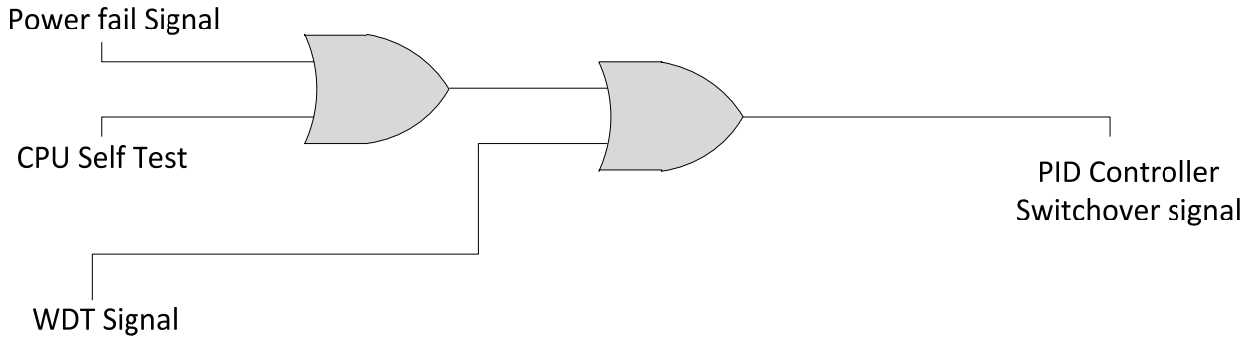

Figure 7-13 Fault detection signaling

During the Fault Injection testing at CCNPP, the watchdog timer was disabled because the fault injection process was causing the main CPU execution to slow down enough to occasionally trip the watchdog timer. The local uncovered fault that was observed escaped detection by the backup CPU and the main CPU self-tests; however, it cannot be concluded that this error would have escaped the watchdog timer because the watchdog timer had been disabled.

7.11. Observations, Lessons Learned and Recommendations

7.11.1. A Task Oriented look at the Overall Effort

The preceding sections have presented the design, development, implementation, and deployment of the UVA fault injection based dependability assessment methodology. The importance of the work in relation to this effort is in what greater lessons were learned that can be used as the research progressed to Phase I and Phase II of this effort. Many of the activities endeavored on this effort were not accounted for in the original methodology. To better understand on this aspect, the activities necessary for applying fault injection and the amount of time spent in each activity were reviewed. Figure 7-14 shows the breakdown in months spent for each task. The order of the tasks in the chart follows the order in which they were conducted. These issues are discussed in the following section.

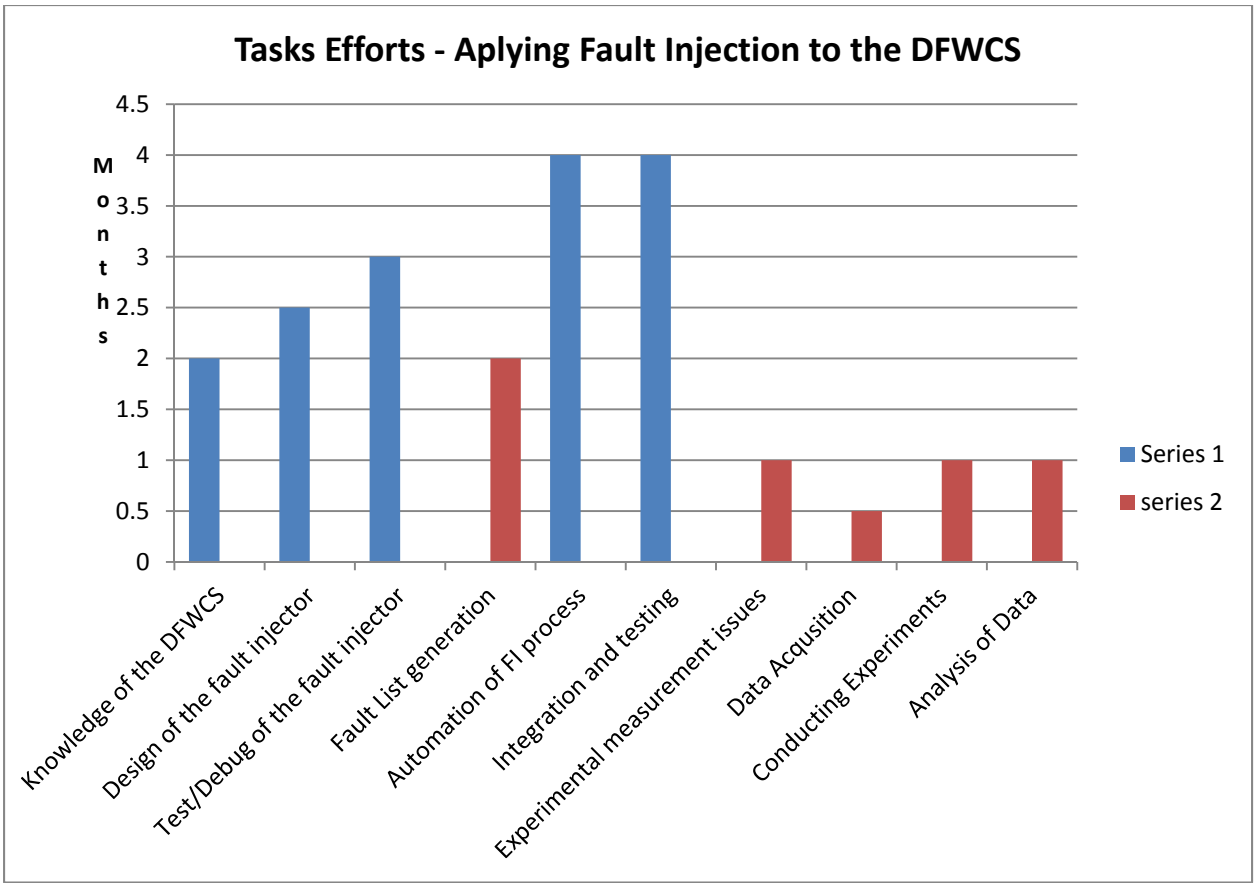

Figure 7-14 Task Breakdown and Level of Effort for Each Task

Referring to Figure 7-14, a majority of the effort (approximately 70 percent) was spent developing the overall means to realize fault injection capability (series 1). The other 30 percent of the effort (series 2) was related to the application of the UVA methodology to the DFWCS. As shown in Figure 7-14, a significant amount of work was needed before the methodology was ready to apply to the DFWCS. In addition, a few of the activities conducted in this effort are absent from the UVA methodology. Namely, the automation of the fault injection, and the integration/testing tasks were not well represented in the methodology. The above figure suggests that significant steps or activities must be integrated into the methodology to provide better guidance on applying fault injection to physical systems. Section 7.11.2 discusses the challenges and observations on the applying the methodology to the DFWCS.

7.11.2. Review of the Problems Encountered

The problems and challenges faced during this effort are briefly summarized to characterize the lessons learned in a broader context. The problems encountered are categorize in three classes: *fault injection, integration, and measurements*. *Fault injection problems* encompass issues with fault models, realizing the fault models with the ICE based fault injector, performance, intrusiveness, controllability, etc. *Integration problems* are concerned with interfacing the fault injector within its operational setting. These issues include, representing the operational profile of the system, automation support of fault injection, fault list generation support, etc… *Measurements problems* are related to the information or data that is needed for estimating the measures of interest such coverage. Specially, the issues here are concerned with viability of acquiring data in way that is meaningful and the uncertainty of the measurements can be characterized.

Fault Injection Issues

The following fault injection issues were encountered during this effort:

- *No-response faults* – faults were injected at random times into the pre-determined locations where variables of interest reside. However, because the injections were randomized in time, the effects of the fault were often masked by the DFWCS program overwriting the corrupted variable with a valid variable. Approximately 1/3 of the faults injected were of this class. Since these no-response faults conveyed no information on the effectiveness of the fault detection and fault tolerance features of the DFWCS, these experiments consumed 1/3 of the experiment time with no contribution.

- *Low Performance of the ICE based fault injector* – The ICE based fault injector was capable of injecting different fault types into the DFWCS with good controllability and precise fault activation. However, the time it took to temporarily halt the processor, modify the contents of a register or memory location, and resume execution was significant. For transient fault models this delay was on average about 150ms. However, on occasions delays over 500ms were observed, and in these cases the process of injecting the fault into the DFWCS would trip the watchdog timer. Both of these delay times are too high. The time intrusiveness of fault injection should be as low as possible on the order of fractions of the control iteration cycle.

- *Fault models* – The ICE based fault injector can emulate both transient and permanent faults. The permanent fault model was implemented through repeated triggering on a specific location by the ICE machine debugger traps. The time penalty for this repeated triggering was substantial. The time delay over 1 iteration cycle (100ms) was from 500ms to 1.5secs for most permanent faults. Again, this amount of time penalty is too high.

- *Fault list Generation* – Fault List were generated from the compiler map and link files. The process was complicated by the fact the DFWCS used dynamic memory management which re-allocated the data and code segments after every reset.

Integration Problems

- *Operational Profile Flexibility* - The DFWCS was integrated into a computer called CPAC which provided recorded plant data to the DFWCS analog and digital inputs. The CPAC computer provided operational data on a few modes operation, namely, low power, high power, and transitional. To better stimulate the operational profile of the DFWCS, the capability to stimulate various event behaviors (e.g., LOCAs) from the CPAC would have been a desirable feature.

- *Interference with onboard fault detection features* – While interfacing the ICE based fault injector to the DFWCS at CCNPP two instances were encountered in which the fault injection process unintentionally triggered the fault detection mechanisms of the DFWCS. The first was the power fail mask, which was described in Section 7.7. The second was the occasional tripping of the watchdog timer when the fault injection process took too long to complete. Both of these issues were resolved, however, these types of problems indicate that fault injection needs to be (1) mindful of the fault tolerance features it is trying to test, and (2) the intrusiveness of fault injection should be keep as low as possible.

- *Automation of fault injection* – The automation of the fault injection process was a significant undertaking. Referring to Figure 7.13, it took about 4 months to design, develop and test. Along with integration, the combined tasks took almost 8 months or about ½ of the total effort of the project. Most of this effort was spent developing a specific fault injection environment for the DFWCS. This type of effort is non-reoccurring, once the fault injection environment is built it can be used over and over again on the DFWCS, but not on other digital I&C systems.

- *Environmental Disturbances* –Several noise and EMI problems were encountered during the development and integration of the ICE-based fault injector into the DFWCS. Most of these issues were due to the electrically noisy environment of the power plant.

Data Measurement and Analysis Problems

- *Data acquisition of error logs and error state* – The way the error logs from the DFWCS were typically generated for plant operations was found not to be optimal for data analysis. In normal plant operations, the error log files are continuous logs of activity over many days. For fault injection, a set of error log files for each fault injection campaign that reflects the errors detected for that campaign only were needed. This is necessary to trace a specific injected fault to set of error responses for that fault. Therefore, the error reporting functions on the DFWCS were manipulated to ouput files after every fault injection, and then clear the error logs after monitoring period was over.

- In addition to error logs, the research attempted to monitor the outputs of specific error detection mechanisms at a lower level than the error log. However, this proved to be problematic for two reasons. The first was locating the outputs of the error detection mechanism in the DFWCS code was not straightforward; however, the variables associated with various assertion tests and diagnostics were found. The second and most limiting factor was the acquiring real-time trace data on the outputs of the error detection mechanisms in way that did not perturb the operation of the DFWCS. Specifically, the real-time trace collection of ability of the ICE machine tended to slow down the operation of the DFWCS CPU. See the next topic for more information.

- *Processor state acquisition* – One of the promises of using an ICE based fault injector is that one can save a *trace* of the processor state at any time during the operation of the digital I&C system. Traces are detailed cycle-by-cycle timing information of each instruction executed on the target CPU. The ability to view wall-clock time of each instruction in the form of a time stamp or incremental time taken by each instruction can be useful for investigating the possibility to find equivalent faults and for implementing fault expansion. It could also be used for fault and error latency measurements.

- The PowerPack ICE machine provides real time full-speed tracing capability. The trace buffers are however limited to 128K frames of address data and signal traces. This restricts the amount of instructions that can be logged at a given time. Due to the small size of the trace buffer, it was observed that only (roughly) 6.5 milliseconds of trace data just before the CPU halts due to memory availability in the trace buffer. This corresponds roughly to about 250,000 instructions. This puts a restriction on the amount of history that is available for analysis.

- A more significant problem was observed when the ICE machine was put into real-time trace mode, the time delay penalty for fault injections went up considerably from about 150ms to over 500ms. At this time, it was not known whether these problems were

specific to the Microtek ICE that was used in this effort or were more or less universal for all ICE machines.

- *Global time reference* – In order to measure time oriented attributes such as error detection latency, there must be a means to establish a base time reference for the measurements. For example, when the fault is injected into the DFWCS, the system responds, records the error response as an event in an error log file, and the DFWCS timestamps the event in error log file. For reliable measurement, the fault injection timestamp and the error response timestamp should be synchronized to a precision time reference. For this effort, the timestamps of the measurements were not synchronized to a precision time reference. The principle reason for this was twofold: The DFWCS timestamps the error messages with its own internal clock, and the fault injector used a different internal clock for its timestamp. Synchronizing the two internal clocks turned out to be somewhat involved, and would have required additional time and effort to complete.

- *Data Analysis* – The price paid for automated fault injection is vast amounts of data, e.g., 100's Mbytes of data are not uncommon for a series of fault injection campaigns. Because manual examination of each of the 2,400 dump files that were created from the fault injection campaign was impractical, a set of parsers were written to extract specific elements of information from these files and then compile the elements into a more suitable format for analysis. Even using automated parsing programs to help with data reduction and filtering, this part of the effort required much more time than anticipated - almost a month.

7.11.3. Summary

The overall effort of applying fault injection to the DFWCS in viewed in a positive light despite the many challenges faced during the effort. Most of the issues and problems were resolvable problems and, with time to refine and iterate, better solutions that what were improvised on a limited timeframe and budget likely could be found. The significant lessons learned that impact the methodology are centered on the support activities (e.g automation for fault injection, measurement based activities, etc.) for the methodology. As alluded to earlier, the UVA safety assessment methodology was developed and refined primarily from simulation based fault injection perspective. As such, some of the issues encountered were specific to physical based fault injection and thus somewhat unfamiliar to the research team. However, with this effort significant knowledge and comprehension of applying fault injection to real digital I&C systems was gained. This was one of the first successful efforts of applying fault injection to digital I&C system beyond Pin-level fault injection.

To improve the methodology in a direction that better supports physical based fault injection, the methodology should be augmented with additional steps called *support steps*. As such, the elements of the methodology can be partitioned into two classes. The first class is what activities are *necessary* to perform fault injection according to the FARM model attributes. These steps are the elements in the methodology now. The new steps are related to what *support* is needed to carry out fault injection. As example, step 4 of the methodology is "define and develop operational profile of the system". A support step of this activity is how you select the operational profile, how you properly integrate it into the digital I&C system environment, what types of equipment are needed for the integration of the operational profile, how do you present the operational profile data to the system so that it is representative, etc. The methodology should be more than a theory based methodology, it should inform the user on what support is needed to realize it.

7.12. References

[Arlat 2003] Arlat, J, Y Crouzet, J Karlsson, P Folkesson, E Fuchs, and G H Leber. "Comparison of Physical and Software-Implemented Fault Injection Techniques." *IEEE Transactions on Computers*, no. 52 (2003): 1115-1133.

[Cutright 2003(b)] E. Cutright, T. DeLong, B.W. Johnson. *Generic Processor Fault Model.* UVA-CSCS-NSE-004, Charlottesville: University of Virginia, 2003(b).

[George 2007] George, N.J. *Robust Fault Injection Through In-Circuit Emulation.* Charlottesville, VA: University of Virginia, Masters Thesis, 2007.

[Pescosolido 2002] Pescosolido, M. "Statistical Models for Coverage Estimation." *School of Engineering and Applied Science Masters Thesis.* University of Virginia, May 2002.

[Maia 2005] R. Maia, L. Henriques, R. Barbosa, D. Costa, H. Madeira. "Xception Fault Injection and Robustness Testing Framework: A Case-Study of Testing RTEMS." *VI Test and Fault Tolerance Workshop.* 23rd Brazilian Symposium on Computer Networks (SBRC), 2005.

[Aldemir 2009] T. Aldemir, S. Guarro, J. Kirschenbaum, D. Mandelli, L.A. Mangan, P. Bucci, M. Yau, B.W. Johnson, C. Elks, E.Ekici, M.P. Stovsky, D.W. Miller, X. Sun, S.A. Arndt, Q. Nguyen, J. Dion. *A Benchmark Implementation of Two Dynamic Methodologies for the Reliability Modeling of Digital Instrumentation and Control Systems.* Regulatory Guide NUREG/CR-6985, NRC, 2009.

8. IDENTIFY KEY CHALLENGES WITH RESPECT TO DIGITAL I&C SYSTEMS

This section discusses and summarizes the important challenges to fault injection-based assessment methods identified in previous Sections. Based on the issues discovered, the challenges, the impact of these issues have on the methodology, and finally the identification of key issues pursued in phase 1 and phase 2 of this research are discussed in this section.

8.1. Better Measurement Tools and Practices

Tools for experimental assessment of dependability properties should be treated for what they really are: measurement instruments. First of all, fault injection being a measurement based assessment process depends on sound measurement practices. Secondly, since measuring a quantity (the measurand) consists in quantitatively characterizing it, a clear and univocal definition of the measurands is of uttermost importance [Bucher 2004]. The science of measurement, *Metrology* has developed theories and good practice rules to make measurements, to evaluate measurements results and to characterize measuring instruments. The main metrological properties of concern to fault injection are uncertainty, repeatability, resolution, sensitivity and intrusiveness should be precisely identified in the methodology (and supporting tools). In a similar way, the results obtained using a tool should include uncertainty evaluation and, when comparing results achieved through different measurement methods, compatibility of measurement results should be assessed.

As noted in Section 7, Digital I&C systems are not typically designed to be monitored to the level required for dependability evaluation by fault injection; a methodological approach for their observation is thus needed. In distributed digital I&C systems things are even more complex, for the lack of central control, and for the difficulties in obtaining a precise global time and an accurate view of the global state of the system are all noted challenges. Measurement issues are very rarely addressed in the major conferences and scientific journals in the area of dependability. The most notable paper in this area [Bondavalli 2007(a)] raised the problem of awareness of measurement theory in evaluating dependability attributes of computing systems. In the paper, a set of well-known tools for experimental assessment of dependability and papers describing results of experimental evaluations are analyzed, identifying whether and to what extent the most important metrological properties and attributes, namely *uncertainty*, *repeatability*, *resolution*, *intrusiveness* and results *compatibility*, are taken into account. Currently, it is the only document that presents a deep state-of-the-art in this area.

Dependability and resilience measurements on computing systems involve a wide variety of measures, from *discrete* measures, such as error counts, loss of messages, to *continuous* measures that refer to the dynamic behavior of the system under evaluation, which include fault and error latency, real-time response, etc…. Dependability-related measurements are very often based on time measurements, for example because the measurand is a time interval, or because the measurement result is obtained through indirect measurements based on timestamps.

Based on the experiences and lessons learned from the DFWCS, certain measurements are a function of the system under test and as such the assessor has limited ability to alter the measurement process to suit the needs of the methodology. Time-stamps on status messages and error messages produced by the target systems are examples. What is more, the assessor of the system has little control over how the time-stamps are applied, when they are applied, and the resolution of the time measurement. In these cases, it is still important to try to

characterize the in-situ measurement and its uncertainty so that a more complete picture of the system response can be obtained accounting for the uncertainties in the measurements.

Part of the effort developing “measurement awareness” for fault injection and how it can be used to inform the methodology was addressed in Phase 1 and Phase 2 of the research.

8.2. Fault Model Selection

Sections 3.3 (FARM model), 4.5 (Fault Models), and Section 5 addressed some the issues regarding fault models for fault injection. The principle concerns raised are:

- *Representativeness* – Are the fault models appropriate for the target system and its operational environment.
-
- *Guidance in using the fault model* – How to apply the fault model so that it is representative of the faults that occur.

Fault model selection almost always begins with first selecting fault models that represent faults that the system was designed to detect/tolerate by the designer, that is, the *fault hypothesis* of the system [Elks 2005]. These may include a number of fault classes as indicated in Figure 1.7, such as hardware faults (permanent and transient), commission faults, omission faults, interaction faults, etc.

However, just because a system was designed to tolerate certain classes of faults, does not mean that it is not susceptible to other types of faults. Thus, the fault hypothesis of the system does not guarantee all representative faults for the system or for that matter any representative faults for the system have been selected.

The evidence of fault representativeness comes from *empirical or experiential* sources that indicate that certain components have certain failure mechanisms under certain conditions. These failure mechanisms are then paired with the most appropriate fault model to represent the failure mechanism.

Empirical evidence of faults/failures in digital I&C systems is often guided by the use of failure databases based upon observed failures of systems in the field. Databases of this type are most useful for characterizing failures when the failure data is homogenous to a specific type or make of system. However, these databases are very rare, and most often proprietary to specific manufacturers. Beyond this use, there are number of concerns about relying solely on failure database for determining fault representativeness.

First, faults and failures that occur in systems are often *technology dependent*. A digital I&C system built from 1990’s electronic technology will not have the same fault susceptibility profile as a contemporary digital I&C system. For example, very large scale integration (VLSI) semiconductor manufacturing processes in the 1990’s were building semiconductors with relatively large transistor feature sizes, and as such, the susceptibility to transient faults due to cosmic particle strikes and EMI were almost non-existent. This is not the case in today’s contemporary ICs.

Secondly, the operational or environmental context of a digital I&C system is important in characterizing its fault and failure susceptibility profile, and this fact is poorly represented in most databases.

Lastly, failure databases often report how the system failed, but rarely report the underlying cause or failure mechanism of the system.

Experiential data is usually collected from accelerated testing methods and/or high fidelity simulations of semiconductor physical processes to determine failure mechanisms. These methods usually determine the upper bound or worst case scenario for failures-in-time (FIT) for a component.

Taken altogether, there are no short cuts toward determining representative fault models or fault-loads for digital I&C systems. The most prudent approach is to employ an "all-sources" approach where all available information is used to select appropriate fault models.

The selection of fault models for digital I&C systems should be more structured and this process should be included in the fault injection methodology as a support step. The steps of the fault model selection process should include the following activities:

- Examination of the fault hypothesis of the system.
- The use of a structured fault taxonomy (e.g., like the one presented in Section 1.7) as a starting point to help guide the fault model selection process.
- Examination of failure data from empirical and experiential data sources that is relevant to the system.
- The operational context of the system.
- The interaction context of the system.

The output of a fault model selection process should produce a set of faults that is relevant to a particular digital I&C system, but more importantly the process of fault model selection should produce an audit or evidence trail so the assumptions and factors for determining the fault models can be assessed during the licensing and review activities.

8.3. Automated Fault Injection Environment

As indicated in section 4.7 and in Section 5, a well formed fault injection environment is one of the most important aspects of a fault injection based methodology. "Well-formed" means the fault injection environment follows and refines the theory of the FARM model (Section 3.5), and allows the fault injection process to be a statistically guided process as described in section 3.7. Furthermore, the fault injection environment plays a crucial role in the data collection and measurement of the responses which are important producing the measures of dependability (e.g., coverage, error latency, etc.). For highly reliable or safety critical digital I&C applications where the probability of system failure is very low (e.g., on the order of 10^{-7} failures/year), statistical fault injection requires that a large number of experiments be performed in order to obtain a commensurate degree of confidence in the data used for model parameters. For example, if a critical component in a target system (e.g., voter) has an assumed fault coverage of 0.99, then to statistically quantify the fault coverage parameter with 10% confidence bounds approximately 1000 fault injection experiments must to be performed. Even though various variance reduction estimation methods can be used to reduce the number of experiments, significant time and resources will be required to perform hundreds or thousands of experiments. Thus, automation of experiments is an essential feature that enables collecting large volumes of data with little need to manually intervene in the experimentation process.

As indicated in Section 5, certain properties of fault injection are required to ensure controllable, repeatable and credible fault injections. Thus, the user must have the ability to manage the

types of faults to be injected into the system, where they are injected, how they are injected, and when they are injected. Additionally, the responses to the fault injections must be acquired in a manner that allows the responses to traced back to the faults so that any fault injection trial can be repeated as needed to reproduce the system response.

As indicated in Section 7 experiences with the DFWCS fault injection effort show that designing, developing a well-formed fault injection environment is a non-trivial matter. In the next phase of this effort a considerable amount of time and thought was devoted to the design, development, and implementation of a well formed fault injection environment for digital I&C systems. While the methodology presented in Section 4 is intended to be somewhat "neutral" to the selection of fault injection techniques, more guidance on what the fault injection environment should provide to the methodology and what the methodology expects from a fault injection technique is needed. The "tie in" from the requirements of fault injection theory (FARM model) to the specification of the fault injection environment is needed.

8.4. Operational Profiles

As indicated in Section 4, operational profiles and workloads of the target system are required to set the operational and environmental context of the system [Musa 1989]. The operational profiles must be representative of the different system configurations and workloads that would be experienced in actual field operations. Context is important in fault injection. For a fault injection based assessment methodology, the operational profiles must represent the input conditions and system interactions that can occur not only nominal operations, but also in off-nominal operations and more importantly during "accident" event scenarios.

Gathering profile real plant data across all of these domains of operations is challenging task. Not all plants in operation have experienced accident events. Data may be limited due to proprietary sensitivities. In order to provide a diverse and representative set of operational profiles for the target system, the use of high fidelity NPP simulator tools to generate nominal, off-nominal, and accident event profiles is the most promising way forward. The challenges in this approach are (1) determining how to integrate the thermo-hydraulic modeling tools like the TRAC/RELAP Advanced Computational Engine (TRACE) [Commission 2011] into the fault injection environment to act as operational profile generator for the target system, and (2) how to coordinate the selection of the operational profiles to the fault injection process. At present, the methodology provides guidance on how to use an operational profile for fault injection, but does not provide detailed guidance on the various ways to realize an operational profile. Further research investigate the use of NPP simulator tools to act as an operational profile generator for fault injection.

8.5. Fault List Generation

Generating fault lists for a fault injection experiment or campaign is a critical activity for fault injection. Most fault list generating techniques found in the literature were developed for simulation based approaches and not physical fault injection approaches [Smith 1993; Benso 2003]. UVA found that these approaches did not translate well into existing I&C systems because simulation-based approaches allow a high degree of controllability and observability which may or may not be achievable in real digital I&C systems.

A *fault list* is a sample set of faults taken from the fault space of the target I&C systems. Specifically, for a single fault notation in a fault list, each entry identifies.

- The type of fault to be injected – Governed by the fault model selection.
- Where the fault is to be injected – Where the corruption is to take place with respect to program execution behavior or component use.
- When the fault is injected – At what time the injection takes place, either relative to an event or when a resource is in use, or randomly selected.
- How long the fault is injected – The persistence of the fault with respect to the time domain.
- The error mask of the fault – What values represent the fault injection process with respect to a resource in use or a component.

The fault list can be thought of as a set of directives to the fault injector apparatus. Each of the directives is under experimental control of the experimenter. The fault list is used to instruct the fault injection process according to a particular campaign purpose. The fault list is strongly tied to the fault injection environment and its capabilities to emulate the faults of concern.

An important aspect of fault list generation is improving the efficiency and effectiveness of the fault injection process. Improving the efficiency and effectiveness of fault injection is often called error acceleration [Chillarege 2002] or more recently pre-injection analysis [Sekhar 2008; Barbosa 2005]. Pre-injection analysis is method to guide the fault injection process to produce more effective and efficient results.

Pre-injection analysis is defined by a set of rules that forces fault injection experiments to push the limits of the measurement on the probability of systems failure. Figure 8-1, shows the system in three states, a good state, an error state and a failed state. Concisely, failure acceleration is achieved when:

$$\text{Error Latency} \rightarrow 0$$
$$\text{Fault Latency} \rightarrow 0$$
$$P(F) \rightarrow 1$$

where *p(F)* is the probability of a fault causing a failure.

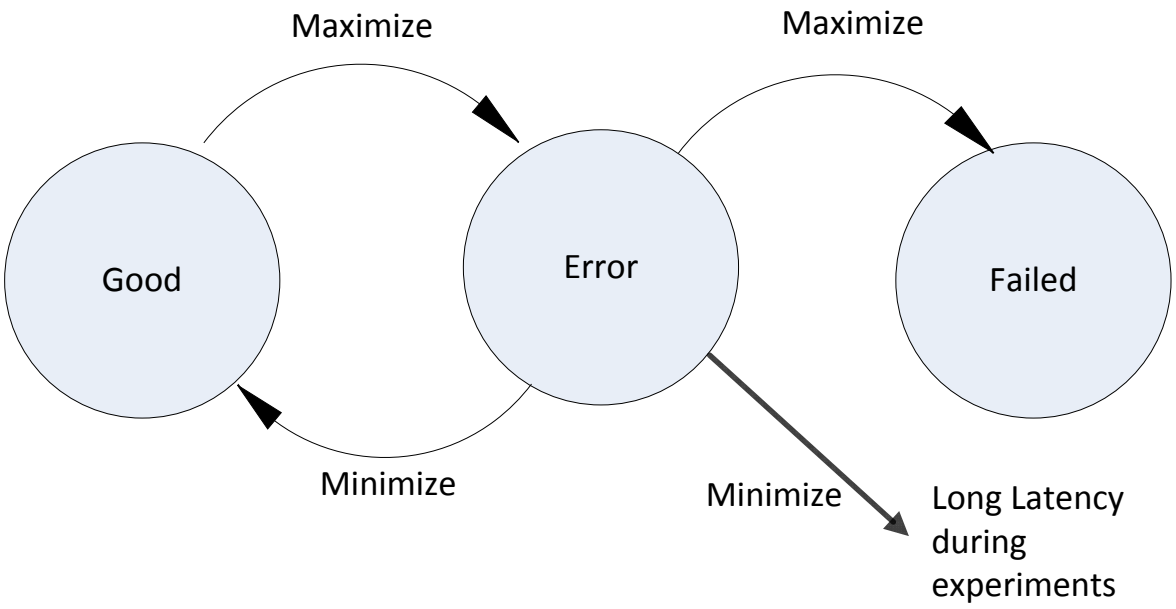

Figure 8-1 Failure and error acceleration through pre-fault injection analysis

The above conditions can be stretched so long as the semantics of the fault model are not altered in the process of altering the controls: namely, the fault injected, and the operational environment. In a fault injection experiment, one constructs a fault, injects it into the system and then waits for the system to fail. This would amount to one run of the fault injection experiment. It is not necessary that each of the fault injections cause a failure. Since it is always possible that the error condition created by the injected fault is compensated by fault tolerance mechanisms, or it is not effective due to non-use of the faulted data. On the other hand, the error condition can lay latent for a long time, and not cause a failure during the time allocated for observation in the experiment.

Pre-injection analysis is a means to reduce or eliminate the "no-response" and the long fault latency problem associated with fault injection. Being a statistical experiment, fault injection testing may require a large number of experiments to be conducted in order to guarantee statistically significant results. Thus, efficiency of the fault injection testing is important. For example, the resources for fault injection testing include in addition to the automated fault injection setup the time required to perform a large number of experiments. Each experiment involves initialization of the system (which includes reset and initialization of the target I&C system), followed by application of an operational profile, the actual fault injection, and then the monitoring of the system for a period of time after the fault injection.

The amount of time required to the initialize digital I&C hardware can be several minutes because of the systematic nature of the diagnostics and self-tests the system initiates at startup. The total time required to perform all steps for one fault injection experiment is typically two to four minutes on contemporary digital I&C systems. This limits the number of fault injection experiments to 300 – 500 experiments per day. Therefore, ensuring that a large percentage of fault injection experiments result in producing a response from the target system is very important.

Referring to Figure 8-2, the error response space is characterized by monotonically decreasing space of fault severities. The largest space is the space of faults that produce no errors.

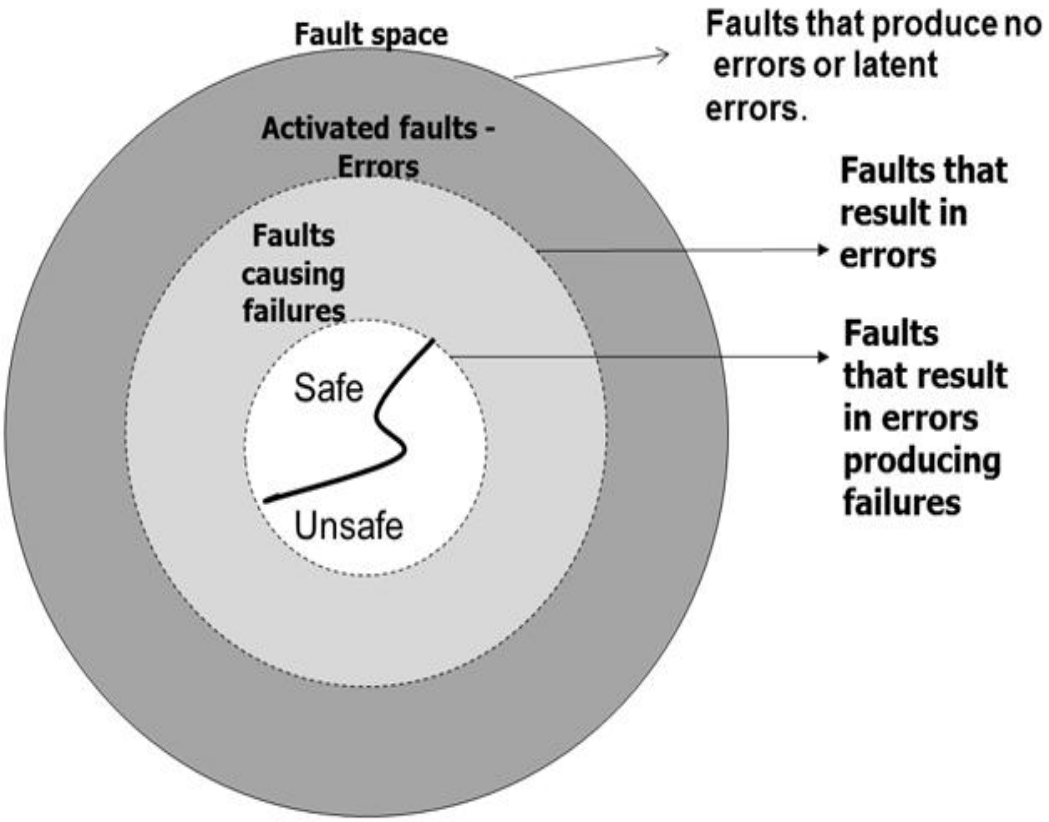

Figure 8-2 Error response space

The next largest space is the space of faults that produce errors, followed by the space that produces failures. From a practical perspective, the error response spaces are really an artifact of the nature of a digital processing system. A typical digital I&C system will have significant memory space (hundreds of megabytes is not uncommon), large number of processor register files, special purpose configuration registers, and (relatively) long control cycle times (50-200ms). With random fault injection experiments (e.g., with no regard to when and where a fault is injected), a large fraction (up to 90%) of fault injection experiments may have no-response outcomes [Sekhar 2008; Barbosa 2005].

A large percentage of these “no-response” outcomes resulting from fault injections are due to non-use of the corrupted data by the executing program. For example, a randomly generated fault could be injected into a memory location that is not used by an application, or could be injected into a processor register that is not in use by the application. These instances in which the injected system would not respond to an injected fault do not convey meaningful information about the fault tolerance capabilities of the system under test. Since time has an associated cost value, if the efficiency of the fault injection campaign is low, then the cost of the fault injection campaign is increased.

As part of the next phase of work, pre-fault injection analysis methods for physical fault injections to improve the efficiency and effectiveness of fault injections were developed. This effort will complement previous fault list generation methods developed by UVA for simulation based fault injection approaches and other methods reported in the literature [Chillarege 2002].

8.6. High Performance Adaptable Fault Injection

As discussed in Section 5 of this report, the need for fault injection techniques to support various fault models for fault injection, and to do so in manner that is minimally intrusive, controllable, repeatable and reproducible is critical to the application of fault injection to digital I&C systems. Section 5 extensively reviewed contemporary fault injection methods to help guide the selection of a particular fault injection for digital I&C systems.

One of the realizations from performing fault injection on the DFWCS, was the challenge of trying to obtain a high degree of controllability and observability while maintaining a low level of intrusiveness. Various methods were investigated (ICE based fault injection and SWIFI) that could only attain controllability or low-intrusiveness but not both at the same time. In addition, it was realized that a fault injection tool or environment should have the capability to choose from several techniques, and provide the capability to inject multiple faults into the system.

It was also, realized that there should be a common portable interface between the fault injector and the fault injection environment. In the DFWCS effort custom interfaces between the fault injector and the fault injection controller that had a low degree of reusability had to be developed. Based on these experiences and lessons learned, it became apparent that a new implementation approach for injecting faults in digital I&C systems was needed if fault injection was to become practical for digital &C systems.

Accordingly, UVA recognized that many of the fault injection techniques UVA and others used in experiments could be united on a single high performance FPGA-based fault injection module. By moving to a single multi-purpose fault injector, designed specifically to support diverse fault injection on digital I&C systems, fault injection could be optimized around performance (e.g., minimal intrusiveness) and controllability simultaneously.

Furthermore, by uniting a variety of fault injection techniques to a common interface onto a single platform, the integration of the fault injector into the digital I&C system is more or less consistent from one digital I&C platform to the next. This aspect becomes important when fault injection is used as *benchmarking activity*. That is, the same set of faults, operational conditions are applied to each digital I&C system by the same fault injection technique to form an objective basis for comparison or compliance. To address these concerns, a FPGA based high performance adaptable fault injector was developed to be applied to both of the benchmark systems.

8.7. References

[Arlat 2003] Arlat, J, Y Crouzet, J Karlsson, P Folkesson, E Fuchs, and G H Leber. "Comparison of Physical and Software-Implemented Fault Injection Techniques." *IEEE Transactions on Computers*, no. 52 (2003): 1115-1133.

[Barbosa 2005] Barbosa, R., Vintern, J., Fokesson, P., Karlsson, J. "Assembly-Level Pre-Injection Analysis for Improving Fault Injection Efficiency." *Lecture Notes in Computer Science, vol. 3463*, 2005: 246-262.

[Benso 2003] Benso, A., S. Di Carlo, G. Di Natale, P. Prinetto, I. Solcia, and L. Tagliaferri. "FAUST: fault-injection script-based tool." 2003.

[Bondavalli 2007(a)] Bondavalli, A., Ceccarelli, A., Falai, L., Vadursi, M. "Foundations of Measurement Theory Applied to the Evaluation of Dependability Attributes." *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks.* Washington, D.C.: IEEE Computer Society, 2007(a). 522-533.

[Bucher 2002] Bucher, J.L. *The Metrology Handbook.* American Society for Quality, 2004.

[Chillarege 2002] Chillarege, R., Goswami, K., Devarakonda, M. "Experiment Illustrating Failure Acceleration and Error Propagation in Fault-Injection." *IEEE International Symposium on Software Reliability Engineering.* ISSRE 2002, 2002.

[Commission 2011] Commission, U.S. Nuclear Regulatory. *Computer Codes.* April 2011. http://www.nrc.gov/about-nrc/regulatory/research/comp-codes.html (accessed 2011).

[Cutright 2003(b)] E. Cutright, T. DeLong, B.W. Johnson. *Generic Processor Fault Model.* UVA-CSCS-NSE-004, Charlottesville: University of Virginia, 2003(b).

[Elks 2005] Elks, C. *A Theory of Run-Time Verification.* Charlottesville, VA: University of Virginia, Ph.D. Thesis, 2005.

[George 2007] George, N.J. *Robust Fault Injection Through In-Circuit Emulation.* Charlottesville, VA: University of Virginia, Masters Thesis, 2007.

[Musa 1998] Musa, J. *Software Reliability Engineering.* McGraw Hill, 1998.

[Pescosolido 2002] Pescosolido, M. "Statistical Models for Coverage Estimation." *School of Engineering and Applied Science Masters Thesis.* University of Virginia, May 2002.

[Maia 2005] R. Maia, L. Henriques, R. Barbosa, D. Costa, H. Madeira. "Xception Fault Injection and Robustness Testing Framework: A Case-Study of Testing RTEMS." *VI Test and Fault Tolerance Workshop.* 23rd Brazilian Symposium on Computer Networks (SBRC), 2005.

[Smith 1993] Smith, D T. "A Malicious Fault List Generation Algorithm for the Evaluation of System Coverage." Tech. rep., In Proc. Annual Reliability \& Maintainability Symp, 1993.

[Aldemir 2009] T. Aldemir, S. Guarro, J. Kirschenbaum, D. Mandelli, L.A. Mangan, P. Bucci, M. Yau, B.W. Johnson, C. Elks, E.Ekici, M.P. Stovsky, D.W. Miller, X. Sun, S.A. Arndt, Q. Nguyen, J. Dion. *A Benchmark Implementation of Two Dynamic Methodologies for the Reliability Modeling of Digital Instrumentation and Control Systems.* Regulatory Guide NUREG/CR-6985, NRC, 2009.

9. RESEARCH AND IMPLEMENTATION PLAN FOR APPLYING FAULT INJECTION TO THE BENCHMARK SYSTEMS

9.1. Introduction

The final step in the research methodology for this report was to create a research and implementation plan to realize the UVA fault injection based safety assessment methodology for the benchmark systems. This plan has two categories of tasks. The first task category includes items that required additional research and development to determine their potential for implementation. As such, it was realized early that this would be on-going work for the project. The second task category involved tasks that needed little or no additional research effort to implement to determine their overall effectiveness. The second category should be viewed as items that need to be accomplished in order to support the overall research objectives.

9.2. Overall Task Plan Structure

After reviewing the lessons learned on the DFWCS project, and assessing the level of effort as shown in Figure 7-14, it was realized that tasks needed to be allocated on the basis of support of research objectives. From previous efforts with the DFWCS project, it was realized that a significant amount of time would be dedicated in the beginning gaining knowledge of the benchmark systems, instrumenting the systems into a test environment, and developing application software for the benchmark systems. These task activities do not directly address the research objectives, however, as support tasks that are necessary so that the research on the benchmark systems can be conducted. The key challenges identified in Section 8 will serve as on-going research tasks in attempt to find resolution to the challenges with respect to Digital I&C systems.

The level of effort for each task was gauged at the outset of each phase of the project and assessed as the research progressed so that (1) adequate resources could be allocated or (2) scoping of the task was done to manage the overall level of effort in order to gain results. Figure 9-1 shows the intended overall task plan structure. These were the major task categories for the effort. This task plan was meant to serve as a guide to carry out the tasks associated with the project. However, this research was highly dynamic with a high potential for encountering new problems, and as such tasks were adjusted, and tasks were added to cope with emerging problems and issues. The tasks are described in the following sections.

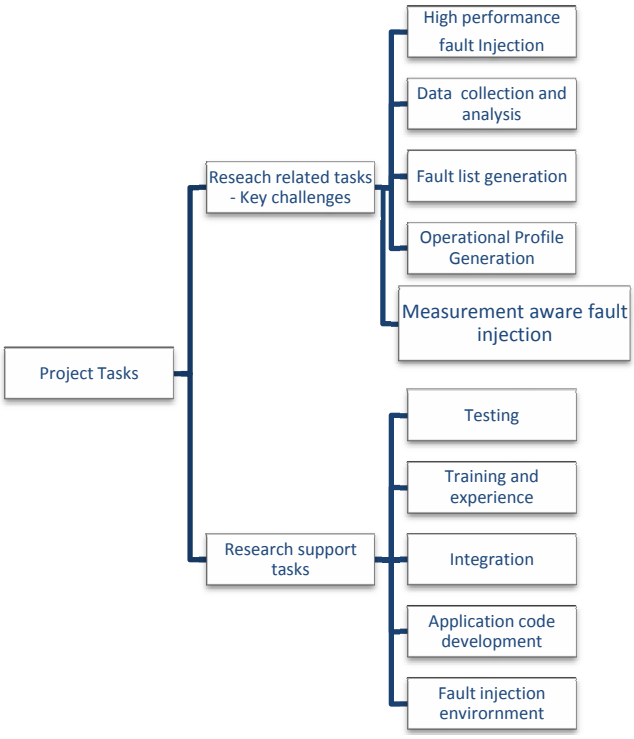

Figure 9-1 Project task organization

9.2.1. Research Oriented Tasks

The research oriented tasks in Figure 9-1 are specifically tied to key challenges identified to Section 8.

High performance fault injection – As discussed in Sections 5 and 8 of this report the need for fault injection techniques to support various fault models for fault injection, and do so in manner that is minimally intrusive, controllable, repeatable and reproducible is critical to the application of fault injection to digital I&C systems. This task will investigate, design and develop and implement new methods to achieve these goals. This effort will be assigned to masters level graduate student for a period of 2 years with supervision from the Co-PI's. The intent is to have the partial capability ready to use in the latter stages of phase 1 and fully deployed in phase 2. This task is considered to be a high risk effort, as such, less risky back up fault injection methods will be developed in parallel to this effort by the Co-PI and another graduate student.

Data collection and analysis – In support of better measurement practices, the needs required for data collection from the benchmark systems were investigated. This included a thorough understanding of the relationship between the error messages from the target benchmark systems and the underlying error detection and fault tolerant mechanisms in the target benchmark systems. Prior experience had shown that vast amounts of data are the norm during long fault injection campaigns. Finding methods to manage the data, establish relationships between the data sets, and reduce the data sets to essential attributes was a key to developing an effective analysis process.

Fault list generation and pre-injection analysis – Generating fault lists for a fault injection experiment or campaign is a critical activity for fault injection. Pre-injection analysis is a method for guiding the fault injection process to produce more effective and efficient results. It is a means to reduce or eliminate the "no-response" problem associated with fault injection. Being a

statistical experiment, fault injection testing may require a large number of experiments to be conducted in order to guarantee statistically significant results. Thus, efficiency of the fault injection testing is important. The objection of this phase of the research was to develop a methodology for pre-analyzing the binaries of the target benchmark systems to reduce no-response fault injections, accelerate error propagation, and improve efficiency.

Operational profile generation – Operational profiles and workloads of the target system are required to set the operational and environmental context of the system. The operational profiles must be representative of the different system configurations and workloads that would be experienced in actual field operations. In order to provide a diverse and representative set of operational profiles for the target system, the use of high fidelity NPP simulator tools to generate nominal, off-nominal, and accident event profiles is the most promising way forward. The work entailed integrating the TRACE thermo-hydraulic NPP simulator into the UVA fault injection environment so that real time process data from the simulator could be used to drive the inputs of the target benchmark systems under various conditions and modes.

9.2.2. Research Support Tasks

Research support task activities were necessary to conduct research on the benchmark systems. As have indicated in Section 7, the amount time to design, development, test, and integrate the various fault injection components together and interface them to the target benchmark system was significant.

Training and experience – To effectively apply fault injections to complex digital I&C systems of the type found in the benchmark systems, the research staff required professional training on the systems by the respective vendors. Once this training was complete, the staff required time to gain additional experience on the systems to fully understand the details of the system from various points of view. This effort required several months to complete before any research tasks or support tasks could be initiated.

Fault injection environment – A well-formed fault injection environment is one of the most important aspects of a fault injection-based methodology to support credible, repeatable, and controllable fault injection experiments. Furthermore, the fault injection environment plays a crucial role in the data collection and measurement of the responses, which are important for producing measures of dependability (e.g., coverage, error latency, etc…). In addition, there must be a capability to manage the types of faults to be injected into the system, where they are injected, how they are injected, and when they are injected. Additionally, responses to the fault injections must be acquired in a manner that allows the responses to be traced back to the faults so that a fault injection trial can be repeated as needed to reproduce the system response.

Beyond the basic functional requirements for a fault injection process, effective fault injection environments also must be practical, adaptable to changing technology, and supportable. Early in the development of the fault injection environment several development goals for the fault injection environment were identified to allow for adaptability of different I&C systems, including:

- Flexible to a wide variety of applications
- Easy to use and familiar to the engineering culture
- Industry-grade, supportable, and open source
- Modular
- Extensible
- Evolutionary

To achieve these goals, the National Instruments™ LabVIEW toolset was selected to develop the basic architecture of the fault injection environment. Proven technology and industry acceptance made this a logical choice. Due to the complex nature of fault injections and the need for tight coordination of several processes (e.g., data acquisition, operational profile sequencing, fault injection, data logging, etc), a cross-platform tool was determined to be the most effective means to support these functions.

Application code development – The benchmark systems were not delivered to UVA with an application embedded on them. Therefore, UVA built a representative RPS was developed.

Integration and testing – Integrating the various components of the fault injection environment, the data acquisition system, and the target benchmark systems required substantial skills and knowledge. The most prominent of these tasks was the integration of the fault injector platform into the target I&C system. This task required considerable modifications to the prototype fault injector platform to effectively integrate the fault injector into the target systems.

9.2.3. Experiment Related Tasks

After the support research tasks were completed, the remaining time was devoted to conducting various fault injection campaigns on the benchmark systems in support of the research objectives. Through the entire project research, development, implementation process, the various challenges, needs and solutions were documented with respect to implementing fault injection on digital I&C systems. The lessons learned along with the results of the effort formed the principle result of the project for the NRC.

The fault injection campaigns conducted on the benchmark systems reflect the application of the FARM model and the statistical models presented in this volume to gauge how applicable they are with respect to digital I&C systems.

10. SUMMARY, FINDINGS AND CONCLUSIONS

This Section summarizes activities (Section 10.1) of this report, and lists the principal study findings (Section 10.2). It also provides observations drawn by the authors by assessing and reviewing the fault injection based assessment methodology from a formal perspective (Section 3) and practical perspective (Section 5 and 7) in the course of the study. In closing, some preliminary observations and recommendations (Section 10.3) are made in order to better refine the fault injection based assessment methodology toward the application to digital I&C systems (Sections 7 and 8).

10.1. Summary of Key Activities and Results

This volume presents a broad and in-depth development of the theory, methodology, the requirements and challenges of realizing fault injection on digital I&C systems as summarized below:

Presentation of a formal taxonomy of faults for digital I&C systems (Section 1.7.3). In Section 1.7.3 presents a taxonomy of faults and threats that may affect a digital I&C system during its lifetime. The purpose of this fault taxonomy is to present a complete and structured view of the domain of faults applicable to digital I&C systems. Since this research effort's main purpose is to assess the applicability and utility of fault injection, it seems reasonable to start with a well-structured and complete view of the fault space. The taxonomy presented here is almost assuredly complete with respect to the types of fault classes in digital I&C systems. The taxonomy presented is complete with respect to the types of fault classes in digital I&C systems, however, this taxonomy is a starting point for further research and discussions.

***Development of a Formal Model of Fault Injection for Digital I&C Systems (Section 3.3)*.** It is important to have a formal model to characterize the applicability and understanding of the fault injection process to ultimately guide its use and facilitate understanding of the results with respect to assumptions. The importance of the formal model is to provide a *reference* for all fault injection based methodologies with respect to the necessary requirements for fault injection.

Development and Analysis of Statistical Models for Fault Coverage Estimation (Section 3.6 and Appendix A). The purpose of the statistical model is to provide a formal basis for (1) conducting fault injection experiments and (2) providing a statistical model for a estimating the measures of a fault injection experiment, such as coverage. Section 3.6 and Appendix A presents a formal and mathematical description of statistical estimation concepts that are fundamental in the assessment of fault coverage. The presentation and analysis provides a sufficient and adequate approach that very well applies to a wide variety of statistical models for fault coverage estimation. Appendix A analyzes two widely referenced and used models for statistical estimation of fault coverage, and present summary findings on both models.

10.1.1. Survey and Characterization of Fault Injection Techniques for Digital I&C Systems (Section 5)

In this section, a characterization schema for fault injection techniques based on eight properties is proposed and developed. These eight properties represent attributes that are desired for a fault injection technique to support the requirements of the FARM model and the fault injection based dependability assessment methodology as presented in Section 4. The purpose of the characterization is to describe fault injection techniques in manner that better informs the NRC on the applicability of specific injection techniques to digital I&C systems. In addition to fault representativeness (i.e., the plausibility of the supported fault model with respect to actual faults) that is one concern that is often raised in conjunction with fault injection

experiments, show a wide range of criteria can be considered to assess the merits of the fault injection techniques. This is particularly important with respect to physical fault injection where complete controllability and reachability are difficult to achieve with just one fault injection technique. This Section is significant because all advantages and disadvantages that must be weighed during the selection process of fault injection methods and techniques with respect to digital I&C systems.

Lessons Learned from Previous Experience (7.10). By examining the application of fault injection on a previous digital I&C system (DFWCS) insight into specific issues and challenges of applying fault injection to digital I&C systems of the type found in NPP operations was gained. The challenges and the resolution of those challenges showed that the application of fault injection to digital I&C systems is complex process involving the integration of several systems to achieve fault injection capability. The integration process involves such diverse systems as the fault injector, the target computer, the fault injection controller to initiate the fault injection process, the operational profile system to supply inputs to the target system, and the data acquisition system to collect data. To improve the methodology in a direction that better supports physical based fault injection, the methodology should be augmented with additional steps called *support steps*. As such, the elements of the methodology can be partitioned into two classes. The first class is what activities are *necessary* to perform fault injection according to the FARM model attributes. These steps are the elements in the methodology now. The new steps are related to what *support* is needed to carry out fault injection.

Challenges to Fault Injection with respect to Digital I&C systems (Section 8). Based on the findings in Sections 3, 4, 5, and 7, six challenges to enable credible fault injection on digital I&C systems and to improve the utility of fault injection process overall are identified. These six challenges are:

(1) ***Better measurement practices and tools*** - Digital I&C systems are not typically designed to be monitored to the level required for dependability evaluation by fault injection; a methodological approach for their observation is thus needed. In distributed digital I&C systems things are even more complex, for the lack of central control, and for the difficulties in obtaining a precise global time and an accurate view of the global state of the system are all noted challenges. Measurement issues are very rarely addressed in the major conferences and scientific journals in the area of dependability.

(2) ***Guidance for fault selection and realization*** – The selection of fault models for digital I&C systems should be more structured and this process should be included in the fault injection methodology as a support step. The steps of the fault model selection process should include at the very least the following activities:

- Examination of the fault hypothesis of the system.
- The use of a structured fault taxonomy (e.g., like the one presented in Section 1.7.3) as a starting point to help guide the fault model selection process.
- Examination of failure data from empirical and experiential data sources that is relevant to the system.
- The operational context of the system.
- The interaction context of the system.

(3) ***Guidance for automation of fault injection processes*** – As indicated in Section 4.7.1.4 and in Section 5, a well formed fault injection environment is one of the most important aspects of a fault injection based methodology. "Well-formed" means the fault injection environment follows and refines the theory of the FARM model (Section 3.3), and allows the fault injection process to be a statistically guided process as described in Section 3.6. Furthermore, the fault injection environment plays a crucial role in the data collection and measurement of the responses which are important producing the measures of dependability (e.g., coverage, error latency, etc…). While the methodology presented in Section 4 is intended to be somewhat "neutral" to the selection of fault injection techniques, more guidance on what the fault injection environment should provide to the methodology and what the methodology expects from a fault injection technique is needed. The "tie in" from the requirements of fault injection theory (FARM model) to the specification of the fault injection environment is needed.

(4) ***Tools for operational profile generation*** – The operational profiles must be representative of the different system configurations and workloads that would be experienced in actual field operations. Context is important in fault injection. For a fault injection based assessment methodology, the operational profiles must represent the input conditions and system interactions that can occur not only nominal operations, but also in off-nominal operations and more importantly during "accident" event scenarios. Gathering profile real plant data across all of these domains of operations is challenging task. Tools to automatically generate profiles along these lines are needed. NPP simulation tools such as TRACE are a possible way forward, however, there may be other approaches to solve this problem.

(5) ***Methods to improve the efficiency and effectiveness of fault injection (pre-fault injection analysis)*** – Generating fault lists for a fault injection experiment or campaign is a critical activity for fault injection. Most fault list generating techniques found in the literature were developed for simulation based approaches and not physical fault injection approaches [Smith 1997; Benso 1998]. UVA found that these approaches did not translate well into existing I&C systems because simulation-based approaches allow a high degree of controllability and observability which may or may not be achievable in real digital I&C systems. In addition, fault list generation methods must address the efficiency and effectiveness issues mentioned in Section 8 related to the "no-response" problem. Pre-injection analysis is method to guide the fault injection process to produce more effective and efficient results. It is a means to reduce or eliminate the "no-response" problem associated with fault injection.

(6) ***High performance fault injection*** – There is a need for fault injection techniques to support various fault models for fault injection, and do so in manner that is minimally intrusive, controllable, repeatable and objective. This is critical to the practical application of fault injection to digital I&C systems. By uniting a variety of fault injection techniques to a common interface onto a single platform, the integration of the fault injector into the digital I&C system is more or less consistent from one digital I&C platform to the next. This aspect becomes important when fault injection is used as *benchmarking activity*. That is, the same set of faults, operational conditions are applied to each digital I&C system by the same fault injection technique to form an objective basis for comparison or compliance.

10.2. Conclusions

Before closing on this Section, some general observations can be drawn from both the work carried out in this research and from the awareness of what the research could not address (at

this point time). It is appropriate to underscore that the nature of this volume reflects the early phase of the investigation. Thus, some of the key findings of this study, and especially those involving the application of fault injection (Section 5 and 8) procedures that have been documented in this volume were further investigated and organized as the methodology was applied to the benchmark systems in Phase 2 and Phase 3. Nonetheless, several of the key findings in this report form the foundation of a technical basis for the application of fault injection to digital I&C systems. The key findings are preliminary, but several key findings stand out as noteworthy for the inclusion of a technical basis supporting fault injections:

Systematic Guidance for Fault Model Selection – Presently there is a significant gap in understanding between faults and failures that have occurred in digital I&C systems, and those that can occur as digital I&C hardware and software technology evolves. Furthermore, since faults and failures are rare occurrences in digital I&C systems, a comprehensive up-to-date database of such faults and failures is needed. A key aspect of the problem that remains open is that of the identification and systematic characterization of potential digital I&C system faults and failures to make possible a selection of the most appropriate fault models for dependability assessment purposes.

The fault taxonomy presented in Section 1.7.3 is viewed as starting point for a *context specific classification* with respect to digital I&C systems. Any classification of fault and failures should be formulated with both empirical data and data from scientific investigations on emerging failures (see reference [Srinivasan 2004] for an example of scientific investigations on emerging failures). In this way, science informs the empirical data to give the complete perspective on potential faults and failures of digital I&C systems. Models for fault injection can then be selected to best represent those faults and failures that have been confirmed by the classification process.

The use of fault injection to stimulate failure modes depends on the underlying fault model. Several fault models (Section 4.5) were identified that are known to be representative of faults that occur in digital systems. This research and the associated insights provided in Section 1.7.3, Section 3.3, Section 4.5, and Section 8.2, identify some of the gaps in theory and practice that will help illuminate this issue in order to provide a substantive baseline for a technical position.

Support Guidance for Fault Injection Based Dependability Assessment Methodology – As indicated in Sections 7 and 8, the application of physical-based fault injection to digital I&C systems requires a significant coordination of processes and systems to faithfully represent the modified FARM model (Section 3.3) and the methodology as presented in Section 4. These levels of coordination and integration are unique to physical based fault injection, and are not a significant issue in simulation-based fault injection. The challenges faced by applying physical based fault injection to the DFWCS revealed this oversight in the methodology. While the steps in the methodology as presented in Section 4 are necessary to perform fault injection, the guidance on how to effectively implement and apply physical based fault injection is something that will be needed to establish a technical basis. The latter stages of this effort (phase 1 and 2) will be conducted with this goal in mind.

Characterization of Fault injection Methods and Techniques –Section 5 presented a comprehensive description of contemporary and emerging techniques for fault injection. For the practitioner or user of fault injection the variety of fault injection techniques and tools are many. The claimed capabilities (often stated without assumptions), and tradeoff space make decisions about fault injection difficult. This report provided a structured survey in order to organize fault injection according to classes. To further aid the reviewer, the benefits, assumptions, and disadvantages of various techniques were summarized so that decision-making regarding the selection or analysis of a fault injection method could be performed in a systematic manner.

This type of information was compiled and guided by extensive experience with fault injection over many years of research and practice in the field. The inclusion of this information in NRC guidance could form a common baseline of understanding of fault injection between the NRC staff and the nuclear industry.

10.3. References

[Benso 1998] Benso, A. "A Low Cost Fault Injection System for Embedded Microprocessor Based Computers". ACM Transactions on Automated Systems, vol. 3(4) 1998: 626-634

[Smith 1997] D.T. Smith, B.W. Johnson, N. Andrianos, J.A. Profeta III. "A Variance Reduction Technique Using Fault Expansion for Fault Coverage Estimation." *IEEE Transactions on Reliability*, September 1997: 366-374.

[Srinivasan 2004] Srinivasan, J., S. V. Adve, P. Bose, and J. A. Rivers. "The impact of technology scaling on lifetime reliability." In the Proceedings of the International Conference on Dependable Systems and Networks. 2004. IEEE Press: 177-186.

APPENDIX A. STATISTICAL MODELS FOR COVERAGE ESTIMATION

A.1. INTRODUCTION

As discussed in Section 3, coverage factor *C* is an extremely important parameter in the safety assessment and safety modeling of dependable fault tolerant systems, of which digital I&C system belong to. The fault coverage available in a system can have a significant impact on the reliability and safety of a system [Smith 1997; Choi 1997; Yu 2004]. The intuitive definition of fault coverage is that it is a measure of a system's ability to perform fault detection, fault isolation, and fault recovery given the existence of a fault. Thus, coverage is a specific measure from the *M* set in the FARM model from Section 3.3.

In reference [Pescosolido 2002] different statistical models available in the literature were analyzed and compared in order to determine their applicability to fault coverage estimation. This appendix addresses two statistical models from that report that are often used: *Stratified Bernoulli model, and the Fault Equivalence model*. For a complete and detailed review of additional statistical models for fault coverage estimation the reader is referred to the original report.

The following sections are 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. The purpose of this appendix is to provide a thorough understanding of statistical modeling concepts as applied to fault coverage and to review the aforementioned statistical models so that informed decisions can be made about the use of the models.

A.2. STATISTICAL MODELING CONCEPTS

A.2.1. The Fault Space

The set of all faults is called the *fault space*, and is denoted in this document by *F*. A number of characteristic attributes, such as location, value, time, or workload, can be used to identify a fault in the fault space. In general, a fault space is regarded as a multi-dimensional space with *d* dimensions, where each fault is identified as a point in the *d*-dimensional space by a set of *d* coordinates. As shown in figure A.1, the fault space can be considered as the Cartesian product of the sets of values that can possibly be assumed by each attribute or axes in the multi-dimensional fault space.

If α_i represents the axis corresponding to attribute A_i, and a_i represents the value assumed by A_i, then the fault space can be defined as,

$$F = \alpha_1 \times \alpha_2 \times \ldots \times \alpha_d \qquad and \qquad f(a_1,\ldots,a_d) \in \mathrm{F} \tag{A.1}$$

where $f(a_1,\ldots,a_d) \in F$ is the fault for which $A_i = a_i$.

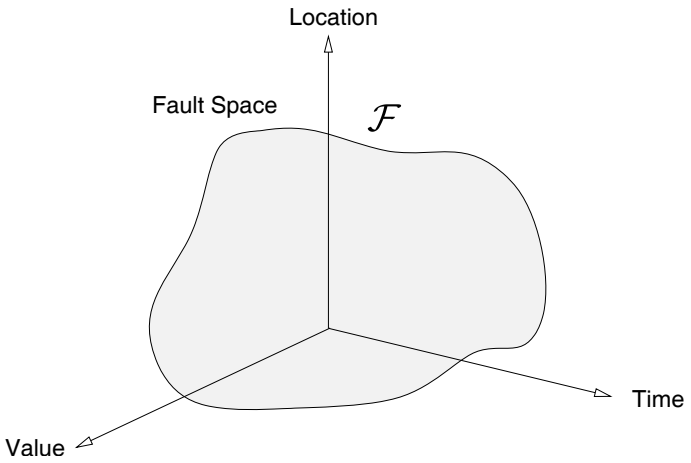

Figure A-1 Dimensional fault space

A.2.2. 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,

$$P(=f) = \frac{1}{|F|} \forall (f \in F) \tag{A.2}$$

where |F| indicates the cardinality of the fault space. The uniform distribution is very commonly assumed in statistical models for fault coverage estimation. The uniform approximation is generally conservative or pessimistic when used as the underlying fault distribution.

A better approximation of the real fault distribution can be obtained if the fault space F can be partitioned into *M* classes F_i for which the relative probability of occurrence is known. Within each class F_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,

$$P(F = f) = \sum_{i=1}^{M} \frac{\delta_i(f)}{|F|} P(F \in F_i) \tag{A.3}$$

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

$$P(F \in \mathsf{F}) = \sum_{f \in F} P(F = f) \tag{A.4}$$

$$P(F \in \Im_i) = \sum_{f \in \Im_i} P(F = f)$$

With

$$\sum_{i=1}^{M} P(F \in \Im_i) = 1 \tag{A.5}$$

Where are ***F*** 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 the underlying fault distribution, 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,

$$P(F \in \Im_i) = \lambda_i \Big/ \left(\sum_{k=1}^{M} \lambda_k \right) \tag{A.6}$$

where λ_i indicates the failure rate of the component corresponding to the fault class $\boldsymbol{F}_i$. 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.

In the literature there are variations on the class-wise fault distribution. One example is the method used by [Smidt 2011] that calculates the probability of software failure from semiconductor failure mechanism of a microprocessor.

A.2.3. 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 (3.10). 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.

$$Y_k = y(f_k) \tag{A.7}$$

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 $\mathcal{F}$ 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 following values.

$$P(\underline{F} = f) \tag{A.8}$$

Notice that the fault injection experiment forces the event "occurrence of a fault $f \in \mathcal{F}$" in the system with the forced distribution as indicated in Section 3.6.3.1 of the main report. 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 Section 3, Section 3.6.3.1.

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 [Ricci 1975].

A.2.4. 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* [Ricci 1975], 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*.

A.2.5. 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 3.6.3.1 the two variables are related to each other by Equation (3.11). 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.

A.2.6. 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.

A.2.7. 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.

A.2.8. 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 or physical 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.

A.2.9. 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 [Powell 1995]. 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.

A.3. STRATIFIED BERNOULLIAN MODEL

A.3.1. 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 [Powell 1996]: if the partition of the fault space is determined so that faults in the same class all occur at a certain location. For example, 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 based primarily on the stratified sampling strategy presented in [Powell 1995; Cukier 1997; Powell 1996].

A.3.2. 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*:

$$\Im = \bigcup_{m=1}^{M} \Im_m \tag{A.9}$$

$$\forall (i,j) \text{ with } i \neq j \quad \Im_i \cap \Im_j = \varnothing \tag{A.10}$$

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

$$P(F = f) = P[(F = f) | (F \in \Im_i)] P(F \in \Im_i) \tag{A.11}$$

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

$$P[(F = f) | (F \in \Im_i)] = 0 \; \forall (f \notin \Im_i) \tag{A.12}$$

The quantity

$$P(F \in \Im_i) \tag{A.13}$$

expresses the relative probability that a fault that has occurred in the system belongs to class $\Im_i$. Ideally, this quantity could be calculated as the limit of the ratio between the number of faults belonging to the class $\Im_i$ 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 A.2.2 several approaches to determine an approximate class-wise uniform fault distribution were described, which apply to his case as well. In fact, the $P(F \in \Im_i)$ are the identical to the *weighting factors* that were there introduced. If the probabilities in Equation (A.13) cannot be determined then the following can always be assumed.

$$P(F \in \Im_i) = \frac{1}{M} \tag{A.14}$$

or that

$$P(F \in \Im_i) = \frac{|\Im_i|}{|\Im|} \tag{A.15}$$

The condition expressed in Equation (A.14) is a stronger condition than Equation (A.15): the latter is obtained when all faults are equally likely to occur (uniform distribution), whereas Equation (A.14) 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 (A.14) the relative probabilities of both classes would be 0.5, whereas using Equation (A.15) 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 (A.15) should always be preferred to Equation (A.14).

A.3.3. Definition of Coverage

Following the development presented in [Powell 1995], the definition of fault coverage can be modified, using Equation (3.10) and Equation (A.11), as follows,

$$\begin{aligned} C &= \sum_{f \in \Im} y(f)P(F = f) = \sum_{f \in \Im} y(f)P[(F = f)|(F \in \Im_i)]P(F \in \Im_i) \\ &= \sum_{i=1}^{M} \sum_{f \in \Im_i} y(f)P[(F = f)|(F \in \Im_i)]P(F \in \Im_i) \\ &= \sum_{i=1}^{M} P(F \in \Im_i) \sum_{f \in \Im_i} y(f)P[(F = f)|(F \in \Im_i)] \end{aligned}$$

$$= \sum_{i=1}^{M} P(F \in \Im_i) C_i \tag{A.16}$$

where C_i is called the *class fault coverage* of $\Im_i$, as follows.

$$C_i = \sum_{f \in \Im_i} y(f) P[(F = f) | (F \in \Im_i)] \tag{A.17}$$

Equation (A.16) 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.

A.3.4. Sampling

The stratified sampling strategy requires the selection of a predetermined number n_i of samples from each class $\Im_i$ of the partitioned fault space.

If the values of the probabilities of Equation (A.13) are known, the number of faults sampled from each class is chosen to be,

$$n_i = P(F \in \Im_i) \cdot n \tag{A.18}$$

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* [Powell 1995].

If the values of the probabilities of Equation (A.13) are not known, different strategies can be employed similar to those discussed in Section 8.2.2. Substituting expressions Equation (A.14) and equation (A.15) into Equation (A.18), the following two relations are obtained, respectively

$$n_i = \frac{n}{M} \tag{A.19}$$

called *homogeneous allocation* [Powell 1995], or

$$n_i = n \cdot \frac{|\Im_i|}{|\Im|} \tag{A.20}$$

which is termed *uniform allocation*.

In general, given a certain assumption on the value of the weighting factors of Equation (A.13), the size of the sample extracted from the corresponding classes are obtained by substituting the assumed value into Equation (A.18).

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

$$P[(\underline{F} = f) | (\underline{F} \in \Im_i)] = \frac{1}{|\Im_i|} \tag{A.21}$$

and the overall sampling distribution can be expressed as.

$$P(\underline{F} = f) = \sum_{i=1}^{M} \frac{\delta_i(f)}{|\Im_i|} P(\underline{F} \in \Im_i) \tag{A.22}$$

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

A.3.5. Point Estimator

With the definition of fault coverage given in Equation (A.16) 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,

$$\hat{C}_i = \frac{1}{n_i} \sum_{k=1}^{n_i} \underline{Y}_k \tag{A.23}$$

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

$$E[\hat{C}_i] = C_i \tag{A.24}$$

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 (A.23) 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 (A.23), 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,

$$\hat{C} = \sum_{i=1}^{M} \hat{C}_i P(F \in \Im_i) \tag{A.25}$$

which is the weighted average of the estimated coverage factors of all the classes in the partitioned fault space. Notice that substituting Equation (A.16) and Equation (A.18) into Equation (A.25), the following is verified,

$$\hat{C} = \sum_{i=1}^{M} \frac{1}{n_i} \sum_{k=1}^{n_i} \underline{Y}_k = \frac{1}{n} \sum_{i=1}^{n} \underline{Y}_i \tag{A.26}$$

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 concern the precision of the estimator and for what concerns its accuracy.

A.3.6. Precision of the Estimator

Analogous to what was done in Section 3.6.4, the random variable $\underline{Y}_i$, describing the fault tolerance event for faults occurred in class $\Im_i$, and the random variable P_i as a function of a fault from class $\Im_i$ that has occurred in the system can be defined with the following probability.

$$P[(\underline{F} = f) | (\underline{F} \in \Im_i)]$$
$$P_i = p_i(f) = \frac{P[(F = f) | (F \in \Im_i)]}{P[(\underline{F} = f) | (\underline{F} \in \Im_i)]} \tag{A.27}$$

The class covariance ρ_i between the variables $\underline{Y}_i$ and P_i is defined as

$$\rho_i = \sum_{f \in \Im_i} [(\underline{Y}_i - \underline{C}_i)(P_i - 1)] = C_i - \underline{C}_i \tag{A.28}$$

It is easy to show the following.

$$\rho = \sum_{i=1}^{M} \rho_i P(F \in \Im_i) \tag{A.29}$$

$$\underline{C} = \sum_{i=1}^{M} \underline{C}_i P(F \in \Im_i) \tag{A.30}$$

The mean of estimator in Equation (3.16) is finally calculated as,

$$E[\hat{C}] = E\left[\sum_{i=1}^{M} \hat{C}_i P(F \in \Im_i)\right] = \sum_{i=1}^{M} E[\hat{C}_i] P(F \in \Im_i) = \sum_{i=1}^{M} \underline{C}_i P(F \in \Im_i)$$
$$= \sum_{i=1}^{M} (C_i - \rho_i) P(F \in \Im_i) = C - \rho = \underline{C} \qquad \text{(A.31)}$$

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

However, the value assumed by the covariance γ is different, in general, from the covariance that would be obtained using the simple Bernoulli model of Section 3.6. 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 A.2.2. 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 (A.27), always assumes values very close to unity. By simple inspection of Equation (A.28), it is clear that $\rho \approx 0$, and from Equation (A.30) it is also verified that $\rho \approx 0$.

However, the values of γ_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 A.3.4.

A.3.7. Accuracy of the Estimator

The variance of the estimator is derived as a linear combination of the variances of the class coverage estimators C_i using the definition of class coverage proportion, the result obtained in Equation (3.22), and Equation (A.1),

$$V[\hat{\dot{C}}] = \sum_{i=1}^{M} [P(F \in \Im_i)]^2 V[\hat{\dot{C}}_i] = \sum_{i=1}^{M} [P(F \in \Im_i)]^2 \left(\frac{\dot{C}_i(1-\dot{C}_i)}{n_i} \right)$$

$$= \sum_{i=1}^{M} \frac{P(F \in \Im_i)\dot{C}_i(1-\dot{C}_i)}{n} \tag{A.32}$$

which can be estimated using [Powell 1995].

$$\hat{S}(\hat{\dot{C}}) = \sum_{i=1}^{M} \frac{[P(F \in \Im_i)]^2 \hat{\dot{C}}_i(1-\hat{\dot{C}}_i)}{n_i - 1} \tag{A.33}$$

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 [Powell 1995] that using the Lagrange multiplier method it can be proven that the variance is minimum if the sample sizes are chosen so that,

$$n_i = n \left(\frac{P(f \in \Im_i)\sqrt{\dot{C}_i(1-\dot{C}_i^2)}}{\sum_{k=1}^{M} P(f \in \Im_k)\sqrt{\dot{C}_k(1-\dot{C}_k^2)}} \right) \tag{A.34}$$

which, in absence of any *a priori* knowledge of the fault coverage, must be approximated with Equation (A.18), the representative allocation, to obtain the following.

$$V[\hat{\dot{C}}] = \frac{\dot{C}}{n - \sum_{i=1}^{M} \frac{P(F \in \Im_i)\dot{C}_i^2}{n}} \tag{A.35}$$

Compared to the results obtained in Equation (3.22), the variance of this estimator is reduced if

$$\sum_{i=1}^{M} P(F \in \Im_i)\dot{C}_i^2 > \dot{C}^2 \tag{A.36}$$

which is not always verified. One case when Equation (3.22) 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 A.3.2, a quite strong requirement. Obviously, it is only possible to verify Equation (A.36) for each specific system. In his thesis [Pescosolido 2002] analyzed the variants of the estimator obtained of this model to the variants of the simple Bernoulli model and found them almost identical.

A.3.8. Confidence Intervals

Using the Central Limit Theorem, double- and single-sided confidence intervals can be defined exactly as the simple Bernoulli model, using the mean and the variance estimator calculated in Equation (A.31) and Equation (A.36), 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 (A.26) with the normal distribution, the same considerations made with simply Bernoulli model apply in this case.

A.3.9. 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 A.3.2. The order of magnitude of the required number of experiments is still the same as discussed in Section A.3.2. 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.

A.3.10. No-Response Problem

A technique called a posteriori stratification is proposed in [Powell 1995] 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 [Powell 1995], 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 A.3.2 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 (A.25) andEquation (A.26), 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 (A.23) is meaningless, since both the numerator and denominator are zero.

A.3.11. Summary and Conclusions

The model discussed in this section, like the simple Bernoulli model described in Section A.3.2, is very commonly used in the assessment of the fault coverage of dependable systems, because of the practical advantages that it provides [Powell 1996].

It was also suggested 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. The stratified approach is motivated by practical considerations, because if no information about the fault distribution is available, this model is equivalent to the simple Bernoulli model presented in Section 3.6 of the main report.

In fact, in absence of information about the fault distribution the sampling strategy can only be chosen as in Equation (A.17) and Equation (A.20). The former represents an allocation 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 is equivalent in many ways to the simple Bernoulli model.

The precision of the estimator is still affected by the value of the covariance. However, it was discussed in Section A.2.2 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 [Powell 1995; Powell 1996; Cukier 1997] 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 was shown in [Pescosolido 2002], 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.

Finally, no optimal solution based on statistical methods 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 (A.18). Undetermined situations can be created if all faults in a certain class result in no-response. For this reason, the issue of no-response faults should be addressed at the fault injection environment level (e.g., pre-injection analysis). These conclusions are summarized in Table A-1.

Table A-1 Summary of properties for the stratified Bernoullian model.

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.

A.4. FAULT EQUIVALENCE MODEL

A.4.1. 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 [Smith 1997], 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 [Smith 1997] 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.

A.4.2. 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 follows.

$$f_h \sim f_k \quad \Leftrightarrow \quad y(f_h) = y(f_k) \tag{A.37}$$

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,

$$\left|f_i, f_j\right| = P[(Y_j = 1)\,|\,(Y_i = 0)] \cdot P[(Y_i = 0)] + P[(Y_j = 0)\,|\,(Y_i = 1)] \cdot P[(Y_i = 1)] \tag{A.38}$$

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,

$$\left|f_i, f_j\right| \leq \gamma \qquad \forall (f \in \Im_i) \tag{A.39}$$

where γ is a confidence parameter.

The model presented in [Smith 1997] only uses Equation (A.37) to induce a pre-determined partition of the space, however, Equation (A.39) can also be used in this model without any change in the results, leaving more flexibility in the setup of the fault injection experiments.

A.4.3. Fault Distribution

The fault distribution is assumed to be uniform in the model proposed in [Smith 1997]. 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.

A.4.4. 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 (A.37), 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 (A.39) 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 equation.

$$f_1 \in \Im \quad \text{and} \quad f_i \in \Im \backslash \Im_{i-1} \backslash \ldots \backslash \Im_{n-1} \qquad \forall i = 1 \ldots M \tag{A.40}$$

Notice that the following conditions are always satisfied by the sets $\Im_i$ determined using the fault equivalence process.

$$\bigcup_{i=1}^{M} \Im_i \subseteq \Im \quad \text{and} \quad \forall i, j = 1 \ldots M, \Im_i \cap \Im_j = \varnothing \tag{A.41}$$

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,

$$n_i = |\Im_i| \quad \text{and} \quad n = \sum_{i=1}^{n} n_i \tag{A.42}$$

which are, in this model, random variables, as their actual value depends on the specific realization of the sampling process.

A.4.5. Point Estimator

Notice that with the terminology introduced in the Section A.3.5 and Equation A.25, this case is equivalent to the case of stratified sampling, and the same estimator previously used can be applied here

$$\hat{C} = \sum_{i=1}^{M} \hat{C}_i P(F \in \Im_i) \tag{A.43}$$

with,

$$C_i = y(f_i) \tag{A.44}$$

The estimator of Equation (A.43) 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,

$$P(F \in \Im_i) = \frac{n_i}{n} \tag{A.45}$$

to obtain the estimator proposed in [Smith 1997].

$$\hat{C} = \frac{1}{n} \sum_{i=1}^{M} \hat{C}_i n_i \tag{A.46}$$

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.

A.4.6. Precision of the Estimator

It is immediate to show that the estimator of Equation (A.43) is unbiased contrarily to the other estimators encountered in the previous sections:

$$E[\hat{C}] = E\left[\sum_{i=1}^{M} \hat{C}_i P(F \in \Im_i)\right] = \sum_{i=1}^{M} \sum_{f \in \Im_i} y(f) P[(F = f) | (F \in \Im_i)] P[(F \in \Im_i)]$$

$$= \sum_{f \in \Im} y(f) P(F = f) = C \tag{A.47}$$

However, the estimator of Equation (A.46) is still biased, unless the assumption of Equation (A.45) 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 (A.46) leads in general to unbiased estimates of the coverage proportion $\hat{C}$.

A.4.7. Accuracy of the Estimator

It is shown in [Smith 1997] that the variance,

$$\hat{S} = \frac{1}{n}\sum_{i=1}^{M}[y(f_i) - \hat{C}]^2 n_i \tag{A.48}$$

is lower than the variance that would be obtained using the simple Bernoulli model if the following condition applies,

$$\frac{n_u}{M_u} < \frac{n}{M} \tag{A.49}$$

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 (A.49) 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 [Wang 1994], 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 [Ricci 1975]), and it is argued that in usual systems the condition of Equation (A.49) is not very often verified. The model presented in [Smith 1997] introduces a cost function that allows determining the convenience of using fault equivalence given the uncertainty of the variance reduction that can be achieved.

A.4.8. 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 [Smith 1997],

$$\widehat{C}_\gamma = 1 - z_\gamma \sqrt{\frac{(n-1)M^2}{(M-1)n^4}} \tag{A.50}$$

which can be used when $M >> 1$ and $N >> 1$. In [Smith 1997] 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 was discussed in Section 10.

A.4.9. 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 [Smith 1997] is estimated using the relation,

$$n \approx \frac{z_{\gamma}}{(1 - C)} \tag{A.51}$$

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.

A.4.10. 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 A.4.4, this strategy is appropriate because faults are sampled and injected sequentially.

A.5. 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 (A.49) 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 (A.49) 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 [Smith 1997] 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 [Wang 1994] suggest that fault equivalence does not, for real systems, provide an advantage in most cases, because the condition of Equation (A.49) is not often verified, also given that the fault equivalence that can be achieved normally is relatively small. These conclusions are summarized in Table A-2.

Table A-2 Properties of the Fault Equivalence model.

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 (A.49) 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 (A.49) must be verified for the model to provide any advantage compared to the simple model.

A.6. SUMMARY

This Appendix has provided a detailed review and analysis of several statistical models for fault coverage estimation found in the literature. The purpose of the appendix is to better inform the NRC on the nature and use of statistical models for fault coverage estimation when used in fault injection based assessment process. A major effort has been devoted to achieving a consistent presentation throughout the whole report, so as to simplify a direct comparison of the models.

Finally, for each model a summary table was provided that describes each of the attributes identified in Section A.2 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.

A.7. REFERENCES

[Smidts 2011] C. Smidts, Y. Shi, M. Li, W. Kong, J. Dai. A Large Scale Validation of a Methodology for Assessing Software Reliability. NUREG/CR-7042, U.S. NRC, 2011.

[Choi 1997] C.Y. Choi, B.W. Johnson, J.A. Profeta III. "Safety Issues in the Comparative Analysis of Dependable Architectures." IEEE Transactions on Reliability, September 1997: 316-322.

[Cukier 1997] Cukier M., Arlat, J., Powell, D. "Frequentist and Bayesian Coverage Estimations for Stratified Fault-Injection." Proceedings of DCCA-6. Grainau, Germany, 1997. 38-57.

[Smith 1997] D.T. Smith, B.W. Johnson, N. Andrianos, J.A. Profeta III. "A Variance Reduction Technique Using Fault Expansion for Fault Coverage Estimation." IEEE Transactions on Reliability, September 1997: 366-374.

[Pescosolido 2002] Pescosolido, M. "Statistical Models for Coverage Estimation." School of Engineering and Applied Science Masters Thesis. University of Virginia, May 2002.

[Powell 1996] Powell, D., Cukier, M., Arlat, J. "On Stratified Sampling for High Coverage Estimations." Proceeding of EDDC-2. Taormina, Italy, 1996. 37-54.

[Powell 1995] Powell, D., Cukier, M., Arlat, J., Crouzet, Yves. "Estimators for Fault Tolerance Coverage Evaluation." IEEE Transactions on Computers, vol. 44, 1995: 261-274.

[Ricci 1994] Ricci, F. Statistica ed Elaborazione Statistica delle Informazioni. Zanichelli, 1975.

[Wang 1994] Wang, W., Trivedi, K.S., Shah, B.V., Profeta, J.A. "The Impact of Fault Expansion on the Interval Estimate for Fault Detection Coverage." 24th International Symposium on Fault Tolerant Computing. 1994. 330-337.

[Yu 2004] Y. Yu, B.W. Johnson. "Coverage Oriented Dependability Analysis for Safety-Critical Computer Systems." The International System Safety Conference (ISSC). System Safety Society, 2004.

NRC FORM 335
(12-2010)
NRCMD 3.7

U.S. NUCLEAR REGULATORY COMMISSION

BIBLIOGRAPHIC DATA SHEET

(See instructions on the reverse)

1. REPORT NUMBER
(Assigned by NRC, Add Vol., Supp., Rev., and Addendum Numbers, if any.)
NUREG/CR-7151
Volume 1

2. TITLE AND SUBTITLE
Development of a Fault Injection-Based Dependability Assessment Methodology for Digital I&C Systems: Volume 1

3. DATE REPORT PUBLISHED
MONTH: December
YEAR: 2012

4. FIN OR GRANT NUMBER
N6124

5. AUTHOR(S)
C.R. Elks, N.J. George, M.A. Reynolds, M. Miklo, C. Berger, S. Bingham, M. Sekhar, B.W. Johnson

6. TYPE OF REPORT
Technical

7. PERIOD COVERED (Inclusive Dates)

8. PERFORMING ORGANIZATION - NAME AND ADDRESS (If NRC, provide Division, Office or Region, U. S. Nuclear Regulatory Commission, and mailing address; if contractor, provide name and mailing address.)
The Charles L. Brown Department of Electrical and Computer Engineering
The University of Virginia
Charlottesville, Virginia

9. SPONSORING ORGANIZATION - NAME AND ADDRESS (If NRC, type "Same as above", if contractor, provide NRC Division, Office or Region, U. S. Nuclear Regulatory Commission, and mailing address.)
Division of Engineering
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001

10. SUPPLEMENTARY NOTES
S.A. Arndt, J.A. Dion, R.A. Shaffer, M.E. Waterman, Project Managers

11. ABSTRACT (200 words or less)
This report is volume 1 of a multi-volume set of reports that present the cumulative efforts, findings, and results of NRC contract JCN N6124 – "Digital System Dependability Performance"

Volume 1 presents the findings of developing a fault injection based quantitative assessment methodology with respect to processor based digital I&C systems for the purpose of evaluating the capabilities of the method to support NRC probabilistic risk assessment (PRA) and review processes of digital I&C systems. Fault injection is defined as a dependability validation technique that is based on the realization of controlled validation experiments in which system behavior is observed when faults are explicitly induced by the deliberate introduction (injection) of faults into the system [Arlat 1990]. Fault injection is therefore a form of accelerated testing of fault tolerance attributes of the digital I&C system under test. The purpose of this work is to help inform the development of regulatory guidance processes for digital I&C systems and potential improvements to the licensing process of digital I&C systems in NPP operations. The work described herein presents; (1) the effectiveness of fault injection (as applied to a digital I&C system) for providing critical safety model parameters (e.g., coverage factor) and system response information required by the PRA and reliability assessment processes, (2) the development and refinement of the methodology to improve applicability to digital I&C systems, and (3) findings for establishing a basis for using fault injection as applied to a diverse set digital I&C platforms.

12. KEY WORDS/DESCRIPTORS (List words or phrases that will assist researchers in locating the report.)
Fault injection, dependability, PRA, digital instrumentation and control systems, I&C

13. AVAILABILITY STATEMENT
unlimited

14. SECURITY CLASSIFICATION
(This Page)
unclassified
(This Report)
unclassified

15. NUMBER OF PAGES

16. PRICE

Federal Recycling Program

UNITED STATES
NUCLEAR REGULATORY COMMISSION
WASHINGTON, DC 20555-0001

OFFICIAL BUSINESS

NUREG/CR-7151
Volume 1

Development of a Fault Injection-Based Dependability Assessment Methodology for Digital I&C Systems

December 2012