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

NUREG/CR-7151, Vol. 3

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

Volume 3

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. 3

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

Volume 3

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

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

Section Page

TABLE OF CONTENTS
(continued)

Section Page

LIST OF FIGURES

LIST OF TABLES

ACRONYMS AND ABBREVIATIONS

A/D	Analog to Digital
API	Application Programmer Interface
BDM	Background Debug Mode
BSC	Boundary Scan Chain
CFR	Code of Federal Regulations
COTS	Commercial Off-the-Shelf
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
D/A	Digital to Analog
DFWCS	Digital Feedwater Control System
DI&C	Digital Instrumentation and Control
DPDR	Development Port Data Register
DPIR	Development Port Instruction Register
ESFAS	Engineered Safety Features Actuation System
ETM	Enhanced Trace Macro-cell
FARM	Faults, Activations, Readouts, and Measures
FDIM	Fault Detection, Isolation, and Mitigation
FMEA	Failure Modes and Effects Analysis
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
GOOFI	Generic Object Oriented Fault Injection
HiPeFI	High Performance Fault Injection
HMI	Human Machine Interface
HW	Hardware
I&C	Instrumentation and Control
ICE	In Circuit Emulator
IEC	International Electro-technical Commission
IEEE	Institute of Electrical and Electronics Engineers
IC	Integrated Circuit
I/O	Input/Output
IOCCOM	I/O Communication Module
IP	Intellectual Property
ISA	Instruction Set Architecture
ISR	Interrupt Service Routine
JTAG	Joint Test Action Group
LOCA	Loss of Coolant Accident
MP	Main Processor
MPU	Main Processor Unit
MTTF	Mean Time to Failure
NPP	Nuclear Power Plant
NRC	Nuclear Regulatory Commission
OCD	On-Chip Debugger
OP	Operational Profile
OS	Operating System
PRA	Probabilistic Risk Assessment
PXI	National Instruments Data Acquisition Controller
RAM	Random Access Memory
RPS	Reactor Protection System
RTE	Runtime Environment
RTOS	Real-Time Operating System
SCIFI	Scan Chain Implemented Fault Injection

SEU	Single Event Upset
SMR	Small Modular Reactor
SoC	Systems on a Chip
SoPC	System-on-Programmable-Chip
SURE	Semi-Markov Unreliability Reliability Estimation
SWIFI	Software Implemented Fault Injection
TAP	Test Access Port
TCK	Test Clock
TDI	Test Data In
TDO	Test Data Out
TMR	Triple Modular Redundant
TMS	Test Mode Select
TRACE	TRAC/RELAP Advanced Computational Engine
TRST	Test Reset
UNIFI	Universal Platform-Independent Fault Injection
UART	Universal Asynchronous Receiver/Transmitter
UVA	University of Virginia
VDC	Volts D/C
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VI	Virtual Instrument
V&V	Verification and Validation
VLSI	Very Large Scale Integration

1. INTRODUCTION

1.1. Background

This report is Volume 3 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 System II- 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 3) presents the findings of applying a fault injection based quantitative assessment methodology (presented in Volume 1) to a processor-based digital I&C system for the purpose of evaluating the capabilities of the method to support NRC probabilistic risk assessment (PRA) and review processes for digital I&C systems. The further 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 for digital I&C systems in nuclear power plant (NPP) operations. The work described herein broadly presents;

(1) the development of the fault injection methods and techniques that were applied to Benchmark System II,

(2) the development of a fault injection environment for digital I&C systems

(3) development of pre-injection analysis methods for automatically generating fault lists for digital I&C systems,

(4) (results of the application of the fault injection method to Benchmark System II,

(5) the challenges to applying fault injection to contemporary digital I&C systems, and

(6) the findings for addressing these challenges and establishing a basis for implementing fault injection to digital I&C platforms.

1.3. Background and Motivation

Given the revitalization of the nuclear power industry in the United States (US), there is near uniform agreement in the nuclear industry that significant technology and production challenges must be addressed to enable efficient 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 years 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. 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 and new human-machine integration capabilities, 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 of 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 1993; Yu 2004; Smith 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 system crashes 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 system 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 as safety critical systems employing extensive fault detection/tolerance and design diversity features to ensure proper operational and fail safe behavior in the event of a system failure. For example, Fault Detection, Isolation, and Mitigation (FDIM) software and online diagnostic functions of the benchmark systems in this research effort account for as much as 40 to 50 percent of the executable system software code [Barton 1990; Palumbo 1986; Young 1989]. This code is rarely challenged during normal operations 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 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 the reliability of digital systems in the context of PRA and the level of

detail that digital systems require in reliability modeling. Nonetheless, it is clear that PRA models must adequately represent the complex system interactions that can contribute to digital system failure modes. The essential research aim of the PRA technical community is to accurately model digital I&C system behaviors to take into account interactions of the system fault handling behaviors, coverage of fault tolerance features, and the view of the system as an integrated software and hardware system.

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) that allows one to place stronger justification or refutation on claims of the overall safety of an I&C system. Fault injection as part of a quantitative assessment process is a robust testing process that can support verification and validation (V&V) and quality assurance activities to gather evidence that the digital I&C system can perform its safety functions in the presence of faulted and failure conditions in compliance with NRC regulations. In addition, those aspects of Appendix B of Title 10 of the Code of Federal Regulations (CFR), Part 50 (10 CFR 50), the NRC Standard Review Plan (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.

1.4.1. Relationship to NRC Research Activities

The research conducted under this contract was done with the 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 could have the potential to be used in other research efforts. Therefore, the researchers 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 research 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:

Objective 1
Demonstrate the effectiveness of the University of Virginia (UVA) 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 fault injection-based process that make it easier and more effective to apply to a wider spectrum of digital I&C systems.

Objective 3
Document the limitations, sensitive assumptions, and implementation challenges that would encumber the application of fault injection processes for digital I&C systems. Also 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 scale due to budget constraints on equipment availability. Therefore, the benchmark 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.

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 should be 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.

Fault injection-based methods 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, reliability estimation requires (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 or hardware faults in the system. As such, the methodology developed and presented in this set of reports is aimed at providing empirical data in support of estimating system fault response data, such as fault detection, error propagation, fault latency, and timing delays.

1.4.3. Work Tasks for Phase II

Section 9 of Volume 1 presents the basic research plan of the project, which was used as a guide to realize the UVA fault injection-based dependability assessment methodology on the benchmark systems. This plan has two categories of tasks. The first task category includes items that require additional research and development to determine their potential for implementation. As such, this was on-going work for the project. The second category comprises 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 needed to be accomplished in order to support the overall research objectives.

1.4.3.1. Research Oriented Tasks

The research oriented tasks listed below are specifically tied to key challenges identified in Section 8 of Volume 1 and findings from Volume 2.

High performance fault injection – Lessons learned from applying fault injection to Benchmark System I suggest a critical need for fault injection techniques to support various fault models for fault injection in a manner that is minimally intrusive, controllable, repeatable and reproducible. This task investigated, designed, developed, and implemented new methods to achieve these goals.

Data collection and analysis – In support of better measurement practices, the research investigated the requirements for data collection from the benchmark systems. This included a thorough understanding of the relationship between the error messages from the target benchmark system 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 were key to an effective analysis process.

Enhanced fault list generation - Generating fault lists for a fault injection experiment or campaign is a critical activity for fault injection. Function block fault list generation extends the capabilities of the map file fault list generation presented in Volume 2 by identifying high-level function block code and data segments in the locatable image files. Nearly all digital I&C systems in proposed nuclear power plant safety applications use a function block software programming environment that employs libraries of pre-determined function blocks needed to design a particular application. The user or software engineer builds the application from these libraries of function blocks. Regulators, designers, and safety reviewers understand the system from this point of view, so it was important to provide the capability to guide fault injection campaigns from this perspective.

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 a promising approach. To support this task, the research effort developed a process for integrating TRAC/RELAP Advanced Computational Engine (TRACE) thermo-hydraulic NPP simulator results 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 system under various conditions and modes.

1.4.3.2. Research Support Tasks

Research tasks are necessary to support research activities using Benchmark System II. As in previous fault injection process efforts, the amount of time to design, develop, test, and integrate various fault injection components and then interface these components into the target benchmark system can be significant. These research support tasks are:

Training and experience – To effectively apply fault injection to complex digital I&C systems of the type found in the benchmark systems, the research staff at UVA required professional training by the respective vendors of the benchmark system platforms. Once this training was

completed, the staff required time to gain additional experience on the systems to fully understand the details of the system platforms from various points of view.

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 campaigns. The fault injection environment plays a crucial role in the data collection and measurement of the responses, which are important to produce the measures of dependability (e.g. coverage and error latency).

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 be traced back to the faults so that any fault injection trial can be repeated as needed to reproduce the system response.

In addition to the basic functional requirements for a fault injection, effective fault injection environments must also be practical to implement and use, adaptable to changing technology, and supportable. Several development goals for the fault injection environment to allow for adaptability for different I&C systems include:

- Flexibility for a wide variety of applications
- Easy to use and familiar to the engineering culture
- Industry-grade, supportable, and open source
- Modular
- Extensible
- Evolutionary

Due to the complex nature of fault injection and need for tight coordination of several processes (e.g. data acquisition, operational profile sequencing, fault injection, data logging, etc), a cross-platform tool was most effective to support these functions. To achieve these goals, the National Instruments™ LabVIEW® toolset was selected to develop the basic architecture of the fault injection environment because of its proven technology and industry acceptance.

Application code development – The benchmark systems were not delivered to UVA did not have embedded application. Therefore, the researchers built a RPS application as the benchmark application.

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 into the target I&C system. This task required considerable modifications to the fault injector to effectively integrate the fault injector into the target system.

1.4.4. Scope of Study

Fault injection-based methodologies are but one part of a comprehensive process for estimating the reliability of a digital system (hardware and software) for PRA applications. From the highest level perspective, the essential information needed for reliability estimation is (1) knowledge of the likelihood of faults (software and 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 or hardware faults in the system. How a digital I&C system responds to a fault and

mitigates the fault are essential elements for accurate system reliability modeling. As such, the methodology developed and presented in this report is aimed at providing empirical data in support of developing system fault response data, such as fault detection, error propagation, fault latency, timing delays, etc.

1.5. Project Organization and Timeline

This project consisted of three phases. The first phase, which is reported in Volume 1, developed and refined the methodology so that it could be applied to a benchmark system. The second phase of the work applied the methodology to Benchmark System I based on the recommendations and plan of action from the first phase of the work. The third and final phase of the work, which is described in this volume, applied the methodology to Benchmark System II 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.

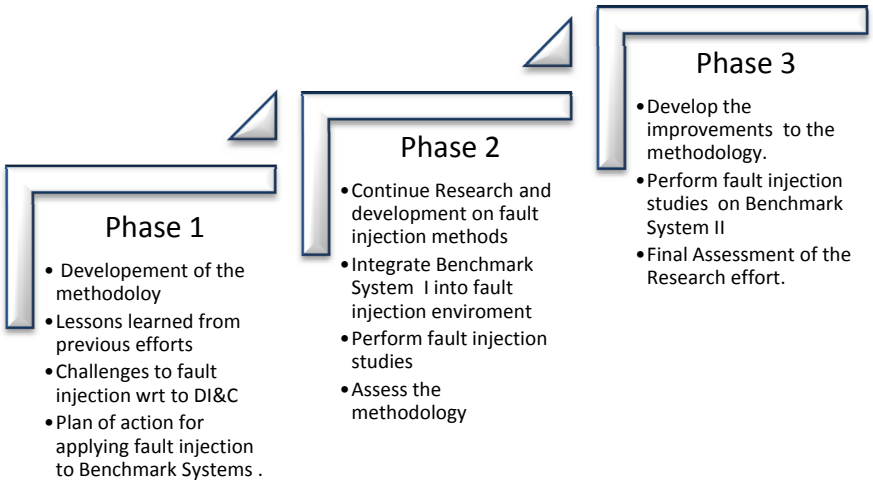

Figure 1-1 Phases and activities of the research effort

1.6. Organization of this Report

This report provides a contemporary and comprehensive perspective on fault injection for digital I&C systems. Additionally, this report also serves the greater digital I&C community by providing a broad and deep perspective of fault injection with specific focus on digital I&C systems.

This report is organized around the following main themes:

(1) Selection of appropriate fault injection techniques for Benchmark System II

(2) Development and implementation of a new high performance fault injection technique for digital I&C systems

(3) Application of the fault injection-based dependability assessment methodology to Benchmark System II

(4) Describing the results, outcomes and challenges associated with the application of fault injection to the benchmark systems.

The Sections build on and connect to previous Sections. Section 1 and Section 2 provide an overview of the fault injection-based dependability assessment methodology, the research methodology of this phase of the research effort, and a foundational understanding of the concepts of fault injection with respect to digital I&C systems. Section 3 presents a detailed overview of Benchmark System II and an overview of the RPS code development. Section 4 presents analysis and selection of candidate fault injection techniques for Benchmark System II. Section 5 describes the development and implementation of the UVA High Performance Fault Injection (HiPeFI) module. Section 6 presents methods to generate fault lists for Benchmark System II including a function block-oriented fault list generation method. Section 7 discusses the integration of the Benchmark System II into the UNiversal Platform-Independent Fault Injection (UNIFI) fault injection environment. Section 8 discusses the fault injection campaigns and experiments conducted on Benchmark System II and the results of those investigations. Section 9 presents the findings, insights, and lessons learned from this research effort.

1.7. Overview of Fault Injection

Section 3 of Volume 1 presents a detailed discussion of fault injection concepts and theories needed for the development of a fault injection methodology for digital I&C systems. This section presents a brief overview of fault injection to reacquaint the reader with the associated principles.

Consider the target digital I&C system shown in Figure 1–2. When fault injection is applied to the target system, the input domain corresponds to the following sets:

(1) A set of faults ***F*** taken from a class of faults "$\boldsymbol{F_{class}}$"
(2) A set of activations, ***A***, that specifies the domain used to functionally exercise the system
(3) An output domain corresponding to a set of readouts, ***R***
(4) A set of derived measures ***M***.

Together, the Faults, Activations, Readouts, and Measures (FARM) set constitute the major attributes that can be used to fully characterize fault injection.

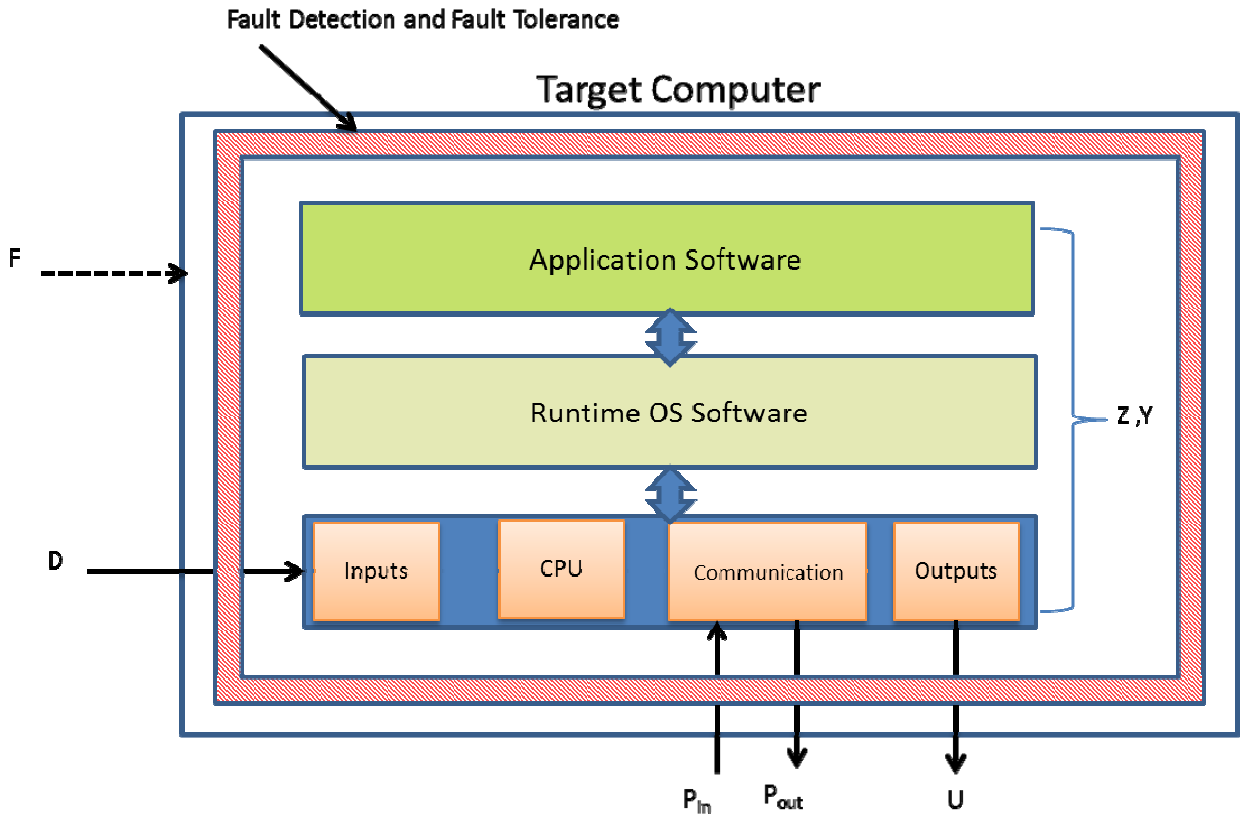

Figure 1-2 Fault injection model for digital I&C

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. Expanding the ***F*** to include the fault space dimensionality of time, location, value, and fault type, 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

v = value of the fault mask

f_m = a specific fault type as sampled from fault classes

The basic concept of a fault injection experiment is shown in Figure 1-3, which shows that faults from ***F*** are sampled from the fault space (discussed in Section 3.7 of Volume 1). These faults are elaborated by the fault type f_m, the fault duration Δ, the fault location ***l***, the value of the fault mask, time of occurrence ***t***, and the set of inputs ***a*** to characterize the 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 a 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 discuss the statistical theory behind the coverage estimation and the dependent parameters of coverage.

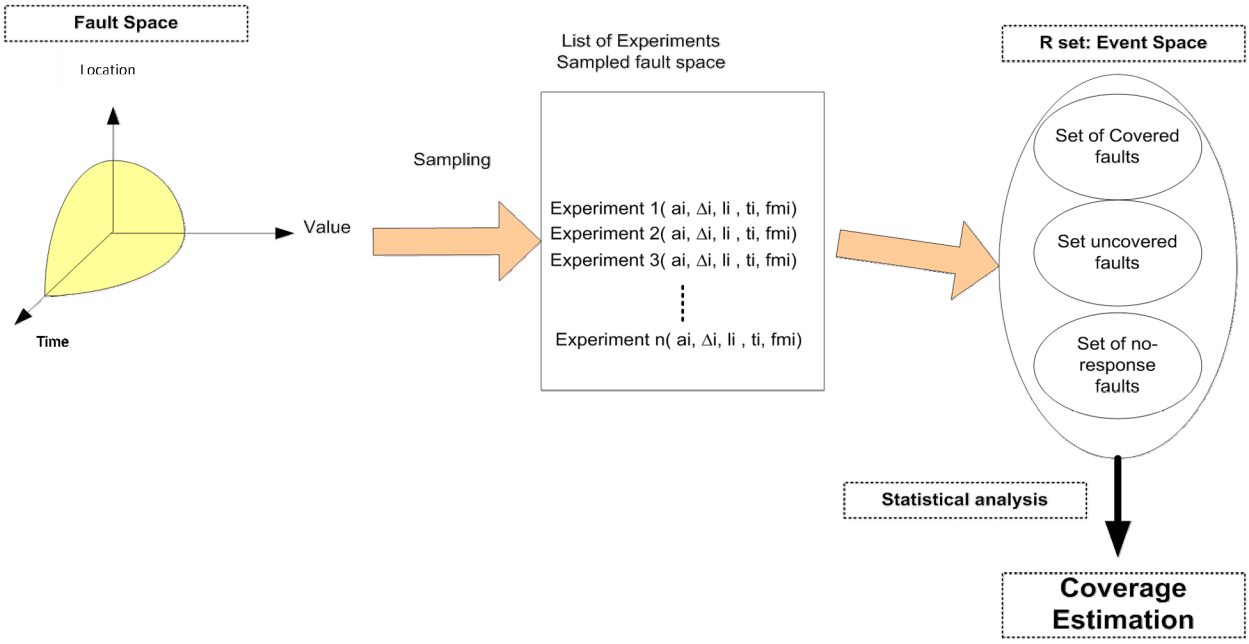

Figure 1-3 Fault injection experiment

1.8. Overview of the Fault Injection-based Dependability Assessment Methodology

The UVA fault injection-based dependability assessment methodology was developed realizing that a fault injection approach may serve different goals and purposes. Thus, the methodology was designed to be as flexible as possible to the needs of the assessor and designer.

The goal of the dependability assessment methodology described in this report was to provide a generic, formal, systematic means of characterizing the dependability behavior of digital I&C systems and their full plant interactions in the presence of anomalous behaviors, faults, and failures. This is termed the *full system approach* to fault injection. The goal methodology provides practical means for characterizing digital I&C system/plant dependability attributes that will assist developers in improving V&V processes, while helping regulatory entities make informed confirmatory decisions about licensing I&C systems for critical plant operations.

Figure 1-4 shows the basic conceptual framework of the fault injection-based dependability assessment process. 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, and these are instantiated back into the PRA models to enhance better predictions of the PRA models. The methodology is described in more detail in Section 4 of Volume 1. The characteristics of each step in the process are described in this section.

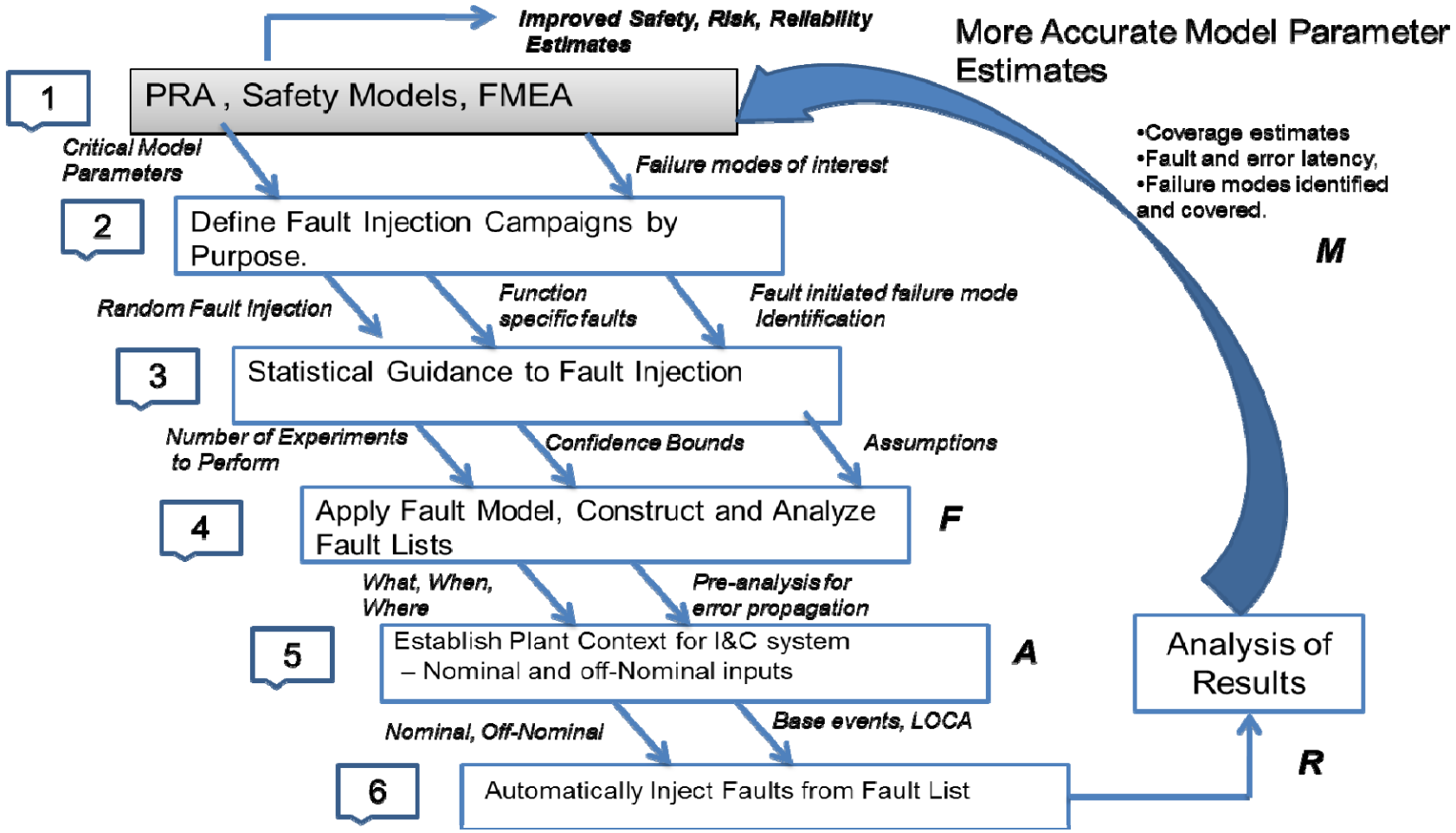

Figure 1-4 Operational view of the fault injection-based dependability assessment methodology

1.8.1. Step 0: Defining the Dependability Metrics

The assessment process begins with defining or selecting the dependability metric of interest. The metrics that can be used in I&C systems include but are not limited to

- system reliability
- probability of coincident failure
- system safety
- probability of failure on demand
- mean time to system failure
- mean time to unsafe system failure
- steady state unsafe system failure

For instance, an actuation system such as a RPS would be more accurately characterized by probability of failure on demand, and an instantaneous availability metric rather than a mean time to failure (MTTF) or a system reliability metric. So, how the system is employed in the context of the plant is very important to the selection of an appropriate metric. In the case of the RPS, it is a reactive system.

A reactive system is characterized by its ongoing interaction with its environment, continuously accepting requests from the environment and continuously producing results [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 system computations usually do not terminate. If the computations do terminate, it is most often due to the fact that an exception event has occurred. Example applications of reactive systems include process control systems, actuation systems, operating systems, and telecommunications.

1.8.2. Step 1: Support for PRA Activities

The purpose of a 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, (3) what are the consequences? Referring to Figure 1-4, the starting point in the methodology is to understand what is needed from the PRA process.

The PRA modeling process usually begins with defining or selecting a set of measurement-based attributes that are appropriate for informing the risk assessment process. These attributes typically 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, measures such as 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 are often seen.

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 estimate quantitatively the behavior model parameters of the system.

A behavioral model parameter is a measure of how a system behaved or responded with respect to a stimulus (e.g., a fault or a set of inputs). The important coverage factor parameter presented in Section 3.6 of Volume 1 is a behavioral parameter in the PRA model. Equally important is stating the underlying assumptions the models or model parameters produce 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 a accurate risk assessment to inform the risk oversight process.

1.8.3. STEP 2: Fault Injection by Purpose and Type

The fault injection process is used for different purposes in order to get a complete picture of a system behavior response. As indicated in Section 3.4 of Volume 1, fault injection is used in validation processes and design processes of digital I&C systems.

From a broader stance, fault injection is viewed as a measurement-based process that provides important experimental techniques for assessing and verifying fault-handling mechanisms. It allows researchers and system designers to study how digital systems react in the presence of faults. Fault injection processes are used in many contexts and can serve different purposes, such as

- Supporting on-line monitoring so that system performance can be effectively monitored.
- Assessing the effectiveness of fault-handling mechanisms in software and hardware.
- Studying error propagation and error latency in order to guide the design of fault-handling mechanisms.
- Providing evidence to support conclusions regarding the resiliency of a system to unexpected faults and failures.

All fault injection techniques have specific drawbacks and advantages as indicated in Section 5 of Volume 1. Since fault injection can be used for many purposes, it is necessary to identify as early as possible the fault injection method and measurements that will be used, and whether fault injections will be applied to a physical system or a model of the physical system. The comprehensive survey and characterization of fault injection methods and techniques presented in Section 5 of Volume 1 serve as a guide toward selecting the fault injection process for a target digital I&C system.

1.8.4. Step 3: Statistical Guidance for Fault Injection

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. As developed in Section 3 and Appendix A of Volume 1, the statistical model supports four specific needs of the fault injection-based dependability assessment methodology:

- Characterize the fault injection experiment in formal statistical framework.
- Quantify and characterize the uncertainty of model parameters.

- Characterize and define the assumptions of the estimation process.

- Statistically estimate the number of observations required to estimate a parameter to a known confidence level based on the assumptions of the model and model parameters.

1.8.5. Step 4: Fault Model Selection

Digital I&C systems are subject to faults and failures from a variety of sources that can be manifested in many ways, as was discussed in the fault taxonomy discussion in Section 1.7.3 of Volume 1 [Avizienis 2004]. Fault models are abstract representations of real faults. For example, a single event upset (SEU) 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 tests, 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 are key components of fault injection-based assessment processes.

Selecting the appropriate fault model for a fault injection campaign is a crucial decision. In Figure 1-5 are the fault classes that were selected to apply to the benchmark systems based on the research on fault and failure behavior of digital I&C systems. These fault models and their justifications were discussed in detail in Section 4.5 of Volume 1.

The output of a fault model selection process should produce a set of faults that is relevant to a particular digital I&C system. More importantly, the process of fault model selection should produce an audit trail or evidence trail so the assumptions and factors for determining the fault models can be verified during licensing and review activities.

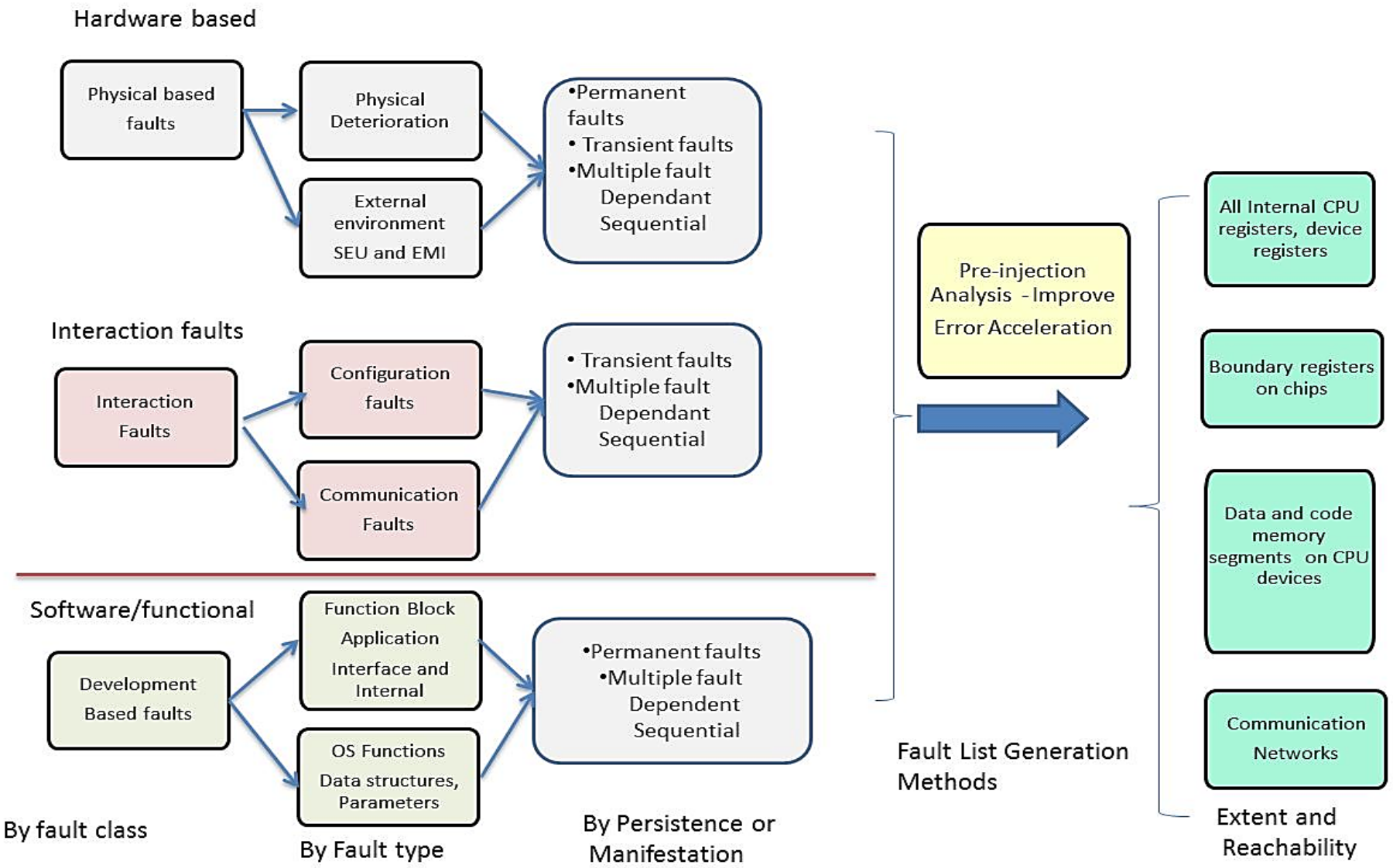

Figure 1-5 Fault model classes for benchmark digital I&C systems

After a representative set of fault models has been selected, the next step is to determine a means for organizing and applying the faults to a digital I&C system. This activity is called *fault list generation*. Generating fault lists for a fault injection experiment or campaign is a critical activity for fault injection.

A *fault list* is a sample set of faults taken from the fault space of the target I&C system. 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 (the values that represent the fault injection process with respect to a resource 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 the 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, which is often referred to as *error acceleration* [Chillarege 2002] or pre-injection analysis [Sekhar 2008; Barbosa 2005]. Pre-injection analysis is defined by a set of rules that enables fault injection experiments to increase the probability of a system failure by identifying the most likely fault injection experiments. Section 8 of this report presenat new fault list generation methods that produce efficient and effective fault injection results for digital I&C systems.

1.8.6. STEP 5: Establishing Operational Profile and Workload

An *operational profile* (OP) is a quantitative representation of how a system will be used within its use environment [Musa 1998]. An OP models how users interact and use the system, specifically the occurrence probabilities of system and user modes over a range of operations. Traditionally, an OP is used to generate test cases and to direct testing to the most used system functions, thus the potential for improved reliability with respect to the use environment is achieved. The OP 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 comprise a system.

As discussed in Section 4 of Volume 1, digital I&C systems that are real-time and reactive operate on a deterministic, time-triggered basis. 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, an operational profile is defined in the context of its application-specific nature (i.e., RPS).

Real time operational profiles to be used in the fault injection experiments must be selected to be representative of the system under various modes of operation and configuration. 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 include sufficient combinations of these configurations to ensure a thorough evaluation of their behavior in the presence of faults.

1.8.7. STEP 6: Injecting Faults into the Target System

Most fault injection tools have been developed with a specific fault injection technique in mind, targeting a specific system, and using a custom-designed user interface. Extending such tools with new fault injection techniques, or porting the tools to new target systems is usually a cumbersome and time-consuming process. Since one of the objectives in this research was to apply fault injections to digital I&C systems of the type found in NPP systems, the need for a flexible and portable fault injection environment was a requirement for efficient application of the UVA fault injection-based dependability assessment methodology. To this end, the research effort developed the UNIFI fault injection environment to manage and coordinate the tasks associated with automated fault injection in physical I&C systems. UNIFI is described in detail in Section 5 of Volume 2. A summary of the UNIFI system is described in the following paragraphs.

Figure 1-6 shows the UNIFI tool with different plug-ins and how the tool interfaces with a target system and target system software development environment. In the UNIFI framework, the various fault injection plug-ins, the database that stores information, and results from experiments are located within the host computer.

The *operational profile generator* module receives as input a special pre-processed input file from the TRACE thermo-hydraulic simulation tool that provides all of the sensor data that the target system will acquire in its operational setting. The *experiment set up and control* plug-in function selects fault injector(s), configures the fault injectors, and initializes the UNIFI tool for a fault injection campaign. The *fault list generation plug-in module* generates a fault list that is parameterized with fault models of interest, locations of fault injections on the target system, types of fault injection, and when the s are to be injected. The *real time data monitoring and collection module* interfaces to the target system diagnostics and error monitoring server to collect error messages and error logs after each fault is injected into the target system. The *fault injection engine plug-in module* allows different types of fault injection techniques to be used by the UNIFI tool.

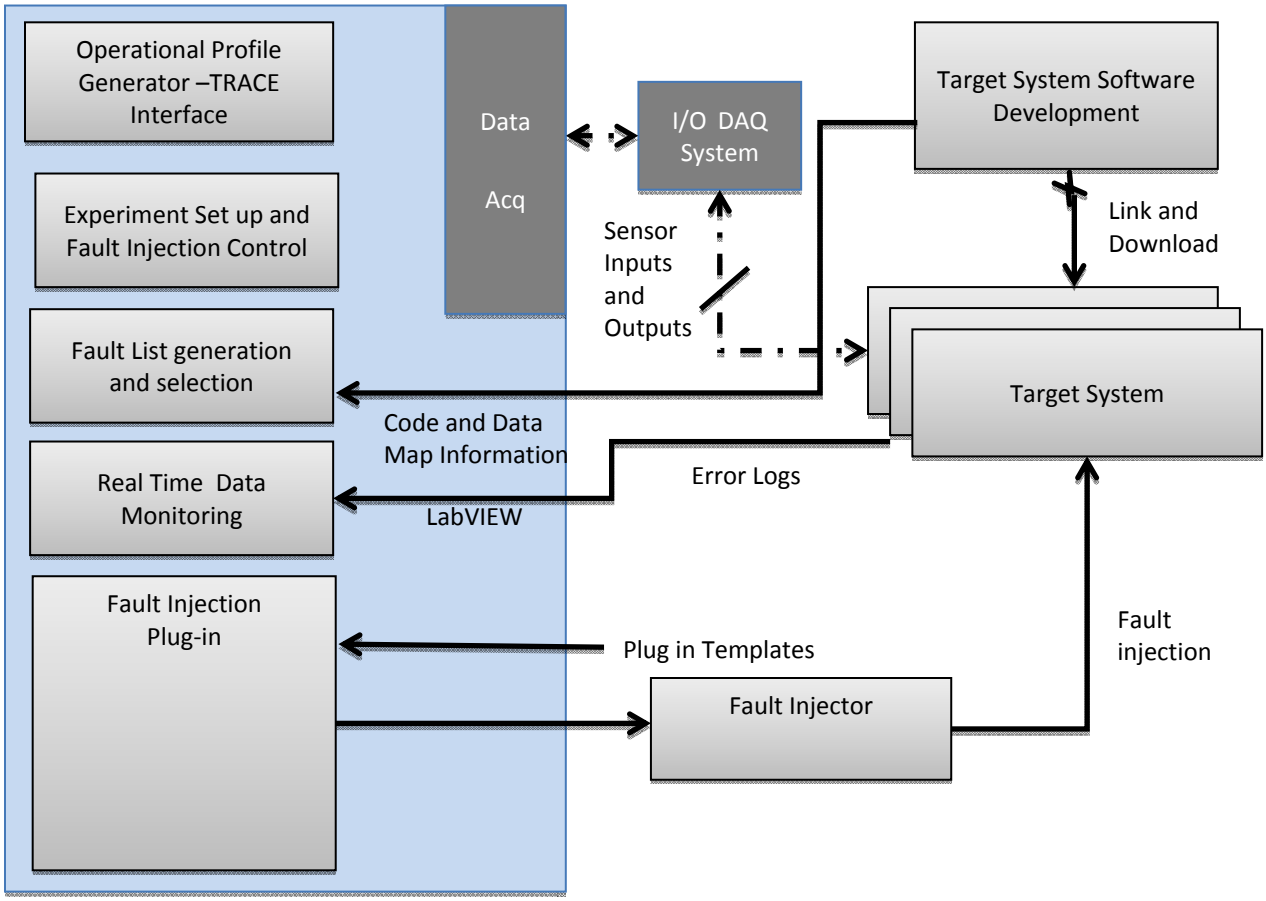

Figure 1-6 UNIFI Fault Injection Environment

1.9. 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 2993] Arlat, J, A Costes, Y Crouzet, J-C Laprie, and D Powell. "Fault Injection and Dependability Evaluation of Fault-Tolerant Systems." *IEEE Trans. on Computers*, no. 42 (1993).

[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.

[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).

[Smidt 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.

[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.

[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.

[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.

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

[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.

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

[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.

[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.

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

2. RESEARCH METHODOLOGY

2.1. Overview

The research methodology for this report (Volume 3) consists of six main steps, which are described in the remaining Sections of this report:

(1) Identification and selection of appropriate fault injection methods for Benchmark System II

(2) Developing the RPS software for Benchmark System II

(3) Analysis of measurement practices and uncertainty for fault injection

(4) Development of high performance fault injection techniques

(5) Development of function block-oriented fault injection

(6) Conduct of fault injection campaigns according to methodology

2.2. Identification and Selection of Appropriate Fault Injection Methods for Benchmark System II

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

There are many different types of fault injection techniques that may or may not be suitable for physical-based fault injection of a real digital I&C system. Building on lessons learned from the application of fault injection to Benchmark System I, the techniques typically employed for fault injection experiments were not optimal for fault injections on time-sensitive digital I&C systems.

Specifically, the use of In Circuit Emulator (ICE) machines proved to be overly time intensive. Software Implemented Fault Injection (SWIFI) required detailed knowledge of the operating system (OS) that was not attainable by non-vendor parties (such as UVA). Accordingly, the research effort defined and developed a set of requirements for high performance fault injection that would address the challenges faced with commercial-based ICE machine fault injection.

In addition, it was recognized that the tight process synchronized operation of Benchmark System II would require fault injection techniques that were virtually non-intrusive in time and space. The feasibility of implementing new fault injection techniques were assessed from the perspective of (1) working knowledge of the system, and (2) the technical working knowledge of the digital I&C system manufacturer.

This step of the research process supported step two and five of the assessment methodology. This step in the research process also helped narrow down the best candidates for fault injection for not only the Benchmark System II but also defined a set of objectives for high performance fault injection for digital I&C systems.

The result of this step was a set of candidate fault injection techniques for the benchmark systems, and design requirements for the High Performance Fault Injection module that was built as a part of this research effort.

2.3. High Performance Fault Injection for Digital I&C Systems

As discussed above, the need for fault injection techniques to support various fault models for fault injection in a manner that is minimally intrusive, controllable, repeatable, and reproducible is critical to the application of fault injection to digital I&C systems. Section 5 of Volume 1 extensively describes contemporary fault injection methods that helped guide the selection of a particular fault injection for digital I&C systems.

One of the realizations from performing fault injections on Benchmark System I was the challenge of trying to obtain a high degree of controllability and observability while maintaining a low level of intrusiveness. Various methods that were investigated (ICE-based fault injection and SWIFI) could only attain controllability or low-intrusiveness but not both at the same time. Additionally, 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 a system. Also realized was that there should be a common portable interface between the fault injector and the fault injection environment (i.e., UNIFI).

In the digital feedwater control system (DFWCS) effort discussed in Section 7 of Volume 1, custom interfaces between the fault injector and the fault injection controller were developed that had a low degree of reusability. Based on these experiences and lessons learned, it became apparent that a new implementation approach for injecting faults into 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 single high performance Field Programmable Gate Array (FPGA)-based fault injection module. By moving to a single multi-purpose fault injection platform designed specifically to support diverse fault injection methods on digital I&C systems, fault injection experiments 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 digital I&C systems would be consistent from one digital I&C platform to the next.

This aspect of consistency becomes important when fault injection is used as a *benchmarking activity*. That is, the same set of faults and operational conditions may be 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, the research effort designed, developed, and applied an FPGA-based HiPeFI module to Benchmark System II with outstanding success.

2.4. Development of the RPS Application

The UVA research team and the NRC technical manager selected a RPS multi-dimensional trip function that used a number of reactor variables. The RPS used for Benchmark System II was functionally equivalent to the RPS developed for Benchmark System I. As with Benchmark System I, the RPS function was similar to the RPS used in Smidt [Smidt 2004], in that it was a reduced model that did not incorporate all of the typical reactor measurements for the trip functions in an NPP RPS.

The RPS function in Benchmark System II used three process variable measurements: reactor coolant system flow, hot leg pressure, and steam generator pressure. These reactor process variables were monitored to prevent NPP power operation in an off-nominal basis, such as Loss of Coolant Accident (LOCA). The RPS function was developed using the software development

tools and environment for the benchmark system using advice from the NRC on the design of the RPS.

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 function as would be typically done for a licensed digital I&C system. The prototypical RPS was developed using the function block oriented autocode generation tools from the vendor of the benchmark system, which are qualified to produce code compliant with the NRC standards.

2.5. Analysis of Measurement Practices and Uncertainty for Fault Injection

As indicated in Volume 1, tools and techniques for experimental assessment of dependability properties should be treated as measurement instruments. First, as a measurement-based assessment process, fault injection depends on sound measurement practices. Second, since measuring a quantity (the measurand) requires quantitatively characterizing the quantity, a clear and univocal definition of the measurands is of uttermost importance [Bucher 2004].

As noted in Volume II, digital I&C systems are not typically designed to be monitored to the level and extent required for dependability evaluations by fault injection; therefore, a methodical approach for their observation was needed. In distributed digital I&C systems the situation is even more complex as the lack of central control, difficulties in obtaining a precise global time, and an accurate view of the global state of the system are all noted challenges. Moreover, as encountered on the efforts with Benchmark System I, the measurement of continuous time quantities such as fault and error latency are challenging due to unknown time delays in the propagation and the reporting of the error.

It was also recognized that measurement-based assessment processes must characterize the various types of uncertainty that can occur during the assessment process to better inform the types of reasonable conclusions can be made about the measured results and how those results can be interpreted in a broader, more general context. This becomes especially important when trying to relate or compare fault injection-based findings from one type of system to another.

In this effort the lessons learned in Phase 1 and Phase 2 were used to develop a characterization or theory of uncertainty with respect to fault injection-based assessment processes. Sincethere had been very little research in this area, the development of this important aspect of fault injection was considered to be initial work and should be viewed as a starting point for future research in the general I&C community.

2.6. Development of Techniques to Support Function Block Fault List Generation

In Section 7 of Volume 2 the concept of *pre-fault injection analysis* to improve efficiency and effectiveness of fault injection on digital I&C systems was introduced and developed. Continuing the research on effective fault list generation and fault modeling methods, Phase 3 of the research effort investigated methods to support the use of software-based fault models.

Developmental based faults, or software errors are viewed as potentially significant contributors to digital I&C system failures. As part of this research, UVA used a very conservative approach to support software domain fault emulation. The approach developed is *function block-oriented fault injection*.

Most digital I&C systems of the type found in NPP systems are programmed by function block programming (e.g. programming based on International Electro-technical Commission IEC 61131). 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 software fault models can be applied to a function block. The research described in this report was not aimed at promoting or endorsing a specific fault model, but rather was focused on identifying or developing techniques to allow the application of fault models to functions and 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). The applications development engineer understands the engineered I&C features of the I&C system through the function block representation. While function block oriented fault injection is strictly not a fault model, it 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 objective 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, emulating function block faults can assist in identifying the failure modes or errors that result, and under what conditions those failure modes or errors arise.

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.

2.7. Conduct Fault Injection Campaigns on Benchmark System II

This research step applied the UVA fault injection-based dependability assessment methodology to Benchmark System II. Using the fault injection environment and the fault injection techniques developed for Benchmark System II, a number of fault injection experiment campaigns were conducted to assess the capability of the methodology to support PRA modeling activities and supply system dependability information that could be used in the regulatory review of digital I&C systems.

The fault injection experiments were conducted using operational profiles generated from the TRACE-based operational profile generation tool. The fault injection was applied to the RPS model function and function blocks, the OS function variables and data structures, OS memory locations, registers in the central processing unit (CPU) of the processing module, an in the peer-to-peer communication functions. Approximately 15,000 fault injections were conducted. Statistical estimation of the parameters of interest include fault coverage factors, latency in fault detection, real-time responses under faulted conditions, and block faulted behavior.

2.8. References

[Bucher 2004] Bucher, J.L. *The Metrology Handbook.* American Society for Quality, 2004.

[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.

3. Description of Benchmark System II and RPS Configuration

3.1. Introduction

The last Section of Volume 1 introduced by way of an overview the general architectural features of the benchmark systems use in this study. This Section supplements the overview Section in Volume 1 with additional information on the operation of Benchmark System II, in particular the self-testing and fault tolerance features of the system. These features are noteworthy of discussion because they were explicitly tested by fault injection methods develop by this research effort.

3.2. Benchmark System II

Benchmark System II was developed to model a safety grade, qualified digital I&C system developed for safety or high reliability functions in NPPs. The benchmark system received from the NRC for this study is a scaled version of a typical 4-channel 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. Its scalability permits development of solutions for a spectrum of safety-related tasks within the NPP system. Typical applications include the RPS and the Engineered Safety Features Actuation System (ESFAS).

It should be noted that the benchmark systems used in this research were testing platforms to exercise the fault injection methodology. In that regard, the benchmark systems represent the complexity of RPS processing and fault tolerance from both a hardware and software perspective. However, typical in-plant RPS digital I&C systems are considerably more complex in their fault tolerance and diversity attributes than the representative benchmark systems used in this research. Therefore, the results of this study are intended to be a reflection of the ability of the methodology to accommodate fault injection experiments on digital I&C systems, and should not be construed as representative of the performance and suitability of the benchmark systems for RPS applications.

3.3. Architecture and System Description of Benchmark System II

3.3.1. Architecture and System Description

The Benchmark System II configuration consists of two chassis assemblies. As shown in Figure 3-1, each chassis includes termination panels, power supply modules, three main processor modules, redundant input/output (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.

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

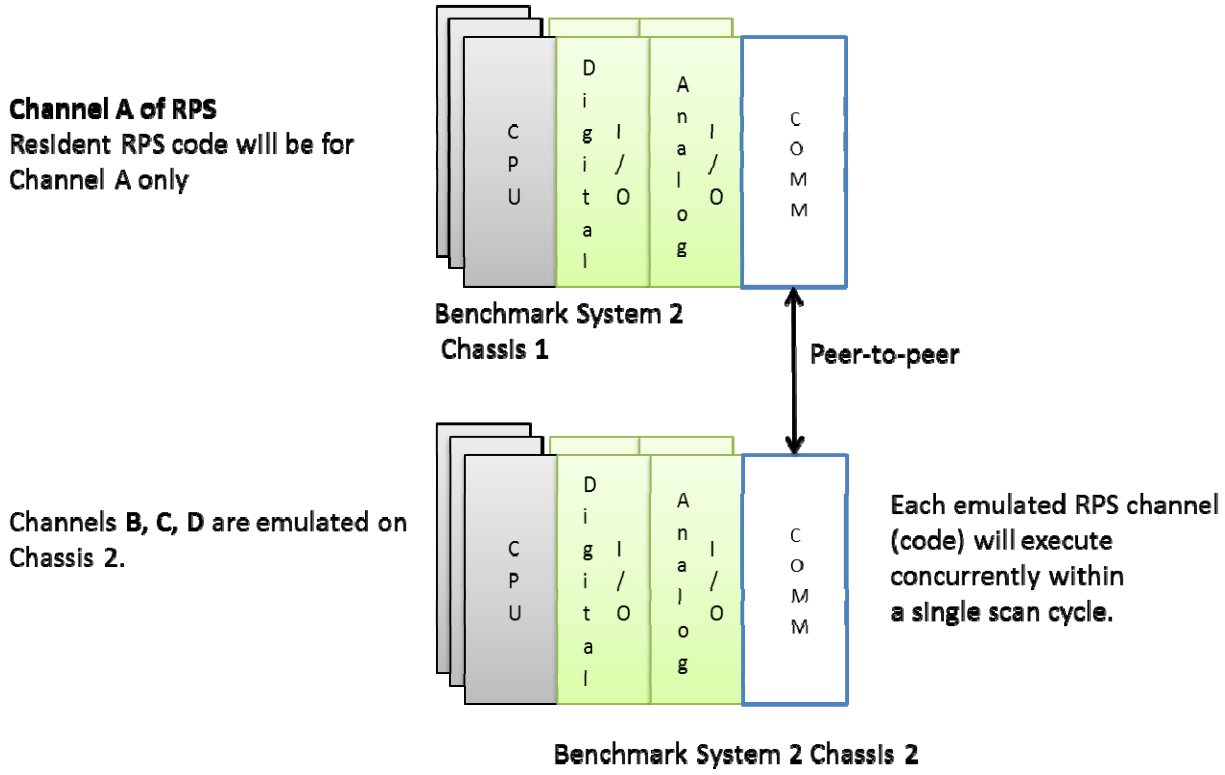

Figure 3-1 Benchmark System II architecture and configuration

Each channel in the configuration is triple modular redundant (TMR) from input terminal to output terminal, as shown in Figure 3-1. The TMR architecture is intended to allow continued system operation in the presence of any single point of failure within the system. The TMR architecture is also intended to allow a system to detect and correct individual faults on-line, without interruption of monitoring, control, and protection capabilities. In the presence of a fault, the system alarms the condition, removes the affected portion of the faulted module from operation, and continues to function normally in a dual redundant mode. The system returns to the TMR mode of operation when the affected module is replaced. Thus, the system acts as fault masking architecture with no active reconfiguration.

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 a module under power while the system is running, with no impact on the operation of the application.

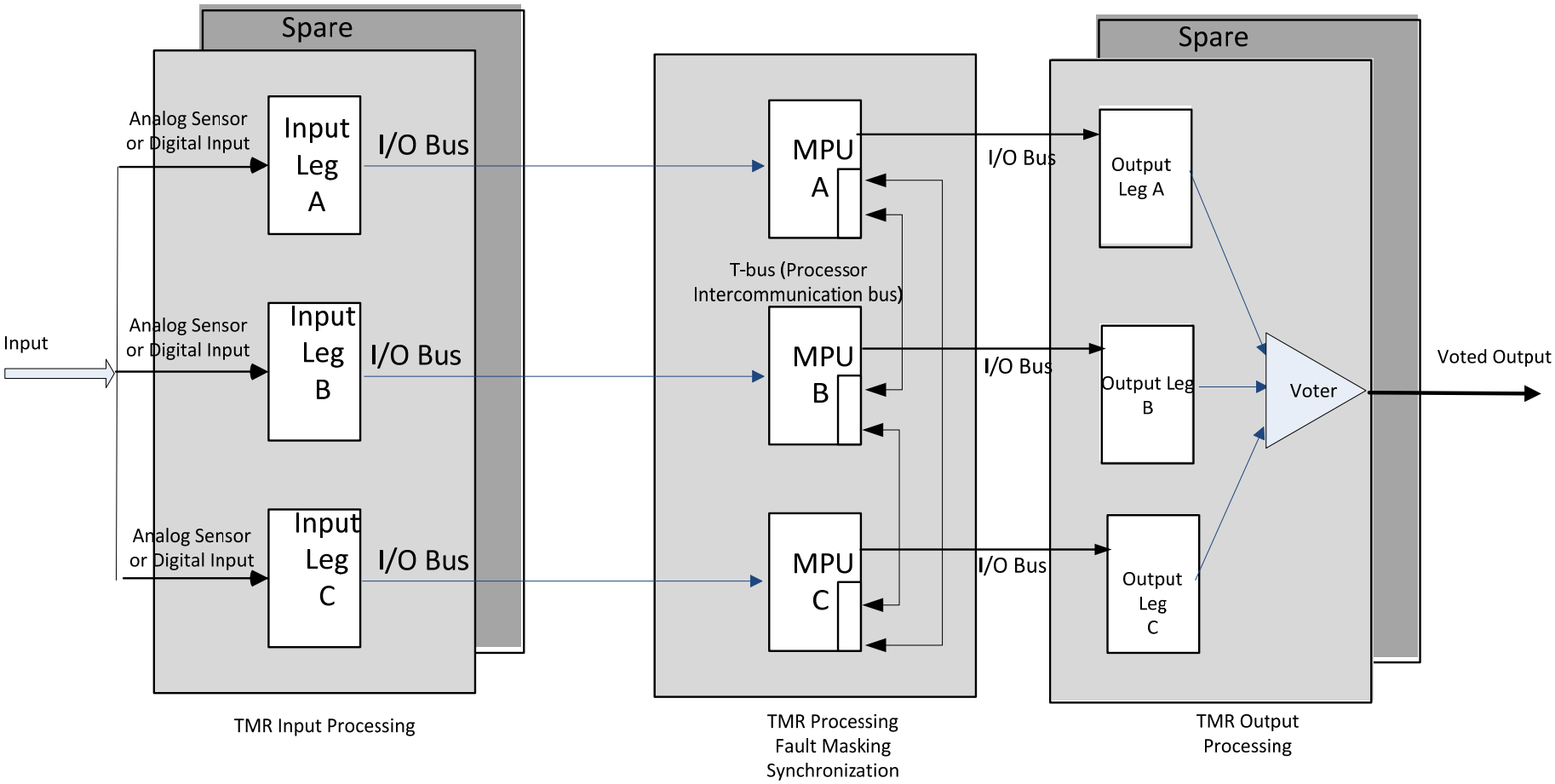

Figure 3-2 Benchmark System II detailed architecture

Figure 3-2 also shows the arrangement of the input, 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 MPU 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 local memory. Signal conditioning, isolation, and processing required for each leg is also performed 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.

3.3.2. Fault Tolerance Operation

Benchmark System II uses 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 (i.e., 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. Each MPU module operates independently with no shared clocks, power regulators, or circuitry.

Each MPU module contains two 32-bit processors and controls one of the three signal processing legs in the system. 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 (i.e., the RPS in Benchmark System II) and all system diagnostics at the MPU module level. Between the 32-bit MPU processors is a dedicated dual port random access memory (RAM) that allows direct memory access data exchanges.

Communication of data between the main processor modules and the input and output modules is accomplished over the triplicated I/O data bus. The system uses cyclic redundancy checks (CRCs) to ensure the health of data transmitted between modules. Should a main processor module lose communication with its respective leg on any of the input modules in the benchmark system or the CRC reveals that the data has been corrupted, Benchmark System II will retry the data transmission up to three times. If unsuccessful, input tables at the MPU module level are nulled with data to reflect a de-energized state. Errors such as an open circuit data bus, short circuit data bus, or data corrupted while in transit will force the input table entries to the de-energized state.

At the beginning of each application task cycle, each primary MPU takes a snapshot of the input data table in dual port RAM and transmits the snap shot to the other MPU modules over the fault tolerant triplicated bus (T-bus). This transfer is synchronized using a fault tolerant clock. Each MPU module then independently forms a voted and consistent input table based on respective input data points across the three snapshot data tables. If a MPU 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.

For digital inputs, the voted input table is formed by a 2-out-of-3 majority vote on respective inputs across the three data tables. The voting scheme is designed to de-energize to actuate trip applications, always defaulting to the de-energized state unless voted otherwise. Any single leg failure or corrupted signal feeding a MPU module is corrected or compensated for at the MPU 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 MPU is compensated at the MPU module level when the voted data table is formed. All input and output modules include self-diagnostic features designed to detect single failures within the module. Significant errors are alarmed.

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 MPs. Unlike the I/O modules, however, the three communication bus interfaces are merged into a single microprocessor. That microprocessor votes on the communication messages from the three MPs 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 MPs.

The communication paths to external systems have appropriate levels of 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 MPU diagnostics monitor the health of each MPU as well as each I/O module and communication channel. The MPU 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. All discrepancies are flagged and used by the built in fault analyzer routine to diagnose faults. The MPU diagnostics perform the following functions:

- Verification of fixed-program memory;
- Verification of the static portion of RAM;
- Verification of the dual port RAM interface with each I/O communication module (IOCCOM);
- Checking of each IOCCOMs ROM, dual port RAM access, and loopback of RS-485 transceivers;
- Verification of the fault tolerant clock interface; and
- Verification of the triplicated voter bus interface.

When a fault is detected on a MPU module, it is annunciated and voted out, and processing continues through the remaining two MPs. When the faulty MPU is replaced, it runs a self-diagnostic to determine its health. When the self-diagnostic is successfully completed, the MPU then begins the process of “re-education,” where the control program is transferred from each of the working units into the returning MPU. All three MPU then resynchronize data and voting, and the replacement MPU is allowed in service.

If a standby module is installed in the paired slot with the faulty module, and that module is deemed healthy by the MPs, 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.

Benchmark System II fault tolerance architectural features are typical of end-to-end modular redundancy systems found in avionic systems and process control systems. The critical sub-systems that were targeted for fault injection are shown in Figure 3-3,. Each stage of processing provides both masking-type fault tolerance and dynamic fault tolerance (self-tests) to ensure fault free processing in the presence of a single fault. These locations were examined for fault injection during the course of the research.

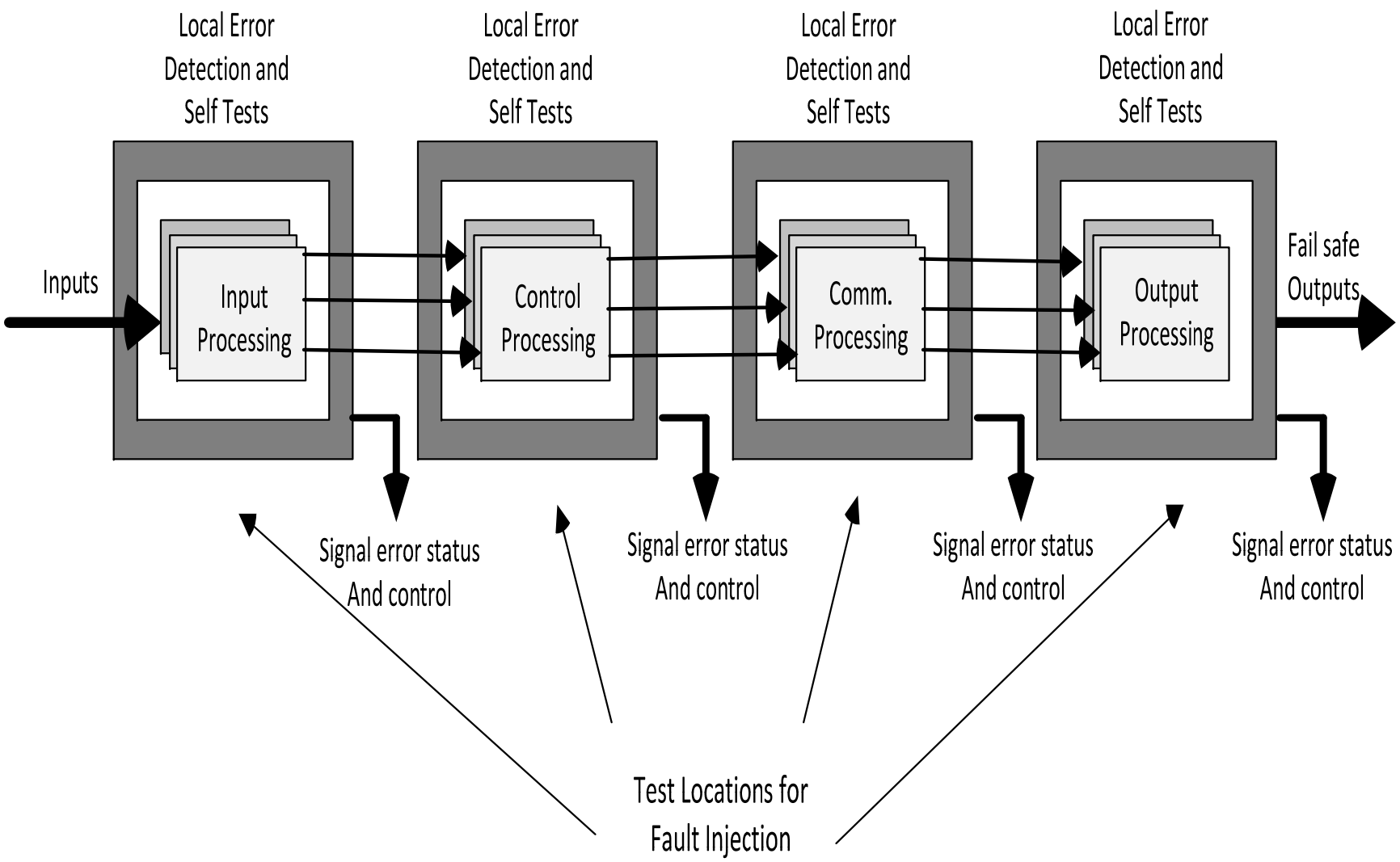

Figure 3-3 Benchmark System II sub-systems targeted for fault injection

3.3.3. Monitoring Interface

External communication to non-safety monitoring stations for the purpose of monitoring the operation of the RPS application was facilitated by the Diagnostic Monitor (DM) interface and the Sequence of Events recorder. The DM gathers system level diagnostic health messages and application level messages from the Benchmark II System and forwards this information to the operator monitoring station. Referring to above Figure 3-1 and Figure 3-2, each stage of processing is associated with a collection of error detection mechanisms. Any detected error at any stage of processing results in a mitigation response and a set of error messages for that error condition are sent to the DM via the communication modules, which in turn forwards the messages to the operator work station. The DM and the messaging protocol are designed to be non-interfering with respect to the safety functions operating on the Benchmark II system.

3.3.4. Runtime Software Operation

The Benchmark System II software consists of the operating system that is resident on the various microprocessors within the system and the application program itself. The Benchmark System II OS software consists of the firmware that resides on the microprocessors in the MPU, the I/O, and the communication modules.

Two sets of firmware exist on the MPU. The primary 32-bit microprocessor has the operating environment firmware. The IOCCOM microprocessor interface has its own firmware to communicate with the I/O and communication modules. The primary microprocessor firmware includes all the built-in self-diagnostics and triple modular redundancy functions.

Upon power up, the processor module goes through the power up initialization and diagnostics. The power up sequence includes a series of power up diagnostics – microprocessor tests, RAM tests, flash memory tests, watchdog tests, clock calendar test, etc. Upon successful completion of power up sequence, the processor begins execution of cyclic application tasks in the Scan Task group.

After successful power up, the primary processors (MPU A, MPU B and MPU C) for each channel 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 identical input data value representations. After the MPs complete the control , from the MPs are provided to the I/O bus microprocessors through dual port RAM. The I/O bus microprocessors then transfer that data to triplicated microprocessors on the output modules. The output modules then set the output hardware appropriately on each of the triplicated sections verify correct operation and vote on the appropriate state.

The principle part of the OS is the RTE. Figure 3-4 shows the processing behavior of the RTE. The RTE 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. The RTE consists of three processing groups: the Scan Processing group (rate group for application tasks), the Communication Processing group, and the Background Processing group.

The Scan Processing group operates in a non-preemptive states and executes the main I&C function application at the dynamic sample time of the plant (typically every 10 ms to 200 ms). The Communication Processing group runs at the next level of priority and processes the data tables for the IOCCOM processor and the communication messages for peer-to-peer communication between the chassis. Any time left over from these two processing groups is surplus time, and in this time the Background Processing group runs the system self-test tasks.

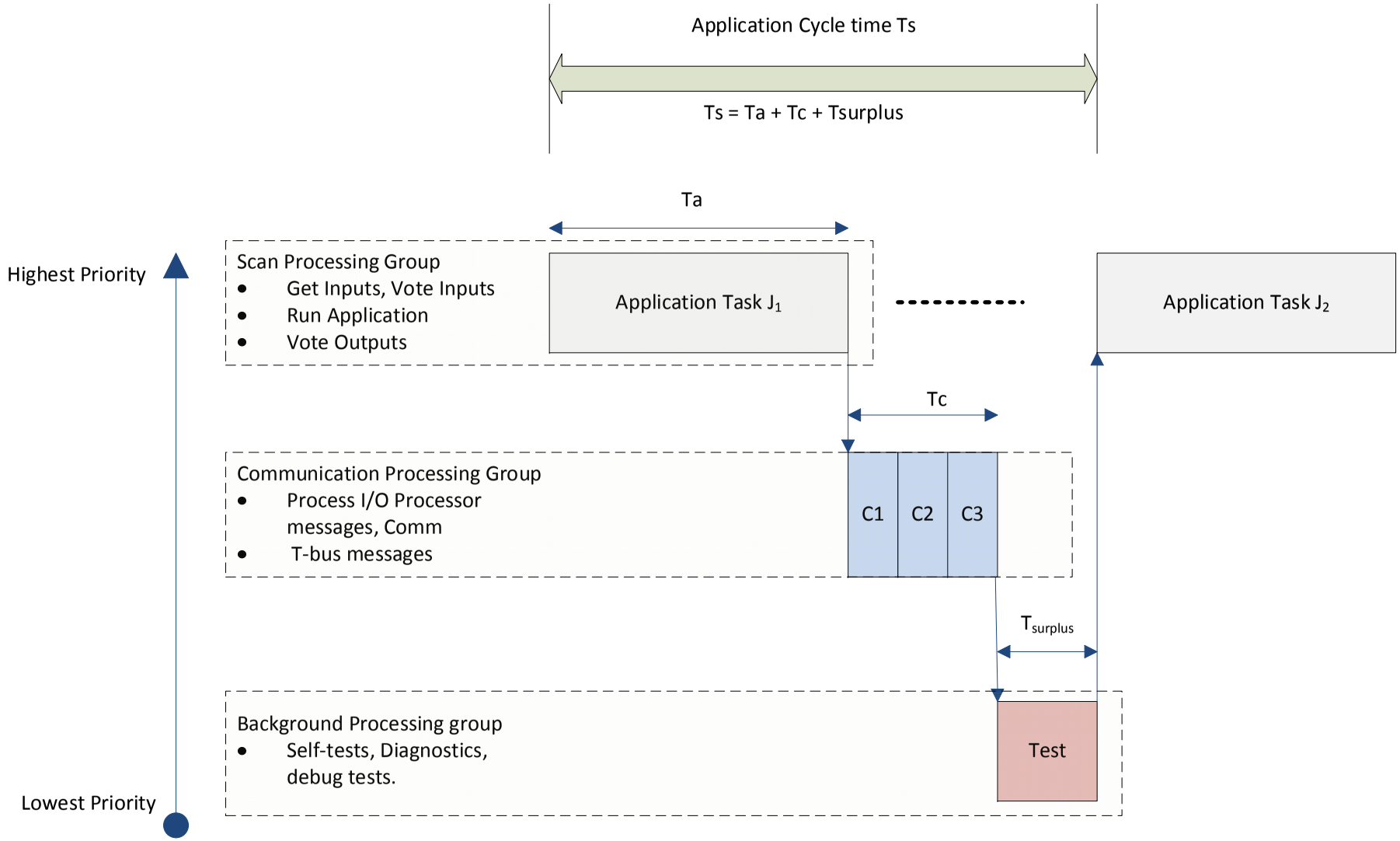

Figure 3-4 Scan Processing group, Communication Processing group, and Background Processing group prioritization

3.4. RPS Configuration for Benchmark System II

The RPS software development environment for Benchmark System II starts with the specification of the I&C system by composing function diagrams and hardware diagrams using the *function block editor* tool. This tool performs a series of consistency and plausibility checks on the I&C 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. The function block diagram approach for developing I&C systems is a standard approach in the nuclear I&C industry. The software development provides a number of engineering functions in support of the overall process of creating, testing, and verifying the operational functionality of the developed I&C code, including:

- Specification of I&C functions and hardware topology
- Automatic code generation from the I&C system representation
- Verification of generated code
- Validation of I&C functions in a simulation environment
- Compilation and linking of the software for the target system
- Loading the software onto the target system

Using the software development and testing environment of Benchmark System II, the RPS configuration for Benchmark System II was configured as a two out of four voting system for three monitored reactor process signals. The signals were hot-leg pressure, coolant flow, and steam generator pressure. This means that if any two channels indicate that any of the measured sensor variables from the reactor are out of safety range, the Trip Logic initiates a shutdown command to the reactor to shut down. If a channel becomes faulty and it is detected as such, the affected channel of the Benchmark System II gracefully degrades from a TMR system to a dual configuration to allow continued operation in a limited capacity while maintenance and service can be performed using off-line.

Figure 3-5 is a process diagram generated by the function block development system for the hot leg pressure function in channel A of the Benchmark System II RPS. The following discussion describes the RPS scan processing task functions.

Referring to Figure 3-5, each RPS channel gathers three redundant reactor sensor signals for each of the monitored the hot leg pressure variables ((see C10 through F10 in Figure 3-5). The process values are acquired by the AIN analog input function blocks (see C9 through F9 in Figure 3-5), which convert the signals from analog values ranging from 4 ma to 20 ma or 1 VDC to 5 VDC to integer real numbers in the range of 819 to 4095. The max and min parameters of the AIN function blocks allow the inputs to be scaled between the upper and lower engineering units of interest.

The three hot leg pressure signals are then compared to the upper and lower safety set point values for the RPS using greater than and less than function blocks (see the GT and LT function blocks in C8 though F8). In Benchmark System II, the upper limit of safe operation was set to 2500 psig and the lower limit of safe operation was set to 1800 psig based on the TRACE model.

The output of the three set point comparisons are pair-wised “OR’ed” by the OR function blocks to form a channel alarm indicator for the measured hot leg pressure signal (see C8 through F8 in Figure 3-5). The outputs of the three channel A OR gates are provided into a final channel A OR gate to produce a channel A signal for alarm and trip functions (see C7 through F7 and E6 in Figure 3-5).

The channel A signal is combined with the hot leg pressure OR signals from channel B, channel C, and channel D that are transferred to chassis 1 using a direct peer-to-peer redundant communication link between the two chassis. The communication of data (channel alarms) between chassis is accomplished using the TR_URCV_BOOL and TR_URSEND_BOOL function blocks (see B6 and E2, respectively, in Figure 3-5). Thus all channels exchange their trip signals with each other to form a distributed 2 out of 4 vote comparison.

The three channel alarms are threshold compared using the NUMBITS function (see E5 in Figure 3-5). The NUMBITS function counts the instances of the channel alarms. If number of channel alarms equals or exceeds two alarms, a trip signal is generated for chassis 1 by the GE function (see E4 in Figure 3-5).

The above description of operation for monitoring the reactor process variables is replicated for each RPS signal in each channel. The final trip signals for each channel are forwarded to the UNIFI/Labview measurement and monitoring module where they are recorded and time stamped when they change state (see E3 in Figure 3-5).

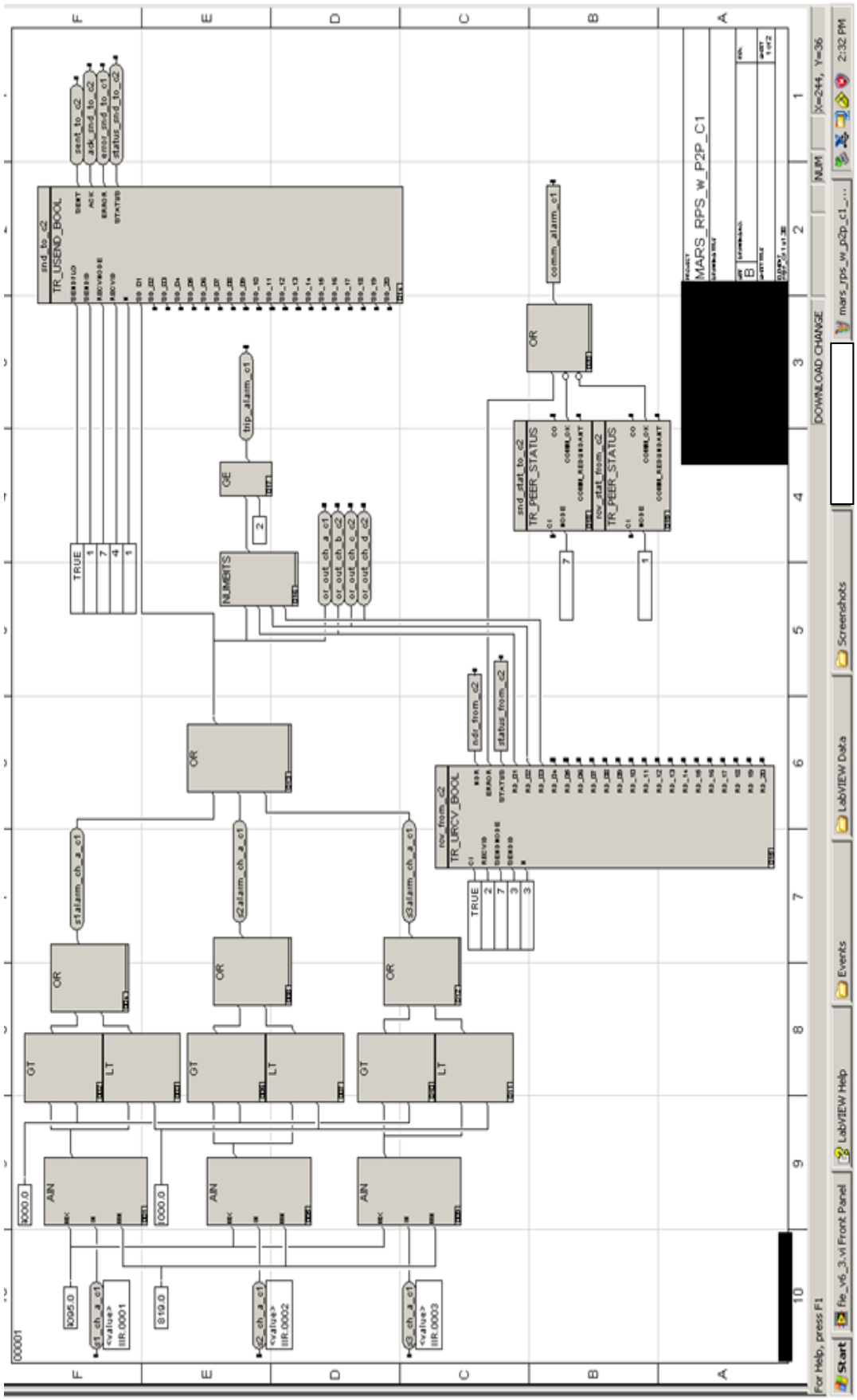

Figure 3-5 Channel A RPS for Benchmark System II

4. Identification and Selection of Fault Injection Techniques for Benchmark System II

4.1. Introduction

This Section describes identification and selection of fault injection techniques for Benchmark System II. The research and development process for this task is as follows:

(1) Identify from Section 5 of Volume I appropriate fault injection techniques for Benchmark System II.

(2) Determine the feasibility of implementing the identified fault injection techniques based on time, effort, and vendor support required to implement the techniques.

(3) Select the appropriate fault injection techniques.

4.2. Identification of Fault Injection Methods for Benchmark System II

In the previous Section, Figure 3-3 identified the points where fault injection should be considered with respect to Benchmark System II. These points included input processing (both analog and digital), the processor modules (Main CPU and IOCCOM CPU), the peer-to-peer bus communication module, and the output processing module (digital). Each of these modules have a number of error detection and fault tolerance functions and self-tests in support of high levels of safety critical I&C operations.

An effective fault injector must be able to emulate various fault models or fault classes so the assessor of the dependable system can test the fault tolerance mechanisms under the effect of various types of faults that a digital I&C system may be subjected to over its operational life. The ability to accommodate various fault models or fault classes using the same fault injection environment is a valuable feature. Furthermore, the ability to use several different fault injection techniques from a single environment certainly aids in the overall usability of fault injection from one platform to another.

Another critical factor in selecting an appropriate fault injection method for a digital I&C system is the amount of intrusiveness that can be tolerated by the system. While complete unobtrusiveness is a desirable goal, it is not feasible to attain such a goal when performing in-situ fault injection (i.e., injecting faults internal to the devices in the system). Thus, time intrusion must be minimized with respect to the operations of the system. Practically speaking, fault injection stimulus should be constrained by two axioms:

(1) The real-time behavior of the system at its environment interface should not be altered by the fault injection process.

(2) The behavior of error detection and fault tolerance mechanisms onboard the system under test should not be any different from normal operations due to the activation of the fault injection process.

The first axiom states that all of the input and output behavior should remain unchanged in the presence of a fault injection process. The important point is the that process of injecting a fault should not alter the behavior of the system but should alter the system behavior. An unaltered input and output behavior means that all inputs are collected at the same time, and all of the

outputs are commanded at the same time with respect to the cycle time interval of the benchmark system despite the act of injecting a fault into the system while it is operating.

The second axiom, states that the process of fault injection should not change the behavior of the fault tolerance mechanisms, which means that the process should not inadvertently cause a fault or error detection event to occur. A fault injection process itself, if it is too intrusive, may be viewed by the system as abnormal behavior and thus cause the system error detection mechanisms to respond to the fault injection process and not the injected fault. An example of this unwanted behavior is tripping a time-out function or watch dog timer.

The Benchmark System II architecture is a synchronous data and message passing system. In order to ensure that data processing operations remain consistent from inputs to outputs the three processing modules (i.e., processing legs) are frame synchronized twice every scan cycle or sample cycle. These synchronization events take place at the beginning and end of every scan or cycle interval. This is in contrast to Benchmark System I, which was based on an asynchronous message passing operation. The key point is that tightly synchronized I&C systems such as Benchmark System II do not tolerate excessive time intrusions from fault injection processes. If the time intrusion due to fault injection is too long, the intrusion tends to un-synchronize the processors before the fault injection occurs. Therefore, fault injection techniques that are minimally intrusive in time must be used with systems like Benchmark System II.

4.2.1. Identifying and Selecting Fault Injection Techniques for the Input/output Processing Modules

The key processing functions involved in the input/output processing are:

(1) Multiplexing of the analog input signals
(2) Analog-to-digital (A/D) conversion of analog signals
(3) Coordination of the A/D conversion by the microcontroller
(4) Voting of the digitized signals
(5) Transfer of the voted digitized signals between modules
(6) Self-testing

Faults can occur at each of these input and output processing functions. The key component in the Benchmark System II I/O modules is the onboard triplicated micro-controllers. These modules are responsible for coordinating the actions of the signal conversion, signal processing, voting, data transfer, and self-tests. Therefore, the micro-controller in the I/O modules was selected as a component for fault injection.

The analog input processing module has no user accessible ports (i.e., serial or Ethernet) that allow access to the operations of the micro-controller and provide a control path for fault injection. The same is true for both the digital input and output processing modules. The lack of a user accessible input port on these modules ruled out fault injection techniques such as SWIFI, which require a port to interface to the fault injection exception handlers that execute on the target processor.

The next alternative that was examined was use of a Joint Test Action Group (JTAG) boundary scan fault injection process. JTAG boundary scan is an Institute of Electrical and Electronics Engineers (IEEE) testing standard that allows the boundary of a chip (e.g., the pins) to be used to detect faults in a module. JTAG fault injection involves invoking the JTAG serial port of a device and loading corruption values into the boundary registers of the device. The user manuals did not reference a JTAG test port capability; however, JTAG test ports were

discovered in the Benchmark System II digital output and analog input boards. It was concluded that JTAG fault injection would be a viable option for in faults into Benchmark System II through the digital output and analog input boards.

Another option considered was on-chip debugger (OCD)-based fault injection. However, the microcontroller used for the input and output processing modules did not have full OCD support, so OCD fault injections were not an option.

The JTAG fault injection technique appeared to be the most straight forward approach to implementing fault injection experiments on the input and output boards. The JTAG fault injection capability that had been developed in Phase II of this research was evaluated for use as the principle fault injection process for the input and output modules. As noted in Section 5 of Volume 1, JTAG fault injection has limited capability to reach the internal registers of a microcontroller. The JTAG process is used mainly to test the input/output ports and signals of a device. SWIFI fault injection techniques have greater reachability to the internals of microprocessor, but they require an external port to communicate with the Interrupt Service Routine (ISR) that implements the fault injection module onboard the micro-controller. In addition, through the UNIFI fault injection tool input signals to Benchmark System II can be corrupted with various environmental disturbance functions such as Gaussian noise and loss of signal.

4.2.2. Identifying and Selecting Fault Injection Techniques for the Processing Modules

For processor-based fault injection, the survey results in Section 5 of Volume 1 and knowledge of the benchmark system were used to identify several fault injection techniques that could be successful on Benchmark System II. These techniques were selected based on the following criteria:

- Applicability to the benchmark system – The techniques could be accommodated by Benchmark System II as it was configured.

- Controllability – The techniques allowed precise control over the attributes of the fault model parameters – time, value and location.

- Support for a variety of fault models – The fault models listed in Figure 1-6 of Section 1 served as the fault classes for fault injections. These fault classes include:

  (1) Hardware based faults including transient and permanent faults,

  (2) Interaction faults including communication faults and configuration faults, and

  (3) Development based faults including faults in software I&C based functions and operating system based functions.

  Techniques that can maximally support these fault classes are favored.

- Support for precise fault activation – This essential feature is related to the requirements of the FARM model. 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 of the fault injection experiments.

Based on the above criteria, three candidate fault injection techniques were identified for Benchmark System II:

- OCD based fault injection
- JTAG Based fault Injection
- SWIFI based fault injection

Each of these techniques has advantages and limitations. The aim of the research was to use several of these techniques jointly in a complementary fashion to maximize the reachability and effectiveness of the fault injection studies. each techniques is described in the following sections.

4.3. IEEE 1149.1 JTAG Based Fault Injection

Most current integrated circuits have external input and output pins linked together in a set called the *Boundary Scan Chain* (BSC). JTAG was designed to access the BSC by means of a virtual register (Boundary Register) connected to its input and output pins. It is possible to alter the contents of the BSC and hence alter the current signals on the pin-outs by serially shifting in data into the Boundary Register. At the same time, bits from the Boundary Register are serially shifted out to the output pin of JTAG controller. Because of the common occurrence of the JTAG port on current processors and integrated circuit (IC) devices, there has been fairly extensive work done on attempting to perform fault injections via this technique.

One of the first works on the use of the JTAG controller for fault injections claimed that IEEE Std 1149.1 (the JTAG standard), when used as a standalone fault injection technique by using the BSC architecture, cannot satisfy the requirements for successful fault injection campaigns and would require additional logic and functionality for meeting performance requirements and accessibility to internal components [Santos 2003].

However, researchers at the have shown that Scan Chain Implemented Fault Injection (SCIFI), which is performed by accessing built in logic specified by IEEE Std 1149.1, can be used for dependability validation of embedded computer systems. Further, the process improves the controllability, reachability and observability of the system over standard hardware-based fault injection techniques [Folkesson 2003]. Moreover, the Chalmers University of Technology research showed that the SCIFI technique provides significantly faster performance than software implemented fault injection.

Since then, researchers have performed validations of various systems by performing fault injections based on accessing the capabilities of the JTAG interface. The tests were performed either by connecting the fault injector to the backplane of a system [Chakraborty 2007], or by creating a switch module for accessing all of the components [VanTreuren 2007]. Unfortunately, both of these fault injection campaigns required additional hardware to be introduced into the system to perform the fault injections.

The issue with adding hardware into a system to perform fault injections was solved in [Pignol 2007] by using a commercial off-the-shelf (COTS) JTAG in conjunction with software running on a host computer. The fault injection campaign was successful, but it was noted that real-time

operation of the software was problematic and could lead to significant overhead. Additoonally, the experiments used intrusive software running on the target system.

More recent solutions for performing JTAG-based fault injections employ an FPGA that was programmed to perform experiments with a predefined set of faults [Portela-Garcia 2007] as the fault injector. This solution is unobtrusive as it does not require additional hardware on the target system and the performance is not degraded by including a host computer as the driver. However, preprogramming an FPGA with fault injection sequences deemed this solution to be specific to the fault injection campaign. Also, the performance capabilities of fault injection campaigns with real-time target systems could not be established.

To summarize this review of JTAG fault injection systems:

Advantages:

- It is possible to perform successful fault injection campaigns by using the functionality provided by a JTAG OCD.
- The process can provide a means for injecting faults internal to integrated circuits, processors, FPGAs and ASICs when other techniques are difficult to implement.
- It usually provides good controllability and observability.

Disadvantages:

- Fault injection experiments are limited to locations that can be reached by the boundary scan registers.
- The method requires external hardware to be designed or purchased to access the JTAG port of the device.
- The device under test is temporarily taken out of operational mode and put into test mode while the fault injection experiment is active.

The attraction of the JTAG fault injection method is that it complements other fault injection methods (such as ICE-based fault injection or OCD-based fault injection) by providing a diverse method that reaches different portions of the internal IC under test. In addition, since most processors and complex IC devices have a JTAG port, the method provides a means to gain access to the IC for fault injection when other means are not possible. This is particularly true for devices like FPGAs that are principally hardware devices and usually do not have a software component.

For these reasons, of a JTAG fault injection module was designed and implemented for the UNIFI fault injection environment, which is described later. However, it was also realized that the JTAG fault injector would probably be used late in the project and therefore would not be the principle means of fault injection for this research.

4.4. OCD-based Fault Injection

Experience with ICE-based fault on Benchmark System I and the DFWCS fault injection study indicated that the method is a sub-optimal fault injection method for real-time digital I&C systems. Given that Benchmark System II was more sensitive to time intrusions of fault

injections methods like ICE-based fault injections were not practicable. The time delays for halting and resuming the target system processor to insert faulst into the processor or memory of the target system can be significant (10's of milliseconds or greater). For this reason OCD-based fault injection methods are better for minimizing time delays associated with fault injections. However, OCD-based methods require that the target system processor have built-in OCD hardware functions to support the fault injection process. Fortunately, in the case of Benchmark System II, both the main processor and the IOCCOM processors support OCD fault injections through the MPC860 processor background debug mode (BDM) port.

To support OCD-based fault injection techniques for Benchmark System II, two methods were employed: (1) the use of a COTS debugger tool called iSystems 3000, and (2) the development of a high performance fault injection module utilizing FPGA technology to implement OCD-based fault injection. The latter will be described in the next Section. These two methods are compared in the next Section to inform the reader of the relative merits of both fault injection methods.

4.5. Software Implemented Fault Injection (SWIFI)

Software-implemented fault injection encompasses techniques that inject faults through software executed on the target system. 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 or ISR, 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.

Software-implemented fault injection (SWIFI) relies on the assumption that the effects of actual 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. This assumption usually holds true for transient faults, but for permanent faults it presents some difficulties due to the repeated invocation of the fault injection exception handler every time a register or memory location is referenced.

Interest in SWIFI-based fault injections is centered on the enhanced performance the SWIFI-based method can provide compared to ICE-based fault injections, and because it is a processor-independent method. SWIFI-based fault injections can execute processor register and memory fault injections in a few milliseconds or less. With appropriate internal processor support for breakpoint registers, the SWIFI method can trap target system states on user variables, operands, instructions, and control flow conditions, which is a desirable feature for precision fault injection experiment control.

One example of a SWIFI-based fault injection method is Xception [Kanawati 1995(b)]. Xception injects faults through an ISR executing in kernel mode. The ISR is typically added to the OS software as a build or patch. The fault injection ISR is typically loaded with a fault list from a host computer. The ISR, therefore, must have an available dedicated external port on the target system to use for communication with the host computer.

The fault injection ISR can be triggered by the following CPU events:

- Operating code fetch from a specified address
- Operand load from a specified address
- Operand store to a specified address
- A specified time elapsed since start-up

Using these triggers it is possible to emulate transient faults and in some cases permanent faults.

The disadvantage of the SWIFI method is that it requires modifying the system software or adding software to the target system (e.g,. a low level ISR). This can be challenging if the complete details of the system software are not known, particularly with respect to implementing the ISR so that it is properly nested in the processor interrupt chain. In addition, an available serial port or Ethernet port must be available on the target system to communicate with the fault injection host computer. In the case of Benchmark System II, there was a user-defined serial port available.

Although the SWIFI method was a feasible option for Benchmark System II, this method was pursued beyond assessing the feasibility of the approach. Instead, the research focused on te OCD-based fault injection methods, which provide the most flexibility and performance with the least amount of modification to a target system. However, the SWIFI method may be a worthwhile fault injection technique for Benchmark System II that the vendor may want to pursue. As indicated in Volume 1 Section 5, there are number of academic tools and a few commercial tools that support the SWIFI-based fault injection that may be of interest to the vendor of the equipment and developments system for Benchmark System II (e.g., Xception, Generic Object Oriented Fault Injection (GOOFI), and the UNIFI tool developed by this research).

4.6. Summary of Fault Injection Techniques for Benchmark System II

Table 4-1 presents suggested physical-based fault injection methods for Benchmark System II. The table reflects what is possible given sufficient technical information about a system. The relatives advantages of the fault injection methods are:

- The most prevalent and accessible fault injection method for Benchmark System II was the OCD-based fault injection method. The CPUs on the MPU modules support the BDM test interface.

- The next most promising fault injection methods was the JTAG-based fault injection method because the microcontrollers on the input and output processing modules and the communication module had accessible JTAG BSCs.

- The SWIFI-based method is a fault injection option for the processor module, but requires adding a special purpose ISR handler to the system software.

- The SCIF-based method and the OCD-based method provide the least amount of investment in terms of time and resources, in that order.

Communication or interaction faults would be best represented by using two different fault injection methods. The first method would be a fault injection method such as SCIFI to introduce faults into the communication CPU of the communication module. The second fault injection method would use an interceptor or jammer module that corrupts various bit fields in the Ethernet or TCP/IP packets as they are being transmitted over the peer-to-peer bus. The fields that are bolded represent techniques that were planned in this research to implement fault injection experiments based on the level of information known about Benchmark System II

Table 4-1 Summary of fault injection techniques for Benchmark System II.

Module	OCD	JTAG/SCIFI	Interceptor	SWIFI	Pin Level
Input: Analog and Digital	Not Supported	Recommend	N/A	Possible, but may be difficult to implement	Only if JTAG cannot used
Output: Digital	Not Supported	Recommend	N/A	Possible, but may be difficult to implement.	Only if JTAG cannot used
Main Processor	Recommend Highly	N/A	N/A	Recommend, if OCD is not used	Last option
Communication	Not supported	Recommend for processor level faults	Recommend for packet level faults	Possible, but may be difficult to implement	Last option

4.7. References

[VanTreuren 1007] B.G. VanTreuren, A. Ley. "Jtag System Test in a Microtca World." *IEEE International Test Conference.* ITC'07, October 2007. 10-10.

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

[Santos 2003] L. Santos, M.Z. Rela. "Constraints on the Use of Boundary-Scan for Fault Injection." In *Lecture Notes on Computer Science*, 39-55. Berlin/Heidelberg: Springer, 2003.

[Portela-Garcia 2007] M. Portela-Garcia, L.-O. Celia, M. Garcia-Valderas, L. Entrena. "A Rapid Fault Injection Approach for Measuring SEU Sensitivity in Complex Processors." *13th IEEE International On-Line Testing Symposium.* IOLTS'07, July 2007. 101-106.

[Folkesson 2003] P. Folkesson, J. Aidemark, J. Vinter. *Assessment and Application of Scan-Chain Implemented Fault Injection.* Technical Report, Goteborg, Sweden: Chalmers University of Technology, 2003.

[Pignol 2007] Pignol, M. "Methodology and Tools Developed for Validation of Cotsbased Fault-Tolerant Spacecraft Supercomputers." *IOLTS*, 2007.

[Chakraborty 2007] T.J. Chakraborty, C. Chen-Huan, B.G. Van Treuren. "A Practical Approach to Comprehensive System Test and Debug Using Boundary Scan Based Test Architecture." *IEEE International Test Conference.* ITC'07, 2007. 1-10.

5. Development of a High Performance Fault Injection Module for Digital I&C Systems

5.1. Introduction

Fault injection methods for digital I&C systems require flexibility to adapt to different target processors while maintaining low intrusiveness. Digital I&C systems often contain multiple types of processors and FPGAs from different manufacturers in their design. These configurations can exacerbate physical fault injection implementation processes because multiple processor types usually require developing fault injection campaigns specific to the processor type. This is particularly true of ICE-based fault injection methods where a new ICE pod for each different CPU must be purchased and fault injection instrumentation software must be written for each processor. As observed in the Benchmark System I fault injection research, COTS debugging tools and environments for embedded systems (e.g., ICE machines and interactive debuggers) are not designed with fault injection in mind; thus, their performance in this capacity is limited. The most significant issues reported in the literature and confirmed by the research on Benchmark System I and the DFWCS include the following:

- Excessive communication overhead between the fault injection host computer and the fault injection hardware
- Intrusiveness to the system under test, such as the introduction of additional hardware or software to the system to implement fault injection experiments
- Inability to temporarily freeze the timing of a commercial safety grade system to nullify fault injection intrusiveness
- High overhead incurred by the fault injection process that impedes fault injection experimenting in real-time systems due to timing constraints
- Lack of support for diverse fault models and scenarios
- Complexity associated with implementing multiple fault injection techniques on a single target system

In order to address these issues, the goal of this phase of the research was to improve upon contemporary, in-situ fault injection methods by implementing an advanced fault injection system developed on a COTS FPGA board. The motivation for this research is described in Section 5.2.

5.2. Motivation

The ever-increasing use of new digital technologies and complex applications in safety-critical I&C systems requires effective, objective, and repeatable review processes. As the diversity of different digital I&C technologies increases, physical fault injection methods must keep pace with these changes in a way that allows developers and regulators to establish a stable base that remains consistent with changing technology. One approach to address this challenge is to adopt fault injection methods developed from existing organization standards for IC testing and embedded software debugging. These standards include IEEE JTAG/SCIFI standards, the IEEE 5001 Nexus Standards, the Intel XDP standard, and the Motorola BDM standards. These standards-based interfaces are available on nearly all semiconductor reference designs –

CPUs, FPGAs, embedded cores, etc. Semiconductor products that have these types of test and debug ports are widely available on IC products manufactured after 2003. Prior to this date, standardized test port availability on IC products was variable. Developing a custom fault injection capability around these standardized test interfaces specifically for digital I&C systems can obviate many of the implementation challenges facing the use of physical fault injection processes.

The main goal of the research effort was to investigate, design, develop, and implement a modular approach to fault injection that was developed around existing standards and that was highly optimized for fault injection. The designed system addresses the following issues:

- Precise control of location, duration, and time of fault injection experiments
- Support for fault injection-based corruptions at the information processing level
- Maximized performance with minimal intrusiveness in both time and space
- Support for different fault models and scenarios
- Flexibility to allow new fault injection modules to be developed as needed

Additionally, the need to inject concurrent faults into multiple locations of the digital I&C system under test was recognized. This feature is extremely useful when evaluating distributed real-time safety-critical systems because these types of systems usually consist of multiple redundant processing and I/O modules to support enhanced fault tolerance and safety processing. The fault injection method developed by this research is capable of simulating different fault scenarios, including sequential faults across all processor modules, and coincidental faults injected at pre- defined processor delays.

5.3. Design of the High Performance Fault Injector

5.3.1. Background and Motivation

FPGAs are a specific type of IC that provide the ability to be repeatedly reconfigured for specific hardware implementations after manufacturing. The functionality of other types of ICs is finalized before manufacturing such that the IC hardware logic cannot be altered once it has been implemented on the IC. Therefore, FPGAs provide the unique possibility of implementing different designs without the necessity to manufacture each iteration of the design in order to verify its functionality. For this reason, often FPGAs are used to prototype system designs and to validate various configurations before the design is finalized.

Furthermore, contemporary FPGA technology allows designers to select from an assortment of building blocks to build systems on a chip (SoC). These SoC include processor cores in flash memory, real time OS, I/O, network interfaces, etc. For these reasons, FPGA technology is a perfect target choice for realizing the design goals of the UVA high performance fault injection module.

As described above, most current processors include dedicated OCD logic for debugging. This circuitry allows a designer to access internal resources by means of a software tool [Portela-Garcia 2007]. OCD infrastructures are increasingly common, particularly in microprocessors used in embedded processing applications, where prototype validation contributes heavily to the total development effort [Fidalgo 2008].

These OCD capabilities can be used to modify registers and memory contents, providing a useful mechanism to inject faults in processors when a circuit description is not available [Portela-Garcia 2007]. This is exploited by the Xception project work that utilizes the advanced

debugging and performance monitoring features existing in modern processors (presented by OCD logic) to inject more realistic faults by software and to monitor the activation of the faults and their impact on the target system behavior in detail. This process can directly emulate physical transient faults in internal target processor units, main memory, and other peripheral devices that can be accessed by software [Maia 2005].

The ability to access internal parts of the microprocessor or a programmable logic board by a standardized interface has the potential to make physical fault injection widely accessible for embedded systems and digital I&C systems. More importantly, fault injection via OCD methods can be achieved without the need for additional hardware logic added to the target system besides the functionalities that the OCD interface provides by itself.

Most of the recent microprocessors implement at least one of the following OCD interfaces:

- IEEE 1149.1 JTAG;
- BDM;
- IEEE-ISTO 5001-2003 Standard (Nexus);
- Intel Enhanced Trace Macro-cell – ETM.

An overview of these common OCD interfaces is provided in the following sections.

5.3.1.1. IEEE 1149.1 JTAG Interface

The IEEE 1149.1 JTAG interface was developed in 1985 and became a standard interface in 1990 [Santos 2003]. The JTAG interface allows access to processor internal components such as the input and output pins. Due to the common occurrence of theJTAG interface in current devices, use of the interface fault injection experiments gained attention.

A study by [Folkesson 2003] revealed that SCIFI utilizing boundary scans used for validating the dependability of embedded computer systems improved the controllability, reachability and observability of the system over Pin Level hardware fault injection techniques. This conclusion encouraged further research in this area [Portela-Garcia 2007; Chakraborty 2007; VanTreuren 2007; Pignol 2007]. Unfortunately, each one of the respective techniques had shortcomings such as

- Introducing additional hardware into the system [Chakraborty 2007; VanTreuren 2007]
- Adding communication overhead and software intrusiveness [Pignol 2007]
- Needing to pre-program fault injection sequences into the system software [Portela-Garcia 2007]

Despite the capability to execute fault injections, each research effort noted that injecting faults into a real-time system with short control cycle times was difficult without affecting the system under test. This was mostly because of the fault injection setup overhead, which was incurred for every fault injection experiment.

5.3.1.2. BDM Interface

The BDM is a specific mode of operation present on most Motorola and Free-scale microprocessors. By communicating through the BDM development port, the fault injector can force the processor to enter the debug mode of operation and allow access to most of the microprocessor resources. While in the debug mode, the microprocessor fetches instructions from the development port. The fault injector can read and write to general purpose registers, special purpose registers and memory locations as needed [Freescale Semiconductor 2004].

Therefore, the BDM interface is significantly more versatile than the JTAG interface and greatly enhances the options for executing fault injections.

Early fault injection attempts to utilize the BDM interface incorporated the host computer printer port [Carvalho 2005b] or an additional serial peripheral interface [Rebaudengo 1999] for communicating between the BDM port on the target system and the host computer. In this configuration, the host computer had the burden of controlling the target BDM port and performing all of the computations at the same time. While performing fault injections was successful, it was found that performing a single fault injection required 71 times to 96 times longer to perform the same functionality than in the normal mode of operation [Rebaudengo 1999]. This slowdown was caused by the required data communication between the target systems BDM port and the host computer through the slow interface.

Another significant overhead was introduced by the prerequisite to reinitialize the setup of the fault injection environment for every injected fault. Przygoda [2006] eliminated the problem of slow communication between the host computer and the target system by implementing a specific USB device driver. In addition, the BDM logic was created on an FPGA for faster reaction times. Despite this improvement, machine cycles were performed in 4 clock cycles. This performance improvement significantly enhanced the rate at which fault injections could be executed, but the overhead was still significant enough to impact the ability to inject faults into fast-cycle time real-time systems.

5.3.1.3. IEEE-ISTO Std 5001-2003 (Nexus) Interface

The efforts to create a standardized, rich debug feature set that would allow for real-time debugging of microprocessors resulted in the implementation of IEEE-ISTO Std 5001-2003 (Nexus) interface for a global embedded processor debug interface [IEEE-ISTO 2003]. Nexus capitalizes on the existing JTAG standard, but extends its functionality by adding an extensible Auxiliary Port for advanced debugging features [IEEE-ISTO 2003]. Research in performing fault injections using the Nexus OCD capabilities was focused in two areas: (1) implementing intrusive additional OCD circuitry on the system under test [Fidalgo 2008; Fidalgo 2006(a); Fidalgo 2006(b)], and (2) using COTS-based fault injectors [Fidalgo 2006(b); Pardo 2006; Ruiz 2005; Fidalgo 2006(c)]. While the methods in the first category were able to execute fault injections in demanding real-time situations, the necessary hardware modifications required for the target system required hardware description language models of the target system to be available, which is almost never the case in commercial safety-critical systems.

COTS-based fault injectors were implemented using commercial debugging tools manufactured by iSystem [iSystem II 011] or Lauterbach [Lauterbach 2012], with the host computer running fault injection software that controlled the fault injector. However, due to communication delays, the window of opportunity for injecting faults often would be missed; thus, the fault injection expeiments would not be successful [Fidalgo 2006(c)].

This research focused on improving the OCD fault injection method and implementing BDM fault injection methods for testing real-time target systems by designing the fault injector on a FPGA and allowing the user to “plug and play” fault injection campaigns with different fault injection modules. The FPGA did not control only the logic and communication between fault injector and target system BDM port as it would be typically implemented. To achieve the desired ability to perform fault injection experiments on a real-time machine, most of the typical functionalities of the host computer were off-loaded and designated to the FPGA, such as decision-making, acquiring data, and analysis. This solution allowed some degree of parallelism to effectively accelerate the fault injection set-up processes, effectively eliminating the communication bottleneck between the fault injection host and the fault injector.

Since Benchmark System II incorporated a Freescale MPC8a60 microprocessor BDM test interface and JTAG interface, the concept of a FPGA-based fault injector concept was developed to use these interfaces. The motivation for creating a highly flexible fault injector was to create a modular system that could be extended to contain multiple fault injection modules that could be driven from one control module.

5.4. Design of FPGA-based High Performance Adaptable Fault Injection

This section presents an abridged presentation on the design and development of the FPGA-based HiPeFI. A more detailed presentation can be found in [Miklo 2011]. The key concept of the UVA fault injection platform is that different types of "plug and play" fault injection modules can be used to fully support digital I&C system testing. These different modules could be BDM, NeXUS, JTAG, ETM, SWIFI, or user defined. Each of the fault injection modules is implemented in full hardware so they can operate independently from one another, thus allowing full parallel operation. This type of architectural concept allows for better coordination when injecting multiple faults or sequence-dependent faults into a system.

As discovered from previous fault injection applications and noted in the Volume 2 findings, using COTS debugging tools for executing fault injections requires access to low level application programmer interfaces (APIs) to bypass high level interfaces and graphic user interfaces (GUIs) to access memory and register locations efficiently while incurring minimal additional system processor overhead. Despite these optimizations, using COTS debugging tools for OCD-based fault injections may introduce unacceptable communication delays and may contribute to excessive fault injection set up time, which can reduce the ability to perform fault injections on fast, real-time target systems. To achieve the highest levels of performance and flexibility, the requirements for the UVA FPGA-based fault injection tool design were:

- Exploit standard OCD hardware logic to achieve real-time fault injections
- Minimize intrusiveness to the target system to the maximum extent possible
- Reduce fault injection set-up time and communication overhead by off-loading these tasks to the onboard FPGA processor
- Eliminate unnecessary elements found in COTS OCD debugging tools and capitalize on task parallelism
- Exploit the natural parallelism found in FPGA technology
- Support injecting faults into multiple processors and subsystems
- Emulate sequential and coincidental failure sequences
- Provide an extensible and portable solution to different target systems with minimal re-design.

From the experience gained from developing a fault injector for Benchmark System I (X-bus fault injector), the research determined that programming and using FPGAs for executing fault injection campaigns was a very powerful strategy. Therefore, as a base for the UVA FPGA-based fault injection platform design, a low-cost FPGA board and the Altera NIOS II development kit, Stratix II Edition [Altera 1995-2012] were chosen to develop the system.

Besides using custom hardware fault injection modules, the UVA design incorporates a System-on-Programmable-Chip (SoPC), which includes several intellectual property (IP) cores such as Altera NIOS soft-core CPU, RAM, FLASH, programmable load lists (PLLs), and memory-mapped interconnects.

To decrease the development time and increase the feature set, most of the performance non-critical features were moved to software that executed on the soft-core CPU of the FPGA, which was shown to be a crucial design decision. It would not have been possible to implement all of the functionality purely in hardware. The design tools provided by Altera for their FPGA line were used to simplify development of the UVA fault injection platform. The basic architecture of the UVA design is shown Figure 5-1.

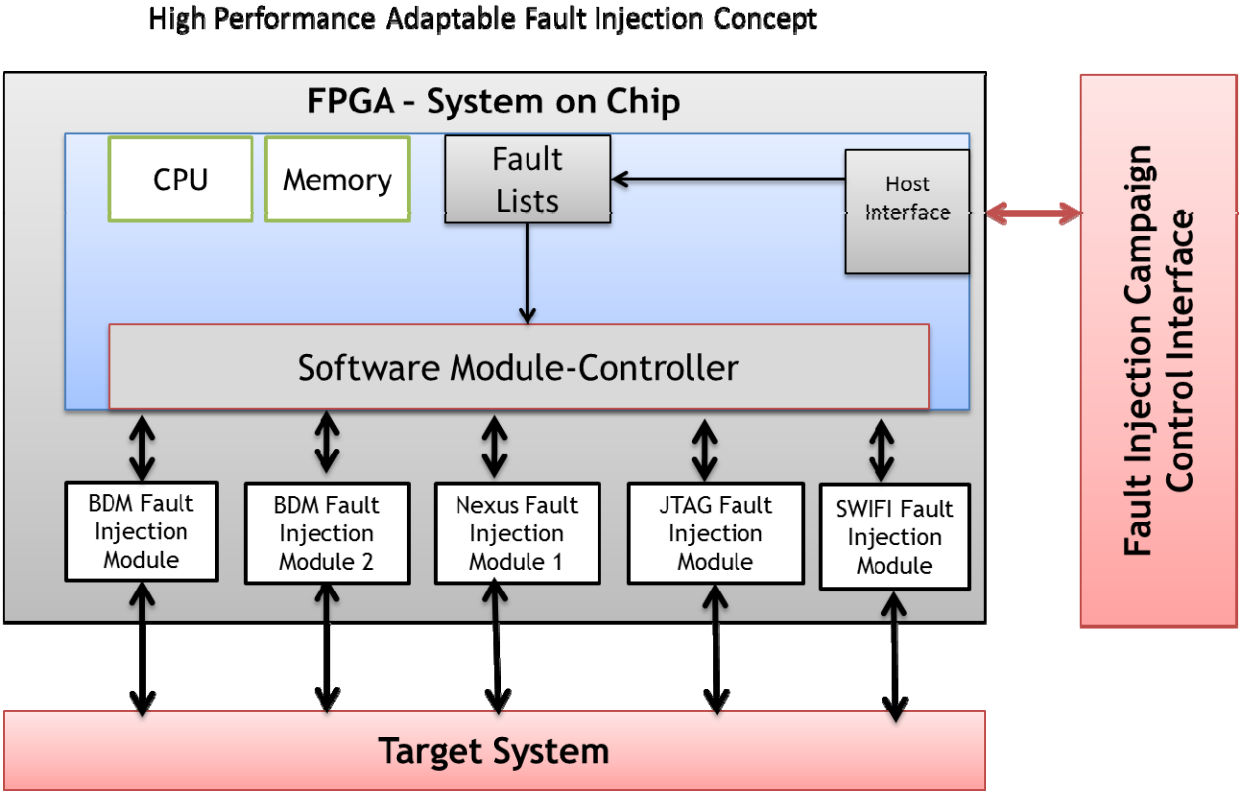

Figure 5-1 Conceptual design of the FPGA-based HiPeFI

It was necessary that all the modules be implemented on the selected FPGA development board because the fault injector design had to adhere to strict timing requirements for executing unobtrusive fault injection experiments on real-time safety-critical target systems without tripping watchdog timers. Therefore, the fault injection modules had to be implemented in hardware and described in Very High Speed Integrated Circuit Hardware Description Language (VHDL). Further, the control module functionality was not time-critical so it was moved to a software module. This design solution provided the capability to add additional features to the UVA fault injection platform by implementing them in software instead of tampering with the time-sensitivity of the hardware modules. The additional design requirements established for the fault injector are as follows:

- The platform should be modular to allow adding multiple modes to the fault injection campaign

- The system had to be self-contained so a fault injection could be started either manually or automatically by a connected device without reconfiguring the fault injection platform
- The system should implement visual debugging aids such as information codes displayed on LEDs and more extensive information available on an LCD display
- After completing a fault injection experiment, each of the modules had to be returned to their initial state to allow another fault injection without a hard reset of the fault injection platform
- In cases of system failure, the system had to return to its initial state.

5.4.1. Fault Injection Hardware Modules

Referring to Figure 5-2, low level communication between the fault injector and the target device was controlled by the fault injector hardware module in order to eliminate system overhead. Once a data stream for executing a fault injection was received from the software module, the fault injection was executed to its completion solely by the hardware module. Both the JTAG and BDM modules were implemented in VHDL and optimized for high performance to minimize overhead.

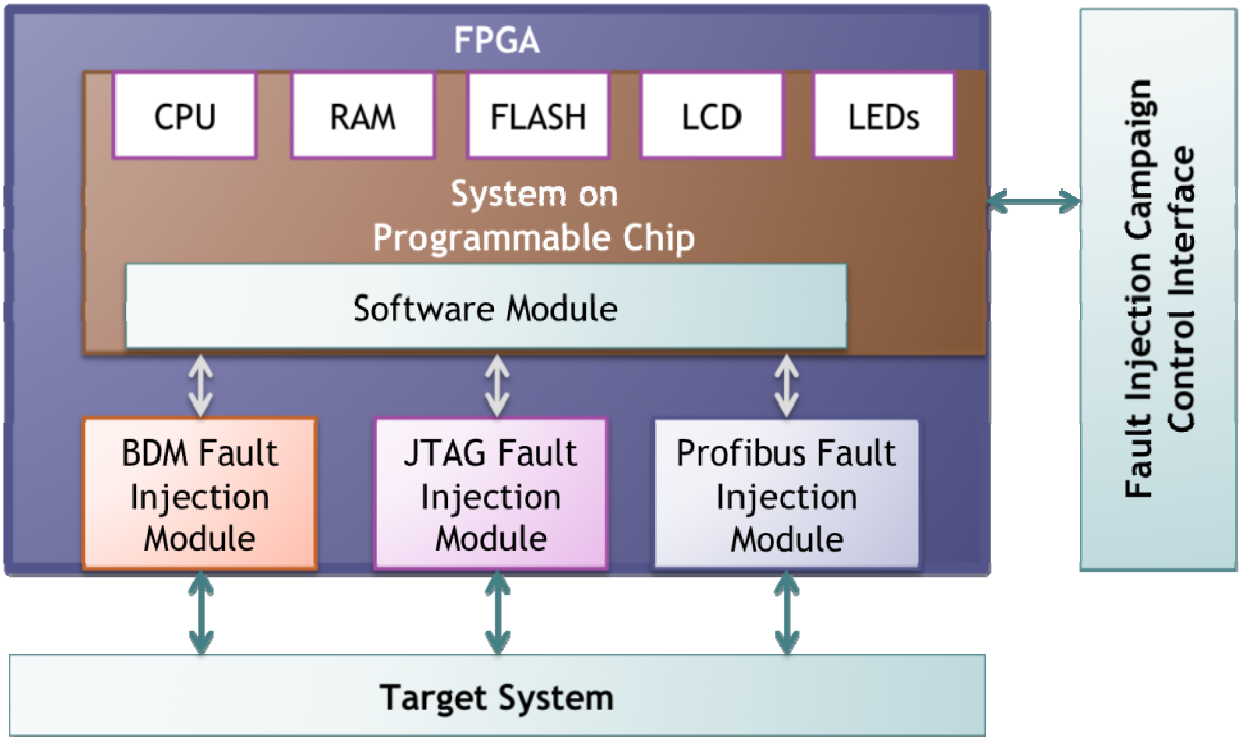

Figure 5-2 FPGA-based HiPeFI architecture final design for Benchmark System II

5.4.2. BDM Fault Injection Module

The BDM is a special mode of operation of the target processor in which the core begins fetching and executing instructions located in the Development Port Instruction Register (DPIR) implemented on the Development Port Support Logic part of the microprocessor. This logic

block is shown in Figure 5-3 below with relevant descriptions of the registers and signals used in BDM operation. Similarly, if an instruction requires additional data input, it is obtained from the Development Port Data Register (DPDR). This register also serves for holding the response sequence of the Development Port to the previously executed instruction; this sequence is later shifted out to the output pin of the Development Port.

There are very few limitations on the type of instructions executed in the processor core while it is in the debug mode. It is possible to gain access to any register, both general and special purpose, as well as to almost any memory location. The capability for executing a fault injection by shifting in a sequence of instructions from the processor instruction set architecture (ISA) allows almost limitless possibilities for fault injection targets.

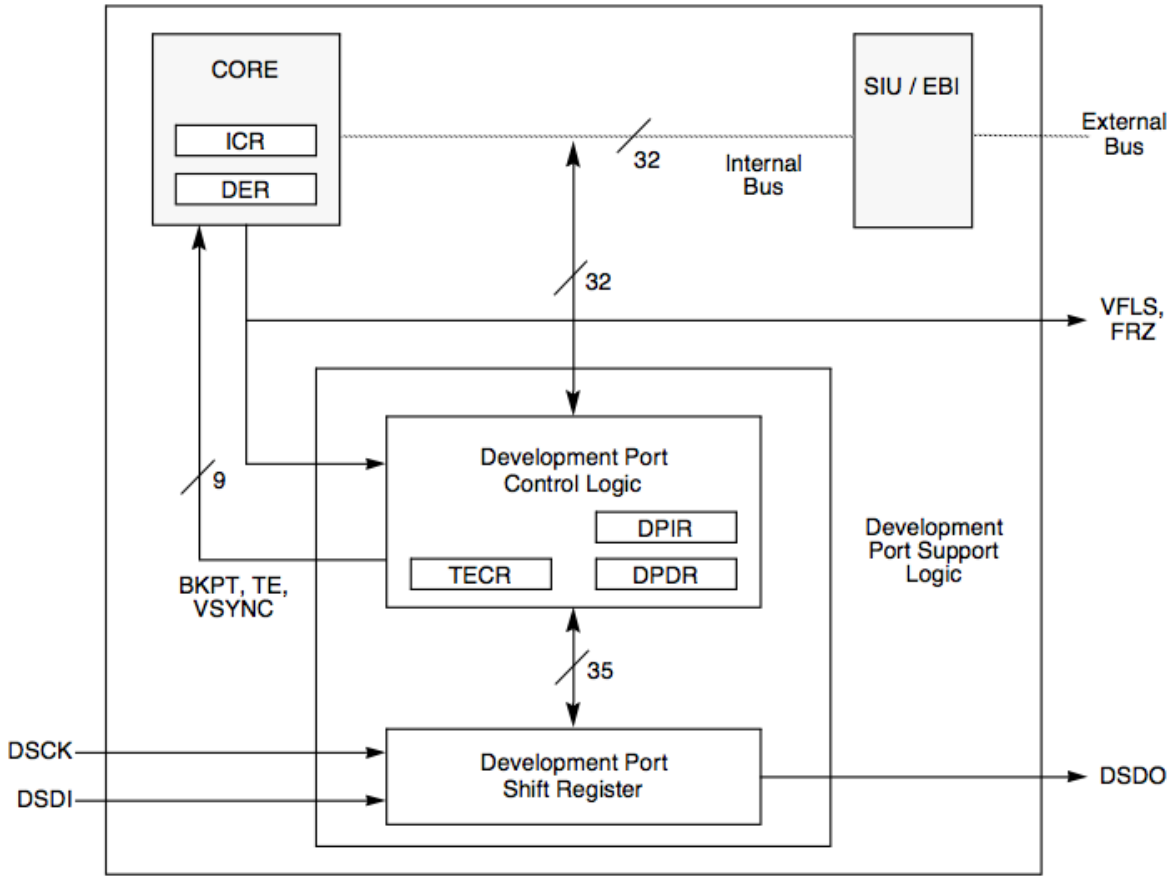

Figure 5-3 BDM functional diagram of the MPC860 Debug Mode support

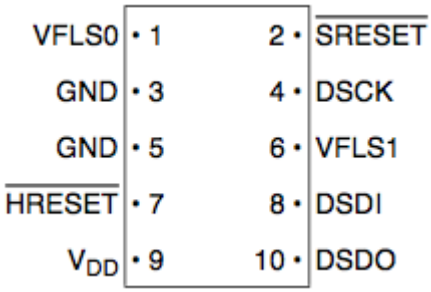

Figure 5-4 Physical representation of the BDM port

Referring to Figure 5-3 and Figure 5-4 above, The DPSR (see Figure 5-3) is a 35-bit wide development register that acts either as a DPIR or as a DPDR depending on whether an instruction (DPIR) or data (DPDR) is being shifted into the development port control logic. The desired bit sequence is shifted into the DPSR through the DSDI pin signal (see Figure 5-4), while at the same time the response to the previous instruction or requested data, is shifted out to the BDM fault injection module on the DSDO pin. After the instruction or data has been shifted into DPSR it is transferred in parallel to the processor core. The development port logic can be clocked either asynchronously by an external clock signal present on the DSCK input pin, or synchronously by the processor core itself.

The selection of synchronous or asynchronous clock mode is made at reset. Use of the external clock option is advantageous for several reasons:

- It is often impossible to gain access to the processors internal clock signal.
- The propagation times of DSDI and DSCK signals can be easily determined and adjusted for achieving the correct timing.
- Actual performance of the fault injection module can be determined by counting the number of clock cycles that are necessary to execute the desired fault injection.

The design of the BDM fault injection module must precisely follow the specifications in order to achieve the desired functionality. All available sources from Free-scale were used to implement the BDM specification [Freescale Semiconductor 2004]. The target BDM interface provided very complex functionality and the fault injection module had to be designed carefully. Similarl to the JTAG fault injection module, the BDM implementation was based on utilizing the strategy of a state machine implementation.

During the first stage, all of the data necessary for performing the BDM-based fault injection was acquired from the input signals. The second stage included initiating the reset of the target processor and proper set up of the BDM interface functionality. The BDM fault injection module had to wait until the target processor entered the debug mode type of operation. The last stage involved shifting the fault injection instruction bits to the development port. At the same time, the output from the development port was observed at the DSDO pin. The fault injection module required this data because the data contained important information regarding the correct operation of the development port. The last instruction then released the target processor from the debug mode, and then resumed executing its original program flow.

5.4.3. JTAG Fault Injection Module

Using the JTAG fault injector developed previously and discussed in Volume II Section 4, design was ported that to the fault injector FPGA with some modifications necessary to enhance operations.

The JTAG interface is shown Figure 5-5. The interface consists of two input pins (TDI and TMS), one output pin (TDO), a test clock pin (TCK) and an optional reset pin (TRST). Additionally, the interface contains two test registers and an instruction register. The first test register serves as a single-bit bypass register for connecting TDI inputs to TDO outputs for testing the correctness of connections within the internal components. The second register is the BSR, which contains bit values of all digital input and output signals inside the logic board. This register can be placed between the TDI and the TDO tod function as a shift register. The Instruction register (IR) is a 4-bit register (without parity) that determines the function that is being performed by the Test Access Port (TAP) controller.

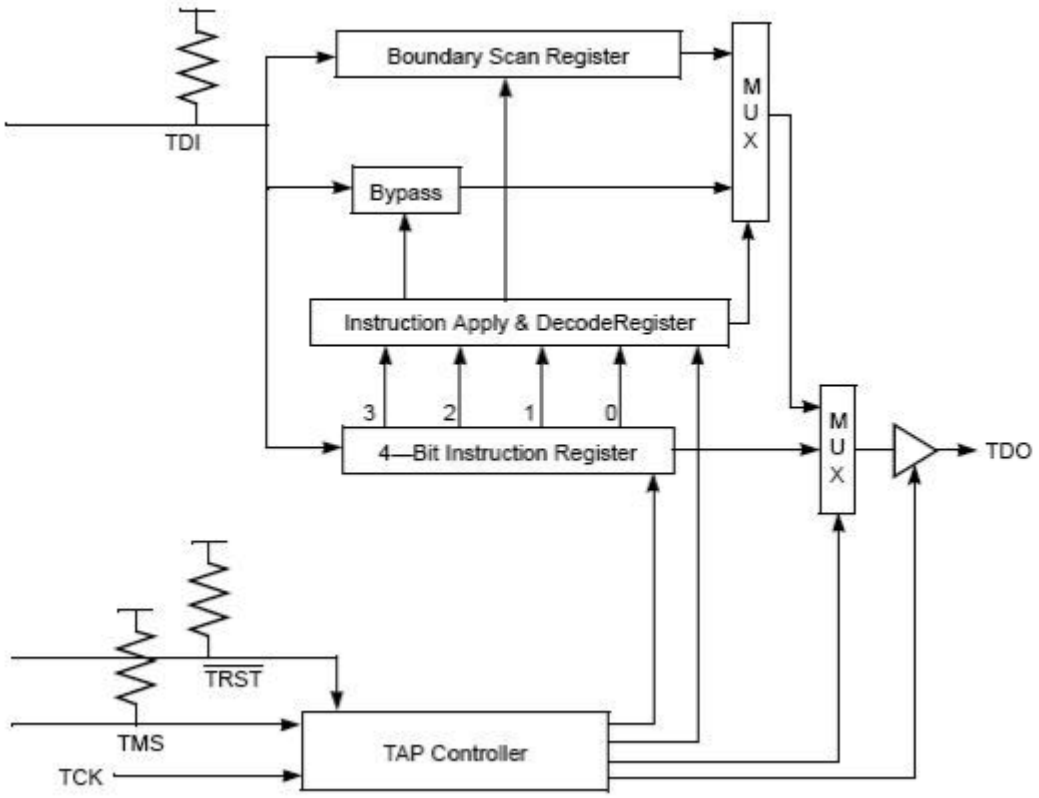

Figure 5-5 JTAG TAP controller test logic diagram

The state machine inputs are obtained from the TMS input pin (see Figure 5-5). The inputs determine the steps of progress through the state machine graph. Once the graph has reached the state SHIFT IR or SHIFT DR, the input from the TDI is shifted into the IR or BSR, depending on the current state. By this implementation, a function code can be shifted into the IR for selecting the desired function, or a bit can be shifted in the BSR for performing the desired corruption. The TAP shown in Figure 5-6 is designed as a state machine clocked by the TCK shown in Figure 5-5.

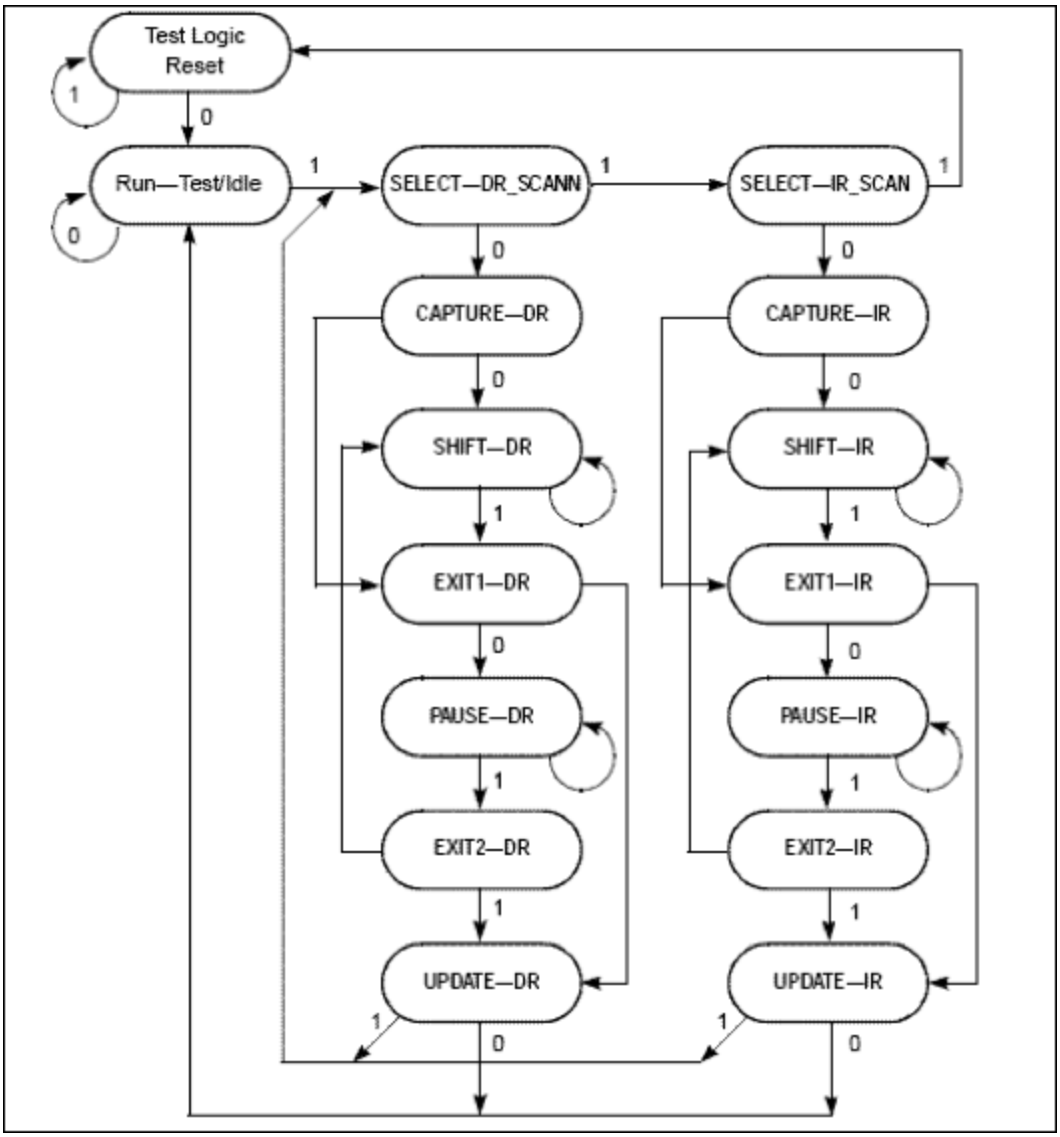

Figure 5-6 JTAG TAP controller state machine

The functionality of the fault injection module is based on a state machine that progresses through several states (i.e., Initialize, Reset, Preload, Extest, Write, and Done). The Initialize state sets up the module and clears the counters.

When the JTAG fault injection module receives a start signal from the SoPC module, the module progresses to the Reset state, which resets the target system JTAG interface so that a fault injection can be initiated at the appropriate time. This state is necessary for achieving proper initial values of all signals and variables, especially for performing multiple subsequent fault injections when the internal signals could still contain residual data.

During the Preload states and the Extest state, the interface follows a sequence through the state machine of the host TAP controller by selecting the appropriate TMS bit signals. At first the module must reach the SHIFT IR state, after which the proper sequence representing the PRELOAD/SAMPLE instruction is sent to TDI and shifted into the IR of the TAP controller.

After the sequence returns to the Run Test/Idle state, it is directed back to the SHIFT IR, where the sequence for the EXTEST function is shifted into the IR. The sequence then enters the idle state again.

At this point, the target JTAG interface is set up to receive the transmission from the developed JTAG fault injection module. The TMS signal guides the TAP state machine to reach the SHIFT DR state. During this state, the it value presented on the TDO output signal is shifted into the BSR, while at the same time an output bit from the BSR is placed in the TDI input signal. The interface remains in this state for the number of clock cycles represented by the number of bits contained in the corruption sequence.

The proper sequencing and synchronization of each step is achieved by using internal counters implemented within the JTAG fault injection module. One counter is used for propagating the sequence through the states by selecting the proper value for the TMS signal, while a different counter is used for counting the number of bits remaining to be inserted into the IR or BSR. Each counter also serves as an index for selecting the current TDO and TMS output bit values transmitted to the target JTAG interface.

5.4.4. System-on-Programmable-Chip (SoPC) module

The inclusion of the soft-core CPU allowed implementing advanced functionality in software through a C algorithm, such as communication with other systems or data parsing. With the option to program the CPU core in C or C++, it was possible to handle and process stored and received data internally in the software core module. This eliminated the need for pre-processing data on a host computer, which can be a bottleneck in contemporary fault injection systems. Furthermore, this approach simplified the process of transmitting data to and from the hardware fault injection modules, which significantly reduced the time required for developing the fault injection functionality.

Besides a C-programmable CPU, the design included several different types of memory. On-chip memory was used to improve performance, while RAM was used during data processing. FLASH memory was incorporated to hold a non-volatile version of the software and hardware image of the FPGA-based fault injector. The system included a large number of parallel I/O modules that were used for connecting the JTAG and BDM modules to the software module. Several Phase Locked Loops (PLLs) were used to drive signals to the hardware fault injection modules at the required frequency. All of the modules were easily accessible to the software core through a memory-mapped interface.

5.4.5. Software algorithm and functions

Having created the hardware fault injection modules and software controller for implementing the fault injection system, the final step was to create an algorithm that would properly operate the fault injector modules as well as process the data transfers correctly. The devised algorithm for the software functionality is illustrated in Figure 5-7.

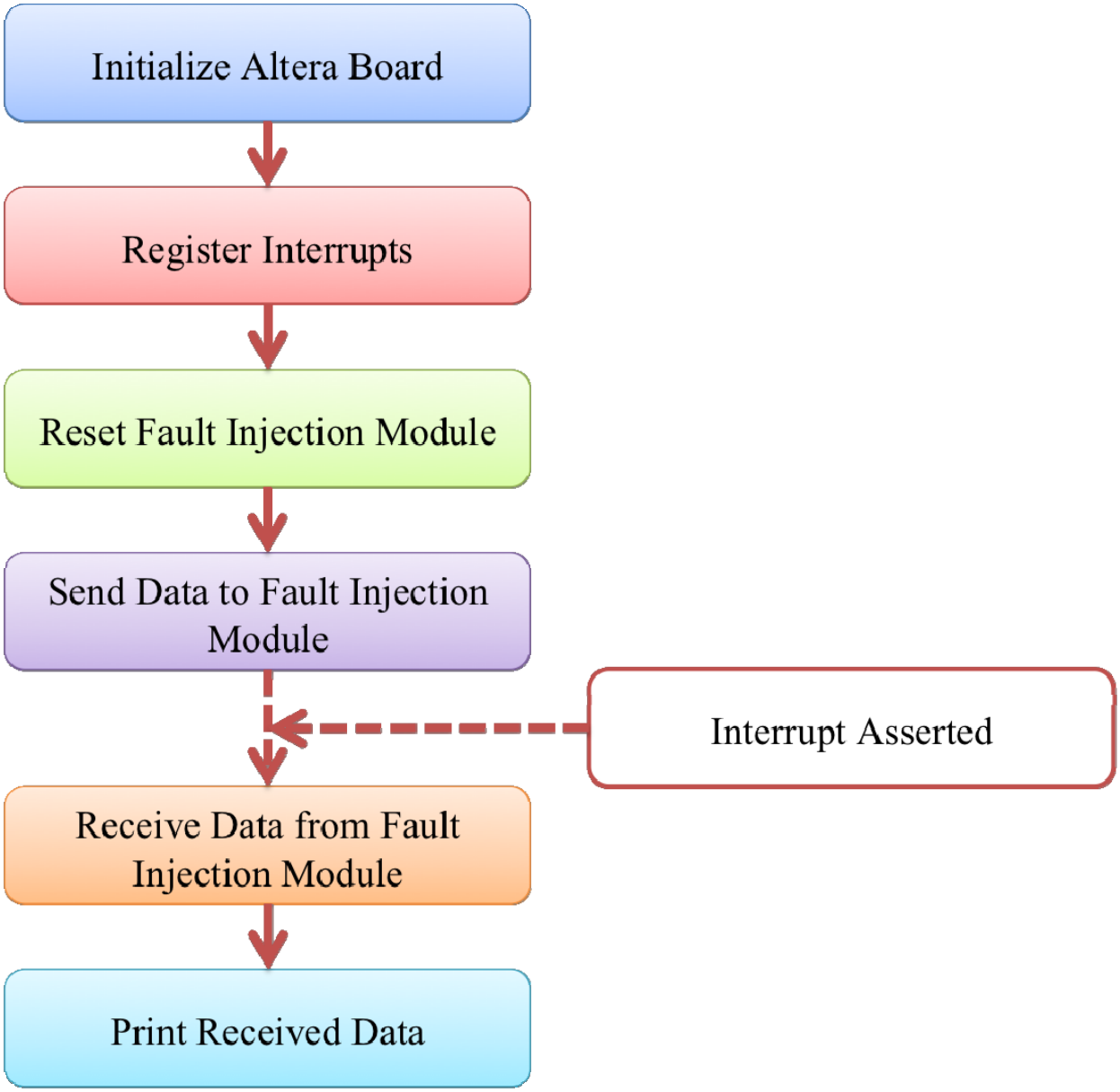

Figure 5-7 Implemented software algorithm for executing BDM-based fault injection

The algorithm is similar to algorithms found in programming software. The only difference between general software programming and programming the FPGA board was in working with signals instead of variables. The signals were addressed with Altera-specific commands that provided the means to read and write signals. The algorithms for each stage are as follows:

- Initialize Altera Board
    - All of the modules are initialized, including LCD, LEDs, etc.
    - Initial message is printed out to the LCD and console
- Register Interrupts
    - BDM interrupt handling routines have to be registered

- Interrupts for the signal must be enabled for proper functionality

- Reset Fault Injection Module
  - Assert high on signals BDM_FI_RESET; which resets the module
- Send Data to Fault Injection Module
  - Handles sending data to the Input FIFO of a fault injection module

    The constant setup signals are asserted
    - BDM_FI_INST_COUNT
    - BDM_FI_DEBUG_MODE_TYPE
    - BDM_FI_DEBUG_MODE_CLOCK_TYPE
  - For loop iterates on the following sequence (number of iterations determined by BDM_INST_COUNT)
    - Assert desired data value onto the signal BDM_FI_DATA_FROM_NIOS_TO_FIFO_IN
    - Assert high value on BDM_FI_DATA_WRITE_REQ_FROM_NIOS_TO FIFO_IN; this tells the input FIFO to read and store data signal on the next clock cycle
    - Assert low value on BDM_FI_DATA_WRITE_REQ_FROM_NIOS_TO_FIFO_IN; this does not negate the previous high assertion because it is performed several clock cycles later and the ONE_CYCLE module on the input side already acknowledged the presence of high signal.
  - Start fault injection by asserting high on BDM_FI_START_FROM_NIOS
  - Assert low to the previously mentioned signals; same logic as with transferring data
- Receive Data from Fault Injection Module
  - Function started when Interrupt was asserted (Interrupt routine calls this function)
  - Reads data from Output FIFO and stores them into internal structure for future manipulation
  - Assert high on BDM_FI_READ_REQ_FROM_NIOS_TO_FIFO; this causes Output FIFO to assert the current data value onto the output signal; multiple reads are eliminated by integrating ONE_CYCLE module
  - Assert low on the previous mentioned signal just to prepare the next cycle
  - Read data from signal BDM_FI_DATA_FROM_NIOS_TO_FIFO and store them in an internal software structure (array)

    - Iterate over the previous three steps for the same number of times as specified in the Send Data to Fault Injection Module Function

- Print Receive Data
    - Iterate over the acquired data and print each value to the LCD display and console
    - Pass data for further analysis or for the transmission to the host computer

5.5. FPGA Design Tools for Realizing the Design

5.5.1. Altera SoPC Builder

Altera SoPC Builder is a software tool for selecting modules and building blocks for the implementation of a SoPC. The tool provides a simple interface with options to select components from a wide range of categories such as Bridges, Phase-Locks Loops (PPLs), Interface Protocols, Memory, Peripherals and many others. The only CPU that this builder provides is the Altera NIOS II CPU.

After the necessary components were selected, alterations were performed to generate the system as desired. The final generation of the system created various VHDL (or Verilog, based on specification) files for inclusion in the final hardware design so it could be properly programmed on the FPGA board.

5.5.2. Altera NIOS II IDE

After the SoPC design was implemented and generated by the Altera SoPC builder, it was used by the software development interface Altera NIOS II IDE as a general processor. The interface is based on the Eclipse programming environment, which allows the developer to be quickly familiar with many of its features and inherently decreases the necessary learning curve. Programming of the SoPC can be performed in C or C++, and includes special commands for handling external signals, such as reading and writing on a bus, handling interrupts, and others necessary to create a fully functional system that comprises not only the SoPC core but also additional hardware modules and interfaces. The interface contains debugging options as well as the ability to perform simulations of the created software and testing by temporarily uploading the designed software to the FPGA board. The ability to debug the design by using the console window while the design is tested on the actual FPGA board and not just in simulation was informative and helpful.

5.6. Design Challenges

Implementing a JTAG fault injection capability was a straightforward process with the requirement to properly synchronize TCK, TDI and TDO signals of the target system with the JTAG fault injection module. However, numerous challenges were encountered during the design and implementation of BDM fault injection module due to the complexity of the protocol. These issues are discussed n more detail in the following sections.

5.6.1. Re-entering the Debug Mode

The main functionality of the BDM hardware fault injection module is executing two types of operations: initializing the target processor for fault injection; and the fault injection process. To initialize the target processor for a fault injection experiment,

(1) The processor was restarted
(2) The BDM debug mode was initialized via a specific instruction sequence
(3) The processor was returned to its normal state of operation through a special instruction.

To inject a fault, the processor is forced to re-enter the debug mode through an interrupt signal, the fault injection is executed via standard instructions for manipulating registers or memory. Finally. the processor is instructed to return to normal state of operation, and the results of the injected fault are observed.

While it was easy to enter the debug mode right after restarting the target processor, injecting a fault into a ROM-less system was not very useful. Therefore, to effectively inject a fault, it was necessary to achieve the capability of reentering the debug mode at any point during a normal state of operation of the target processor. This was accomplished by carefully constructing the BDM initialization sequence. As mentioned earlier, the BDM port can accept virtually any instruction from the PowerPC ISA. The BDM initialization sequence is a set of PowerPC instructions that modify special purpose registers of the target processor in order to allow for successful re-entry into the debug mode upon command to inject a fault. This sequence of instructions is loaded into the BDM fault injector module queue and is transferred to the target processor after a forced reset and negotiations of the BDM port speed and parameters. The steps performed during this sequence are the following:

(1) Read internal memory map base address for accessing memory mapped special purpose registers

(2) Configure load/store watchpoint operations

(3) Disable software watchdog

(4) Configure instruction breakpoint operations

(5) Enable debug mode re-entry

(6) Clear comparator registers

(7) Configure load/store address breakpoint operations

(8) Clear breakpoint counter and machine status register

(9) Clear interrupt cause register

(10) Return to normal mode of operation and enable software watchdog

This BDM initialization sequence was constructed based on the information obtained from MPC860 User Manual [Freescale Semiconductor 2004] and from communications with the Freescale support staff. Devising the correct sequence of setup instructions was a tedious process, as an incorrect sequence would cause the processor to immediately restart itself due to the internal checks of the safety-critical real-time target system without providing any feedback data.

5.6.2. Avoiding Timeouts

Most modern processors include a variety of both hardware independent and software-based watchdog timers, which restart the processor upon timer expiration. This trigger protects the processor from being unresponsive or stuck in a faulty state. When the processor enters the BDM mode of operation, it freezes its normal execution, including regular refresh of the watchdog timer; the normal execution flow resumes upon exit from BDM mode.

To avoid this problem in the UVA fault injector system, the fault injector disabled the software watchdog timer during the BDM initialization sequence, which is a reasonable constraint, then re-enabled it during normal operation for the actual fault injection.

Additionally, multi-module real-time systems usually have inter-module synchronization watchdog timers that restart a module if it does not respond within a specific timeframe. These error detection mechanisms can be a significant obstacle for fault injection techniques that are time intrusive in the range of milliseconds. However, because of the HiPeFI design, the FPGA-based fault injector never tripped a watchdog timer or caused the processors to lose synchronization even when long instruction sequences were communicated between the fault injector and the target processor.

In retrospect, removing the processor software watchdog timer might not have been necessary due to the extremely fast execution achieved by the UVA design. However, this feature can be enabled or disabled in the software module of the fault injector without the need to recompile the whole system. Furthermore, information concerning whether to enable or disable the target processor watchdog timers for a certain fault injection can be a part of the initialization data that is communicated from the host computer. This simplicity of altering the behavior of the fault injector design and amending its functionality demonstrated the correct decision to move non-critical functionality to the software module.

5.6.3. Temporary Data

While in debug mode, the target processor executes regular ISA instructions. A register or a memory content must be changed to execute a fault injection, During this process, additional registers on the target processor are used to store temporary data. However, by altering the content of these registers, the program flow can be influenced significantly.

A fault might occur because of the new contents of the registers used for storing temporary data and not due to the corrupted contents of the actual fault injection target. This is an undesirable effect. Therefore, it is necessary to store the contents of registers used for managing temporary data at the beginning of the fault injection and restore their contents before returning the process to its normal state of operation. This is accomplished by incorporating a hardware register file directly in each BDM fault injection module.

To eliminate register store/recover penalties, the hardware design of the module provides the register file with direct access to the data received from or sent to the BDM port. This is achieved by sending a control sequence appended to the PowerPC instruction from the software module to the BDM fault injector module. The control sequence instructs the fault

injector module whether to store the data coming from the target processor, which in this case constitutes contents of a register used for storing temporary data, directly into the fault injector module internal register file. This method essentially mirrors the contents of the target processor register file with virtually no penalty on performance.

When the data needs to be restored at the end of a fault injection, the control sequence instructs the fault injector module to transmit data from its internal register file back to the target processor, instead of using the original data from the instruction queue. Contemporary fault injectors and debuggers use the host computer for storing temporary data, thus incurring costly communication overhead twice; once for storing the data and a second time when the data is retrieved and restored to the target processor. The UVA method of incorporating an internal register file directly inside the fault injector module allows the injector to store and transfer dynamic data contents from and to the target processor without any overhead, which significantly improves the performance.

5.6.4. Communication with Host Computer

A fault injector design must be capable of communicating with a host computer to receive information about the desired fault injection campaign. This communication interface is the performance bottleneck for most commercial systems as it is often used for controlling certain vital functions of the fault injector. In the UVA fault injector design, the communication interface is used solely for receiving data describing the fault injection to be performed, and for retrieving log data upon completion of the fault injection. The FPGA-based fault injector performs all information processing and parsing inside the software module of the FPGA. The software module then delegates critical performance tasks to the custom hardware fault injection modules. No performance-critical action is dependent on the speed of the communication interface, nor its latency. This design achieve high performance and executes fault injection campaigns in very demanding real-time situations, which was previously not possible. Eliminating the communication overhead was one of the goals and novel features of the UVA fault injector design.

The communication interface is a basic Universal Asynchronous Receiver/Transmitter (UART) serial port. The main advantage of this interface is that virtually any computer or device can connect to the fault injector through this interface and instruct it to execute a fault injection. The description of a fault injection is provided through a simple text stream. The data stream is then parsed in the fault injector software module and translated into an instruction sequence for the BDM hardware module.

The versatility of the UART communication interface presents a great advantage for executing fault injection campaigns with respect to field deployments and allows it to be incorporated into existing fault injection environments. This is achieved by the logic inside the software module, as it can accept a description of the fault injection to be executed as parameters in the input stream such as target, location, delay, breakpoint settings, etc. The parameters are then translated into PowerPC instructions and control sequences that execute the desired fault. With this setup, fault injection environments over and beyond the UVA UNIFI fault injection environment can use the FPGA-based fault injector.

5.7. Multi Fault Injection Capability

A novel feature of the UVA fault injection system is support for injecting faults concurrently into different CPUs or target ICs without accruing performance or reconfiguration penalties. This allows evaluation of systems consisting of multiple modules by simulating sequential faults across modules, or coincidental faults injected at pre-defined delays.

Each hardware-based fault injection module (JTAG, BDM) was designed as a separate entity that can be easily replicated and added to the FPGA-based design without additional reconfiguration of each respective module.

In order to support the multi-module system, an Arbiter module was implemented to properly schedule the execution of each fault injection experiment and initiate a start command for respective fault injection modules. The Arbiter receives information from the software module that specifies which fault injection module should be enabled for the specific fault injection experiment and when each module should schedule execution of injecting a fault immediately, after a predefined delay, or after a different fault injector has injected a fault into its target.

The current design of the UVA FPGA-based fault injector (Figure 5-2) consists of two BDM fault injection modules and one JTAG module. The only limitation that prevented inserting more modules was due to the number of physical pins on the FPGA board that could be used for connecting to the target system. The solution has been tested and was able to execute multi-mode fault injections in parallel without incurring any performance overhead.

5.8. Performance Measurements

The performance measurements presented in this section were obtained from Benchmark System II. The target processor for the fault injections was a Freescale MPC860 running operating at 50MHz, which was the main processing unit in each processor module.

There were two common types of fault injections: 1) register corruption and 2) memory corruption. To properly execute each type, it was necessary to issue a set of PowerPC instructions to the processor in order to corrupt the desired bits. In the implementation, injecting a fault into register required 16 instructions, and injecting a memory location required 17 instructions.

The BDM hardware fault injection module frequency was 8.33MHz for most of the fault injection campaign. The measured execution times for each action were:

- BDM initialization sequence: 401µs
- Register corruption: 83µs
- Memory corruption: 109µs

The UVA design is not limited to a module frequency of 8.33MHz, but rather by the specifications of the BDM port on the target processor. For the MPC860 processor, the maximum allowed frequency of the BDM clock input (DSCK pin) was one third of the nominal frequency of the processor, so in the Benchmark System II case, the module frequency was 16.666MHz. When the BDM hardware module frequency was increased 16.666MHz, the following results were obtained:

- Register corruption: 43µs
- Memory corruption: 56µs

As expected, the time required to execute each fault injection decreased by approximately 50 percent. Depending on the target hardware, the frequency of the injection module could be increased further, thereby resulting in even better performance results. This shows that the UVA fault injector design is readily scalable for faster processor frequencies. As noted earlier, performance remains consistent when increasing the number of fault injection modules as each

module is a separate hardware entity, rendering performance independent on other parts of the system.

An important objective was to significantly decrease the time necessary to inject a fault into a target system compared to commercially available fault injectors. It was noted in [Elks 2010(a)] that a typical commercial fault injector can execute a single register read in 500µs to 1ms. A sequence of steps necessary to execute a full fault injection, a read-modify-write cycle, required approximately 4ms to complete when a commercial fault injector was used. Based on the results of the research and the fault injection campaign, the FPGA fault injector design is orders of magnitude faster than a commercial OCD-based fault injector.

To place the performance results in perspective in terms of unwanted time intrusiveness, the number of processing cycles lost on the target microprocessor due to a fault injection can be estimated. The target microprocessor frequency of System B (Freescale MPC860EN) was 50 MHz, thus the number of lost cycles due to the execution of a fault injection was 348 processor cycles to 530 processor cycles when corrupting a register and a memory location, respectively. When compared to the number of cycles necessary for triggering a software watchdog implemented as a 16 bit counter (which is 65535 processor cycles prior to its activation) it is obvious that the number of processor cycles lost due to the execution of a fault injection is insignificant. The time to un-sync the processors on Benchmark System II due to delay in synchronization rendezvous was approximately 5ms. At 43µs per fault injection, the UVA fault injector performance was approximately two orders of magnitude below that trip level.

The JTAG fault injection capabilities are limited by the simplicity of the interface. The capability for inserting varying numbers of bits into the BSC, which alters the values on input and output pins of the device is faster than usual methods because the entire BSC does not have to be read and reinserted. However, the tradeoff is controllability of the injected fault. The initialization of the JTAG interface and return to normal mode of operation requires approximately 30 clock cycles. Inserting one bit into the BSC requires only one additional clock cycle. Executing at 20MHz, injecting one bit takes 1.5µs; inserting each additional bit requires 0.05µs.

5.9. Bibliography

[Fidalgo 2006(a)] A. Fidalgo, G. Alves, J. Ferreira. "A Modified Debugging Infrastructure to Assist Real Time Fault Injection Campaigns." Design and Diagnostics of Electronic Circuits and Systems, 2006(a): IEEE.

[Fidalgo 2006(b)] A. Fidalgo, G. Alvez, J. Ferreira. "Real Time Fault Injection Using a Modified Debugging Infrastructure." 12th IEEE International On-Line Testing Symposium. IOLTS 2006: IEEE, 2006(b). 6.

[Fidalgo 2008] A.V. Fidalgo, G.R. Alves, M.G. Gericota, J.M. Martins-Rerreira. "A Comparative Analysis of Fault Injection Methods via Enhanced On-Chip Debug Infrastructures." 21st Annual Symposium on Integrated Circuits and System Design. 2008. 22-27.

[Altera 1005-2012] Altera. Nios II Development Kit, Stratix II Edition. 1995-2012. http://www.altera.com/products/devkits/altera/kit-niosii-2S60.html (accessed 2010).

[VanTreuren 2007] B.G. VanTreuren, A. Ley. "Jtag System Test in a Microtca World." IEEE International Test Conference. ITC'07, October 2007. 10-10.

[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).

[Fidalgo 2006(c)] Fidalgo, A, M Gericota, G Alves, and J Ferreira. "Using NEXUS compliant debuggers for real-time fault injection on microprocessores." SBCCI'06. Minas Gerais, Brazil: ACM, 2006(c). 214-219.

[Freescale Semiconductor 2011] Freescale Semiconductor, Inc. "MPC860 PowerQUICC(TM) Family User's Manual." http://www.freescale.com/files/netcomm/doc/ref_manual/MPC860UM.pdf, July 2004.

[iSsytem 2011] iSystem. iSystem: Get the Most Out of Development and Testing. 2011. www.isystem.com (accessed 2010).

[Carvalho 2005(b)] J. Carvalho, A. Carvalho. "Experimental Analysis of Outage Times for PROFIBUS Networks." 31st Annual Conference on IEEE Industrial Electronics Society. 2005(b).

[Przygoda 2006] K. Przygoda, D. Makowski, G. Jablonski, A. Napieralski. "USB-Based Background Mode Debugger for Freescale Processors." International Conference on Mixed Design of Integrated Circuits and System - MIXDES'06. June 2006. 734-737.

[Santos 2012] L. Santos, M.Z. Rela. "Constraints on the Use of Boundary-Scan for Fault Injection." In Lecture Notes on Computer Science, 39-55. Berlin/Heidelberg: Springer, 2003.

[Lauterbach 2012] Lauterbach. Lauterbach Development Tools-Leading Through Technology. 2012. www.lauterbach.com/frames.html?home.html (accessed 2011).

[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.

[Portela-Garcia 2007] M. Portela-Garcia, L.-O. Celia, M. Garcia-Valderas, L. Entrena. "A Rapid Fault Injection Approach for Measuring SEU Sensitivity in Complex Processors." 13th IEEE International On-Line Testing Symposium. IOLTS'07, July 2007. 101-106.

[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.

[Folkesson 2003] P. Folkesson, J. Aidemark, J. Vinter. Assessment and Application of Scan-Chain Implemented Fault Injection. Technical Report, Goteborg, Sweden: Chalmers University of Technology, 2003.

[Pignol 2007] Pignol, M. "Methodology and Tools Developed for Validation of Cotsbased Fault-Tolerant Spacecraft Supercomputers." IOLTS, 2007.

[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.

[Chakraborty 2007] T.J. Chakraborty, C. Chen-Huan, B.G. Van Treuren. "A Practical Approach to Comprehensive System Test and Debug Using Boundary Scan Based Test Architecture." IEEE International Test Conference. ITC'07, 2007. 1-10.

6. Characterization of Uncertainty Sources for Fault Injection

6.1. Introduction

Part of the research effort was to assess the dependability assessment methodology with respect to any oversights or deficiencies that need to be addressed in the context of physical fault injections for digital I&C systems. First, 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 unequivocal definition of the measurands is of importance [Bucher 2004]. A *measurand* is defined as "a particular quantity subject to measurement" or "the quantity intended to be measured" [ISO 1995]
In spite of steady advances in fault injection research over the past 20 years, quantitative evaluation of dependability attributes remains a complex task lacking standard processes and techniques. Until recently, measurement practices and uncertainty analysis for fault injection testing has been largely overlooked. As fault injection has moved from the laboratory to the working environment of various industries, a greater awareness for the need to incorporate metrology concepts into dependability studies has arisen. To do so, it is necessary to regard tools such as those used for fault injection and data collection as measuring instruments, and to obtain measurement results that can be compared and reproduced by others.

As noted in Volume 2, typically, digital I&C systems are not designed to be monitored to the level and extent required for dependability evaluation by fault injection; a methodological approach for their observation is thus needed. In distributed digital I&C systems, the issue is 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.

Finally, measurement-based assessment processes must characterize the various types of uncertainty that can occur during the assessment process to better inform the conclusions made about the measured results, and how those results can be interpreted in a broader, more general context. This becomes especially important when trying to compare fault injection-based findings from one type of system to another.

The methodology presented in Volume 1 and Section 1 of this Volume has not developed this important topic to the degree required for the evaluation of safety critical systems. Since very little research has been done in this area, the development of this important aspect of fault injection is initial work and should be viewed as a starting point for future research in the general I&C community. In this section, the lessons learned to date were used to develop a characterization or theory of uncertainty with respect to fault injection-based assessment processes.

6.2. Background

The difficulties in comparing and reproducing results of dependability measurements arise from a wide range of uncertainties associated with measurement procedures, non-representativeness, and instruments and monitoring processes of the target systems. Examples of non-representativeness are non-realistic workloads and fault-loads. Errors can occur when the wrong instants in time are chosen for collecting measurements. Also, approximate implementation of a measurand definition can result in an error as well as instrument uncertainty due to misconfiguration of the measurement tools used in the process. Such factors cause variations in measurement results. It is therefore necessary to the best extent possible to characterize and estimate the magnitude of such variations and to assess the

metrological compatibility of measurement results. That is, whether different measurements, possibly obtained in different studies, relate to the same measurand.

Over the years, numerous fault injection tools and techniques have been described in the literature (see Section 5 of Volume 1 for a comprehensive survey). The purpose of the fault injection process is to introduce realistic faults/errors into a system and then measure the relevant aspects of the system's dependable operation (e.g., error coverage of specific mechanisms and the error latency). Measurands such as coverage and latency are well defined and accepted by the dependability community (see Section 3 of Volume 1). However, due to the uncertainties associated with each type of fault injection technique, the results of injecting a given fault may differ to some extent. The fundamental questions are the sources of uncertainties in fault injection experiments, and how these uncertainties can affect the results. The next section describes possible sources of uncertainties that may arise in fault injection experiments.

6.3. Sources of Uncertainty

Generally, uncertainty expresses doubt about the validity of a measurement result and may also provide quantitative measures of that doubt. In that context, a measurement result is associated with an uncertainty parameter, which characterizes the dispersion of values that could be attributed to a measurand. Uncertainty is often distinguished by two major classes: Aleatory uncertainty and Epistemic uncertainty.

Aleatory uncertainty is the inherent variation associated with a physical system or the environment under consideration. Aleatory uncertainty occurs in random experiments such as dice throws or chaotic systems. It is often referred to as irreducible uncertainty, that is, more experiments will not reduce the uncertainty because the uncertainty is an inherent property of the system [Limborg 2008].
On the other hand, *Epistemic uncertainty* describes uncertainty of an event or observation due to lack of knowledge or information. Epistemic uncertainty is uncertainty of an outcome due to a lack of knowledge or information in any phase or activity of the assessment process or modeling process. This is uncertainty in the dependability assessment process itself. Epistemic uncertainty reflects uncertainty in "beliefs-about-the-world" [Limborg 2008].

Epsitemic uncertainty is not an inherent property of a physical system. A gain in information about the system or environmental factors of the system can lead to a reduction in the uncertainty. Thus, Epistemic uncertainty is a reducible uncertainty.

With no claim of being exhaustive, this section describes uncertainties specific to fault injection experiments.

6.3.1. Sensor or System Input Uncertainty

Uncertainty in the sensor values or system inputs is typically a stochastic (aleatory) uncertainty and is attributed, in part, to any errors in converting the sensed analog values to an appropriate digital equivalent, or digital values to physical analog values. All conversion processes have a finite amount of imprecision in the conversion process whether it be digital-to-analog (D/A) or A/D conversion. Typically, these errors are very small compared to the converted value or measurand (parts per million). Another aspect of input uncertainty is not knowing significant values of certain inputs that could arise or when they could arise. This is more typical in the real world environment rather than in a laboratory testing environment. The uncertainty lies in not knowing beforehand which circumstances (e.g., inputs to a program) will cause an unusual event and when these circumstances will arise during the operational execution of the program.

6.3.2. Non-representative Sampling

The use of representative workloads and fault-loads is essential in fault injection experiments. Hence, *non-representative sampling* is an important source of uncertainty. This type of uncertainty is epistemic in nature. Operational profile and workload context make the collected dependability measures meaningful for a system under expected operating conditions of the system under test. The uncertainty in this case arises from not knowing or having incomplete knowledge about the true operational profile behavior of the system in either normal and off-normal conditions.

Fault load uncertainty reflects an incomplete understanding of the representative failures that may occur in a specific digital system in a specific environmental context. For instance, one may deduce from the error log of a system that communication data failures occur at a certain frequency. However, it may not be known whether the communication data failures are the result of environmental interference (noise) or the noise is an underlying problem with the timing on the communication hardware/software. The fault models for environmental disturbances are different from the fault models for timing and value corruptions; therefore, it may not be known beforehand which model applies.

Fault load uncertainty is often addressed by gaining knowledge about the faults that are possible and representative of the system in its environmental context. For example, it is known that transient faults occur in digital systems due to environmental disturbances such as single event upsets, electromagnetic fields, etc. Applying these faults would be justified and would be representative of the system response to environment conditions.

Non-representative sampling uncertainty concerns not only the OP and fault-load, but also the time instants when output data is sampled on the target system. This infers that the output of a given system task must be sampled with a given frequency that is appropriate for the context of system plant interactions and that the data items that represent the output must be chosen appropriately.

6.3.3. Determinism and Observability

The degree of determinism or non-determinism of a target system may affect the outcomes of an experiment from one experiment to the next. Most real time systems used in nuclear applications are designed to be functionally deterministic, meaning their behavior is predictable from a task execution and input/output perspective. Determinism is needed for fault injection results to be reproducible.

It is also necessary to ensure repeatability, which is an important property of the fault injection techniques. Repeatability refers to the ability to repeat the injection of a specific fault and obtain the same result. However, there is usually some level of non-determinism at the lowest levels of digital operation such as rounding effects in floating point values, performance improving features in microprocessors such as multi-level caches and branch prediction, and the use of non-deterministic or qausi-deterministic communication protocols.

From a fault injection perspective, these types of uncertainties are not a significant factor in system level deterministic operation. Rather, uncertainties in the ability to precisely observe a set of conditions on the target system and reproduce those conditions from one experiment to the next is an uncertainty that can affect fault injection experimentation. To reproduce a set of conditions in processor-based digital system requires an ability to observe the state and the inputs to the system to the degree necessary to be able to halt the system at any given instance of state condition. In effect, this implies the ability to observe processor instructions and input

conditions and initiate triggers on the basis of those conditions to reproduce the fault injection event.

Fault injection techniques such as OCD-based fault injection, SWIFI, and ICE-based fault injection can provide the level of observability and controllability to achieve reproducibility. Fault injection techniques such as pin-level fault injection cannot produce this level of deterministic behavior, therefore there will be an uncertainty with respect to reproducibility of the results. The key point is that observability and reproducibility uncertainties can be inherent to the fault injection technique chosen. This type of uncertainty is both aleatory and epistemic.

6.3.4. Resolution

Finite resolution of instruments affects measurements in the time and value domains. The binary representation of values in memory and registers can usually be accessed without losing precision. Nevertheless, resolution is a problem if a processor uses an internal precision that is higher than what can be observed by the measurement or monitoring instruments.

Resolution of instruments used in fault injection processes is mostly a concern when measuring time and time delays. This uncertainty is a significant issue when trying to estimate fault and error latency detection in a target system to develop a set of injected faults. As encountered in Benchmark System I studies, the resolution of time-stamps used by the target system and the application of those time-stamps to the error messages produced results that led to imprecise estimates. Later sections discuss practices for better measurement of time based measurands.

6.3.5. Approximation of a Measurand

Measurements are sometimes obtained using an *approximation of a measurands definition*. Sometime this occurs because the researcher is not fully aware of the assumptions of the measurand definition, or has not taken into account all of the factors that may affect the precision of the measurement. In other cases, the experimenter may only be able to measure the outcomes if certain approximations are made.

In dependability theory there is usually a precise definition of measurement-based system attributes with underlying assumptions. Examples include error detection coverage, fault and error latency, mean time to shutdown, etc. However, translating these definitions into physical measurements can be a non-trivial exercise.

For example, in measuring fault coverage one would assume that observing the error detection and fault tolerance behavior under the influence of an injected fault would be sufficient to gauge whether or not the system properly covers the fault. However, there is an implicit time measurement element to the fault coverage definition. The error detection response and mitigation must occur before the real-time deadline of the system.

Therefore, when measuring coverage one must also measure the output response of the system, which is not intuitive from just reading the definition of coverage. These types of oversights can occur in experimentation, and can lead to uncertainty in the measured dependability attribute. These errors were made in the research, thus lead to the conclusion that this type of uncertainty must be considered in the methodology.

In principle, defining and applying good metrology practices before starting the fault injection experiment definition phase will inform the fault injection process on how to measure observations and account for imprecision and uncertainty.

Instrumental uncertainty is often neglected in dependability studies. The tools and techniques involved in dependability measurements are usually complex, thus requiring some form of validation. The fault injection component should be verified against a reference component whenever available, to ensure that it does inject the intended faults. Furthermore, the components that analyze the data collected during each experiment may also contain software defects, leading to incorrect classification of experiments.

6.3.6. System Initialization

Initialization uncertainty can influence the outcome of an injected error. Processor-based systems are usually set to a known state at startup, but the memory may not be initialized in its entirety by the target program. After an error is injected, a program may use non-initialized values; in such cases the contents of unused memory locations and registers can influence the impact of the injected error. The parts of memory that are used by the program in normal conditions are likely to have initialized values (otherwise the target system would probably contain some defects). Consequently, even if a specific fault/error is injected several times into the location, the outcome of such experiments may differ depending on how a given memory location is initialized.

6.3.7. Spatial Intrusiveness

Spatial intrusiveness reflects modifications made to the system under test to incorporate fault injection and allow monitoring for experimental evaluation. Such changes typically include adding special software (ISRs) to the target program for injecting faults, as in the case of SWIFI techniques. By doing such a modification, the system memory and code segments are different from the original program and instead hold valid instructions or data in the modified program. The outcomes of fault injections in these two program versions can differ, as an injected fault in the modified program can access memory values that are not present in the original program.

Spatial intrusiveness also arises if the memory layout of the target program is changed. This happens, for example, when a program is loaded into a different memory segment due to dynamic memory management or is deliberately placed in a memory segment for the purpose of experimental evaluation. Dynamic memory management has been seen to cause spatial intrusiveness in one case, the DFWCS described in Section 7 of Volume 1. In this case, memory code segments were remapped every time the processor reset.

Spatial intrusiveness can occur with specific types of fault injection. OCD and JTAG fault injection typically do not modify the target system, and these techniques have no intrusiveness with respect to space. The exception-based or SWIFI-based fault injection techniques, however, require that the target program be extended with additional instructions in the way of ISRs. For example, the Xception SWIFI fault injection tool requires approximately 5Kb of code and data for most applications [Kanawati 1995b].

6.3.8. Temporal Intrusiveness

Temporal intrusiveness is especially important in real-time systems. Assessing the impact of faults in such systems requires the fault injection tool to have a negligible impact on the target system with respect to time. To obtain meaningful measurements, the temporal intrusiveness must be lower than the resolution used for output timing measurements (usually on the order of milliseconds or tens of milliseconds).

Temporal intrusiveness was investigated for the several of the techniques have used in the fault injection study of the benchmark systems. For the OCD-based injection, it is mainly caused by the time needed for sending commands and data between the fault injection tool and the target

system. The FPGA-based HiPeFI described in Section 5 of this report minimizes these interactions to the point where the time intrusion is several orders of magnitude below the resolution of the output timing measurement. The JTAG fault injector has similar time intrusion characteristics as well. The commercial iSystems OCD-based fault injector had a time intrusion of 1ms to 2ms for register fault injections and about 4ms for memory fault corruptions. This amount of intrusion is marginal for Benchmark System II. However, the ICE-based fault injection used on Benchmark System I and the DFWCS was highly intrusive (on the order of tens of milliseconds or more). The communication between the host computer and the ICE machine components in ICE-based fault injection generates high and not entirely deterministic temporal intrusiveness, which is unsuitable for real-time systems.

Although the SWIFI approach was not used in these studies, the temporal overhead for th SWIFI-based fault injection is caused by the execution of the ISR. In previous implementations, the ISR routine requires at most 100 machine instructions to handle an exception, inject an error, and configure a breakpoint that is used to terminate the experiment. These instructions are executed only once for each injected fault. Thus, the onboard temporal intrusiveness of exception-based injection is constant and very low. The communication between the ISR and the fault injection environment also must be considered when implementing SWIFI, as communication overhead can be a factor.

6.4. Open Issues

After identifying potential sources of uncertainties in fault injection, further research must be conducted to determine how these uncertainties affect fault injection and how they can be minimized.

Evidence from comparative analysis experiments should be collected to see how different fault injection techniques are comparable in their results, taking into account the uncertainties that may arise.

Guidance for sound metrological practices should be incorporated into a fault injection-based dependability assessment methodology as a fundamental process in the methodology. The inclusion of sound metrological practices is within the reach of a modest research effort.

Characterizing and investigating the types of uncertainties that may affect fault injection should be addressed by the dependability research community and the I&C community together.

The uncertainties presented in this section do not undermine the veracity of fault injection or measurement-based assessment methodologies, rather they present an open and transparent perspective that informs the knowledge base.

6.5. Bibliography

[Bucher 1995(b)] Bucher, J.L. *The Metrology Handbook.* American Society for Quality, 2004.

[ISO 2007] ISO/IEC Guide 99:2007 - "International Vocabulary of Metrology (VIM)", ISO (International Organization for Standardization) Standards, 2007

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

[Limborg 2008] Limborg, P. *Dependability Modeling Under Uncertainty.* Springer-Verlag, 2008.

7. Integration of the Benchmark System II into the UNIFI Fault Injection Environment

7.1. Introduction

This Section describes the integration of Benchmark System II into the UNIFI environment, configuration of the benchmark system for fault injection, and discussion of better measurement practices for fault injection set-up and testing.

7.2. Overview of UNIFI

Section 5 of Volume II describes development of fault injection environment for digital I&C systems. The fault injection environment developed in this research is the UNIFI platform. A more through discussion of UNIFI can be found in that Section; an overview is presented in this section to set the context for this effort.

The objectives of UNIFI are to provide a user-friendly fault injection environment and support for adaptation to new target systems and new fault injection techniques. To achieve the first goal, a UNIFI graphical user interface (GUI) was developed to standardize fault injection experiments. The second goal was achieved by providing a plug-in based framework. New techniques and target systems can be added through the UNIFI-LabVIEW plug-in interface. A major advantage of this architecture is that a new plug-in can be added to UNIFI without the need of a regression test since the system is not affected by bugs in the added plug-in. UNIFI does not have to be recompiled when a new plug-in is added and the new plug-in will automatically be found when UNIFI is restarted in the LabVIEW environment.

Figure 7-1 shows the UNIFI tool with different plug-ins and how UNIFI interfaces with a target system and a target system software development environment. In the UNIFI framework, the various fault injection plug-ins and the database that stores information and results from fault injection experiments are located within the host computer.

The TRACE thermo-hydraulic simulation tool generates sensor data for nominal, off-nominal, and accident scenarios. A special pre-processed input file developed from TRACE calculated results provides input data to the *operational profile generator* module.

The *experiment set up and control* plug-in function selects fault injector(s), configures the fault injectors, and initializes the UNIFI tool for a fault injection campaign. The *fault list generation plug-in module* generates a fault list that is parameterized with fault models of interest, locations of fault injection on the target system, type of fault injection, and when the fault is injected. This fault list is then loaded into a file for the experiment control plug-in to access during a fault injection campaign.

The inputs to the fault list generation plug-in module are dependent on the type of fault injection selected. Typically for processor-based fault injection, the map files from the target system compile and link processes are used as the inputs. For JTAG fault injection the boundary scan registers map from IC vendor is used. For communication-based fault injection, the control and data packet structure of the communication messages are used to identify where and when to corrupt message traffic.

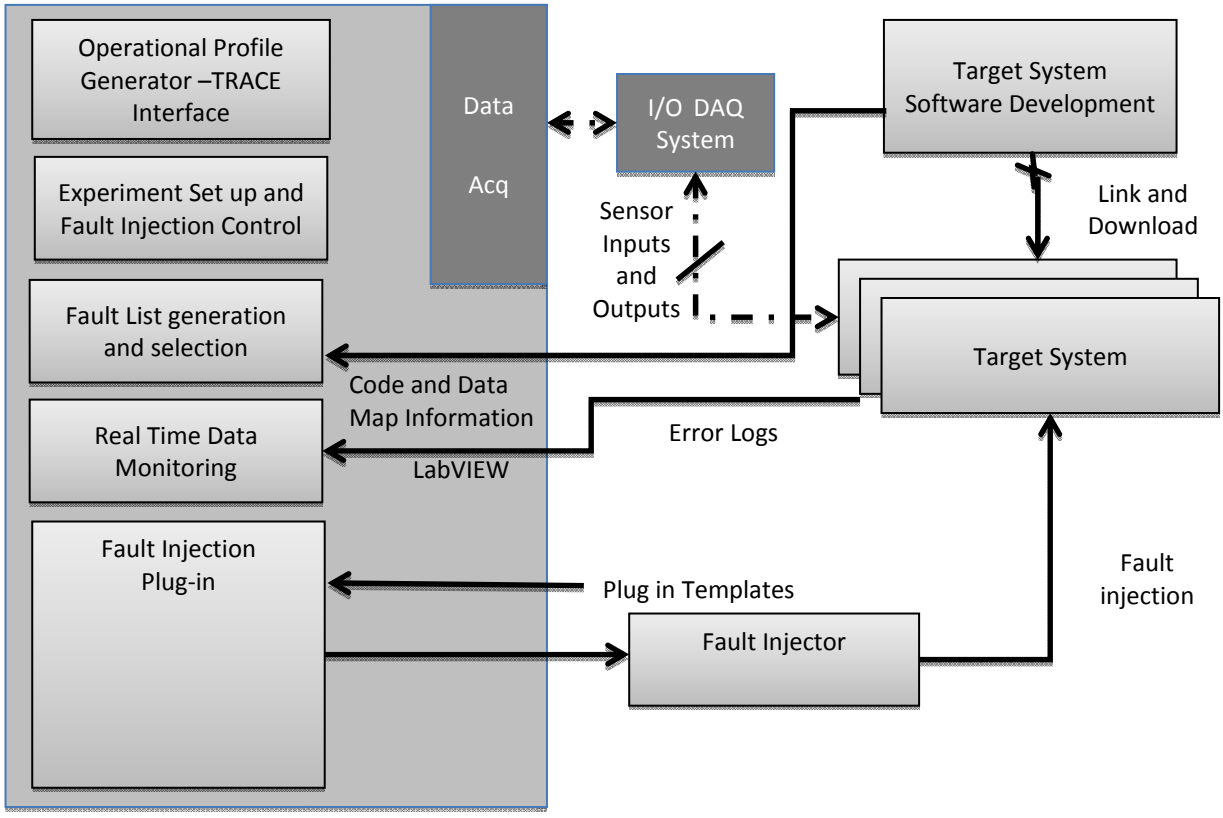

Figure 7-1 UNIFI fault injection environment

The *real time data monitoring and collection module* interfaces to the target system diagnostics and error monitoring server to collect error messages and error logs after the fault is injected into the target system. These error logs and timing files are stored in a database that allows the error log to be correlated with the experiment control information such as the type of fault that was injected, the operational profile conditions, the time the fault was injected, where the fault was injected, etc. This allows experiments to be reproduced as needed. In addition, the outputs and feedback loops of the target system are sampled by the LabVIEW interface to obtain a complete time response of the target system for each fault injection experiment.

The *fault injection engine plug-in module* allows different types of fault injection techniques to be adapted to the UNIFI tool. At this phase of research, plug-in modules have been developed for ICE-based fault injection for the Pentium processor in Benchmark System I and the X-bus fault injector for corrupting message and control traffic on the Benchmark System I.

Switching between different target systems involves minimal effort, which is primarily focused on fault injection plug-ins, data monitoring, and the I/O subsystems. Because UNIFI does not support all target system chip architectures, some customization (e.g., designing a new plug-in module) may be necessary, however the flexible nature of the UNIFI tool reduces the amount of time from conception to implementation.

Fault injection requires the joining together of several processes and coordinating these tasks to achieve the overall goal of automated fault injection. An example is the coordination of processes required for configuration, fault injection set-up, fault injection campaign management, and data acquisition and monitoring.

7.3. Configuring and Selecting a Fault Injector

An important step in configuring UNIFI for a target system is to select and interface an appropriate fault injection technique for the target system. Generic templates for important processes to assist the user in adapting the UNIFI to their target system have designed. The modules typically used are file open, file close, file return, string to array, call function, and return function.

For Benchmark System II, the FPGA-based HiPeFI in UNIFI was used as the principle fault injection module. In UNIFI, the configuration of the fault injector is driven by the application programmer interface (API) interface of the fault injector. As discussed in Section 5.6 of this volume, the FPGA-based fault injector communicates with the UNIFI fault injection environment to achieve low overhead fault injection. The following five basic steps are coordinated between UNIFI and the fault injector for each fault injection:

(1) Reset of the FPGA Board – Resets the modules, software, and state machines to an accepting state.

(2) Initialization of the FPGA board – UNIFI sends an initialize command to the FPGA board that initializes all modules.

(3) Send parameter data to the Fault injector - UNIFI passes a file of ASCII strings of fault injection parameters to the fault injector. Each line in the fault list file is a single fault experiment. A fault string typically includes fault type, fault module type, fault mask, fault location, fault trigger time, fault delay time, etc. The entire fault list is sent to the fault injector so this step occurs only once per campaign. Figure 7-2 shows a typical fault list. The parameter string for a single fault injection begins with the “s” which means start, and ends “x.”

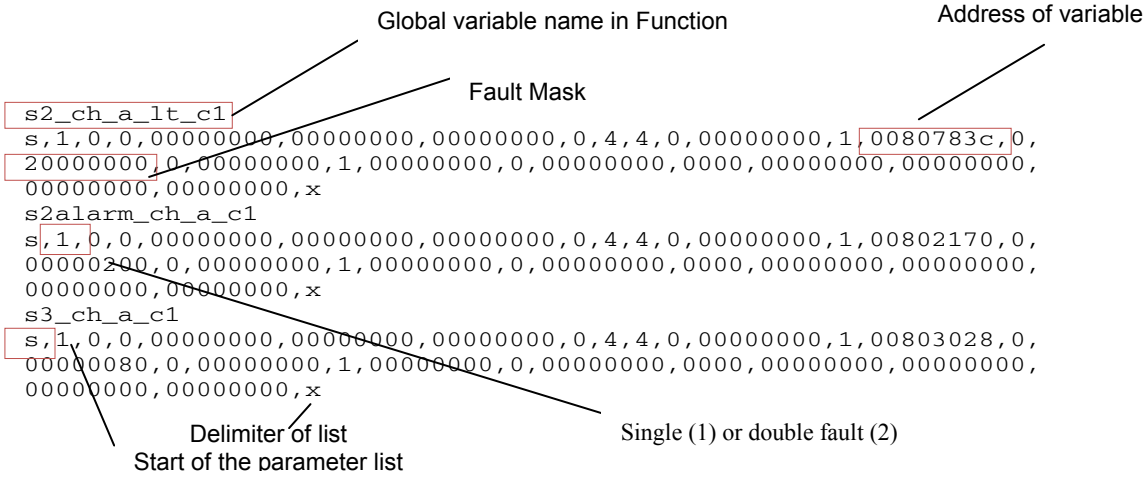

Figure 7-2 Fault list snippet

(4) Fault Inject Start – UNIFI sends a command to initiate a fault injection based on the fault list sent.

(5) Fault Successful Ack – The Fault Injector sends an acknowledgment back to UNIFI if the fault injection was successful (e.g., the fault was injected).

Once the fault injector is configured, the Campaign Manager GUI of UNIFI is used to automate the fault injection process.

7.4. Set up of Fault Injection Campaigns

The set-up phase is used for setting up fault injection campaigns and generally involves three steps in UNIFI. In the first step, the user enters data about the campaign in the campaign setup tab in the master controller window (see Figure 7-3). Then, specific information about where and when faults will be injected are defined in the fault injection setup tab. Finally, the registers and memory positions that the user wants to observe are defined in the observation setup tab.

From the menus in the GUI, fault injection campaigns can be configured by starting the corresponding plug-in for a chosen target system and fault injection technique. The campaign name, the number of experiments in the campaign, and the time-out value for the experiments must also be entered. A fault injection experiment can be terminated when a time-out value has been reached, an error has been detected, or the execution of the workload ends, whichever comes first. The workload may consist of a program that either terminates or is executed as a cyclical task.

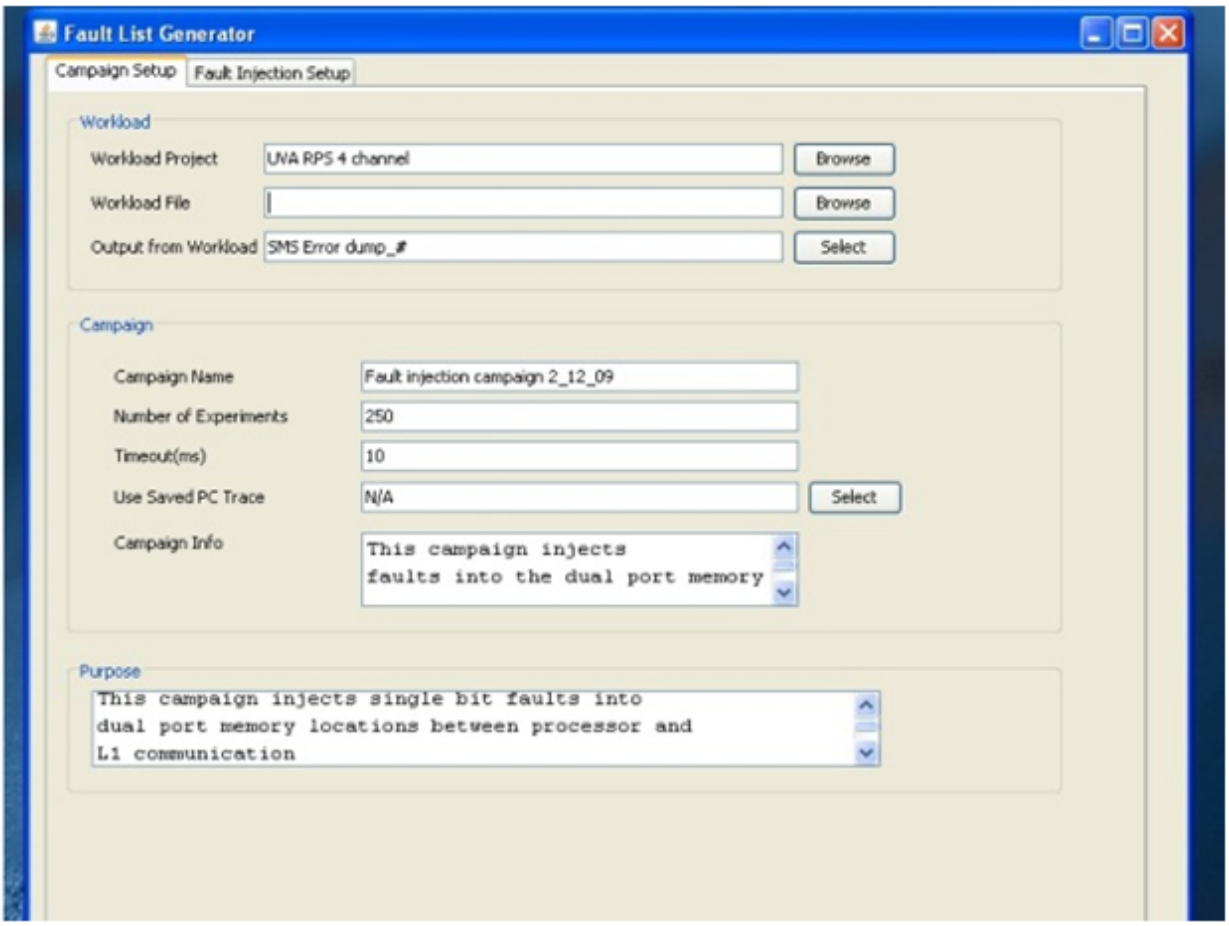

Figure 7-3 Screenshot of Master Controller window

The user may also choose a pre-injection analysis to improve the efficiency and maximize the error acceleration of the fault injection experiment if the processor supports pre-injection analysis. Pre-injection analysis and error acceleration is discuss in Section 8 of Volume II.

7.4.1. Fault Injection Set Up

The fault injection set-up is where the detailed fault lists are created for the fault injector. Figure 7-4 shows the process for generating a fault list in UNIFI. There are two basic modules to the fault injection set up tool: a front end GUI for defining the fault injection parameters, and backend parser module to parse the map files from the target system compiler.

The fault injection set up tool was designed with two separate modules to enhance portability between digital I&C platforms. The back-end module is specific to the target system native object code and map file format that is generated from the compiler. This information usually changes from one digital I&C system to the next. The front end GUI is the high level interface, which contains the relatively established fault injection parameters that are used on most digital I&C systems. These parameters include type of fault, fault mask, memory locations, etc. Occasionally, some modifications may be necessary to the front end GUI for a particular target processor. In these cases, the GUI is easily modified due to its Java-based design. The open source free-ware Net-Beans Java creation tool was used to create the front end GUI.

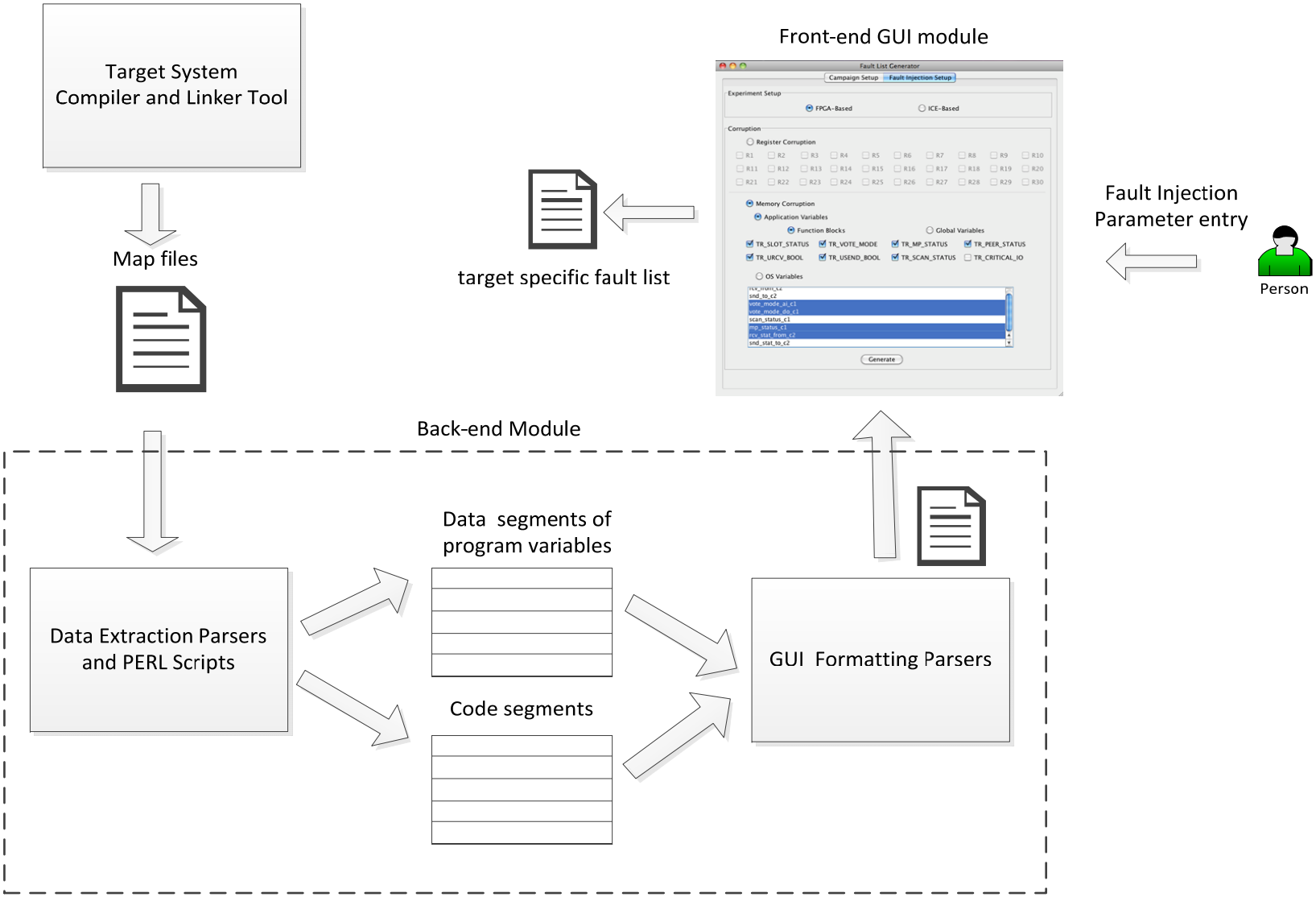

Figure 7-4 Process for generating a fault list using UNIFI

For Benchmark System II, the following options for fault list generation are available:

- Single or multiple fault injection - Fault lists can be created for single faults or multiple faults. In the case of multiple faults the fault list is annotated with information with respect to where the second fault occurs, when it becomes active, and if it is triggered by an event.
- Fault type – Transient or Permanent. Transient faults are injected once per fault injection execution. Permanent faults are the repeated triggering of breakpoint set of specific register or memory location.
- Function and Function Block fault lists - Faults are generated with respect to specific function blocks selected by the users in the fault injection set-up GUI.
- Register or Memory – User selects physical processor registers and physical address locations.
- Global variable - Global variables of the application and/or operating system.
- Operating system – Functions, variables of the OS, stacks, etc.

Fault list front end GUIs are shown in Figure 7-5.

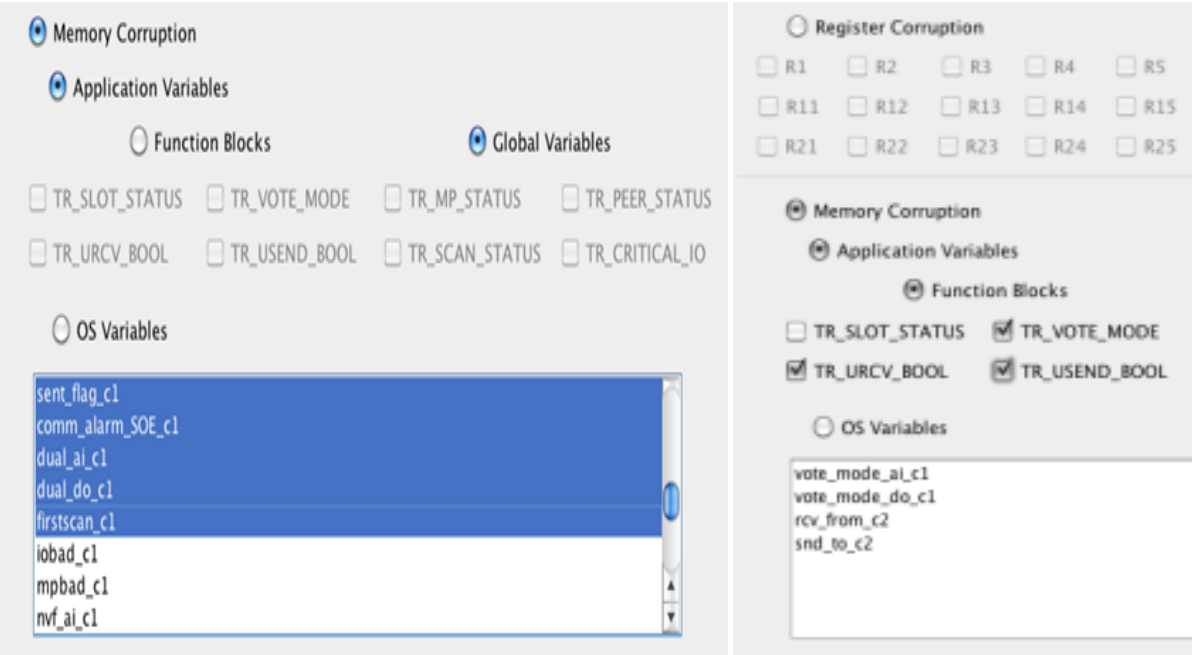

Figure 7-5 Screenshots of the fault list generation GUI

7.5. Benchmark System II Test Configuration

The Benchmark System II configuration consists of two chassis assemblies. As shown in Figure 7-6, each chassis includes termination panels, power supply modules, three main processor modules, redundant digital and analog 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.

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

In the configuration shown Figure 7-6, RPS channel A is isolated from channels B, C, and D by an independently-powered chassis component rack. Each RPS channel has its own set of analog inputs for the process variables coolant flow, SG pressure, and hot leg pressure. These inputs operate on a 4ma to 20ma current loop, as is typical for digital I&C systems in NPPs. There are 3 redundant analog signals for each measure process variable for a total of 12 analog inputs for the 4 channels. Two digital inputs were used to indicate when the benchmark system was in TMR mode and ready for fault injection.

The digital outputs of the RPS channels were the channel alarms for each redundant process variable, channel trip signals for each monitored process variable (i.e., coolant flow, hot leg pressure, and steam generator pressure) resulting in 12 digital outputs. All channel alarms from each channel were exchanged with all other channels to form a distributed 2 out of 4 voting RPS via the peer-to-peer communication network. In addition to the RPS digital outputs, six additional digital outputs were used to indicate fault status and assist in more accurate timing of fault detection events. Table 7-1 lists the signals for the RPS system.

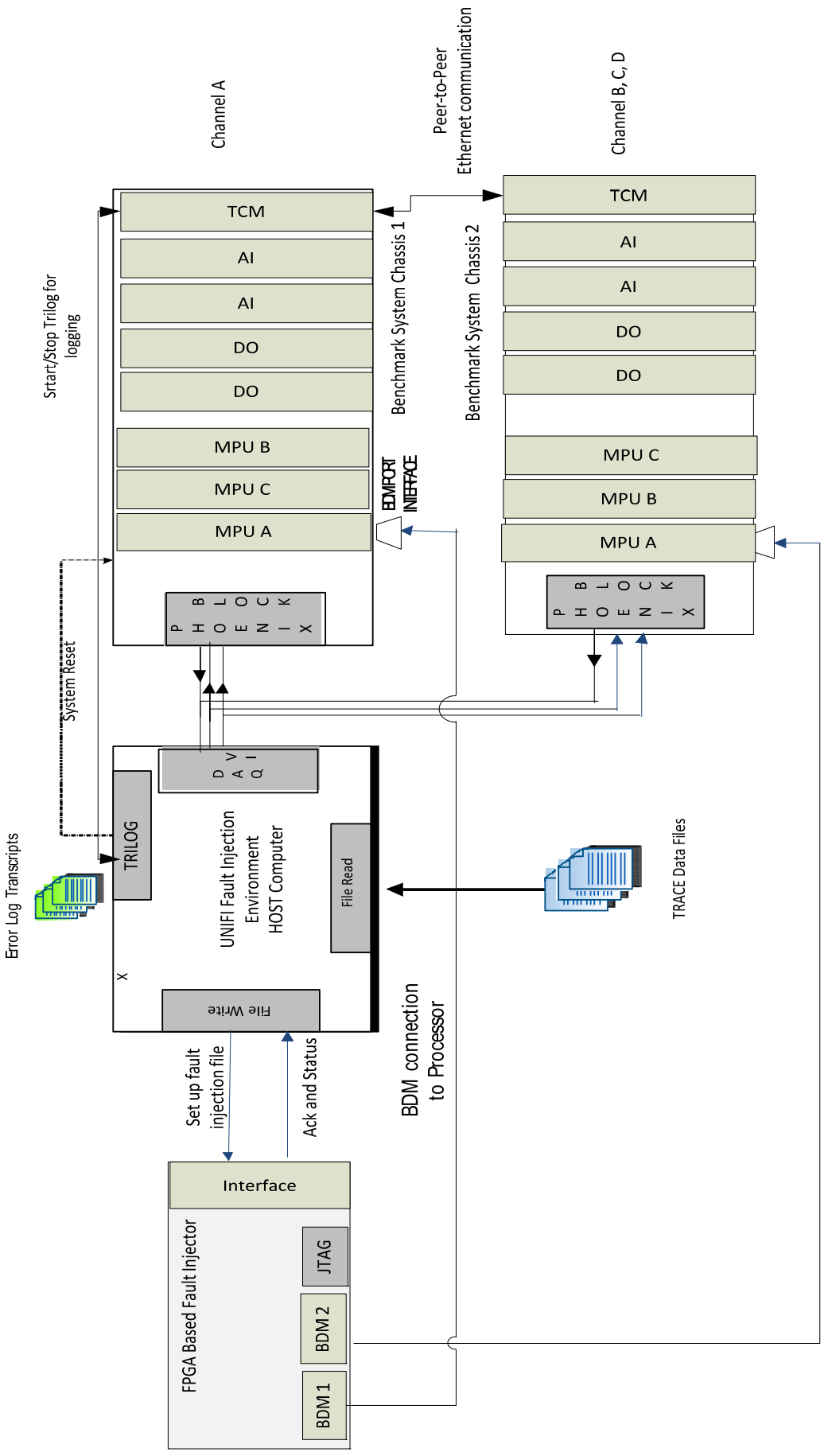

Figure 7-6 Test configuration of Benchmark System II

Table 7-1 Inputs/Outputs in the Benchmark System II RPS.

Signal Name	Description	Number of signals per channels	Analog Inputs	Digital Outputs	Digital Inputs
SG Signal Alarm	Sensor readings exceeds set-points	3		X	
Coolant Flow Signal Alarm	Sensor readings exceeds set-points	3		X	
Hot leg Pressure signal Alarm	Sensor readings exceeds set-points	3		X	
SG trip Alarm	2 or more Sensor readings exceeds set-points	1		X	
Coolant Flow trip alarm	2 or more Sensor readings exceeds set-points	1		X	
Hot leg pressure trip alarm	2 or more Sensor readings exceeds set-points	1		X	
Steam generator sensor signal	Input to RPS	3	X		
Coolant flow sensor signal	Input to RPS	3	X		
Hot leg pressure signal	Input to RPS	3	X		

Table 7-1 Inputs/Outputs in the Benchmark System II RPS.

Signal Name	Description	Number of signals per channels	Analog Inputs	Digital Outputs	Digital Inputs
Comm_alarm	Error in Communication between chassis	1		X	
MP_bad	Error detected on main Processor	1		X	
IO_bad	Error detected on I/O processor	1			
TMR vote mode	Three processors are running in chassis	1		X	
Dual Mode	Only two processor running	1		X	
Single Mode	Only one processor running	1		X	

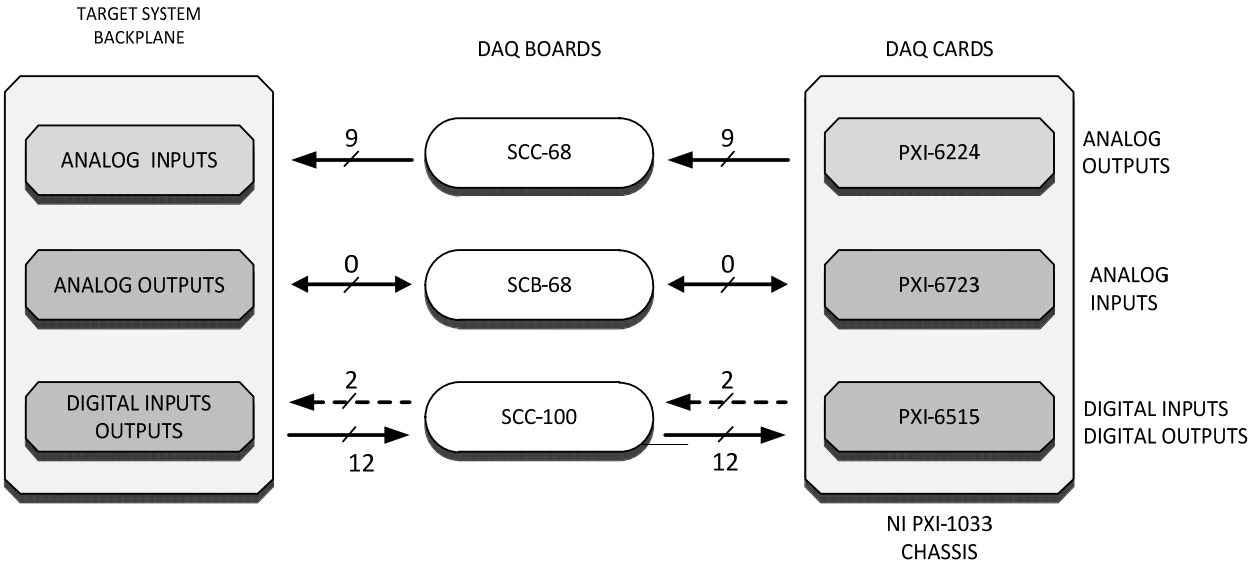

Figure 7-7 Signal connections to benchmark system

7.5.1. Integrating the UNIFI I/O data acquisition system to the benchmark system

The UNIFI I/O Interface module or template is a collection of LabView block diagrams that provide a software/hardware interface between UNIFI and the target I&C system. Referring to Figure 7-6 and Figure 7-7, the benchmark system I/O backplane, which consists of an array of Phenix block connectors, is wired to the National Instruments™ PXI data acquisition signal connection breakout boxes. These breakout boxes form the connection between the benchmark system and the data acquisition modules in the PXI-1033 controller. The signal wiring conforms to the Benchmark System II interface standard as shown in the digital output example box in Figure 7-7. Analog input signals are connected to the SCC-68 module; digital input and output signals are connected to SCB-100 module. Both of the breakout boxes are connected to PXI-1033 controller by a wiring harness.

The PXI-1033 controller chassis contains A/D, D/A, and signal conversion boards to interface all I/O signals entering and exiting the benchmark system. The PXI-1033 controller chassis is connected to the UNIFI host computer by a PXI express connection to a PXI port on the host computer. The LabView design environment recognizes the PXI-1033 and loads the drivers for the PXI-1033 chassis onto the host machine so Labview can recognize the controller. Control and configuration of the data acquisition cards in the PXI-chassis is accomplished through the LabView interface and libraries. All of the data acquisition, control, and measurement of the signals are accomplished by UNIFI LabView function blocks.

Analog input signals (sent to the benchmark system) are generated from the TRACE operational profile generator tool (described in the next Section) and are placed into a profile file. LabView reads this file and converts the sensor and signal data from the operational profile into digital and analog signals that are fed into the target system by the PXI-1033 controller.

Response data from the benchmark system (digital outputs) is collected, time-stamped, and logged. The response data includes trip alarms, failure event flags, feedback signals from the benchmark system. These signals and events are recorded using the recording functions in the

measurement and recording plug-in. These tasks are accomplished using the special library of virtual instrument (VI) functional blocks provided by LabView from National Instruments™. Since LabVIEW monitors all signals input and output by the PXI-1033 controller D/A and A/D cards, UNIFI provides a non-intrusive means of observing sensor and feedback signals.

7.6. Measurement Revisited

A lesson learned from the Benchmark System I investigations was that time stamps from the error logs were not very reliable for calculating system response and fault/error latency attributes. As such, the measurement practices were modified to reduce the uncertainty and imprecision in the measurements of these time stamp values.

The approach taken was to use the PXI-1033 data acquisition sampling capability to sample certain digital outputs of the benchmark system via the precision sampling card on the PXI-1033 data acquisition system. The digital outputs of most importance were:

- MP_bad – Indicates that the injected fault caused an detectable error on the main processor.
- IO_bad - Indicates that the injected fault caused a detectable error on the I/O processor.
- TMR vote mode – Indicates that the system is in full TMR mode (i.e., the injected fault had no effect on the configuration).
- Dual Mode - Indicates that the system has degraded to dual mode (i.e., the injected fault caused the configuration to degrade to two working processors).
- Single Mode - Indicates that the system has degraded to a simplex mode (i.e., the injected fault caused two working processors to fail).

One of the timers used in UNIFI is a fault injection countdown timer. This timer establishes when the fault injection is to take place. A timer value can be absolute (X seconds from the start of the sequence), or relative (inject the fault after some event), or randomly selected. One of these timers was used to start sampling of the benchmark system status output just prior to fault injection. The outputs were sample at 1-ms intervals (50 times the scan time of the benchmark system). The sampled data were stored and recorded in a data file. Each data file was created with the signal file name, date, and time appended to that it so it can be traced back to the fault injection experiment.

With this type of measurement process, all time values of the benchmark system were measured using a single precision clock. This greatly simplified the calculation system response time, and allowed the precision of the measurement to be attributed to the sampling interval used in the measurement instrument which was fully characterized from the instrument measurement specification.

7.7. Pre-fault Injection Analysis Revisited

Section 7 of Volume II describes pre-fault injection analysis, which is an innovative method for improving the effectiveness and efficiency of fault injection experiments and campaigns. Pre-injection analysis is a means to reduce or eliminate “no-response” and long fault latency issues associated with typical fault injection methods.

Being a statistical experiment, fault injection testing may require that a large number of experiments be conducted to guarantee statistically significant results. Thus, efficiency of the fault injection testing is important. However, the application of the method developed in this research relies on obtaining execution traces from the benchmark system, which proved to be problematic with Benchmark System I. The main issue with Benchmark System I was related to using tools such as ICE machines or interactive debuggers to gather trace information and store these results in trace buffers or files to be pre-analyzed by the pre-fault injection analysis algorithms. The first problem associated with this approach is that the assessors of the system under test may not have access to such development tools. The second problem with this approach is that tools may not be capable of acquiring a sufficiently long execution trace that captures the entire behavior of the application. The third problem is that the system under test may not support the use of these development tools because the processor used in the Benchmark System Is not supported by the development tool vendors.

However, simulation studies have shown that pre-injection analysis can have a significant impact on fault injection studies by improving efficiency and effectiveness, while also allowing fault list reduction methods to be used. Therefore, research was conducted to determine how pre-injection analysis can overcome these implementation obstacles and become more accessible to the digital I&C system community. The following section describes how pre-injection analysis can be implemented in general way.

7.7.1. Pre-Injection Analysis using Interactive Code Analyzers

For all of the reasons mentioned above, physical trace-based pre-injection analyses, in which actual trace data is collected from the system under test, are only applicable in circumstances where the development tools can support the full needs of acquiring an execution trace from the system. These instances do not appear to be promising in light of experiences with the benchmark systems.

A possible way forward that obviates the need for physical-based execution traces is the use of advanced code analysis tools such as IDA Pro. IDA Pro is a powerful commercial interactive disassembler and debugger with a set of visualization tools that allows a user to completely map out the behavior of the binary code of a system. The use of IDA Pro for pre-injection analysis was suggested and partially investigated in preliminary work with developing the algorithms [Sekhar 2008]. The benefits of using IDA Pro for pre-injection analysis is that the tool can be used with many different processor families, not just the processor family of the system under test or interest. Therefore, the extensibility of the tool to a variety of systems addresses the generalization objective described mentioned above. The cost of IDA Pro is approximately $900 dollars, which is much less than the cost of an ICE machine or interactive OCD tool.

7.7.2. Using Static Disassembler Tools for Pre-fault Injection

Static analysis is the analysis of code prior to execution. While dynamic analysis is used to determine the program behavior for a specific input sequence, static analysis can be used to study program behaviors for different input sequences. Static analysis can be performed with control flow and data flow graphs. Static analysis can be conducted on the disassembled binary code or on the source code, if it is available. For fault injection experiments, static analysis on the disassembled binary code is the preferred choice as the binary load image is exactly what is executing on the target machine. The use of IDA Pro® was examined to determine areas in the code that can be identified as suitable candidates for fault injection purposes. This section describes the use of IDA Pro for pre-injection analysis to identify and extract code segments.

IDA Pro® is a binary analysis/disassembler tool that supports many instruction set architectures (Intel, Motorola, ARM, etc.) [Pierce 2008]. IDA Pro® is considered the de-facto binary code

analysis tool on the market. For that reason, it is widely used in the security arena for security vulnerability analysis of embedded systems. The tool disassembles binary code and allows the binary code to be loaded at a desired offset in memory. IDA Pro® can identify symbolic information if the binary code has been compiled for debugging purposes. This allows IDA Pro® to identify function names and other strings present in the source code from the binary machine code listing.

A pre-fault injection analysis is based on the assumption that symbolic information is available for the disassembled binary code, which often is the case. An important feature in IDA Pro® that is of interest for pre-fault injection analysis is the graphing utilities. These utilities provide control flow graphs of the disassembled binary code as well as graphs of cross references to the functions, as illustrated in Figure 7-8.

Referring to Figure 7-8, the disassembled code is shown in the left portion of the screenshot of IDA Pro. However, this representation is not very informative for pre- injection analysis purposes. Rather the code structure should be viewed. Observing program code flow on a graph usually gives a better global view of function structure rather than examining the disassembled code. The *Flow Chart* command of IDA Pro draws such graphs. The graph on the right in Figure 7-8 shows the control flow of the disassembled program. An important feature of the control flow graphic capability is that locations in a function for applying pre-injection analysis can be identified using this feature. For example, the function *strngcopy* at the top of the graph has a window of opportunity with the *ebp* register.

While graphing provides the ability to decompose the code into its structural behavior, the feature itself does not pre-analyze the code for fault injection windows of opportunity. The pre-injection analysis algorithms developed in this research do that. There are several methods for accomplishing pre-injection analysis using IDA Pro. The first and most user-friendly method is to design a “plug-in” for pre-fault injection analysis on IDA Pro. IDA Pro has open API architecture that allows users to write their own plug-in tools and to customize IDA Pro for their own purposes. A second method is to extract (copy and paste) all of the disassembled code for each function block and store it in a file for pre-injection analysis algorithms to parse. While this can work, it is is susceptible to transcription errors. These approaches for pre-injection analysis may be addressed by UVA in future developments of the methodology.

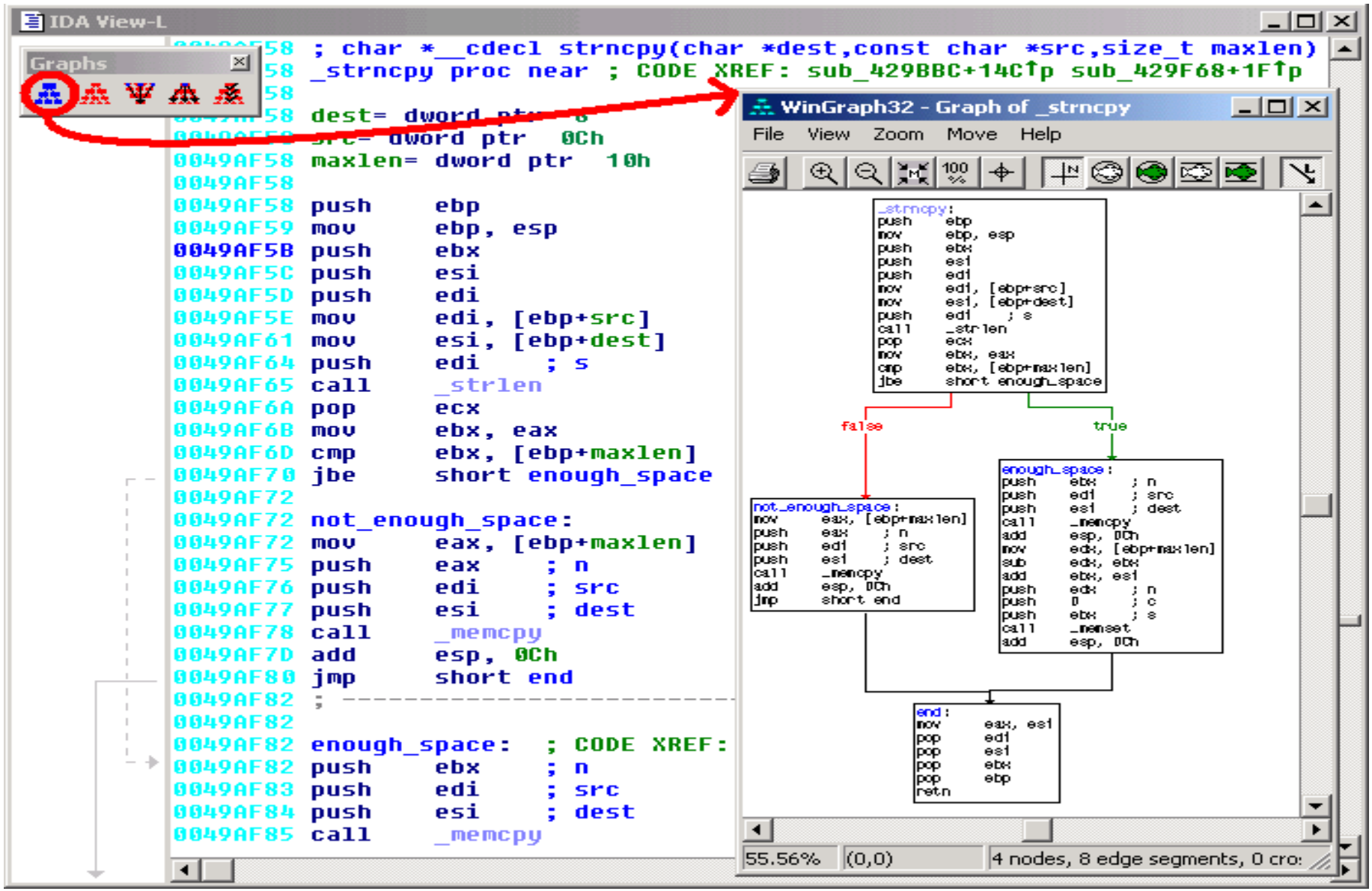

Figure 7-8 Graphing binary code with IDA Pro

7.7.3. Using IDA Pro Dynamic Debug Capability

While static analysis using IDA Pro function graphs is one method to enable and implement pre-injection analysis, another possibility exists in IDA Pro for benchmark systems that use a commercial-based real time operating system (RTOS) such as Linux Real time or VxWorks that allows dynamic analysis using tracing functions. Neither of the benchmark systems in this research employed a commercial RTOS. However, VxWorks is used extensively in the embedded systems world.

Dynamic analysis is analysis that is conducted by observing running code. Dynamic analysis is usually conducted with the help of execution information such as execution traces. The original pre-injection analysis methods used in this research project were developed for dynamic analysis. The technique of dynamic analysis by examining execution traces was used to determine resources (registers and memory locations) that were used by the application. IDA Pro has the capability to extract dynamic execution trace information from executing programs provided that those programs are hosted by one of the OS platforms that IDA supports.

Tracing allows the user to record various forms of tracing information or “trace events” during the execution of an application. IDA Pro writes this trace event information in a "trace buffer". The size of this trace buffer can be limited (in this case, newer trace events will overwrite older ones) or unlimited (which may require substantial computer memory resources). IDA Pro offers different tracing mechanisms:

- Instruction Tracing: IDA Pro will record the execution of each instruction, and save the resulting register values. By using this information, the execution flow of the application and the registers modified by a given instruction can be determined. The computer running the IDA Pro interface is the "debugger client".

- Function Tracing: IDA Pro records all function calls and function returns.

- Read/Write-Write-Execute tracing: IDA Pro records all access to a specified address. Internally, Read/Write, Write, and Execute tracings are nothing more than non-stopping breakpoints.

For pre-injection analysis *instruction tracing and read/write execute tracing* would be used to collect the data traces for the pre-injection analysis algorithms. These tracing options allow the user to collect all trace information needed to perform a pre-injection analysis and generate a fault list. Again, the debug trace options are only available for supported commercial OS and not proprietary operating systems.

7.8. References

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

[Pierce 2008] W. Pierce, J. Larson, L. Miras. *Reverse Engineering Code with IDA Pro.* Elsevier Science, 2008.

8. Application of Fault Injection to Benchmark System II: Results

8.1. Introduction

Once the UNIFI fault injection environment, operational profile generator, and RPS code activities were completed, fault injection campaigns were conducted intermittently over 8 months on the Benchmark System II. These campaigns were conducted near the end of Phase 2 of the project, and therefore were limited in the time available for conducting the fault injection campaigns. Nonetheless, a significant number of fault injection campaigns were conducted on the benchmark system. The fault injection campaigns were run in groups of experiments to assess applicability factors of fault injection with respect to a digital I&C system, to determine the ability to produce information that would support PRA assessment activities, and to produce information to support claims of system operation.

Overarching these objectives was the “process of discovery”, that is, collating the lessons learned during the research up to the culmination of results. This Section presents the types of quantitative and qualitative results that were obtained by applying the fault injection-based dependability assessment methodology to the benchmark system. It is important to note that the data and results presented in this Section are meant to be interpreted as the types of information that can be acquired from a fault injection-based methodology, and are not an assessment of the capabilities of the benchmark system. The benchmark systems that were used as test platforms in this research were scaled representations of a digital-based RPS and thus do not encompass all of the features that could be found in a typical digital based RPS in a NPP.

8.2. Typical Fault Injection Sequence For Benchmark System II

After UNIFI has been properly configured and interfaced to a target digital I&C system, a fault injection campaign can proceed using the Master GUI. Figure 8-1 shows the typical automated operations that are performed by the UNIFI master GUI for a single fault injection experiment. Typically, an experimenter will run hundreds of fault injection trials per campaign.

The fault injection sequence begins with resetting the target processor prior to a fault injection experiment to ensure the system is in an error free state. Diagnostic routines and self-tests that are executed during boot up are the primary means to determine that error effects from previous fault injection experiments have been purged from the system.

The benchmark system has the capability to externally signal its health status to UNIFI with a TMR_status mode signal, which indicates the system is in TMR mode. If the target computer signals to UNIFI that it is operational, then UNIFI will proceed with the fault injection sequence, otherwise, UNIFI aborts the sequence with a NO-GO message to the user. From this point forward, UNIFI automatically runs the fault injection sequence to completion.

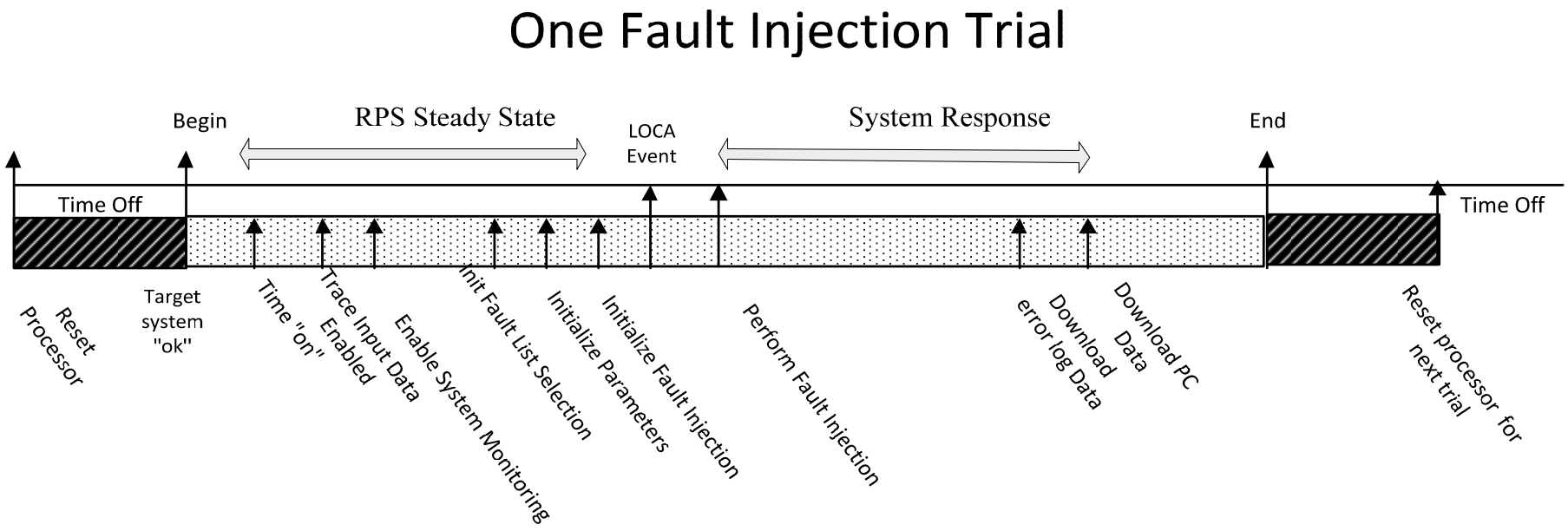

Figure 8-1 Fault injection sequence for Benchmark System II

Referring to Figure 8-1, the first few steps in the process are associated with initializing the measurement systems, and providing operational inputs to the target system. These steps include

- enabling the count-down timers
- setting the length of the censor time
- initializing the measurement instruments in UNIFI
- sending operational profile sensor data to the target system through the I/O interface module
- establishing a connection to the benchmark system TRILOG monitor

The TRILOG monitor observes the target system during operation and stores error messages from the benchmark system. These error messages form an error log of events during a fault injection experiment. During this phase of the fault injection experiment sequence the system is provided with steady state inputs from the TRACE operational profile generator.

The next steps in the UNIFI process initialize the fault injection experiment processes in the FPGA fault injector module. The fault list is initialized and incremented to the current active fault to be injected into the benchmark system. The fault injection parameters such as fault type and fault mask are then indexed and loaded into the fault list.

One of the timers used in UNIFI is an operational profile countdown timer. When this timer reaches zero, the TRACE operational profile file is accessed by UNIFI to initiate a plant event, in this case a LOCA. In addition to the operational profile timer, there are the fault injection timers. These timers establish when the fault injection is to take place. A timer can be absolute (X seconds from the start of the sequence), or relative (inject the fault after some event) or random. In the above timeline, the fault injection is triggered after the LOCA event is initiated. This scenario was used uniformly throughout the campaigns.

The target system is allowed to run until the censor time of the experiment expires. This is set by the user during campaign set up. The sensor time for the experiments with Benchmark System II was 30 seconds.

The last steps in the UNIFI sequence are to download and store all of the recorded system response and error information in the database. Once all of the data is collected, UNIFI sends a reset to the benchmark system, and the fault injection process for the trial is completed. After the sequence is completed, UNIFI initializes all fault injection parameters to begin the next fault injection experiment sequence.

8.3. Experiment Definition

A number of fault injection experiments were executed on Benchmark System II that represented the types of fault injection tests that typically would be conducted by an assessment organization or the digital I&C equipment vendor during the course of a V&V activity. These experiments were chosen to provide a basis for determining the utility of the methodology to support system safety assessment activities such as license reviews, failure modes and effects analyses (FMEA), and PRA activities. The quantitative results presented in this report only reflect the ability to obtain such information. Table 8-1 summarizes the types of the fault injection experiments conducted on Benchmark System II. The fault injection

experiment details and results for each of these types of fault injection experiments are presented in the following sections.

Table 8-1 Types of fault injection experiments conducted on Benchmark System II.

Experiment	Purpose	Type of fault Injection	Fault model used	Fault Injection reference time point	Applied to Module " "	In support of verifying
1 - CPU register Fault coverage	Provide data to quantify the error detection coverage of register faults	FPGA-based fault injection BDM	Transient and permanent	After LOCA	Channel A main processing module	Self-tests and error detection mechanisms for processor-based faults
2 - Memory Fault coverage	Provide data to quantify the error detection coverage of memory faults	FPGA-based fault injection BDM	Transient and permanent	After LOCA	Channel A main processing module	Self-tests and error detection mechanisms for processor-based faults
3 – Function and Function Block	Provide data to quantify the error detection coverage wrt functions and function blocks	FPGA-based fault injection BDM	Transient Single and multi-bit flips	After LOCA	Channel A main processing module	Self-tests and error detection mechanisms for processor-based faults
4 - Digital Output Trip function timing response (faulted and Non-faulted)	Provide data that the digital output signals are actuated in a timely manner	FPGA-based fault Injection BDM	Transient	After LOCA	Channel C and Channel D main processing module	Actuation timing
5- Fault and error Latency analysis	Provide data to quantify the error detection latency.	FPGA-based fault Injection BDM	Transient	After LOCA	Channel A main processing module	Error detection and Fault masking, self-tests for processor-based faults
6- Double fault injection	Determine the response of the System to two faults	FPGA fault injector BDM 1 and BDM 2	Transient	After LOCA	Channel A and Channel B main processing module	Error detection and Fault masking, self-tests for processor-based faults

8.4. Processor-based Fault Injection Experiments

8.4.1. CPU register corruptions

Fault injections experiments were conducted for various registers of the MPC860 processor of Benchmark System II. These fault injections were transient single-bit and multiple-bit corruptions injected into one of the 32-bit registers of the 32 register files in the processor. Table 8-2 shows the details of the registers used in the fault injection experiments.

The fault injection experiments were conducted in the following manner. The selection of the register to be corrupted was randomly picked. This accounts for some of the variation seen in the fault injection totals for each register. The location of the fault in the register was selected randomly as well. After these random selections were made, the selections were written to a file so the results of the fault injection could be traced. All register fault injections were of the breakpoint read-modify-write format. This ensured a high percentage of faults resulted in error corruptions. Very few no-response faults were encountered. A small number of register-based fault injections were conducted in the investigation, as most fault injection experiments were focused on the application level and operating system level.

Note that all CPU register corruptions that were not of the “no-response” class were detected properly. The criterion for “error detected or not detected” was that a valid error log was produced for the fault injection experiment (see Figure 8-2). No-response experiments were those experiments that did not produce an error log and did not produce any other observable errors such as output deviations.

Table 8-2 Details of registers used in fault injection experiments.

Register type	Single-bit fault transient	Multiple-bit fault transient	Bit location in register	No-response	Effectively detected	Not detected	% Total detected
R0	37	24	Random	0	61	0	100
R1	13	24	“ ..”	0	34	0	100
R2	10	34	“ ..”	3	44	0	100
R3	22	18	“ ..”	1	40	0	100
R4-12	34	12	“ ..”	0	46	0	100
R13-31	101	34	“ ..”	0	135	0	100
lr	6	3	“ ..”	0	9	0	100
Total	223	149	“ ..”	4	369	0	100

8.4.2. Memory-Based Fault Injection

The Memory-based fault coverage experiments and the function and function block fault injection experiments were performed to determine whether the Benchmark System II with one faulty channel would fail to engage a reactor trip during a loss of coolant accident in the reactor. There were 110 different locations containing global variables in memory space targeted for fault corruptions. These 110 fault locations were corrupted with multiple random bit flips. That is, the fault mask in the fault list parameter file contained at least two bit locations that were used to “flip” bits in the memory cell (e.g., change a bit from 0 to 1). Each campaign was conducted with a fault list of varying size, most often each fault campaign was set to 200 faults, which was required approximately 17 hours of runtime. The campaigns were repeated and run at slightly different fault injection times to expose Benchmark System II to different fault

activation intervals. Approximately 1200 faults were injected for this set of experiments on Benchmark System II. No uncovered faults were revealed during the course of this set of experiments.

8.5. Function Block-Oriented Fault Injection

Many digital I&C systems use a function block software programming environment that employs libraries of pre-determined function blocks to design a particular application. The developer builds the application from these functions using an ensemble of connected function blocks. Regulators, designers, and safety reviewers understand the system from this point of view, so it is important to provide the capability to guide fault injection campaigns from this perspective. Function block-based fault list generation extends the capabilities of map file fault list generation by identifying high level function block code and data segments in the locatable image files.

Referring to Figure 8-2, to construct fault lists for functions and function blocks it is important to have two pieces of information. The first is a low level description of how the function is composed structurally in the data and code segments of the image file. The second piece of information is where the instances of the function block code are located.

A detailed map file may contain a low level description of how a function is composed structurally in the data and code segments of the image file in the form of a data structure. This information may change from compiler to compiler, but it is always contained somewhere in the detailed linker listing files, such as map files, sym files, OLM files, etc. Figure 8-2, shows the results of parsing the map files to find the instances of the function blocks and the results of parsing the map linker file to find the type structure of the function blocks. Once both files are parsed, the files are unrolled and merged to produce a fault list. The fault list at the bottom is for one instance of the function block **TR_Critical_IO.**

Target specific function blocks in the safety function or supporting functions.

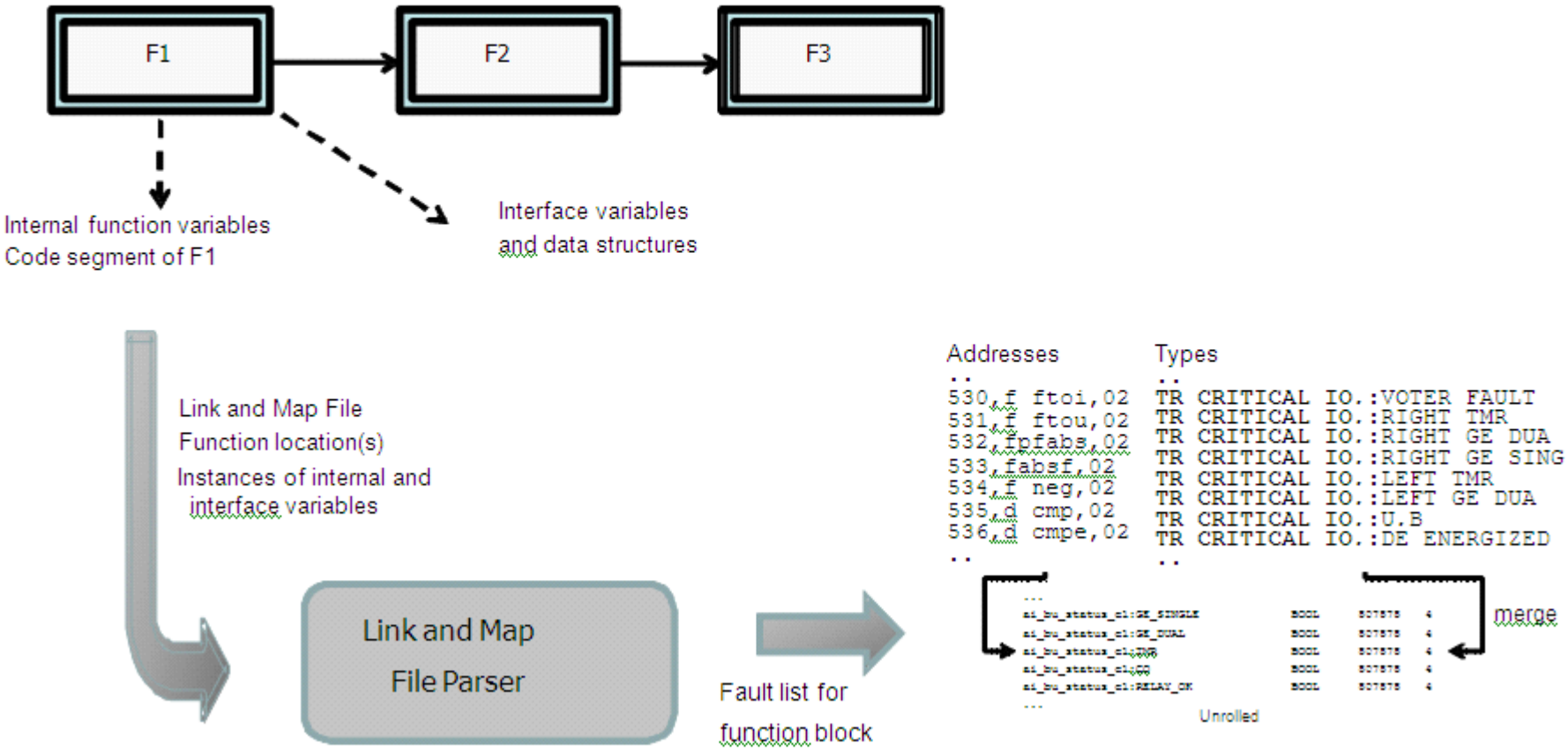

Figure 8-2 Function block fault list generation

Using the fault list generation tools developed and described in Section 6, these tools were extended to include function block fault list generation for the application code area, and to function in the real time operating system. The Benchmark System II OS has over 100 functions used in various support tasks for scheduling, task management diagnostics, error detection, etc. The UNIFI GUI allows the user to select any of the functions and global variables in the operating system. Functions were selected that would be of interest to a verification effort. These were functions associated with dispatching, fault tolerant data management, and synchronization. On the application side, all of the functions in the RPS block diagram were selected for fault injection. Table 8-3 shows some of the functions and function blocks that were targeted for fault injection.

Table 8-3 List of sample functions and function blocks.

Function Block	Description	Location Application	Location Operating System
AIN	Analog input processing	x	
OR	Or function	x	
LT	Less than	x	
GT	Greater than	x	
TR_URCV	Peer-to peer Communication Receiver	x	
TR_USEND	Peer-to peer Communication Send	x	
TR_MP_Status	Main Processor Status	x	
TR_Vote_mode	Vote Status	x	
comprocess	OS function		x
heap	OS function		x
tribus	OS function		x
voteint	OS function		x

Both global and local variables are associated with each of the above function blocks and functions. The global variables are mainly interface variables to the function blocks. Local variables typically are associated with the internal code of the function block. Once a function block is selected by the fault list generator, its variables are revealed to the user. The user can select all of the variables or select specific variables of interest. For example, the global variable `or_out_ch_b_c2` isssociated with the first OR gate of the RPS of channel B in chassis 2. This variable is located at x`00802170`.

The experiments for function block fault injection were divided into two major groups: RPS-related function blocks and OS-related functions. Experiments were further divided into transient and permanent faults for each group.

In an exploratory effort to see how the fault list generation methods could support injection of software-based faults, software-oriented faults were injected into OS functions. These fault injection experiments included incorrect reference calls, wrong return values, and misaligned

pointers. The results of the software fault injection experiments were encouraging. The researchers were able to identify the data structures of interest from the parsed map and .NCS files, and then emulate the software faults. However, the goal of this effort was not to conduct research in software-based faults, so this research was not pursue any further. Table 8-5 presents the distribution of function blocks targeted for fault injection.

Table 8-4 Distribution of function blocks targeted for fault injection.

Experiment group	Number of function blocks (FB) targeted	Number of Channel A target FBs	Number of Channel B, C, D target FBs	Fault model type transient ?	Fault model type permanent ?	Fault location	Fault occurrence
RPS	80 (all)	19	19	Yes	No	Chassis 1 MPU A	Single fault in MPU
OS	22 of ~100	22 of ~100	0	Yes (major)	Yes (minor)	Chassis 1 MPU A	Chassis 1 MPU A

The RPS function block experiments were executed first and required approximately 3 months over a 6-month period to complete. The OS fault injection experiments were then executed, requiring about 2 months to complete. The reason for spreading the experiments out over time was to allow students to complete work on other projects.

The statistical experiment designed for the function block fault injection process consisted of partitioning the sample space into two groups: the RPS function block fault space defined with respect to RPS function blocks and their associated data structures; and the OS function fault space defined with respect to OS functions and their associated data structures.

A function block was selected non-randomly for each experiment to ensure that all function blocks were selected fairly over the course of the campaign. The data structures associated with the selected function block were then selected at random. For all experiments, the fault mask was a multi-bit flip mask.

Experiments were replicated a number of times to gather multiple data responses for each function block. The measurement instruments for recording the error response data and the output response data were enabled 1 second before the fault injection. The fault was injected at the LOCA initiation time in the OP (12 seconds after the start of the OP data). The measurement recorders sampled the response data at 1 ms intervals (50 times the sampling rate of the target system). The coverage estimates for these experiments are specific to the sample spaces of the function blocks. Table 8-6 presents the results of the fault injection experiments.

Figure 8-3 and Figure 8-4 show the total fault injection response data as proportions. The notable aspect of these graphs is low percentage of no response faults due to using the break-point fault injection feature of the FPGA-based fault injection module for a large percentage of the experiments.

Table 8-5 Function block experiment characterization and results.

Sample space	Function blocks	Fault type (single)	Number of experiments	Number of replicated experiments	Effectively detected	No-response	Crash faults	% Total detected
RPS	RPS Processing	Transient	6505	~5/function block	6159	346	12	100%
OS	Dispatch, Data management System services	Transient	2250	~3/function block	1931	319	17	100%
OS	**“..”**	Permanent	128	None	128	0	0	100%
Totals			8883		8218	665	29	

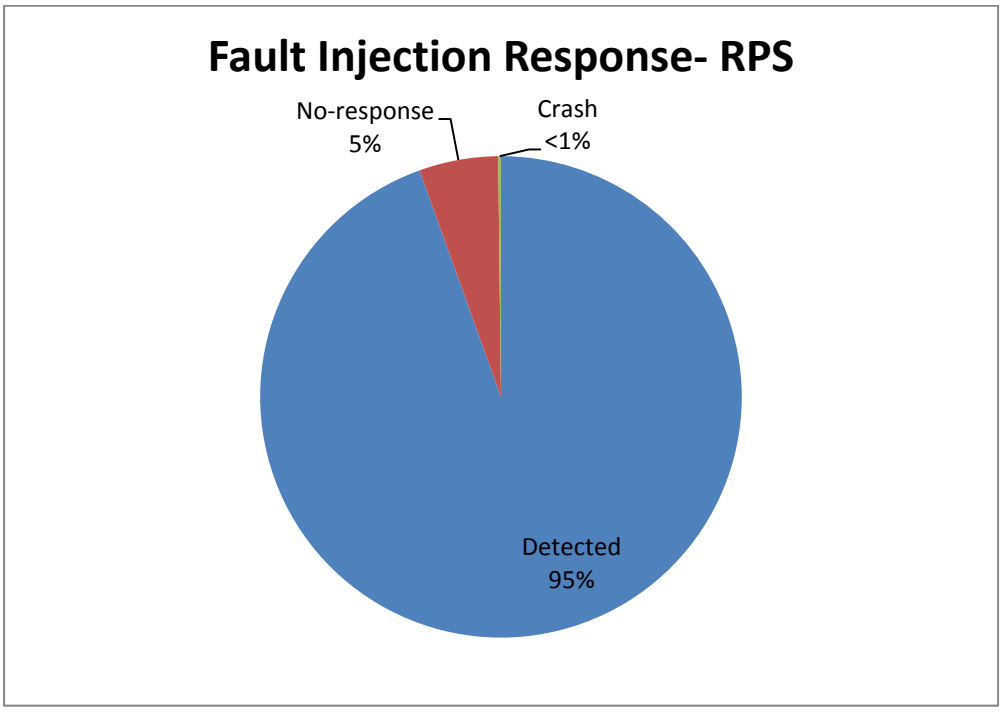

Figure 8-3 Raw fault injection response data - RPS

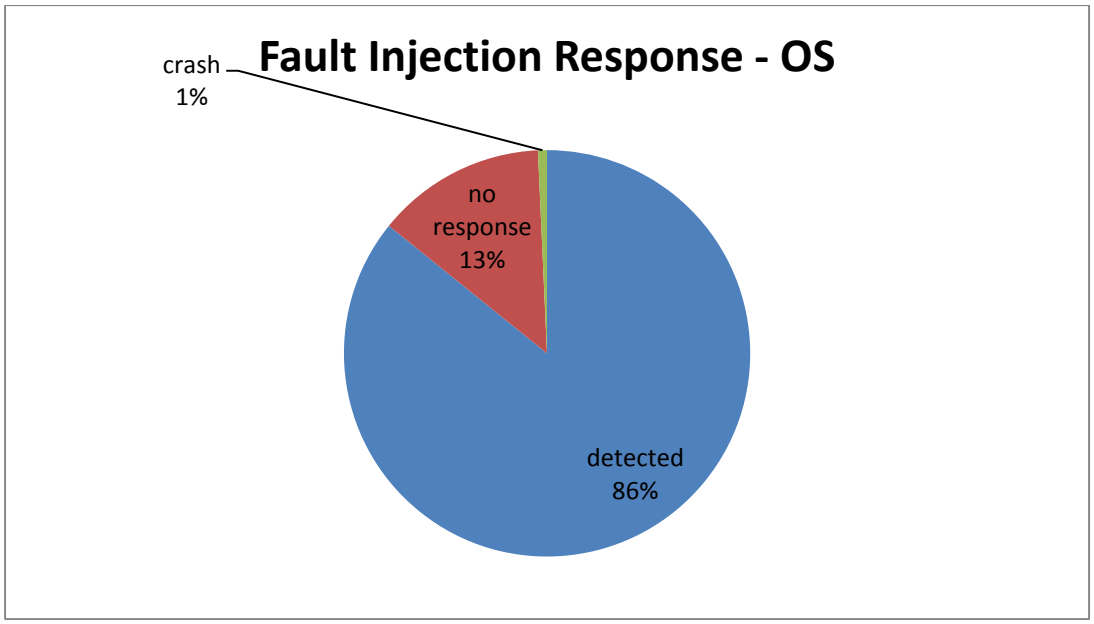

Figure 8-4 Raw fault injection response data - OS

Table 8-6 shows the overall results of the 8883 fault injection experiments on the Benchmark System II RPS and OS function blocks. Of the 8883 fault injection experiments, 665 experiments resulted in a no-response result. That is, the system error logs did not indicate that the fault had an effect on the system behavior. A no-response experiment result could be the result on an error response being overwritten, as pre-fault injection analysis was not used in these experiments. The other possibility is that the fault could have had a longer latency period than the censor monitoring time.

Additionally, 29 injected faults resulted in a “crash state” of the processor. That is, the processor failed to reboot properly after reset. These events required that the processor board be de-energized by extracted the board from the chassis and then re-seatin the board back in the chassis. Most of these crash faults appeared to be caused by the onboard health monitoring diagnostics. The benchmark system diagnostics keep track of the number of faults experienced. If the number of faults exceeds the fault table threshold, the system will not reboot, thereby requiring the board to be removed from the chassis to reset the number of faults counter.

All detected faults were effectively handled, meaning (1) the system output responses were equivalent to the no fault response and (2) the error logs indicated a proper detection and mitigation of the fault via fault masking and re-synchronization.

The coverage of the system error detection mechanism with respect to faults occurring in function blocks calculated using the fault response data shown in Table 8-6. Two statistical coverage estimation models apply to this type of experiment; the partitioned Bernoulli model and the Fault Equivalence model. The partitioned Bernoulli model was used because of the simplicity of the model.

8.6. Coverage Estimation

The partitions for this application were the OS and the RPS sample space. In each case, the Bernoulli model [Pescodido 2002] was applied and coverage was calculated for each sample

space. As with the register fault coverage estimation, the method of variance reduction was used to obtain a conservative point estimate of C.

The Bernoulli model statistic provides a system view of the fault tolerance capabilities of the benchmark system with respect to function block failures. From Table 8-6 it can be seen that 346 experiments in the RPS data domain generated no response and hence were discarded. Normally, these experiments would be rerun to determine whether the experiments were truly no-response or long latency as discussed in Section 8.10. Likewise, 319 faults in the OS domain were no-response and were not included in the estimation.

Table 8-6 Coverage estimates for Benchmark System II functions.

System Function	Number of fault injections	No response faults	Lower bound estimate of C using Eqn 8.1	Lower bound estimate of C using Eqn 8.2
RPS System Safety Function (see Table 8-4)	6159	346	0.999682	0.999514
Operating System Functions (see Table 8-5)	1931	319	0.998985	0.998450
CPU Register Faults (main processor)	369	4	0.994695	0.991915
Global Memory Faults (RPS)	1205	103	0.998337	0.997517

Estimation of coverage can be hindered by the lack of non-covered failures (the number of non-covered failures is zero). In such a case, the estimated mean coverage would be 1.0 (100% coverage), which may not be representative of the likely true coverage state because of various reasons such as imperfect error detection mechanisms, defects in redundancy management software, and incomplete testing. Thus, a value less than 1.0 is used in conservative dependability modeling. Estimating the coverage when testing reveals no failures or uncovered faults is possible using statistical methods such as [Kaufman 2002, Smith 1997, Tang 1997, Miller 1992].

In Gaussian distribution terms, a confidence interval cannot be calculated if the measured coverage distribution has zero estimated variance. One method for overcoming this limitation is to derive a bound on the variance estimate that is more conservative than the original variance estimate. The variance can be estimated for the all-covered case by converting one covered fault injection experiment to an uncovered fault injection experiment. This reduction in the value of the point estimate is necessary to ensure that the estimated variance is never equal to zero. This approximation results in a larger variance estimate and is therefore conservative in nature. The conservative estimate is always equivalent to having a variance estimate based on one uncovered fault injection experiment. This is a conservative assumption since the variance for the point estimates is actually less than the calculated value.

Using the method proposed in [Smith 1997a], the following conservative estimation of coverage can be used,

$$C_{cons} = 1 - z_{\gamma}\left[\frac{n-1}{n^3}\right]^{\frac{1}{2}} \quad (8.1)$$

where z_{γ} indicates the 100% standard normal percentile and *n* is number of covered fault observations.

Another method, proposed in Tang 1997 is

$$C_l = \alpha^{\frac{1}{n}} \quad (8.2)$$

where *n* is the total number of faults that were covered (as observed in the test), and α is the desired significance level (1 −α is the confidence level).

The coverage of the benchmark system error detection mechanisms with respect to system function (e.g. RPS, Operating System, CPU registers, etc.) was calculated using the fault response data shown in Table 8-1 through Table 8-6. Two statistical coverage estimation models apply to this type of experiment; the partitioned Bernoulli model, and the Fault Equivalence model. Since, uncovered faults were not observed during this effort, the lower bound conservative approximations were used (Equation 8.1 and Equation 8.2). The results are shown in Table 8-6 above.

The benefits of function block-oriented coverage estimation are that the system coverage factors can be determined specifically for important I&C safety functions. More often than not, coverage factors associated with system resource faults like processor registers and memory or other components are seen. While these coverage factors are important, having coverage factors that relate specifically to safety functions allows a more complete view of a system detection capabilities in the context to its application and environment. This allows one to answer “what if” questions more definitively about failures in the safety functions and the functions of the operating system.

8.7. Digital Output Response Experiments

Experiment 5 tested the capability of the methodology and the UNIFI measurement systems to gather critical timing information about the actuation and disengagement capabilities of the system. Experiment 5 measured the interval time between the initiation of a LOCA event and when the reactor trip alarm signal goes high on the outputs. This time measurement was facilitated by sampling the trip alarm signals and comparing the times with LOCA initiation times. This experiment was conducted in a no-fault scenario and a faulted scenario. The outputs of the benchmark system were sampled at 10ms intervals by the Labview data acquisition system. These types of experiments allow the real-time output response of an I&C function to be measured in a no-fault or faulted case.

Figure 8-5 and Figure 8-6 show typical trip alarm responses for 1 fault injection campaign (~200 fault injections). State changes from 1 to 0 indicate disengagement. The minimum values occur at 350ms; which is approximately the time after the LOCA event is initiated that the first measured sensor values start to cross the RPS set point thresholds. The values above these minimums occur at 50 ms boundaries which is the sampling (scan) rate of the RPS application on the benchmark system. Both charts show the similar trends. Most of the responses measured were around the 400ms time bin indicating that the system responds within 1 cycle time of the event. Some of the responses at 450ms and 500ms could be due to sampling

misalignment. The key point in these experiments is that the quantitative response data allows you to see trends that inform the qualitative/deterministic system arguments.

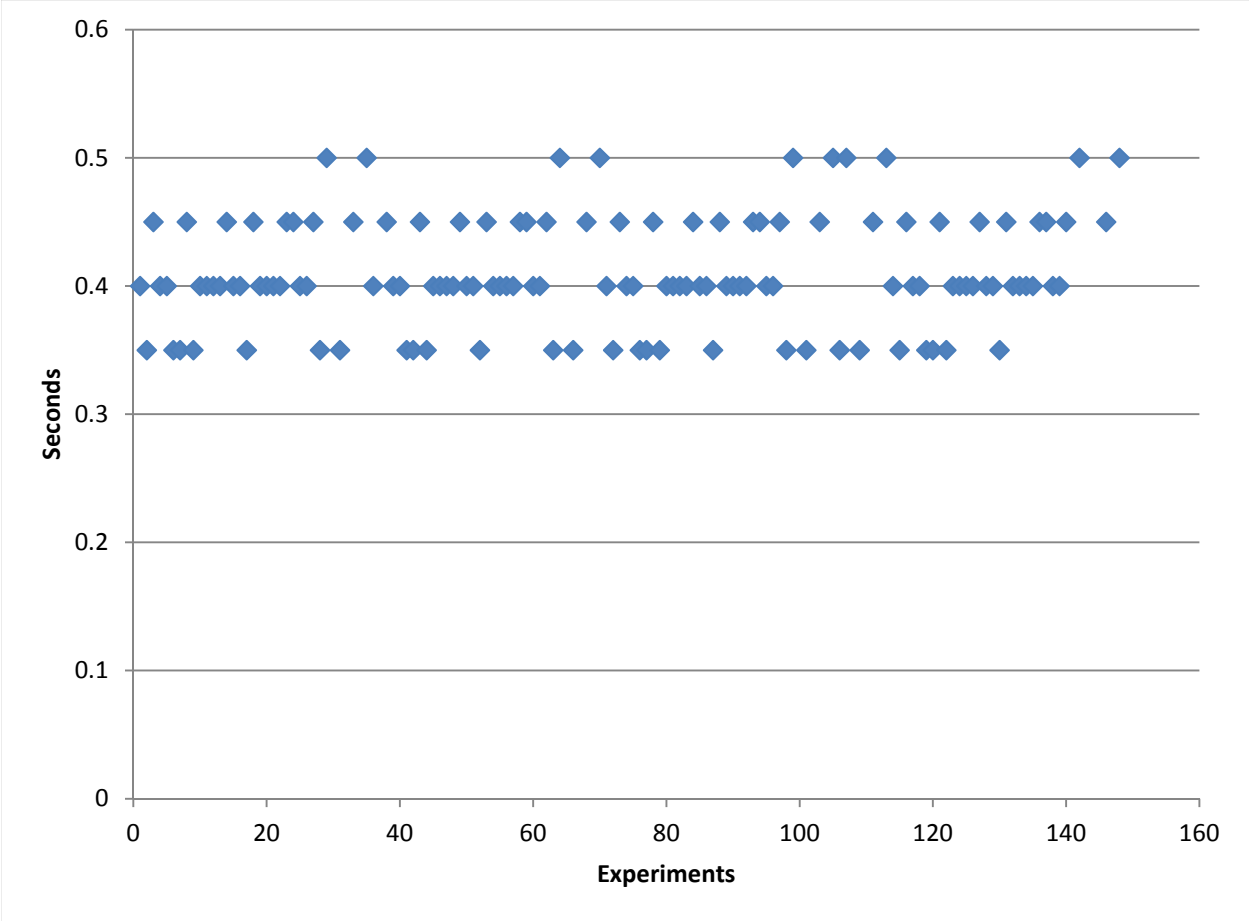

Figure 8-5 Trip alarm response times (faulted)

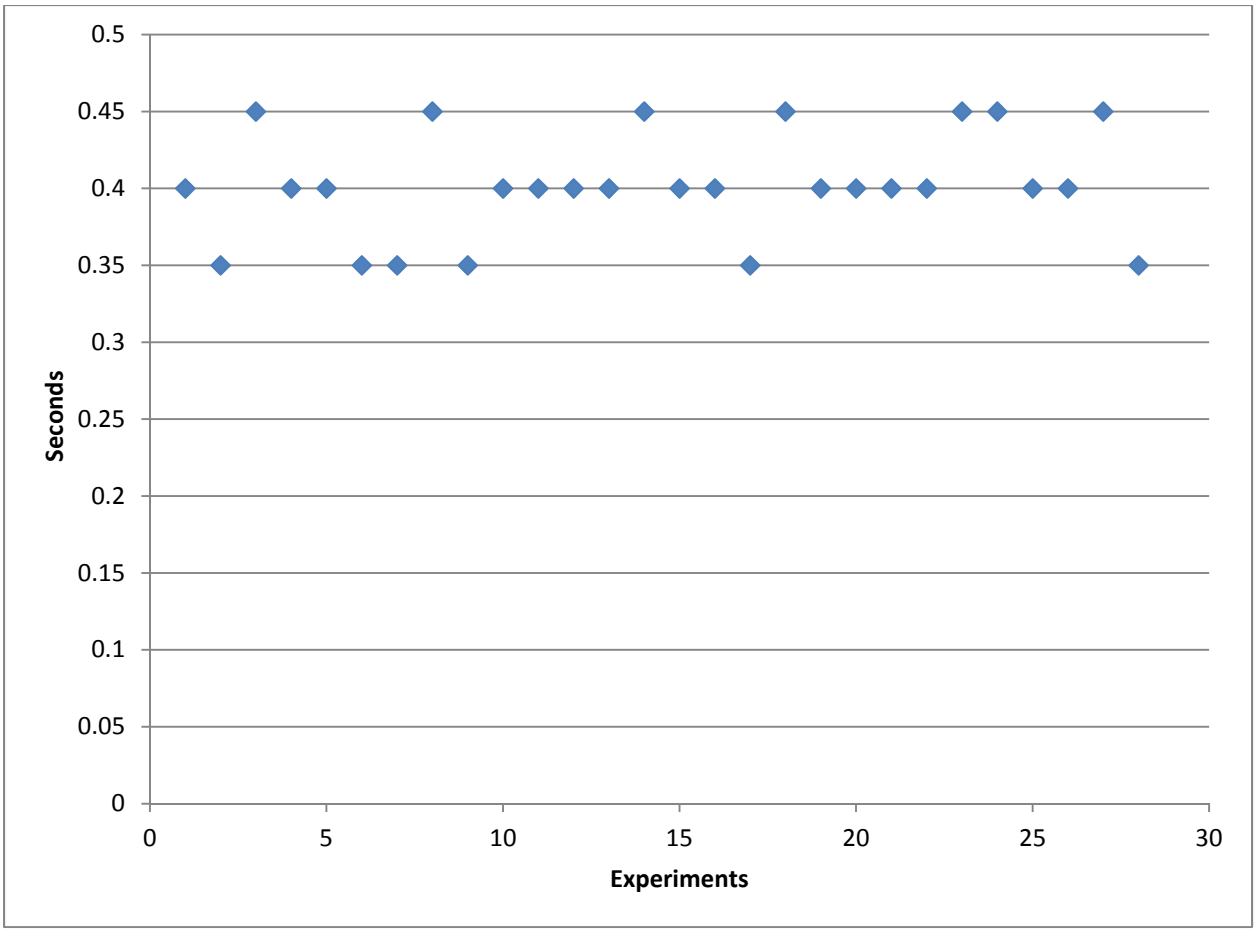

Figure 8-6 Trip alarm response times (non-faulted)

8.8. Fault and Error Latency Analysis

Another important metric used in dependability analysis is error and fault latency. Figure 8-8 shows the concept of fault and error latency. A fault may remain latent until it is activated by use in the program or hardware. Once the fault becomes active it may produce an error that propagates until it is detected by the error detection mechanisms or self-tests of the system. Fault latency is important because the longer an error or fault remains undetected in the system, the probability of the undetected fault colluding with another fault in the system increases with time. The two errors could then manifest in the same time interval requiring the system to handle a double fault situation. Error logs from the system are valuable for determining how the system responded to the fault.

The fault injection experiments allowed the total time to be measured by using the time-stamps of the error messages recorded in the error logs and by sampling of the MP_STATUS function block signal TMR-to-Dual, which was instrumented in the RPS code. In Benchmark System II the diagnostic routines that run in the OS monitor the error detection mechanism outputs and forward these conditions (as error messages) to a monitor program (Trilog) that runs on a host computer external to the benchmark system. Trilog time-stamps these error messages.

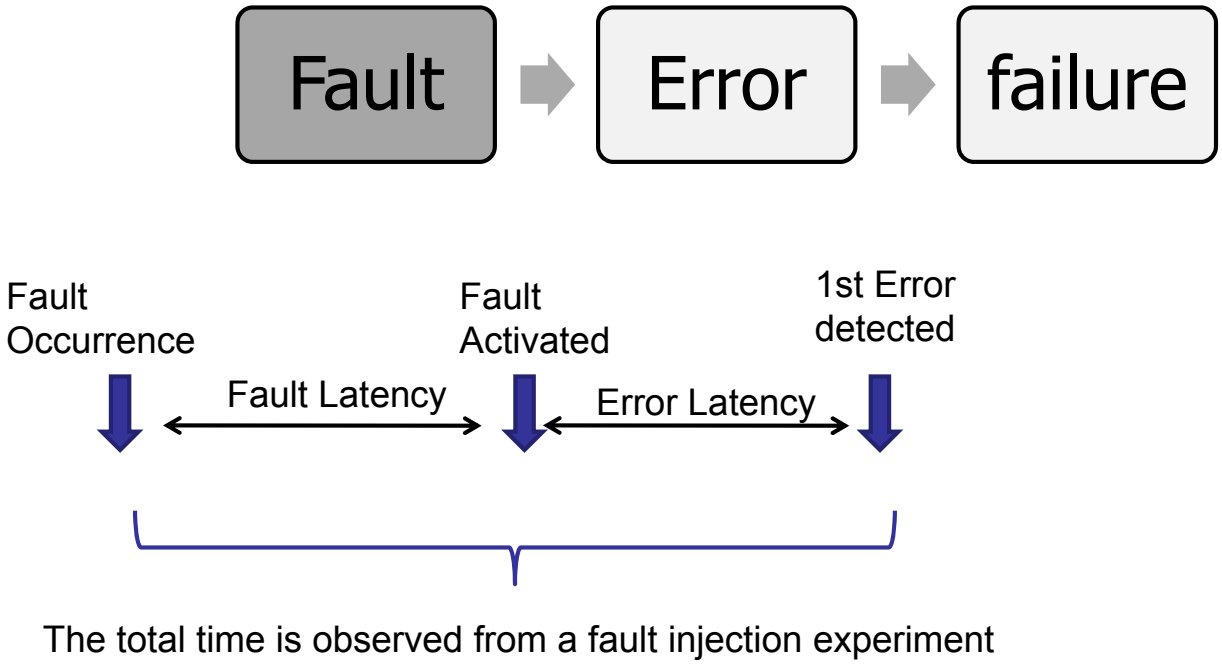

Figure 8-7 Example of fault and error latency

The fault injection experiments allowed the total time to be measured by using the time-stamps of the error messages recorded in the error logs and by sampling of the MP_STATUS function block signal TMR-to-Dual, which was instrumented in the RPS code. In Benchmark System II the diagnostic routines that run in the OS monitor the error detection mechanism outputs and forward these conditions (as error messages) to a monitor program (Trilog) that runs on a host computer external to the benchmark system. Trilog time-stamps these error messages.

In the analysis of fault latency, the timestamps of the Trilog program were not used to calculate fault/error latency because preliminary experiments revealed that the timestamps were not consistent with what was observed in the lab with respect to fault detection times. It was concluded that some form of delay in time stamping was occurring, perhaps due to loading effects on the host computer, transport delay, or both. Nonetheless, the digital output signal that indicates when the TMR degrades to a dual mode was sampled. This measurement provided a conservative estimate of fault/error latency. This was apparent by inspecting a typical error log, such as that shown in Figure 8-8. Comments regarding the error log messages are provided in the italicized bolded text

The error log message reveals that several fault detections occurred before the system signaled **"MP is in the duplex mode".** Also shown in the error log is that the error log timestamps recorded a total time of approximately 8 seconds (13:22:55 – 13.22.47) from the first detection event until the system was in the duplex mode. This amount of time was not observed in the experiment.

9/22/2010 13:22:47.202 ETS A 0x10 3 Us D No cmpr Add(808a1c) Off(1a1c) M(111) US(511) DS(111).
9/22/2010 13:22:47.202 ETS B 0x10 3 Ds D No cmpr Add(80d870) Off(6870) MY(0) US(0) DS(0).
9/22/2010 13:22:47.202 ETS C 0x10 2 My D No cmpr Add(80d870) Off(6870) My(0) US(0), DS(0).

This is the main processor detecting a comparison error between the three MPs. The "Us D No" message has the correct data so in this case MP A had a value of 111, MP B had a value of 111, and MP C had a value of 511.

9/22/2010 13:22:51.102 ETS A 0x10 3 Us D No cmpr Add(808a1c) Off(1a1c) M(111) US(511) DS(111).
9/22/2010 13:22:51.102 ETS B 0x10 3 Ds D No cmpr Add(80d870) Off(6870) MY(0) US(0) DS(0).
9/22/2010 13:22:51.102 ETS C 0x10 2 My D No cmpr Add(80d870) Off(6870) My(0) US(0), DS(0).

This is the second detection of a comparison error.

9/22/2010 13:22:55.002 ETS A 0x10 3 Us D No cmpr Add(808a1c) Off(1a1c) M(111) US(511) DS(111).
9/22/2010 13:22:55.002 ETS B 0x10 3 Ds D No cmpr Add(80d870) Off(6870) MY(0) US(0) DS(0).
9/22/2010 13:22:55.002 ETS C 0x10 1 A MP reset for 27, P1=807000 P2=1a1c P3=1a1c P4=ffffffff.

This is the third detection and the reset of MP C. The 27 indicates that the data area miscompared between the main processors.

9/22/2010 13:22:55.097 ETS A 0x10 3 US BD(0) Invalid Sequence Number Flags(80) seq(ffffffff) CRC(0). TXCR(980) USCR(4100) DSCR(980)
9/22/2010 13:22:55.097 ETS A 0x10 3 US BD(fc) Missing Block Flags(92) seq(ffffffff) CRC(0). TXCR(980) USCR(4100) DSCR(980)
9/22/2010 13:22:55.097 ETS B 0x10 3 DS BD(0) Invalid Sequence Number Flags(80) seq(ffffffff) CRC(0). TXCR(980) USCR(980) DSCR(4100)
9/22/2010 13:22:55.097 ETS B 0x10 3 DS BD(fc) Missing Block Flags(92) seq(ffffffff) CRC(0). TXCR(980) USCR(980) DSCR(4100)
9/22/2010 13:22:55.099 ETS A 0x10 3 US BD(3) Invalid Sequence Number Flags(90) seq(ffffffff) CRC(0). TXCR(980) USCR(4100) DSCR(4980)
9/22/2010 13:22:55.100 ETS B 0x10 3 DS BD(3) Invalid Sequence Number Flags(90) seq(ffffffff) CRC(0). TXCR(980) USCR(980) DSCR(4100)

This is MP A and B detecting that MP C did not rendezvous with A and B.

9/22/2010 13:22:55.099 ETS A 0x10 4 Voting changed from 7 to 3.
9/22/2010 13:22:55.099 ETS B 0x10 4 Voting changed from 7 to 6.

This indicates that the voting changed from triple to dual.

9/22/2010 13:22:55.100 ETS A 0x10 3 **MP is in the duplex mode.**
9/22/2010 13:22:55.101 ETS B 0x10 3 **MP is in the duplex mode**.

Figure 8-8 Error log transcript from the TRILOG server

Figure 8-9 and Figure 8-10 present the data that was used to analyze the fault/error latency of the injected faults in memory as detected by the Benchmark System II error detection mechanisms. These fault latency measurements were obtained from the function block data set. For each fault injection that resulted in a detected fault, the time of the first TMR-to-Dual response was noted.

Figure 8-9 shows a typical response of five consecutive fault injection experiments. The bar graph for each fault injection indicates the fault/error latency interval time between the fault injection time and the assertion of the TMR to Dual signal. The results are well within the bounds of acceptable response for a real-time detection event. The first measured latency was

37ms, the second is 38ms and so on. These fault/error latencies are conservative for the reasons discussed above.

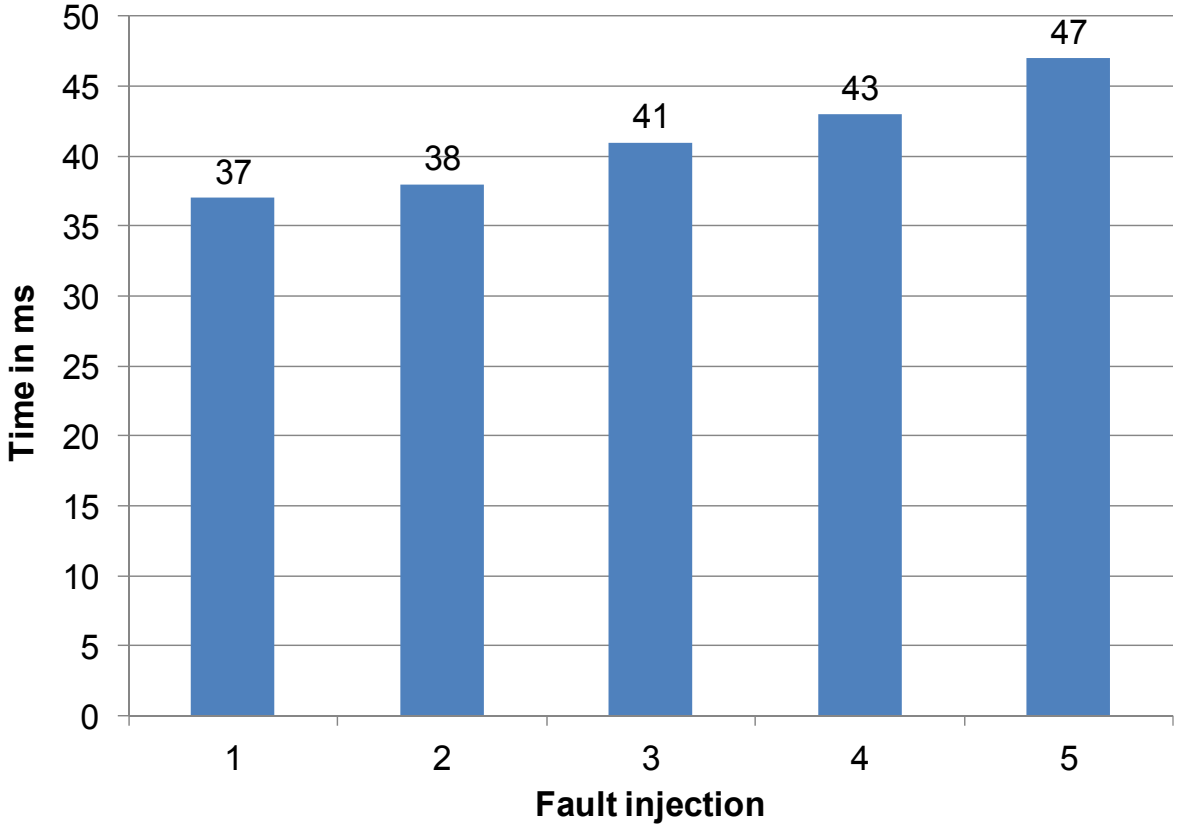

Figure 8-9 Fault/Error latency interval time

Figure 8-10 shows the frequency of occurrence (an empirical distribution) of the fault latencies for one complete fault injection campaign (several hundred fault injections). The first bin is all recorded responses that are between 0 and 10ms. The second bin is all responses that are between 10 and 20ms. Each bin is an interval 10ms wide. Most of the fault latencies occurred between 30ms and 50ms. This trend was consistent from one fault injection campaign to the next.

Figure 8-11 shows the fault latency frequency of occurrence (empirical distribution) for the fault injection campaigns. The larger data set confirms the intuition from Figure 8-10 that the mean fault latency for these experiments is conservatively around 30ms. Again this type of quantitative data helps inform the analysis and assessment of the fault tolerance and fault detection mechanisms of digital I&C systems. Furthermore, measured fault latency parameters are crucial for reliability and safety modeling efforts. These parameters typically are used in models to determine the impact of fault detection latency on system reliability and safety.

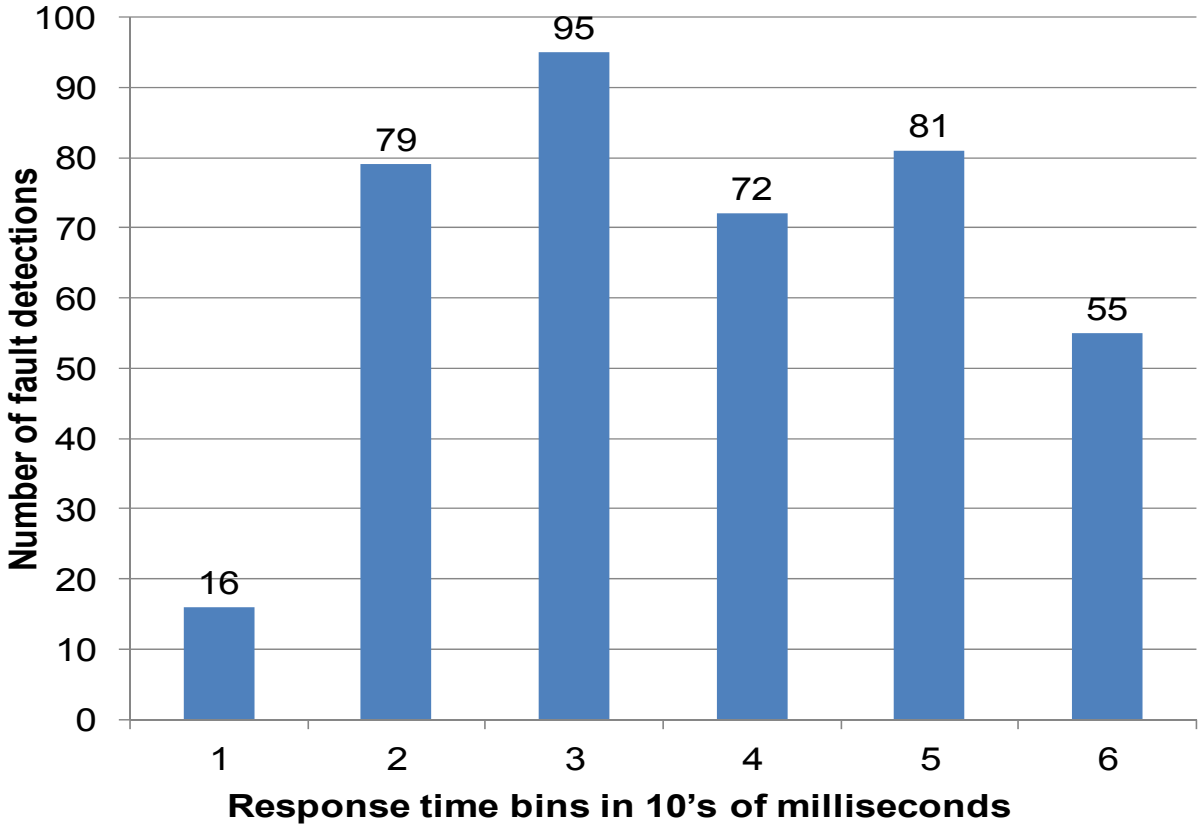

Figure 8-10 Frequency of occurrence of system fault detection responses

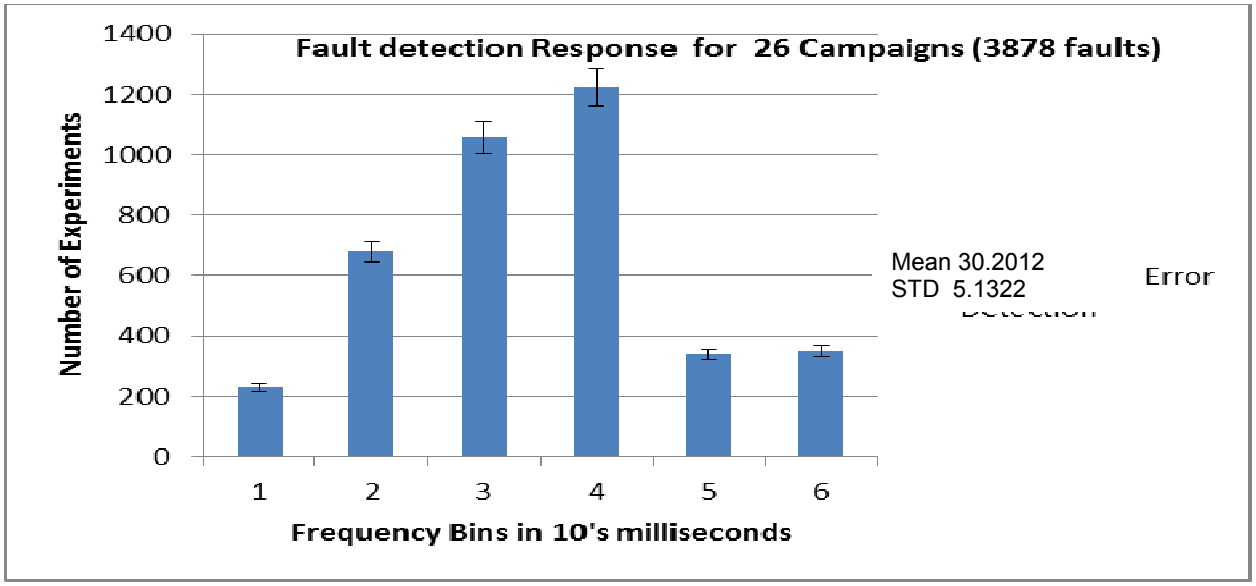

Figure 8-11 Frequency of occurrence of fault latency over set of fault injection campaigns

8.9. Multiple Fault Injection Studies

A small number of multiple fault injections experiments were conducted in the latter phases of the research on Benchmark System II to determine the efficacy of using the FPGA-based HiPeFI module for these types of studies. In these studies, a fault was injected in processor A of chassis 1, and then 100ms later a second fault was injected into processor B of chassis 2. Because three channels of the RPS (channel B, channel C, and channel D) were emulated on chassis 2, a single fault could affect all three channels simultaneously. Keeping this in mind it was realized the results of such experiments were not valid as they did not represent a real RPS physical division separation. The key result from these experiments was confirming the testing the capabilities of the fault injector to support multiple independent fault injections. Several small campaigns on the order of 20 fault injections per campaign were conducted to determine whether the multiple fault injection features worked as intended. All of the multiple fault injections worked as expected. The time to the second fault injection was very close to 100ms for each fault injection. As a side note, the benchmark system tolerated every second fault injection.

8.10. Addressing No-response Faults

In the fault injection experimentation the research did not discern between no-response faults and long latency faults due to time constraints on the project. However, to adhere to NRC single failure criteria testability, no-response faults must be distinguished from long latency faults or hidden faults. The pre-injection analysis presented in Section 7 of Volume II is one means to identify no-response faults and distinguish them from long latency faults. The following process is a suggested method to use the methodology to effectively deal with no-response faults.

(1) Classify faults that result in a "no-response" as "further experiments needed".

(2) Group these No-response faults into classes according to their function - safety block functions, registers, special purpose IC registers, OS, etc.

(3) Conducted Pre-injection analysis to ensure the fault is in a part of the system that has executable code and data.

(4) Re-execute each no-response fault according to the following process:

    (1) Repeat the fault injection with the same fault injection parameters - extend observation time by increments of 5X.

    (2) For all faults detected thru extended monitoring, identify the latency duration. Determine whether these long latencies impact reliability by entering them back into the PRA models. If so, then the faults with reliability impact should be reassessed.

    (3) If no response is observed after three successive fault inection experiments with increasing monitoring, with the last fault injection monitoring period 30 minutes, then classify the fault as "very long latency".

    (4) Classify all "very long latency" faults as potentially dangerous faults.

(5) Alter the fault injection parameters of the very long latency faults (i.e., time, duration, value) parametrically to determine the system sensitivity to these parameters.

(6) Execute different input profiles (trace events and data) to determine sensitivity to input and event space.

If faults still exist that produce no response, the analyst should try to identif why the fault is latent/non-responsive. This may entail the use of dynamic and static analysis of the code to determine why the fault is not detectable. If the fault is in an area of the system code, data, or parameter space then, by definition, it is a latent fault. Classify these faults as very long latency, undetected faults that need complete analysis to determine why they are not detected. The Vendor and regulator should completely analyze these faults, and the corrective plan of action. Finally, the severity of all fault Injections should be categorized by their observed responses by both latency, detection effectiveness, and the system time response.

8.11. Application of the Fault Injection Data to Benchmark System II Safety Models

This section describes how the fault coverage parameters generated in the previous sections can be used to instantiate the safety and reliability models that may be used in a PRA process. The models presented in this section are representative safety models of the benchmark system, but are simplified to illustrate the concepts and protect proprietary information.

As described in section 3, Benchmark System II consists of three replicated main processor modules whose outputs are compared using a voter. The system produces the correct output as long as two of the three replicated main processor modules produce the correct output, and thus only one of the replicated processor modules has failed. Each replicated processor module is characterized by a localized fault detection and mitigation capability (that is, fault diagnostic coverage) defined as C_p. In addition, the output module (majority voter) has a fault detection and mitigation capability C_o, which may or may not be equal to C_p. The input processing module similarly has a fault detection capability defined as C_i. Figure 3-2 (Benchmark System II architecture) is repeated in Figure 8-12 for reference.

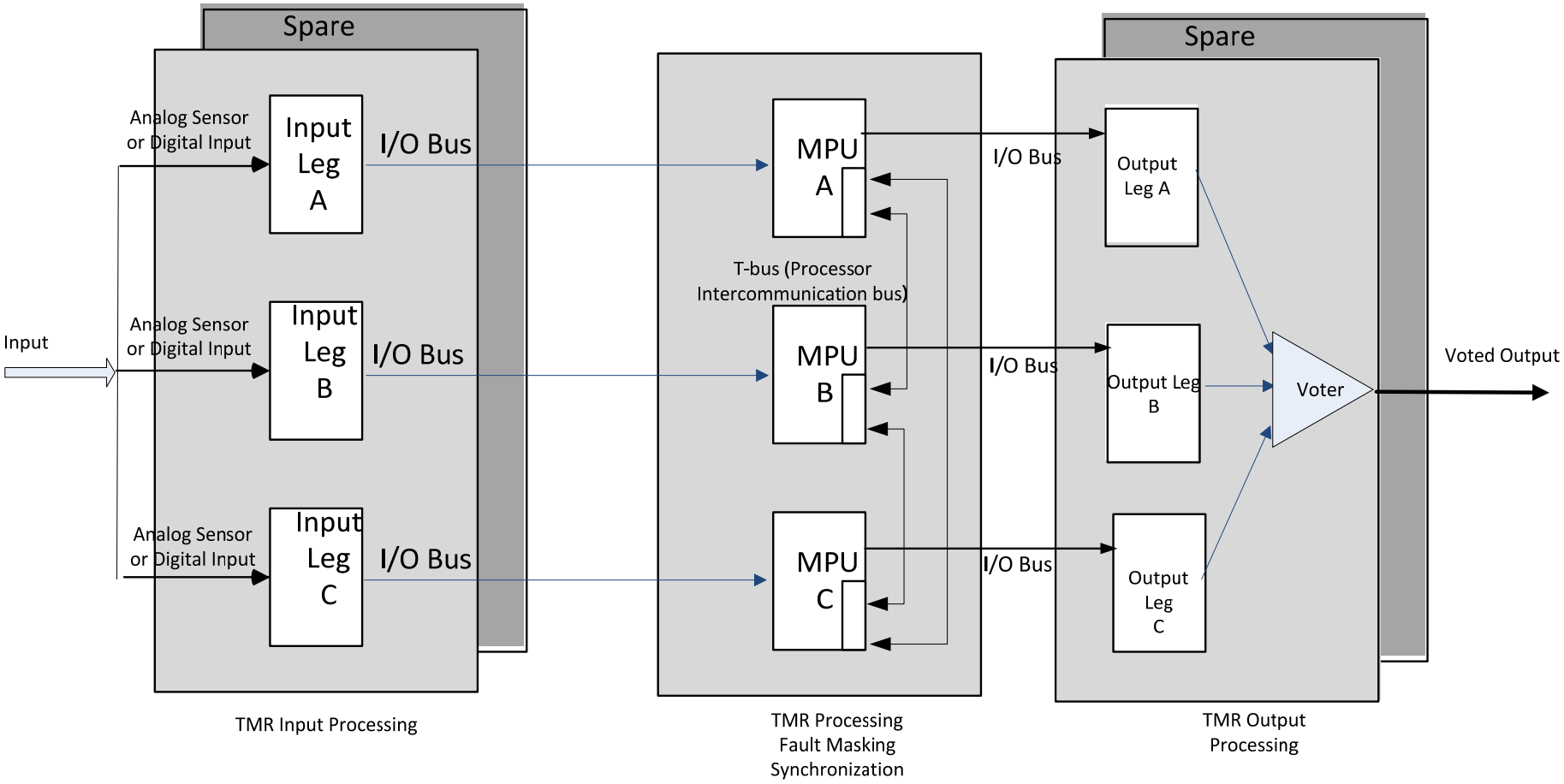

Figure 8-12 Benchmark System II detailed architecture

Metrics are commonly used as measures of system performance for a given attribute. Typically, metrics are computed based on an analytical model that describes the behavior of a system as a function of parameters associated with these attributes. In the most general sense, this analytical model contains the notion of *state*, which when combined with the inputs to a system provides a means to uniquely identify the system at any time, *t*. In general, there two classes of safety and reliability metrics used in safety engineering.

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

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

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

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

- Mean Time to Unsafe Failure, *MTTUF*
- Mean Time To Failure, *MTTF*
- Mean Time To Repair, *MTTR*

Steady-state safety S_{ss} represents the evaluation of the system safety as a function of time, in the limiting case as time approaches infinity. Thus, S_{ss} represents the probability that a given system will operate safely after it is placed in service. Typically, S_{ss} represents the minimum safety for a safety-critical system, and as such provides a lower bound for the system safety under most conditions.

The MTTUF represents the average or mean time that a system will operate safely before a failure occurs and produces an unsafe system state. This metric complements S_{ss} because it provides a quantitative measure of the average period of time that a given system will operate safely, given that the probability of safe operation is characterized by S_{ss}. In [Smith2005], the metrics MTTUF and S_{ss} for were developed safety critical systems.

The *PFD(t)* metric represents the probability that the system has failed in an dangerous manner at a given instant of time. This metric represents the probability that the system has failed dangerously when it is needed to respond to a demand [6]. This metric is particularly useful for quantifying the safety of protection systems (e.g. a RPS), which perform safety-critical interventions in the event of a system failure to bring the system to a safe state.

The *PFDss* probability metric is the evaluation of PFD as a function of time, in the limiting case as time approaches infinity.

Reliability and safety engineers may use one or all of the above metrics to characterize system safety. In this example the focus is on three metrics, *S(t)*, S_{ss}, and MTTUF, to illustrate the use of coverage parameters in safety models.

8.11.1. Markov Model of Benchmark System II

Markov processes are frequently employed to characterize the dynamic failure behaviors of fault tolerant digital systems [BUTLER96]. A *Markov chain* is a mathematical process that transitions

from one state to another between a finite number of possible states [KARLIN 1990]. It is a random process usually characterized as memoryless, that is, the next state depends only on the current state and not on the sequence of events that preceded it. For modeling digital systems, Markov chains with discrete space and continuous time are the most suitable choice. Here the states of the model are discrete representing the various modes of operation, failure, and partial failure that occur. The transitions between states are characterized as not discrete and may take on any real value governed by an underlying distribution.

The first step in modeling a system with a discrete-space and continuous-time Markov model is to represent the state of the system with a vector of attributes that change over time. These attributes typically are system characteristics such as the number of working processors, the number of spare units, or the number of faulty units that have not been removed. The more attributes included in the model, the more complex the model will be. Thus, the smallest set of attributes that can accurately describe the fault-related behavior of the system is typically chosen.

The next step in the modeling process is to characterize the transition time from one state to another. Because this transition time is rarely deterministic, transition times are described with a probability distribution.

Figure 8-13 illustrates a Markov safety model of the benchmark system. In this model, there are 4 states representing the various important modes of operation and failures of the system. This model is typically referred to as a *partitioned failure space model*, because the failure space is partitioned between failsafe and fail unsafe states. The terms failsafe and fail unsafe are specific to a conservative assumption about the detection of failures in the system. That is, failures that are detected and mitigated properly by the system are assumed to be safe. Failures that are undetected or improperly handled by the system are assumed to be unsafe. It is usually not known ahead of time whether a certain undetected or improperly handled failure will result in safety incident, so it is conservatively assumed that a failure could lead to an unsafe state.

The failure transitions from one state to another are governed by a hazard rate function or failure rate function (often denoted as $\lambda(t)$ of the sub-systems in the benchmark system). In this case, hardware failure rates are used that represent the manufacturers data on failures. The C_{xx} terms are the coverage parameters, which characterize the fault detection and mitigation capabilities of the system. These are one the main parameters that are estimated from the fault injection campaign data detailed previously in section 8.5 and section 8.6. It is important to note that the fault injection campaigns did not reveal the presence of undetected or single point failures, it is assumed for this analysis only one undetected failure to be conservative.

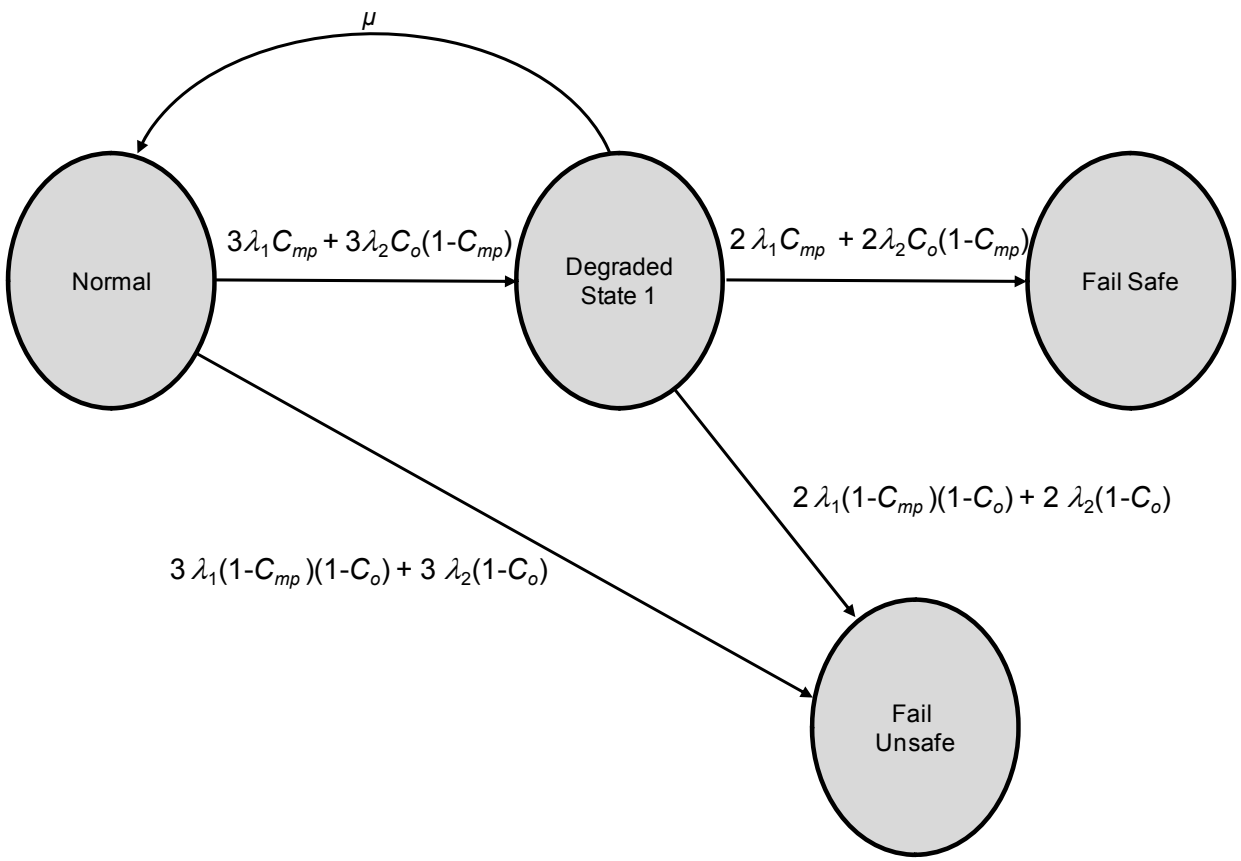

Figure 8-13 Simplified Markov model of the benchmark system

States of the Model

Normal (P0): In this state, all components are operational and functioning correctly.

Degraded State 1 (P1): In this state, only two of the processors are operational (one has failed), or two of the Input/output legs are operational (one has failed). The failure is detected by either the processor fault detection diagnostics or the output voting module.

Fail-Safe (P2): In this state, only one of the processors are operational (two have failed), or only one of the Input/output legs are operational (two have failed). The failures are detected by either the processor fault detection diagnostics or the output voting module.

Fail-Unsafe (P3): In this state, two or more failures have occurred and are not detected by either the processor fault diagnostics or the output voting module.

The model does not explicitly include spare I/O modules. The purpose of this examle is to show the utility of the methodology for safety modeling assessments, and thus some simplifications are made in the modeling. An actual PRA modeling activity would develop models to include spare I/O modules to determine the impact of hot switchovers from a faulted I/O module to a spare I/O module, including scenarios of imperfect switchover, failed spare on demand, etc.

Parameters:

λ_1: Main Processor module failure rate. Nominally 2x10^{-5} failures/hour[1].
λ_2: Output module failure rate. Nominally 2x10^{-5} failures/hour.
C_{mp}: Coverage of RPS functioning. This value is equivalent to the coverage calculated in Table 8-6 (C_{mp} = 0.9995)
C_o: Coverage of the operating system functions and output module functions (C_o = 0.9984).
μ: Repair rate or reboot rate after a recoverable failure: 3 minutes/reboot or 0.05 repairs/hour.

The behavior of Benchmark System II expressed as a system of linear differential equations is given by the following relationships.

Let P(t) be a vector that gives the probability of being in each state at time t.

$$P(t) = [P_0(t), P_1(t), P_2(t), P_3(t)] \tag{8.6}$$

The system of differential equations for the model is given by

$$\frac{dP(t)}{dt} = AP(t) \tag{8.7}$$

where A is the transition state matrix describing the coupling between the states of the model and is given by the matrix **A**

$$= \begin{bmatrix} -(3\lambda_1 C_{mp} + 3\lambda_2 C_o(1 - C_{mp}) & \mu & 0 & 0 \\ 3\lambda_1 C_{mp} + 3\lambda_2 C_o(1 - C_{mp}) & -\mu & -(2\lambda_1 C_{mp} + 2\lambda_2 C_o(1 - C_{mp})) & -(2\lambda_1 G + 2\lambda_2(1 - C_o)) \\ 0 & 2\lambda_1 C_{mp} + 2\lambda_2 C_o(1 - C_{mp}) & 0 & 0 \\ 3\lambda_1 G + 3\lambda_2(1 - C_o) & 2\lambda_1 G + 2\lambda_2(1 - C_o) & 0 & 0 \end{bmatrix}$$

where G = $1 - (1 - C_{mp})(1 - C_o)$

The matrix **A** is easily constructed by thinking of the Markov model in terms of flow in and flow out of each state. Because there is a transition from state P_0 to state P_1,the entry at a_{11} is nonzero. The value of a_{11} is the transition rate $-(3\lambda_1 C_{mp} + 3\lambda_2 C_o(1 - C_{mp})$. The negative sign indicates the transition is leaving the state. The entry at a_{21} (P_1) from state P_0 is $(3\lambda_1 C_{mp} + 3\lambda_2 C_o(1 - C_{mp})$, which indicates state P_1 is accepting a transition from state P_0.

In non-matrix form, the system is described in Equation (8.8) as;

$$\begin{aligned} \frac{dP_0(t)}{dt} &= -((3\lambda_1 C_{mp} + 3\lambda_2 C_o(1 - C_{mp}))P_0(t) + \mu P_1(t) \\ \frac{dP_1(t)}{dt} &= (3\lambda_1 C_{mp} + 3\lambda_2 C_o(1 - C_{mp}))P_0(t) - \mu P_1(t) - \left(2\lambda_1 C_{mp} + 2\lambda_2 C_o(1 - C_{mp})\right) P_2(t) - (2\lambda_1 G \\ &\quad + 2\lambda_2(1 - C_o))P_3(t) \\ \frac{dP_2(t)}{dt} &= (2\lambda_1 C_{mp} + 2\lambda_2 C_o(1 - C_{mp}))P_1(t) \\ \frac{dP_3(t)}{dt} &= 3\lambda_1 G + 3\lambda_2(1 - C_o)P_0(t) + 2\lambda_1 G + 2\lambda_2(1 - C_o)P_1(t) \end{aligned} \tag{8.8}$$

[1] The failure rate parameters stated here are representative, the actual failures are proprietary.

Using the initial conditions P[0] = [1 0 0 0], the equations can be solved to yield the steady state safety and MTTUF expressions for the system.

System safety, *S(t)*, is the probability of being in any state except the failed-unsafe state. That is,

$$S(t) = P_0(t) + P_1(t) + P_2(t) = -((3\lambda_1 C_{mp} + 3\lambda_2 C_o(1 - C_{mp}))P_0(t) + \mu P_1(t) + (3\lambda_1 C_{mp} + 3\lambda_2 C_o(1 - C_{mp}))P_0(t) - \mu P_1(t) - \left(2\lambda_1 C_{mp} + 2\lambda_2 C_o(1 - C_{mp})\right) P_2(t) - \left(2\lambda_1 G + 2\lambda_2(1 - C_o)\right)P_3(t) + \left(2\lambda_1 C_{mp} + 2\lambda_2 C_o(1 - C_{mp})\right) P_1(t) + 3\lambda_1 G + 3\lambda_2(1 - C_o)P_0(t) + 2\lambda_1 G + 2\lambda_2(1 - C_o)P_1(t) \quad (8.9)$$

The steady state safety is the limit of *S(t)* as time, *t*, approaches infinity

$$S\_ss = \lim(t \to \infty)\{P_0(t) + P_1(t) + P_2(t)\} \quad (8.10)$$

which yields

$$S_{SS} = 1 - G^2 + 3G(1 - G)e^{-2\int_0^t(\lambda_1+\lambda_2)dt} + (1 - 3G + 2G^2)e^{-3\int_0^t(\lambda_1+\lambda_2)dt} = G^2 \quad (8.11)$$

Note that S_{ss} is total function of system coverage (*G and* G^2). That is, the safety of the system in the limit is a function of the aggregate coverage of the system. As coverage factors increase for the error detection mechanisms and diagnostic functions in a system, steady state safety becomes dominated by these coverage factors as function of time.

In this phase of the research, due to time and resource limitations, not all of the coverage factors for the various error detection mechanisms could be determined. Thus G was limited to RPS function coverage, OS function coverage, and partial output processing coverage estimates. However, using the methodology developed in this research allows one to gain the necessary additional coverage data for the various error detection mechanisms in the benchmark systems. The important point to note here is that the safety of the system, *S(t),* asymptotically approaches the system coverage as time is increased; therefore, developing an understanding of coverage and its use for PRA modeling activities is significant for accurate safety modeling of digital I&C systems.

The MTTUF of the system is given by [Smith2005]

$$MTTUF = \frac{MTTF}{1-S_{SS}} = \frac{MTTF}{1-C_{sys}} \quad (8.12)$$

where C_{sys} is the total system coverage.

The MTTF of the system is the robability of being in $P_0(t)$ and $P_1(t)$ over the life of the system.

$$MTTF = \int_0^\infty R(t) = \int_0^\infty (3Ge^{-2(\lambda_{Tot})t} + (1 - 3G)e^{-3(\lambda_{Tot})t})dt = \frac{2+3G}{6\lambda_{Tot}} \quad 8.13)$$

where $\lambda_{Tot} = \lambda_1 + \lambda_2$

so that

$$MTTUF = \frac{MTTF}{1-S_{Ss}} = \frac{2+3G}{6\lambda_{Tot}(1-G^2)} \quad (8.13)$$

Instantiating Equation (8.13) with the data from the fault injection experiments where C_{mp} = 0.9995, and C_o = 0.9984 yields

$$MTTUF = \frac{MTTF}{1 - S_{Ss}} = \frac{2 + 3G}{6\lambda_{Tot}(1 - G^2)}$$

$$= \frac{2 + 3(1 - (1 - C_{mp})(1 - C_o))}{6(\lambda_{Tot})[1 - (1 - (1 - C_{mp})(1 - C_o))^2]}$$

$$= \frac{\sim 5}{(2.4\ x10^{-4})(1 - .99985)} = 1.3888889\ x\ 10^7\ hours\ to\ first\ unsafe\ failure$$

One of the advantages of the *MTTUF* metric is that it is calculated using standard dependability metrics *R(t)* and *MTTF*, and system coverage C_{sys}. Typically, *MTTF* is given as one of the reliability metrics by vendors. System coverage, C_{sys} , is obtained either through fault injection experiments or measurement-based dependability analysis. The appropriate interpretation of the metric should be taken in the same caution as when using *MTTF*.

The *MTTF* metric is sometimes misunderstood to be the life of the product instead of the expectation of the times to failure. For example, if an item has an *MTTF* of 100,000 hours, it does not mean that the item will last that long. It means that, on the average, one of the items will fail for every 100,000 item-hours of operation. If the *times to failure* are exponentially distributed, then on average, 63.2 percent of the items will have failed after 100,000 hours of operation. The same logic applies to *MTTUF*, it is an indicator of *the expected* time to unsafe failure. A higher *MTTUF* for a particular system infers the system is expected to have a longer time between unsafe failures.

8.11.2. Example System Safety and Sensitivity Analysis

The MTTUF and S_{ss} metrics provide asymptotic results, which are convenient for making decisions at the earlier stages of an assessment process. Because Markov methods allow modeling the evolution of a system over time, the probability of being in a given state at any time *t* can be determined. Thus, state-based metrics such as System Safety, *S(t)*, and Reliability, *R(t),* are easily determined. In addition, sensitivity analyses by parametrically varying critical parameters to determine the effect of the parameter variations on system safety or reliability can be performed.

Figure 8-14 shows System Safety, *S(t),* as function of mission time. System Safety is defined as being in states *P(0), P(1), or P(2)* of the Markov model at any time in the mission. From Figure 8-12, the the lower bound of the predicted system safety *S(t)* is 0.999 for one year, which is commensurate with fault tolerant triplex systems. The Markov model results in this section were solved using the NASA WinSURE reliability analysis program [ref], where SURE is the acronym for Semi-Markov Unreliability Reliability Estimation tool.

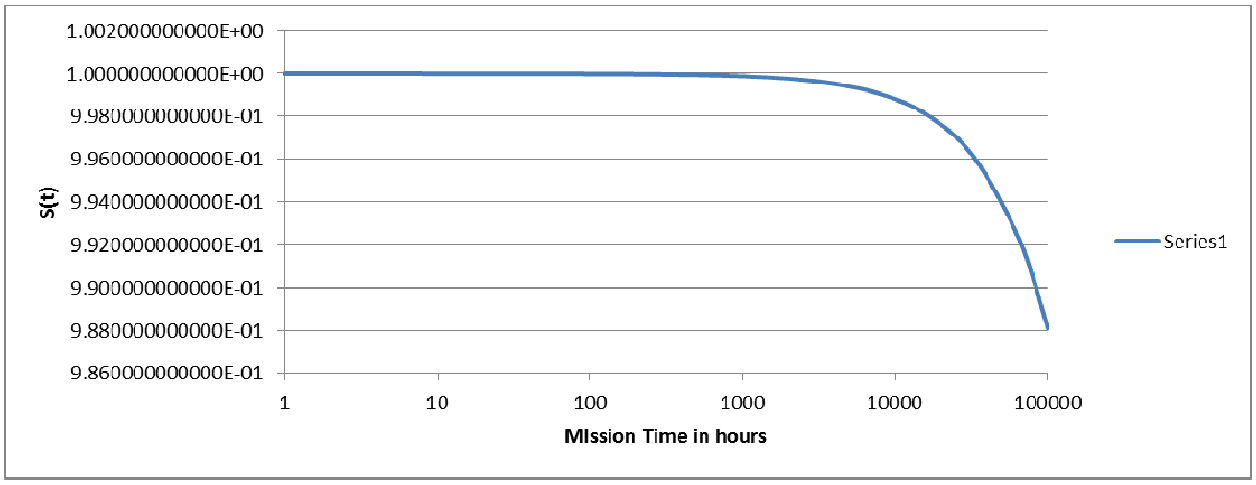

Figure 8-14 Example of system safety as a function of mission time

As described in Section 3.3, the fault tolerance operation of Benchmark System II is composed of various stages of fault/error detection from input to output. It is typically informative to determine whether the capabilities of one stage of error detection mechanisms are more sensitive than another stage of error detection mechanisms. In this case, the measured fault coverage capabilities of the processor (C_{mp}) are compared to those of the Output module (C_o).

Figure 8-15 shows the effect of varying the processor fault coverage from 0.9 to 1.0 on the probability of failure (states 3 and 4). The effect is negligible, and this result is consistent with the design of the architecture. That is, the faults/errors that escape detection at the processor level still have at least one more level of fault detection to propagate through to the output. Conversely, as seen figure 8-16, Benchmark System II is more sensitive to the fault coverage (C_o). In this case the probability of system failure varies over two orders of magnitude for the same parameter variation. For the designer or assessor, this would indicate where fault injection campaigns would be most beneficial to the assessment process.

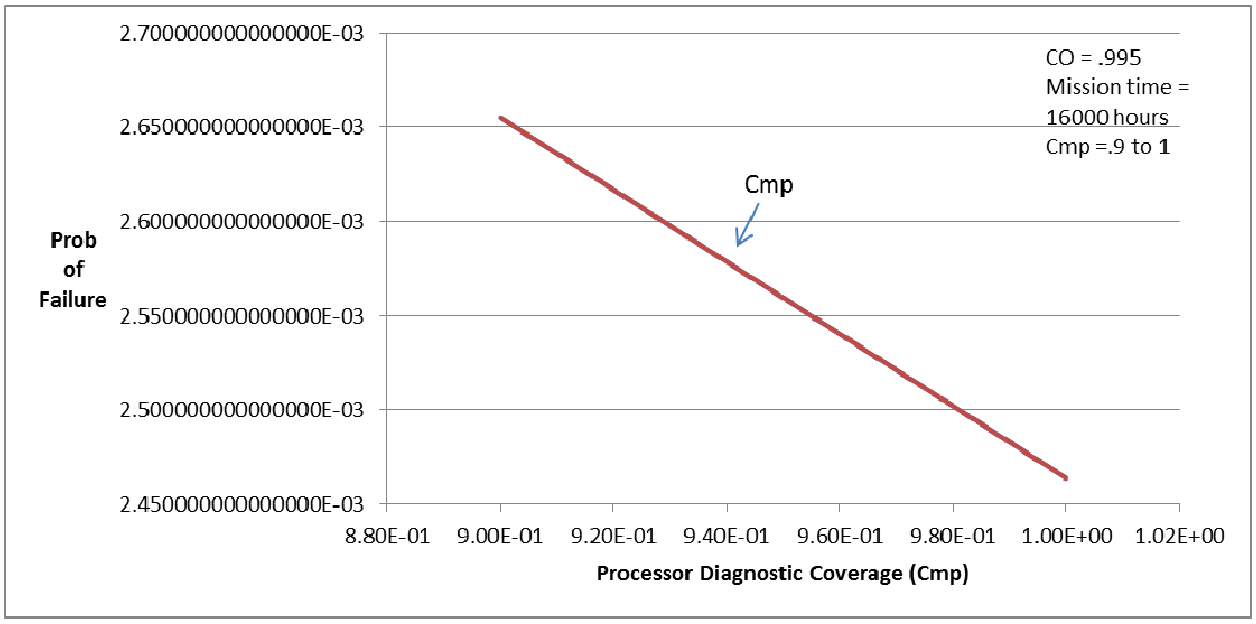

Figure 8-15 Failure probability as a function of processor diagnostic coverage

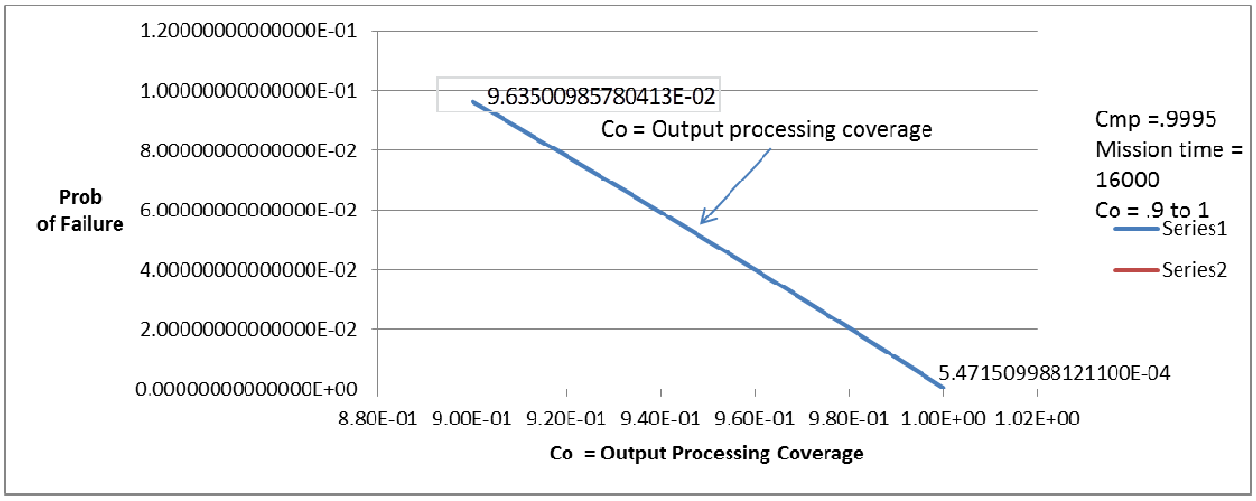

Figure 8-16 Failure probability as a function of output diagnostic coverage

8.12. References

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

[Smith 1997] Smith, D. T., T. A. DeLong, B. W. Johnson, and III Profeta, J. "An algorithm based fault tolerance technique for safety-critical applications." 1997. 278-285.

[Tang 1995] D. Tang and H. Hecht, “Measurement-Based Dependability Analysis for Critical Digital Systems”, *SBIR NRC-04-94-061 Phase I Final Report*, SoHaR Inc., May 1995.

[Smith 1997a] D. Smith and Barry Johnson, “A variance-reduction technique via fault-expansion for fault-coverage estimation”*, IEEE Transactions on Reliability Vol 46, no 3 1997.*

[Kaufman 2002] L. Kaufman, J.B Dugan, B.W Johnson, “Coverage Estimation Using Statistics of the Extremes for When Testing Reveals No Failures”. *IEEE Trans. Computers* 51(1): 3-12 (2002)

[Miller 1992] K. Miller, et al “Estimating the probability of failure when testing reveals no failures” *IEEE Transactions of Software Engineering*, Vol 18 no. 1 1992.

9. Summary, Findings, and Conclusions

This Section summarizes activities (Section 9.1) of this report, and lists the principal study findings (Section 9.2). It provides preliminary conclusions drawn by the authors by applying and assessing the fault injection-based assessment methodology to Benchmark System II. In closing, some final observations and recommendations are made in order to better refine the fault injection-based assessment methodology application to digital I&C systems.

9.1. Summary of Key Activities and Results

The work described herein presents

(1) Developing the fault injection methods and techniques that are appropriate and suitable to the benchmark system

(2) Developing an innovative FPGA-based high performance fault injector for digital I&C systems

(3) Performing a preliminary investigation into the uncertainty factors and sources of uncertainty that could affect fault injection

(4) Presenting the results of the application of the fault injection method to Benchmark System II

(5) Describing the findings from addressing challenges and establishing a basis for implementing fault injection to digital I&C platforms.

The requirements and challenges of realizing fault injection on the Benchmark II system are summarized below:

9.1.1. Identification and Selection of Appropriate Fault Injection Techniques

Section 4 identified appropriate physical fault injection techniques for Benchmark System II based on:

- the types of faults that could affect end-to-end system processing and thus impact the I&C functionality.

- the sub-systems or modules where fault injection should be applied to represent real world faults.

Based on these criteria, a fault injection technique matrix was developed that indicated appropriate fault injection experiments for each sub-system in Benchmark System II. These recommendations are presented from the view of what is possible and what is recommended. Thus, the matrix provides information of what is possible if complete system level documentation is available to the assessor.

9.1.2. Development of a High Performance Fault Injection Environment:

As discussed in these reports, the need for fault injection techniques to support various fault models for fault injection in manner that are minimally intrusive, controllable, repeatable and reproducible is critical to the application of fault injection methods for digital I&C systems.

On the basis of experiences and lessons learned from previous fault injection efforts, it became apparent that a new approach for injecting faults in digital I&C systems was needed if fault injection was to become practical for digital &C systems. Accordingly, innovative and practical methods that address many of the problems encountered trying to perform fault injection on real-time digital I&C systems were developed. The realization of this research is a FPGA-based HiPeFI fault injector.

The HiPeFI fault injector was used exclusively on Benchmark System II with outstanding success achieved in performing over 10,000 fault injections without any significant issue. By moving to a single, multi-purpose fault injector designed specifically to support diverse fault injection on digital I&C systems, fault injection has been optimized around performance (e.g., minimal intrusiveness) and controllability simultaneously. Furthermore, by uniting a variety of fault injection techniques with a common interface onto a single platform, the integration of the fault injector into digital I&C system fault injection experiments is more or less consistent from one digital I&C platform to the next. This aspect becomes important when fault injection is used as a benchmarking activity. That is, the same set of faults and operational conditions can be applied to each digital I&C system by the same fault injection technique to form an objective basis for comparison or compliance.

9.1.3. Investigation of Measurement Practices and Uncertainty Factors for Fault Injection

An objective of this research effort was to assess a dependability assessment methodology with respect to oversights or deficiencies that must be addressed in the context of physical fault injection processes for digital I&C systems. First, fault injection is a measurement-based assessment process that depends on sound measurement practices. In spite of steady advances in fault injection research over the past 20 years, quantitative evaluations of dependability attributes remains a complex task lacking standard processes and techniques. Until recently, measurement practices and uncertainty analysis for fault injection testing has been largely overlooked. As fault injection has moved from the laboratory to the working environment of various industries, a greater awareness of the need to incorporate metrology concepts into dependability studies has arisen.

Measurement-based assessment processes must characterize the various types of uncertainty that can occur during the assessment process to better inform the set of conclusions that can be made about measured results, and how those results can be interpreted in a broader, more general context. This becomes especially important when trying to relate or compare fault injection-based findings from one type of system to another. The methodology presented in Volume 1 and Section 1 of this report has not developed this important topic to the degree required for the evaluation of safety critical systems. This initial work is a first step toward developing a better awareness of (1) better measurement practices for dependability studies and (2) identifying to the best extent possible the types of uncertainty that are germane to fault injection processes. In Section 6, a number of potential sources for uncertainty with respect to fault injection processes were identified and characterized.

Further, guidance of toward better measurement practices is one area of the methodology that requires more consideration and improvement. The UVA intends to continue work on this important area to improve the applicability of the methodology for testing digital I&C systems.

9.1.4. Pre-fault Injection Analysis Revisited

Section 7 of Volume 2 introduced and developed the concept of *Pre-injection analysis* as a means to reduce or eliminate the "no-response" fault injection testing result and to improve the overall effectiveness and efficiency of the fault injection process. However, the application of pre-injection analysis methods to the benchmark systems has not been entirely successful.

The principle reason for this ineffectiveness is mainly due to an assumption that real-time execution trace data was required to realize an acceptable pre-injection analysis. However, acquiring such data was shown to be problematic with both benchmark systems. Given that, it has been shown through simulation studies that the using a pre-injection analysis process can have a significant impact on fault injection studies by only improving efficiency and effectiveness, and allowing the use of fault list reduction methods. As such, efforts were refocused to make pre-injection analysis viable for digital I&C systems. The current idea is to use a powerful interactive disassembler and debugger tool such as IDA Pro to extract control flow and data flow program information that could be used by the pre-injection analysis algorithms to generate highly effective fault lists. This approach is in the initial stages of development, but preliminary results suggest that using tools such as IDA Pro will provide the necessary information needed for pre-injection analysis for large variety of digital I&C systems.

9.1.5. Application of the Fault Injection Methodology to Benchmark System II

The culmination of the research described in this report was the application of the fault injection-based dependability assessment methodology to Benchmark System II. The fault injection experiments represent the types of fault injection tests that typically would be conducted by an assessment organization or the digital I&C equipment vendor during the course of a system V&V. The experiments were chosen to stress the methodology and the supporting tools (UNIFI) and HiPeFI to provide a basis for determining the effectiveness of the methodology for supporting system safety assessment activities (e.g., license reviews and FMEA) and PRA activities.

All of the fault injection experiments were conducted successfully, providing a rich set of information on the fault handling behavior of Benchmark System II that would be supportive of PRA assessment activities. Over 10,000 faults were injected into the benchmark system during the course of this research. Approximately 8 gigabytes of system response data were collected for these fault injection campaigns.

The key findings for this effort were used to develop a novel method of fault list generation to support function block level fault injection experiments on RPS and OS code. Critical PRA modeling parameters such as fault coverage, fault latency, and system trip responses were directly measured from the system under test, thereby successfully validating that the UVA assessment methodology is applicable, feasible, and appropriate for digital I&C systems of the type tested to date.

9.2. Conclusions

This research effort has laid a foundation for vendors and regulators to consider fault injection as a method to help inform the assessment of digital I&C systems in nuclear energy applications. Several findings were significant with respect to applying fault injection to the digital I&C systems:

- The baseline elements and functionality of a fault injection environment for digital I&C systems were established.

- New methods and tools to facilitate full usage of fault injection processes in digital I&C systems were developed.

- Quantitative-based assessment was demonstrated to be not only possible but to offer the potential to augment current review processes in manner that could make the processes more objective and streamlined.

The fault injection methodology applied to the benchmark systems successfully obtained independent information about the system that could corroborate vender and regulator information, and in some cases produced information that would have been very difficult to deduce from vendor information alone.

A final observation is the experience of conducting fault injection experiments in a systematic process often yields more information than just quantifying the fault tolerance aspects of the system. Further, a systematic fault injection process also provides a means to circumspect and comprehend the behaviors of complex fault tolerant I&C systems to support overall assessment activities for both the regulator and the developer. In this regard, quantitative assessment processes such as fault injection are a natural partner to current deterministic review processes.

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 3

2. TITLE AND SUBTITLE
Development of a Fault Injection-Based Dependability Assessment Methodology for Digital I&C Systems: Volume 3

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 3 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 3 presents the findings of applying a fault injection-based quantitative assessment methodology to a prototype triple modular redundant reactor protection system (Benchmark System II). The purpose of this work was to (1) determine 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) develop and refine the methodology to improve applicability to digital I&C systems, and (3) establish a basis for using fault injection 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 3

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

December 2012