2008

Development of a Hardware-in-the-loop Simulation Platform for Safety Critical Control System Evaluation

Drew James Rankin
Western University

Development of a Hardware-in-the-loop Simulation Platform for Safety Critical Control System Evaluation

(Spine title: Hardware-in-the-loop Simulation of Safety Control Systems)

(Thesis format: Monograph)

by

Drew James Rankin

Graduate Program
in
Engineering Science
Electrical and Computer Engineering

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Engineering Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

Abstract

During the lifetime of a nuclear power plant (NPP) safety electronic control system components become obsolete [7]. It is difficult to find replacement components qualified for nuclear applications [50]. Due to strict regulations, replacement components undergo extensive verification and operational analysis [70]. Therefore, the need for a platform to evaluate replacement safety control systems in a non-intrusive manner is evident. Verifying the operation or functionality of potential replacement electronic control systems is often performed through simulation [71].

To enable simulation, a physical interface between potential control systems and computer based simulators is developed. System connectivity is establish using Ethernet and standard industrial electrical signals. The interface includes a National Instruments (NI) virtual instrument (VI) and data acquisition system (DAQ) hardware. The interface supports simulator controlled transmission and receipt of variables. The transmission of simulated process variables to and from an external control system is enabled. This is known as hardware-in-the-loop (HIL) simulation [49]. Next, HIL interface performance is verified and the following are identified; a measure of availability; the effect of varied configurations; and limitations.

Further, an HIL simulation platform is created by connecting a NPP simulator and a programmable logic controller (PLC) to the interface, Canadian Deuterium Uranium (CANDU) reactor training simulator and Invensys Tricon version nine (v9) safety PLC respectively. The PLC is programmed to operate as shutdown system no. 1 (SDS1) of a CANDU reactor. Platform availability is verified and the response of the PLC as SDS1 and is monitored during reactor shutdown. Proper execution of the steam generator level low (SGLL) logic on the PLC and variable transmission are observed. Thus, a platform and procedure for the evaluation of replacements for obsolete electronic control system components is demonstrated.

Keywords: hardware-in-the-loop, simulation, National Instruments, interface, programmable logic controller, Tricon v9, shutdown system, SDS1, steam generator level low, obsolete, CANDU, nuclear power plant

Table of Contents

List of Tables

List of Figures

Abbreviations and Nomenclature

Abbreviations

2oo3	Two-out-of-three voting
AECL	Atomic Energy of Canada Limited
BWR	Boiling Water Reactor
CANDU	CANadian Deuterium Uranium
Ch-X	Channel (identified by X)
CNSC	Canadian Nuclear Safety Commission
CPU	Central Processing Unit
CSV	Comma Separated Values
D_2O	Deuterium Oxide, Heavy Water
DAQ	Data Acquisition System
DarlSIM	Darlington Nuclear Power Plant Simulator
ECCS	Emergency Core Cooling System
EI	Execution Interval
eu	engineering units (kg/s, m, °Celsius)
FBD	Function Block Diagram
H_2O	Hydrogen Oxide, Water
HIL	Hardware-in-the-Loop
I&C	Instrumentation and Control
IAEA	International Atomic Energy Agency
LCV	Level Control Valve
LOCA	Lost Of Coolant Accident
mA	milli-ampere
ms	milli-second

MW	megawatt
MW(e)	Megawatt Electric Power
NI	National Instruments
NPP	Nuclear Power Plant
NW	North-West
OPG	Ontario Power Generation
OS	Operation System
PCI	Peripheral Component Interconnect
PHTS	Primary Heat Transport System
PHWR	Pressurized Heavy Water Reactor
PLC	Programmable Logic Controller
PWR	Pressurized Water Reactor
SCFW	Spurious Closure of Feed-water valve
SDS	Shutdown System
SDS1	Shutdown System no. 1
SDS2	Shutdown System no. 2
SG	Steam Generator
SGLL	Steam Generator Level Low
SIS	Safety Instrumented System
SUT	System Under Test
TCP/IP	Transmission Control /Internet Protocol
TNT	Trinitrotoluene
UDP/IP	User Datagram Protocol/Internet Protocol
USNRC	United States Nuclear Regulatory Commission
v9	version nine
VDC	Voltage (Direct Current)
VI	Virtual Instrument

Nomenclature

av_{HIL}	interface availability
cv_{eu}	controlled variable represented in engineering units
$cv_{eu_{tx}}$	transmitted variable represented in engineering units
$cv_{eu_{rx}}$	received variable represented in engineering units
cv_i	instantaneous controlled variable value
cv_{max}	maximum expected controlled variable value
cv_{min}	minimum expected controlled variable value
cv_{range}	range of expected controlled variable values
f_{sw}	frequency of the sine wave to generate
f_s	frequency of the samples
n_{bit}	resolution in bits for analog to digital conversion
n_{bit}	total number of identified erroneous transfers
$n_{I/O}$	number of configured inputs and outputs
n_s	number of sinusoidal samples to generate
n_{total}	total number of attempted transfers through the interface
res_{analog}	analog signal resolution
res_{cv}	expected controlled variable resolution
$res_{UDP/IP}$	resolution of the UDP/IP packet structure
t_{EI}	period of execution for modules on the simulator
t_{HIL}	maximum time expected for the majority of SUT outputs to return after being requested to do so
$t_{s_{HIL}}$	interface signal settling time
$t_{s_{SUT}}$	system under test signal settling time
$t_{SUT_{EI}}$	system under test control logic execution period
t_{SUT}	system under test control logic execution wait time
t_{te}	total elapsed time for all process variables to be transmitted through the interface
$t_{te_{max}}$	maximum total elapsed time for process variable transmission over entire simulation

Chapter 1
Introduction

The success of integrating nuclear power plants (NPPs) into a society with increasing population density and environmental concern relies heavily on safety system design [14]. Confidence in safety systems to mitigate environmental, economical and public health consequences must be demonstrated [14]. Many NPP safety analysis codes, simulation tools and methods of evaluation exist [6]. However, significant challenges remain in the verification process of electronic control systems for safety critical nuclear control applications [6].

A flexible simulation platform for the operational verification of safety electronic control systems is developed. In doing so, existing tools are interfaced with modern simulation practises and control system technologies. This platform is not intended to replace existing tools. However, it is intended to strengthen the set of currently available tools [6]. The following sections underscore the background, motivations, problems and objectives of the research undertaken in this thesis.

1.1 Background

NPPs are required to abide by increasingly strict safety standards and regulations [70]. Safety critical control and shutdown systems within NPPs are essential in satisfying these regulations, assuring public health and safety, minimal environmental impact and stable plant economics [3]. Therefore, safety related electronic control systems must be evaluated to verify, validate, qualify or certify proper operation and design.

Evaluations are performed by regulatory bodies, manufacturers and systems engineers [6]. Depending on the intended application for the electronic control system, specific qualifications and certifications are required [11]. These requirements are specified by regional, national and international regulatory bodies whom develop NPP instrumentation and control (I&C) standards [8]. However, as mentioned, standards and regulations are becoming increasingly strict, and are also sometimes vague [31].

Therefore, the submission of an electronic control system for evaluation may require the commitment of extensive resources over a long period of time [43]. For many manufacturers, justifying certification procedures for nuclear applications is difficult due to the relatively small nuclear I&C market [50].

Though a small market for certified electronic control systems for nuclear applications currently exists, it is widely believed that the nuclear energy industry is entering a renaissance [21] [20]. The apparent renaissance is the result of an increase in energy demand, public awareness and an acceptance that nuclear power generation is necessary for base load electrical supply [21] [19]. However, as mentioned, manufacturers cannot justify performing extensive and costly industry certifications. The result is a small number of certified electronic control systems,. This places limitations on NPP utilities and systems engineers when developing new or replacement I&C system components. Engineers either; a) choose from few certified electronic control systems; b) design and qualify in-house electronic control systems; or c) attempt to integrate and qualify commercial electronic control systems which lack certification [4].

It is not clear whether manufacturers will respond to the expected increase in commercial nuclear activity and tightening regulations. However, the requirement for a range of electronic control systems covering a variety of applications is evident. Increasingly, the task of verifying, validating and qualifying electronic control systems to satisfy the requirements outlined by regulatory bodies is being placed on NPP systems engineers [18]. To prove system functionality electronic control systems are often evaluated through various simulation and emulation techniques [71]. Thus, the application and development of the platform within this thesis is performed to streamline the evaluation of electronic control systems applied in specific NPP applications.

The International Atomic Energy Agency (IAEA) estimates that total worldwide electrical generating capacity will grow in the range of 37 to 84 per cent between 2006 and 2030. Over the same period, nuclear generating capacity is expected to grow in the range of 21 to 87 per cent [16]. These projections are difficult to foresee given the average age of operational NPPs is over 24 years. In fact, today nearly 85 per cent of operational NPPs are within 25 years of their life expectancies [21] [20] [19]. Therefore, simply to sustain current nuclear power generating capacity there is a great need for major refurbishment and life extension initiatives in parallel with new nuclear build.

Public concern over nuclear proliferation and security causes significant roadblocks when addressing safety system development for nuclear applications [88]. Historically, nuclear power generation has a great safety record [88]. This record has been maintained through regulations, standards and thorough engineering practices which have ultimately resulted in an industry which is poised to respond to the ever growing demand for energy.

The fundamental basis for power generation by nuclear fission both enables and disables its application. Nuclear fission reactions release an immense amount of energy, 20 million times the amount of energy released by Trinitrotoluene (TNT) [2]. However, the fuel used to generate heat in a nuclear fission reaction is naturally radioactive. Further, fuel which has been depleted of its useful radioactive material, or spent fuel, has a higher radioactivity than the original natural or enriched uranium [2]. Therefore, spent nuclear fuel requires extensive monitoring, management, and storage facilities [88].

Currently, 18 nuclear reactors generate 14.6 per cent of Canada's total electrical supply [22]. However, a reactor has not commissioned in Canada for over 15 years. The most recently constructed reactor is Darlington Unit 4, an Atomic Energy of Canada Limited (AECL) design, which went into service in June 1993 [23]. Of the 18 operating Canadian reactors, two units, Pickering 1 and 4, are recently refurbished having been brought back into service within the last five years [37]. Also, in March 2008, New Brunswick Power began an 18-month refurbishment of the Point Lepreau NPP [78].

The estimated life expectancy for Canadian Deuterium Uranium (CANDU) NPPs operating in Canada is 40 years [23]. By 2030, 16 of the 18 operating Canadian reactors will have surpassed their life expectancies [23]. Therefore, it is immediately necessary to plan engineering methods for plant life extension, refurbishment, retrofit and new build projects. Failure to act will result in a reduced nuclear capacity of only 1870 megawatt (MW), Darlington 3 and 4 combined capacity, and a shortage in total Canadian electrical supply [23].

The accelerated development and expansion of NPP sites will require definite assurances that public health, public safety and the environment are not placed at risk [21] [20]. These assurances rely on engineering practices during NPP safety system design. Within the nuclear industry, for example, design basis events are well defined postulated events identified to establish the acceptable performance requirements for

structures, systems, and components [44]. To mitigate the consequences of design basis events, safety critical control and shutdown systems are installed.

In a NPP many systems are classified as safety critical. There are different standards and levels of classification. However, the safety critical and shutdown systems which are most critical to safety in all NPPs are those which prevent the release of radioactive material from a contained volume [29]. In order to maintain containment; pressures, temperatures, flow rates, and other plant processes as well as neutronic stability are monitored and controlled [29]. By controlling the physical plant, limitations of the mechanical components which contain radioactive materials are never reached and accidents are mitigated [29]. Further, conservative safety margins are maintained to assure safe operation [10]. Safety margins are the margins allocated between the limitations of physical operation in any given process and the specified safe limit of the components within the process during anticipated operational occurrences [61]. The reduced performance of ageing and obsolete components nullifies engineering efforts and can result in a reduction of the safety margin [87].

Aside from obsolete systems, many electronic control systems in NPPs are composed of ageing analog circuitry [10]. Concerns often arise when analog controllers are to be replaced by digital controller platforms [67]. The reliability of software based systems, common cause faults and fault tolerance of digital systems come into question [60]. Simple replacement of these systems can lead to minor variances in plant performance, or instability in plant operation [60]. However, through proper simulation methods it is believed that the performance of the digital systems can be measured and either observed variances can be minimized or process performance or safety margins can be improved [56].

The process of engineering replacement systems includes extensive testing and verification [53]. This is especially true for safety critical systems [53]. However, due to hazardous environments, it is not always possible to perform testing on replacements for obsolete systems as they exist in the plant [9]. In these cases, the use of simulation techniques not only assists with verifying proper electronic control system operation prior to installation within the NPP, but is the only means of testing the physical system prior to plant start-up [9].

Traditionally simulations are performed entirely by software [42]. In the Canadian nuclear industry there are many extensively developed nuclear power generation software simulation platforms [15]. These simulators are capable of performing com-

plex and extensive simulations [15]. With the increasing problems of obsolescence and replacement, expansion of software simulators to include connection to external hardware is becoming more common [89]. Process variables from within the software simulator are converted to compatible electrical input signals for input to the electronic control system under test (SUT). Also, SUT output process variables are converted from electrical signals to software variables and received by the software simulator. Connection of the two systems, simulator and SUT forms a loop. Therefore, this method of simulation is known as hardware-in-the-loop (HIL) simulation [79].

This thesis covers two major topics; i) the role of HIL simulation in the development and testing of electronic control systems; and ii) the simulation of a specific CANDU NPP shutdown system application. Throughout, focus is placed on the Canadian nuclear power generation industry where these topics have extensive application as well as motivation for research and development. Both the HIL interface and the simulation platform developed within this thesis are designed to be flexible and modular, enabling installation between various software simulators and electronic SUTs.

1.2 Motivations

There are three distinct motivations behind the research performed in this thesis. All motivations are the result of strict industry regulations and limited availability of replacement electronic control systems discussed previously. The primary motive is the need for both a process and a platform for the qualification of shutdown system hardware and software based on relevant standards through HIL simulation. This is a long term motivation which extends beyond the scope of this thesis. The secondary motive is to better enable the replacement of obsolete digital and analog shutdown system hardware and software. The final motivation is to provide support for the integration of enhanced functionalities and advanced control techniques digital safety control systems.

1.2.1 Qualification

During replacement, a potential electronic control system must demonstrate equivalent functional performance to the system being replaced. Further, plant safety margins must be maintained and safety standards satisfied [87]. Within the nuclear industry, interest in qualifying electronic control systems has grown due to the lack of certifications, discussed earlier. Therefore, a method to qualify electronic control systems through evaluation is necessary.

Application specific qualification routines provide engineers with the ability to prove that potential replacement systems meet the criteria outlined in standards and regulations and do not negatively impact plant performance when installed in a specified application [50]. It is expected that related standards can be identified, reviewed and that system requirements can be quantified for a specific application. The quantified requirements are evaluated against the potential electronic control system and the system is either accepted for installation or removed from contention.

Qualification of a electronic control systems requires functional evaluation. However, installation into a NPP is not an option due to the continuous and expensive nature of NPP operations [68]. Therefore, performance evaluation of a potential electronic control system within an HIL simulation platform is performed to provide full functional evaluation and verification. Along with quantified standard requirements, an integrated platform for streamlined system qualification could be constructed.

1.2.2 Obsolescence and replacement

It is essential to better enable the replacement of obsolete shutdown system components in NPPs [67]. This is true as the rate of obsolescence rises with the age of operating reactors [7]. Also, it is expected that obsolescence will continue despite the predicted nuclear renaissance [33]. In fact, the opposite is true with an increase in anticipated refurbishment, retrofit, and plant life extension projects.

Currently, many projects address obsolete systems within NPPs [7]. However, due to financial, scheduling and regulatory restrictions the replacement of all systems and components is not realistic [7]. Therefore, proper functionality must be established between refurbished and existing systems. Additionally, continued obsolescence issues are inevitable. It is difficult to predict technological trends or support for currently installed systems [65].

Components once commonly available are now either manufactured on a per request basis, or are simply no longer manufactured [50]. Further, many suitable modern electronic control systems are not qualified for nuclear applications [50]. Therefore, reverse engineering techniques have become common practise in maintenance and refurbishment routines [1]. Through reverse engineering, physical, functional and operational replicas are developed to satisfy NPP design standards and regulations [25]. These replicas avoid or reduce the extensive requirements experienced when installing modern replacements [25]. Thus, functionality between old and new is maintained. However, at the system level, complexity proves to be a limiting factor in the process of reverse engineering [25]. Systems are comprised of intricate networks of non-serviceable components and can be very difficult to replicate. If the original manufacturer of the system is no longer willing to manufacture these devices, entire system replacement must be adopted.

For every system replacement, potential replacement systems must be extensively verified to assure confidence in mitigating design basis accidents and maintaining safety margins [79]. Though many methods of evaluation exist, the development of a HIL simulation platform which enables I&C component and electronic control system verification in a standardized, flexible and scalable manner would be a valuable addition to an already strong set of tools [35]. By this method, exhaustive reverse engineering practices would become obsolete. Further, the ability to prove the performance of modern replacement systems would be enabled.

This thesis encompasses the preliminary development of a flexible, scalable and modular interface. The interface is designed with focus on common, standardized simulator and electronic control system connectivity. It is expected that the proposed procedure can be utilized within the nuclear industry to facilitate streamlined integration of existing software simulator platforms and potential replacement systems. The platform developed is not intended to be used for qualifying or validating components. However, the platform enables evaluation of industrial processes and the functional verification of electronic control system logic.

1.2.3 Enhanced capabilities

Obsolescence is the primary reason raised when replacing control systems or components [77]. However, replacement systems often emulate the performance of the

obsolete system [77]. There is a fundamental flaw in this practise. Obsolete systems are no longer manufactured because either modern, more capable products are available, or because problems are identified and the system did not perform as expected for some applications. This being said, when modern technologies are installed to replicate obsolete systems safe operation of the plant may be in jeopardy.

The enhancement of control routines is limited due to strict regulations within the nuclear industry, as previously mentioned [83]. Also, a reduction in nuclear power generating projects over the past decades has resulted in NPP control systems falling behind modern technology and practices [32]. With advancing technology, new algorithms and control techniques are constantly being developed [90]. These techniques have been extensively employed and proven through use in other industries [90]. However, the benefits of these techniques have been ignored by the nuclear industry [90].

Similar to the troubles surrounding the replacement of obsolete systems, the integration of enhanced capabilities requires extensive evaluation. In a NPP, simple control logic is preferred, especially for safety systems [34]. However, modifying safety system control logic through the implementation of built-in electronic control system functions, alarms and other routines can improve performance while maintaining or improving safety margins [34]. Performance metrics include among other parameters, reliability, visibility availability, accuracy and reduced execution time. HIL simulation is an extremely valuable tool for validating correct functionality and identifying the benefits that these enhanced capabilities provide prior to installation. Further, the work required to replace any given component is effectively reduced.

1.3 Problem Statement

There are two primary problems which are addressed within this thesis. The first is the development of an interface for connecting a software simulator to an electronic control system. The second is the application of the interface, of the first problem, to a specific safety critical process within a NPP. These two problems are essential in identifying the role of HIL simulation in electronic control system development and testing and applying modern simulation techniques to a specific NPP application.

1.3.1 Hardware-in-the-loop interface development

Everyday, significant financial and human resources are directed at developing truth and physics models [24]. These software simulators are developed using many different computer platforms and operating systems [59]. Recently, increasing problems associated with obsolescence of electronic and electro-mechanical components have produced a common extension for the application of the simulators [77]. Refurbishment, upgrade and replacement of obsolete components is performed through functional verification and evaluation against real-time bench-mark simulated plant models. HIL simulation platforms provides this capability [41]. In fact, HIL simulation platforms are now developed specifically for the purpose of extending existing simulator functionality [41].

Within the Canadian nuclear industry, as discussed previously, there are concerns regarding the obsolescence of components. Consequently, the extension of well developed and tested simulators into HIL simulation platforms has become a priority of the Canadian nuclear research community [58].

A software, Unix based NPP training simulator, supplied by Ontario Power Generation (OPG) is used to provide extensive training and replication of operational scenarios [12]. However, the computer architecture and operating system on which the simulator was developed, though real-time, is very limited and is no longer supported by the manufacturer [38]. This becomes a problem as evaluating potential replacements for obsolete electronic control systems by HIL simulation requires interface between the actual control system hardware and the training simulator. Upon further investigation, development and analysis; a simple, flexible and inexpensive solution is established.

1.3.2 Hardware-in-the-loop shutdown system simulation

Advances in digital safety control system platforms and algorithms are being constantly developed [90]. These advanced systems are installed in an array of applications and industries throughout the world [5]. Although the installation of digital systems into nuclear control systems, including safety critical systems has been limited, there are many recent ambitious initiatives for increased implementation [57]. Therefore, a renewed necessity is apparent to evaluate modern alternatives to analog

and legacy digital systems in safety critical control applications. In fact, three advanced electronic control systems are now certified for nuclear safety application by the United States Nuclear Regulatory Commission (USNRC) [69] [17].

The Canadian nuclear industry led the world in digital control system implementation in the 1980s [39]. However, reduced public support of nuclear power generation did not exclude Canada. Though AECL has taken minor design steps, limited electronic control system development has resulted [40]. Yet, the resurgence of nuclear interest has sparked refurbishment initiatives throughout the country. The NPP at Point Lepreau refurbishment involves specifically the installation of one of the three USNRC approved electronic control systems. Therefore, the refurbishment directly relates to the research within this thesis. The Tricon version nine (v9) programmable logic controller (PLC) is the electronic control system selected for replacement of shutdown system no. 1 (SDS1) at Point Lepreau [52]. It is also the electronic control system of focus within this thesis. Though the Tricon v9 PLC is considered a commercial-off-the-shelf system, it is a complicated system and requires extensive training and programming experience to implement all available enhancements [86] [84] [85]. Further, the logic solving unit of SDS1 does not rely on complex operations, functions and algorithms. The simplicity of the safety critical algorithms is apparent when studying the emulated station code of the training simulator and SDS1 fundamentals [39].

In conjunction with the previous problem, the implementation of SDS1 logic as it exists in the training simulator is ported to the Tricon v9 PLC. Further, combining the development of an advanced electronic control system platform for SDS1 and an HIL interface for a software simulator, results in the Tricon v9 PLC being implemented with the developed HIL interface to form a HIL simulation platform. Functional and operational evaluation of the Tricon V9 PLC and verification of the installed control logic and the capabilities of the simulation platform are performed.

1.4 Objectives

The objectives for the research performed within this thesis are outlined, in no particular order, in the following list. The two major objectives listed below stem from the problems and motivations stated in Sections 1.3 and 1.2 respectively. Extended objectives are nested below the two major problems.

- **extend the functionality of a NPP training simulator to include hardware-in-the-loop simulation**
  - identify a method for simulator and electronic control system connectivity
  - develop a software module for the acquisition of NPP training simulator process variables
  - develop an interface which supports connectivity between the Tricon v9 PLC and the NPP training simulator
    * propose a procedure for developing an hardware-in-the-loop interface
    * evaluate the performance of the interface prior to simulation
    * identify limitations of the interface
    * develop a method for assuring hardware-in-the-loop simulation results
    * propose a procedure for connecting and verifying the operation of the interface within a hardware-in-the-loop simulation platform
- **demonstrate the functionality of CANDU shutdown system no. 1 control logic on a Tricon v9 PLC through hardware-in-the-loop simulation**
  - replicate shutdown system no. 1 control logic within the Tricon v9 PLC
  - connect the Tricon v9 PLC to the HIL interface
  - establish process scenarios for Tricon v9 PLC operational verification
  - verify signal transmission through the hardware-in-the-loop simulation platform
  - evaluate the performance of the Tricon v9 PLC against bench-mark NPP process simulations

1.5 Organization

The thesis is organized as follows. Chapter 2 includes a review of NPP fundamentals, where the concept of nuclear power generation and related control systems are presented. Chapter 3 proposes a procedure for developing an HIL interface. The proposed procedure is performed and an interface is established. In Chapter 4, analysis of the interface is performed through the execution of two studies. Various timing

requirements and limitations of the interface are observed and discussed. Chapter 5 presents the installation of the interface developed in Chapter 3 within a HIL simulation platform. The simulation platform is designed to enable simulation of a specific CANDU NPP process. Compatible control logic is developed, and an electronic control system is programmed. After installing the electronic control system within the platform, variable transmission and platform timing routines are verified. A benchmark process is then observed and hardware-in-the-loop simulation is performed. During simulation, a software simulator executes CANDU specific NPP processes. The connected electronic control system is responsible for controlling a segment of the NPP shutdown system. The performance of the electronic control system as a replacement system for shutdown, and the developed interface as a tool for verifying proper electronic control system operation are discussed. Finally, Chapter 6 presents the conclusions related to the thesis work, contributions, and an outline of potential future research.

Chapter 2
Nuclear Power Plant Control Systems and Simulation

2.1 NPP Fundamentals

The basic energy cycle for a NPP is shown in Figure 2.1. Fuel for a nuclear fission reaction contains fissile material (uranium, plutonium). This fuel is fed into the reactor core where the fission reaction occurs. The energy produced from the chain fission reactions is released in the form of heat and is used to boil water. The steam produced from the boiling water drives a set of turbines and in turn an electrical generator. The electricity produced is supplied to the commercial electric power grid [27]. In conjunction with feeding fuel into the reactor, spent nuclear fuel is periodically removed from the reactor. Further, a small percentage of the generated electrical energy is utilized for electrical energy consumption requirements of the NPP [27].

There are many NPP designs in world-wide operation today, all of which follow this simplified NPP energy cycle. NPPs are classified by the properties of their moderator. The moderator is a material is contained within the reactor core [63]. Three typical moderator materials are ordinary water, heavy water, and graphite [28]. Most NPP in operation today use water as a moderator. Three main types of water reactors exist including heavy, or light water and heavy water reactors. In brief, light water is composed mostly of hydrogen oxide(H_2O) molecules, whereas heavy water is composed of an increased amount of deuterium oxide (D_2O) molecules [75]. D_2O shares many chemical properties with H_2O but allows for non-enriched uranium, or natural uranium to be used as reactor fuel [66]. This eliminates the requirement for uranium enrichment facilities. Two of the three main types of water moderated reactors are moderated by light water. Namely, pressurized water reactors (PWRs) and boiling water reactors (BWRs). The other is a heavy water moderated

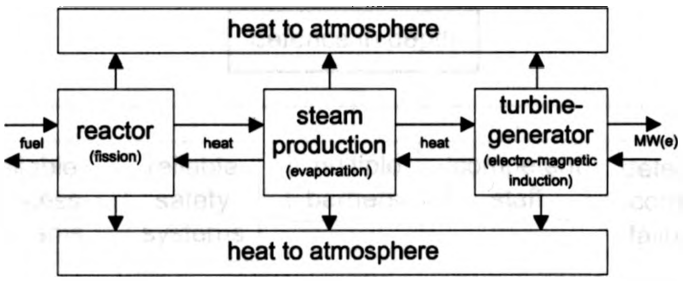

Figure 2.1: Basic nuclear power plant energy cycle.

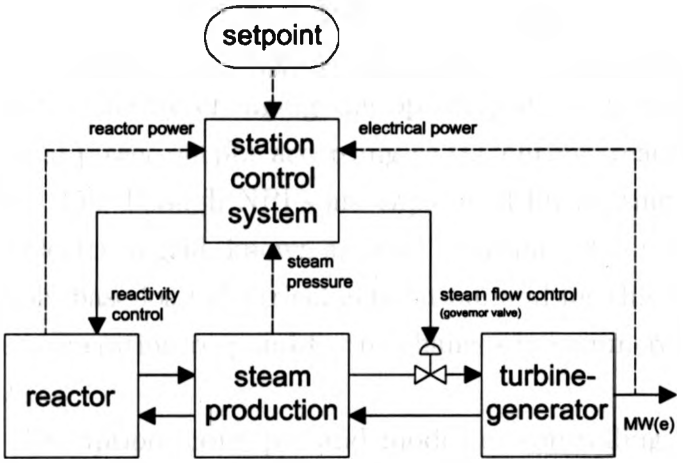

Figure 2.2: Simplified nuclear power plant control system.

reactor or pressurized heavy water reactors (PHWRs) [62]. These basic details of nuclear physics are sufficient in understanding the thesis as a whole.

NPPs are designed to operate for extended periods at constant electrical power output. Thus, they often provide base-load electricity supply to the commercial electrical power grid. A steady state balance must be maintained between the rate of energy released from the fuel in the reactor core and the electrical output of the generator. Many factors affect this balance, including disturbances in energy conversion processes, changes in the load requirements of the electrical power systems and in energy exchanges between the station and the environment [45]. Therefore, a control system must be implemented to compensate for the identified system imbalance [45]. A simplified plant control system is illustrated in Figure 2.2.

Primary inputs to the control system are reactor power, steam pressure and generator output (MW). Two common operating modes exist for NPPs. NPPs can operate in either mode. However, the most common mode is for the control system to operate in 'reactor-leading-turbine,' also referred to as 'turbine-following-reactor'

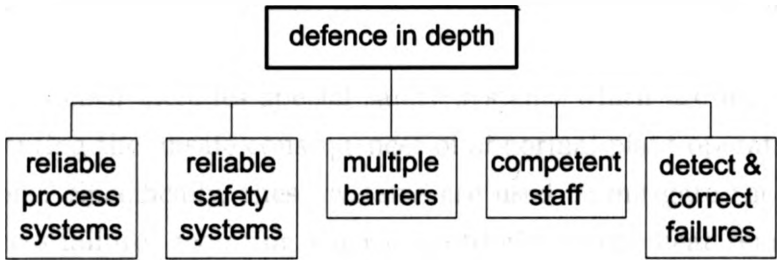

Figure 2.3: Model of 'Defence in Depth' concept.

mode. In this mode, steam pressure is maintained constant while the nuclear fission energy output is corrected to the desired set-point. The control system adjusts the steam flow to the turbine by changing the opening of the governor valve, thereby altering the electrical power output according to the current reactor core power generating capabilities [45]. Though NPPs are capable of fluctuating with the demands of the commercial electrical grid, known as 'load following,' it is not common practise. NPPs usually supply base-load electrical generation, and for this reason, other forms of electrical power generation respond to the changes in commercial electrical power demand [54] [55].

The above description, concepts and modes of controlling a NPP are significantly generalized. However, as indicated in Chapter 1, it is essential to minimize potential threats to the public and environment from the radioactive materials involved in the nuclear power generating process. Specific principles for achieving safe reactor operation are well defined [80]. Reactor safety is maintained if the reactor power is controlled, the fuel is cooled and the radioactivity is contained. For simplicity these principles are shortened to 'Control, Cool, and Contain,' [80].

Reactor safety is further incorporated through a concept known as 'Defence in Depth,' illustrated in Figure 2.3. The concept covers the entire process of designing, constructing, commissioning and operating a NPP. Further, the concept assumes that a) the NPP design will have some flaws, b) equipment will occasionally fail, and c) operating personnel will occasionally make mistakes. It is essential that the employed depth of defence covers both expected and unexpected flaws, failures and mistakes [80].

It is apparent that safety holds significant importance in all aspects of a NPP. At the top level control, cooling and containing are of greatest importance. This is primarily achieved through practising defence in depth as illustrated in Figure 2.3.

To assist in defence in depth practise, the methods documented in Table 2.1 are used [13].

Defence in depth includes special safety systems which are initiated as a last resort for preventing the unsafe consequences of abnormal plant operation during accident conditions. Specifically, these systems are used to mitigate the consequences of serious process failures requiring reactor shutdown, decay heat removal or retention of released radioactivity. Therefore, the systems perform no active function during normal NPP operation. In fact, all safety systems must be actively poised to allow operation of the NPP. For this reason safety system designs often require minimal system unavailability, an aspect which requires efficiently scheduled testing and maintenance procedures [13]. This is considered when developing an interface for satisfying electronic control system verification and the methods discussed in Table 2.1 are practised extensively during special safety system design. Safety systems are the primary focus within the work of this thesis.

2.2 CANDU Fundamentals

CANDU or Canadian Deuterium Uranium refers to a Canadian NPP design. CANDU NPPs are PHWR plants and are designed and maintained by AECL [47] [82]. A heavy water moderator and coolant are applied. This allows for natural as opposed to enriched uranium to be used as fuel. Canada has vast natural uranium reserves. Therefore, no uranium enrichment facilities are required [64]. CANDU NPPs include various additional characteristics which are unique in the nuclear industry. CANDU NPPs have the ability to refuel while the reactor is generating electrical power and connected to the commercial power grid. Other reactor designs replace fuel supplies regularly during extended scheduled shutdown periods. Many PWRs and BWRs are designed with high pressure reactor core and require bulk re-fuelling routines. The AECL NPP has a low pressure reactor core (calandria) which contains the moderator. High pressure tubes accessible from either end of the calandria enable fuel loading via fuel bundles. Primary coolant flows around the fuel bundles and heat energy produced by the fission reaction is transfered to the coolant [64]. These characteristics are known only to be implemented in AECL and CANDU-based reactor designs.

Similar to Figure 2.1, Figure 2.4 illustrates the energy cycle of a CANDU NPP. In addition, main CANDU process systems are included. The calandria contains

Method	Description [13]
Redundancy	Provision of alternative structures, systems and components, so that any one can perform the required function regardless of the state of operation or failure of any other.
Diversity	The presence of two or more redundant systems or components to perform an identified function, where the different systems or components have different attributes so as to reduce the possibility of common cause, or mode, failure.
Reliability	The probability that a system or component will meet its minimum performance requirements when called upon.
Availability (Testability)	The fraction of time for which a system is capable of fulfilling its intended purpose.
Separation	Separation by geometry, by appropriate barriers, or by a combination thereof.
Environmental Qualification	Generation and maintenance of evidence to ensure that equipment will operate on demand, under specified service conditions, to meet system performance requirements.
Quality Assurance	• The function of a management system that provides confidence that specified requirements will be fulfilled. • A systematic programme of controls and inspections applied by any organization or body involved in the transport of radioactive material which is aimed at providing adequate confidence that the standard of safety prescribed in regulations is achieved in practise. • All those planned and systematic actions necessary to provide confidence that a structure, system or component will perform satisfactorily in service.
Codes and Standards	In addition to the above methods, all system should comply with mandatory regional, national and international codes and standards.

Table 2.1: Methods used to ensure NPP safety requirements [13].

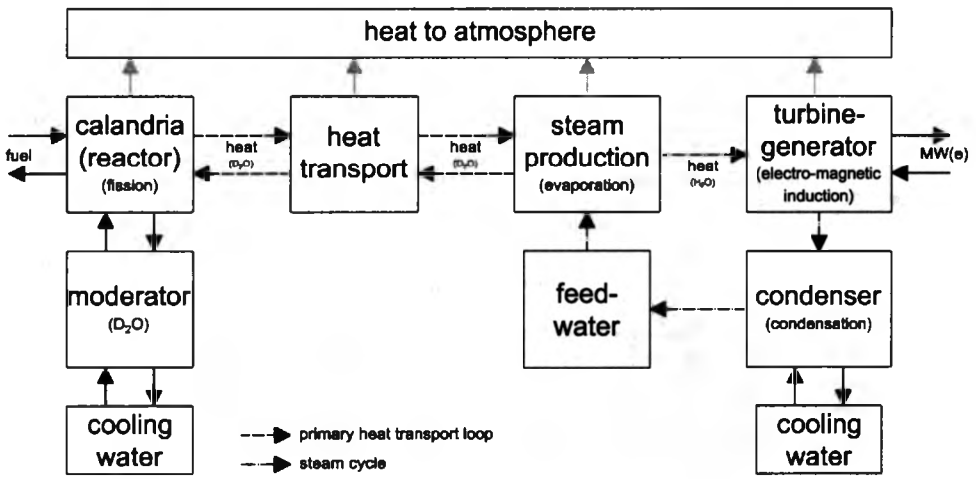

Figure 2.4: Energy cycle and main CANDU NPP systems.

heavy water moderator surrounded by a shield tank. The shield tank is filled with light water, a neutron absorber. Reactivity control is performed by vertical and horizontal reactivity control units installed between pressure tubes from the top and side of the calandria respectively [46].

Description of a simplified primary heat transfer loop begins with the reactor core. This closed loop is the primary heat transport system (PHTS), a system with a figure eight bi-directional coolant flow pattern. The PHTS is responsible for removing the coolant from the calandria and transferring it to light water boilers. An illustration of the PHTS is provided in Figure 2.5. Uranium is loaded into hundreds of pressure tube channels. Heavy water coolant circulates through pressurized tubes, where heat is transferred to the coolant. The heated (300°C) coolant enters the boilers, or steam generators. The coolant is then divided into hundreds of tubes where heat is transferred by conduction to light water. Following energy transfer, the cooler (260°C) heavy water exits the steam generator and is routed to heat transport pumps. The coolant then returns to the pressurized fuel channels. This completes the primary heat transport loop. Approximately 95 per cent of the heat energy generated during the fission reaction is transferred to the steam generator light feed-water with the other five per cent being lost to the moderator. A moderator cooling circuit is also included in the NPP but not discussed here. The heat energy is eventually lost to the atmosphere [26].

A steam cycle begins in the middle of the PHTS. An illustration of the steam cycle is provided in Figure 2.6. Light water in the steam generator is heated by

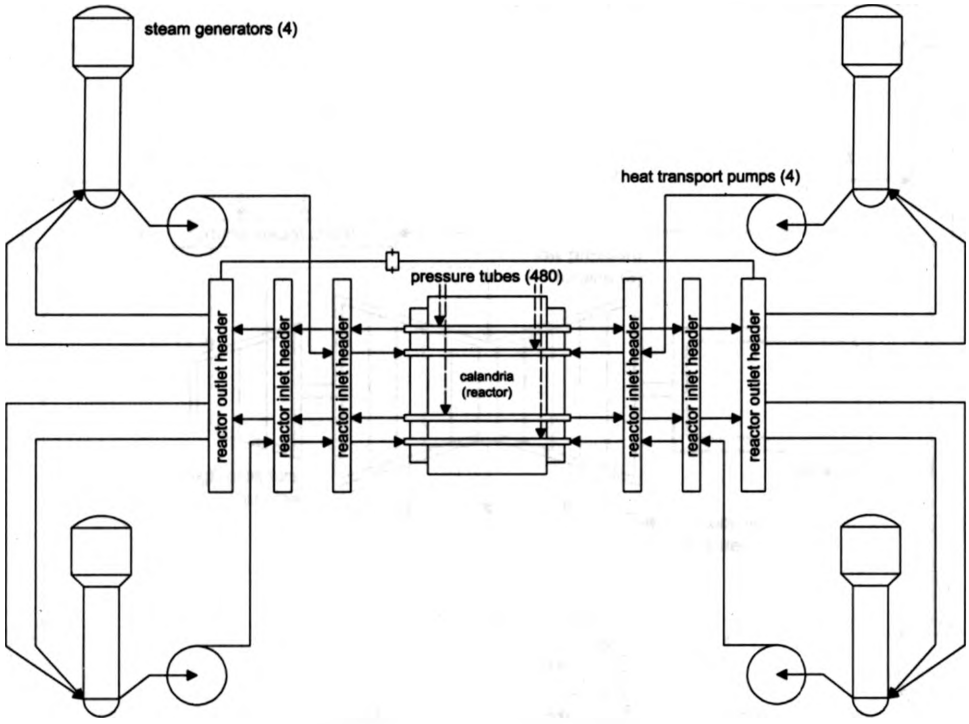

Figure 2.5: Simplified CANDU NPP primary heat transport system (PHTS) [26].

conduction from the heavy water of the PHTS. The light water boils and evaporates into steam. With a lower density than water, the steam rises to the top of the steam generator. A reduction in energy transfer efficiency results. The steam departs the steam generator and flows to the turbine set, both high pressure and low pressure, where it exerts force on the turbine blades. The force on the turbine causes the turbine shaft to rotate. The turbine shaft rotates the coupled rotor shaft of an electrical generator. Heat energy from the steam that does not contribute to driving the turbine blades is lost to cooling water in the condenser. Finally, the condensed steam, or water, is returned to the steam generator feed-water inlet through feed-heating and deaerator systems to complete the steam cycle [30]. Approximately 30 per cent of the total heat energy produced by the chain fission reaction is transferred to the mechanical energy in the turbine shaft [30].

In summary, kinetic energy from nuclear fission heats the heavy water coolant. Heat from the heavy water is conducted to the light water secondary coolant in the steam generator. The light water heat energy is then converted to mechanical energy (turbine) and mechanical energy converted to electrical energy (generator).

The Darlington NPP, located in the Municipality of Clarington near Bow-

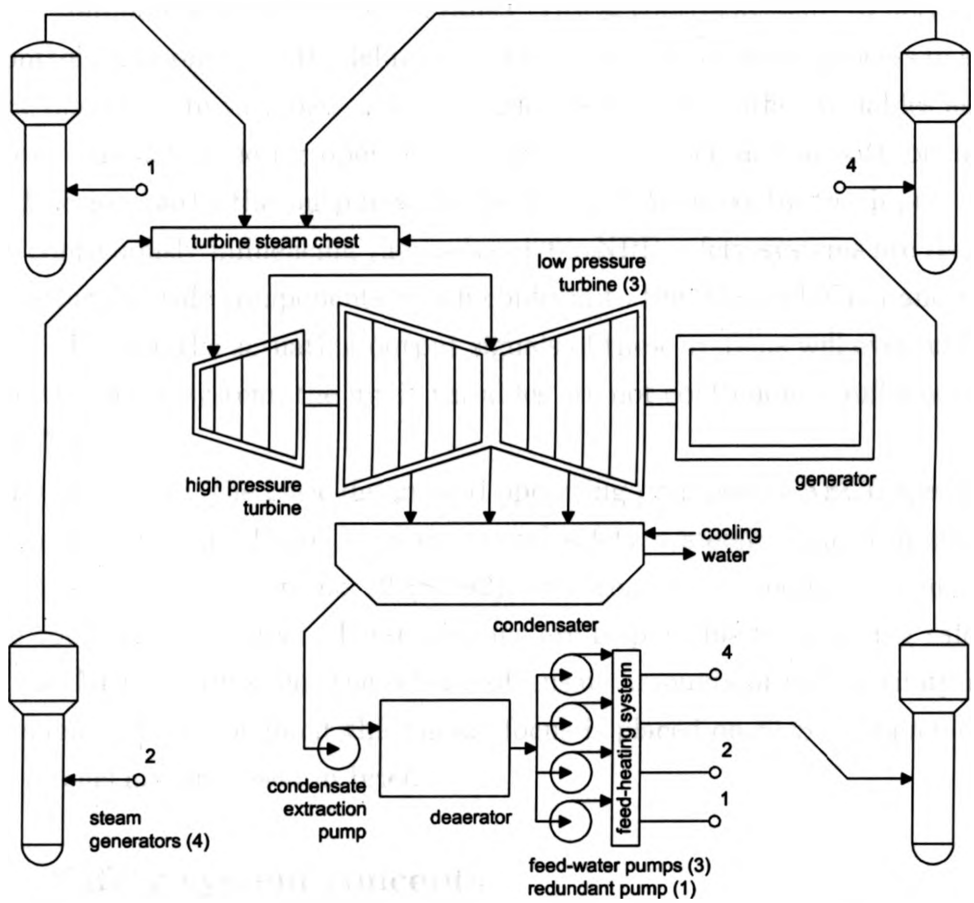

Figure 2.6: Simplified CANDU NPP steam cycle [30].

manville, Ontario, Canada, is the subject of all further NPP descriptions. The NPP is used to illustrate NPP design, operation and safety characteristics. The design and practises of the Darlington NPP provide a basis for control system simulation and development. Darlington NPP is the newest CANDU NPP operating in Canada and is owned by OPG. Darlington NPP is a four-unit station with a total electrical output of 3,524MW(e) [39]. Design and operating specifications of the Darlington NPP are provided in Appendix A.

2.3 Safety Systems

The objective of this research is to enhance safety critical control system practises. With this in mind, the following systems do not conform to classical closed loop control problems. In fact, the algorithms within these systems are relatively simple

[76]. It is common practise for special safety critical control systems to perform open-loop control functions [76]. By definition, open-loop control is any process in a system whereby one or more variables as input variables influence other variables as output variables. Characteristic for open-loop control is the open action path or in case of a closed action path, the output variables being influenced by the input variables are not continuously influencing themselves [76]. NPP safety systems are responsible for actuating fail-safe components which conform to the 'Control, Cool and Contain' principle. Though the actuating output signals of these systems will eventually affect input to the safety system, the input variables are not continuously influenced by the outputs [76].

In this section, the basic design and operating practises of AECL special safety systems are discussed. There are four special safety systems: shutdown system no. 1 (SDS1), shutdown system no. 2 (SDS2), emergency core cooling system (ECCS) and the containment system. These systems are responsible for ensuring safe operation of the NPP, and prevent the release of unsafe amounts of radioactivity into the environment [29]. Throughout this thesis, focus is placed on SDS1. Therefore, other safety systems are discussed in brief.

2.3.1 Safety system concepts

Within CANDU NPPs, safety related systems are grouped to protect against common cause failures which may result from localized damage to the plant. There are two groups of safety systems. Group one encompasses the systems required for normal operation of the NPP as well as SDS1 and containment safety systems. Group two includes SDS2 and the ECCS safety systems [29].

Systems within each group are capable of independently performing reactor shutdown, decay heat removal, control, and monitoring. Physical separation and functional diversity separate the groups. Further, interconnection between groups is practically minimized. However, safety support services (electrical, power, cooling water and feed-water systems) are capable of providing backup support services to each group [29].

Control and monitoring of station events is performed from the main control room of group one. However, if the main control room becomes inoperable or inhabitable, a secondary control room belonging to group two remains functional [29].

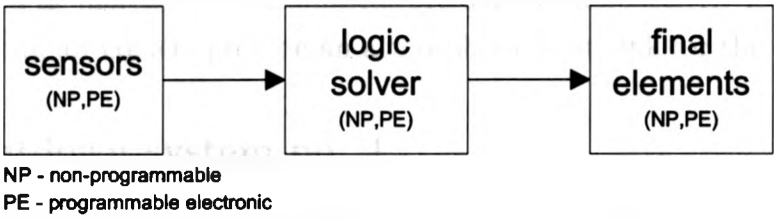

Figure 2.7: Safety instrumented system (SIS) open action path.

The previously described NPP control routines provide some degree of safety protection as they maintain plant balance. However, special safety systems are by definition safety instrumented systems (SIS). An SIS is defined as an instrumented system used to implement one or more safety instrumented functions and is composed of any combination of sensors, logic solvers, and final elements [29]. This is similar to the open action path illustrated in Figure 2.7. There are two primary SISs in CANDU NPPs. One system, SDS1, consists of mechanical shut-off rods while the other, SDS2, injects a neutron absorbing solution into the moderator via liquid poison injection nozzles. The two shutdown systems respond automatically to both neutronic and process signals. Further, either system acting on its own is capable of shutting down the reactor and maintaining shut down for all design basis events [29].

The redundant SDS2 quickly terminates reactor operation when specific neutronic and process parameters enter an unacceptable range. The method of terminating reactor operation is the rapid injection of concentrated gadolinium nitrate solution into the moderator through a horizontal nozzle located on the side of the calandria [29].

The ECCS replenishes reactor coolant in the event of a loss-of-coolant accident (LOCA). This safety function is necessary to assure cooling of the reactor fuel. In the initial ECCS phase, pressurized light water is delivered to PHTS headers. Pressure is provided by ECCS gas tanks. Over the long term, the ECCS re-circulates the light water through the pressure tubes and dedicated ECCS heat exchangers for decay heat removal [29].

Containment is provided by containment envelope and containment isolation. The containment envelope is the reactor building. The envelope provides a pressure-retaining boundary and includes airlocks, piping and electrical penetrations, with isolation valves included as necessary. It is designed to withstand the maximum pressure of any design basis accident. Isolation is achieved automatically by closing

specific valves if an increase in containment pressure or radioactivity level is detected. In the long term, air coolers provide an atmospheric heat sink for the envelope [29].

2.3.2 Shutdown system no. 1

The two NPP shutdown systems, SDS1 and SDS2, are designed in compliance with Canadian Nuclear Safety Commission (CNSC) regulations [29]. Though SDS1 is the focus of this section and the thesis work, general and performance requirements for both shutdown systems include [29]:

- A CANDU NPP must be equipped with two independent and diverse shutdown systems.
- Compliance with regulatory requirements must be approved by the governing body (CNSC, formerly the Atomic Energy Control Board) prior to issuance of construction approval or operating license.
- Designed such that, acting alone, it can ensure that:
  - the reactor is rendered sub-critical and is maintained sub-critical;
  - the reference dose limits are not exceeded; and
  - a loss of primary heat transport system integrity shall not result from any fuel failure mechanism, in the event prompt shutdown is required.
- Ensure fuel in the reactor, with no defects prior to the event, does not fail as a consequence of the event for relevant events listed in Tables B.1, B.2, B.3, B.4 of Appendix B.
- Have the ability to operate when exposed to the most severe environmental conditions that accompany the events listed in Tables B.1, B.2, B.3, B.4 of Appendix B. This capability should be demonstrated and effects analyzed.
- Demonstrate the target unavailability of less than 10^{-3} years per year including availability, testing and maintenance.
- Incorporate redundancy such that the failure of any single component will result in below allowable performance under accident conditions.
- Provide shutdown capabilities regardless of electrical power supply condition.

- Be designed to have consistent safety capabilities in all operating and failure modes. Failed components must fail at a safe state.
- Prevent disabling of manual operator shutdown capabilities, and to respond to redundant and remote location manual shutdown.
- Incorporate diverse, physical and operational independence.
- Be effective and independent of normal process, safety or special safety systems.

SDS1 is the primary method for terminating reactor operation when a specified set of NPP parameters enter an unacceptable range. SDS1 reduces the capability of nuclear fission reactions within the core through the release of spring-assisted gravity-drop shut-off rods [29]. The shut-off rods are composed of the neutron absorbing material cadmium.

SDS1 has three independent channels, designated D, E and F. Each channel detects the requirement for a reactor trip and de-energizes a set of relays which control direct current clutches for the release the shut-off rods. The triplicated design and measurement of safety critical NPP parameters will function even when a single loop component or power supply fails. Therefore, a failure will not incapacitate or spuriously invoke the operation of the safety system [29].

The logic for SDS1 operates using general coincidence. General coincidence operation requires that two of the three (2oo3) channels of SDS1 trip. The parameter on which SDS1 trips does not affect the dropping of the shut-off rods. Therefore, if any two parameters are beyond specified safe limitation set-points, SDS1 initiates shutdown routines [29].

SDS1 and SDS2 operation differ when more than one trip parameter is beyond normal operating range. Both SDS1 and SDS2 use a 2oo3 voting mechanism. SDS1 will trip on any two parameters, illustrated in Figure 2.9, which are beyond an acceptable range, known as general coincidence. On the other hand, SDS2 requires that the two parameters that are not satisfying proper NPP operation are produced by the same parameter on both channels. This is known as local coincidence [29]. Both coincidence voting techniques are illustrated in Figure 2.8.

The parameters which can initiate a general coincident reactor trip through SDS1 are listed below. Further, the parameters and the systems that the measured variables are captured from are illustrated in Figure 2.9. There are nine measured variables which must remain within specified ranges in order for the reactor to remain

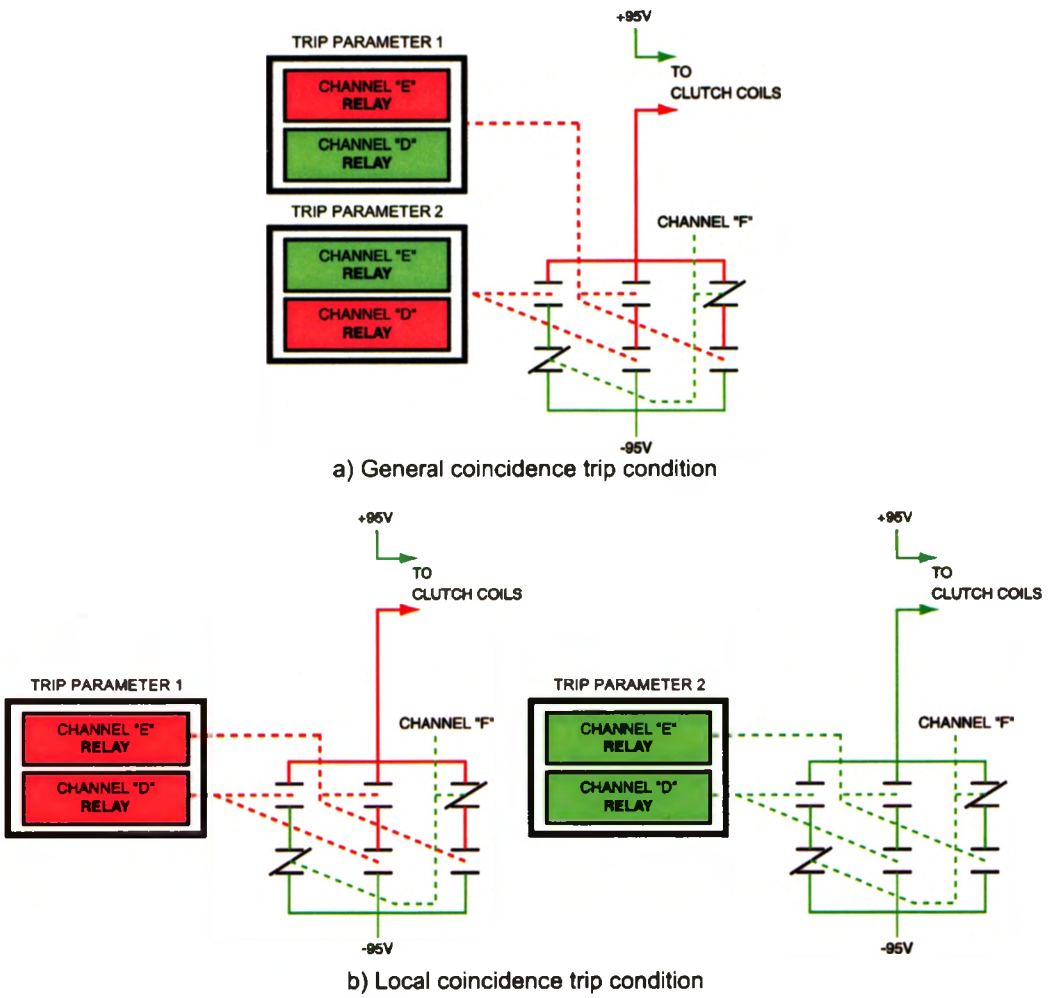

Figure 2.8: a) General (SDS1) and b) local (SDS2) coincident two-out-of-three (2oo3) trip mechanisms.

in an operating state. The selection of variables is performed so that all categories of process failures identified in Appendix B are protected against. Also, the manual initiation of SDS1 is supported. This is performed by an operator from the main control room or secondary control area [36].

1. *neutron power* - Inconel in-core flux detectors. A linear amplifier converts detector current readings to a voltage signal. The set-point for neutron over power is altered depending whether abnormal reactor operating conditions are observed. Shutdown system design ensures protection coverage during normal fuel manoeuvring or single device failure.

2. *rate log neutron* - Ion chamber detectors. Current signal proportional to the

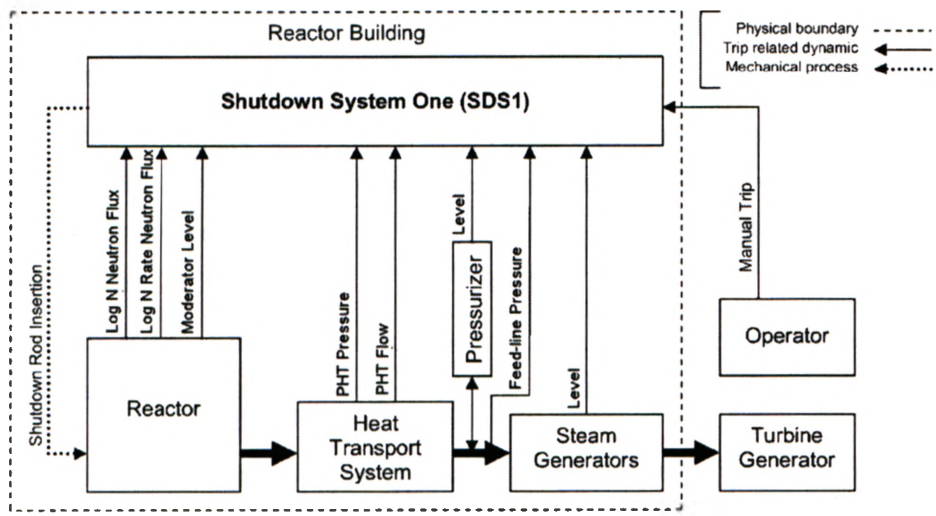

Figure 2.9: CANDU SDS1 trip parameters and originating systems.

thermal neutron flux and amplified to linear and logarithmic neutron power and rate logarithmic signals. The rate logarithmic signal is used as a direct trip parameter. Set-point conditioning for rate log neutron trip is provided by neutron power levels.

3. *heat transport system flow* - Mass flow feeders. The flow feeders are divided into two groups, half are used for SDS1 and the other half for SDS2. Measurements are taken from each coolant pass within the calandria. Heat transport system trip is conditioned out for very low power, or shutdown conditions.

4. *heat transport system pressure* - Pressure transmitters. Both high and low pressure are used trip conditions are enabled. However, low pressure is conditioned out for low neutron power levels. Pressure transmitters are also used to control auxiliary relief valves.

5. *reactor building pressure* - Pressure transmitters. Triplicated measurement effective for primary and secondary side breaks within containment.

6. *pressurizer level* - Effective for small loss-of-coolant accidents. Conditioned out at very low neutron power levels to allow for draining during maintenance.

7. *steam generator level* - Effective for protecting against secondary side failures including secondary side heat removal. If secondary side heat removal is not functioning the energy transfer from the PHTS to the steam generators, and

subsequently the turbines, is disabled. Further, the mechanical structure of the steam generators may be compromised. This parameter is also conditioned out at low neutron power level.

8. *steam generator feed-line pressure* - Pressure in each steam generator feed-line is monitored. Protection against secondary side failures which could result in the loss of the steam generators as heat sink. Conditioned out at low neutron power levels.
9. *moderator level* - Effective for loss of moderator cooling, moderator pipe breaks and channel failures. Both high and low levels provide trip conditions.

The three independent channels of SDS1, D, E and F have completely independent power supplies, parameter sensors, and other instrumentation, trip computers and annunciation devices [29]. Shutdown rods are divided into two banks. Each bank is supplied with redundant power supplies. The shut-off rods are suspended above the calandria by energized direct current clutches and coincident logic. Each clutch coil is held energized by the contacts of a separate relay. If a single relay fails, the remainder of the shut-off rods will drop into the calandria [29].

SDS1 has various other operational features. Every variable within SDS1 can be overridden within the NPP. In this way, SDS1 logic can be functionally tested at regular intervals [29]. Also, rod drop tests can be performed for each individual rod. The ability of the rod to drop freely into the calandria is essential for proper reactor shutdown. During a drop test the clutch mechanism is de-energized, and re-energized almost immediately by a time delay relay. The distance the rod travels is recorded and determined either to be suitable for shutdown. If the rod does not pass the rod drop test maintenance is performed [48].

The physical location of components within SDS1 enables maintenance routines while practising separation. Drive mechanisms are located on the reactivity deck above the calandria. Controlled access to the clutches, motors, potentiometers, gear boxes and winches is required. Separation between clutch and withdrawal motor circuits is essential. Regulating and shutdown system channels cannot be on the same channel. Therefore, separate cables and junction boxes are installed. Trip computers and relay trip logic for each SDS1 channel are located in the main control area. All SDS1 reactor measuring devices are field mounted. However, they are distributed in a manner which minimizes the possibility of common-mode failures with SDS2 and

reactor regulating system devices. Cables are run in three physically separated cable runs to the main control room [51].

One trip computer is utilized per channel in SDS1 and SDS2. The trip computers are digital platforms and have replaced analog trip comparators, programmable digital comparators and relay logic used in previous plants. Final 2oo3 voting, seal in routines and shut-off mechanism actuation is performed using various relay circuits [67]. The relay circuits are controlled directly from the digital outputs of SDS1 and SDS2.

The required insertion time of the shut-off rods following a trip detection is less than or equal to two seconds. When SDS1 is in a tripped state not more than two shut-off rods are fully withdrawn. Reactor poison removal procedures and adjuster and mechanical control absorber withdrawal are disabled to assure maximum shutdown capabilities. Also the D_2O moderator system is isolated from the reactor core to prevent the addition of pure moderator to the poisoned moderator. Conversely, when SDS2 is in a tripped state the same measures are taken and shut-off rods withdrawal is disabled [81].

Chapter 3
Hardware-in-the-loop Interface Development

The transmission of signals between simulated and actual systems within an HIL simulation platform is illustrated in Figure 3.1. Within the following chapters the HIL simulation platform is referred to simply as the platform. The platform is composed of simulator, system under test and interface sub-systems. During development a simplified platform is used. This platform is referred to as the development platform.

Conversion of software process variables to electrical signals enables the installation of an electronic control system as it exists in an actual NPP environment. Therefore, HIL simulations are commonly used to evaluate, validate and verify the operation of electronic control systems. However, to establish process variable transmission between the simulator and the electronic control system an interface must be developed and installed.

The fundamental function of the interface is to acquire simulated process variables from a simulator and to convert the variables into analog or digital electrical signals. The opposite is also true. Electrical signals are returned to the simulator from the SUT through the interface. The remainder of this chapter focuses on the development of the interface [41].

3.1 Hardware-in-the-loop Interface Development Procedure

During HIL simulation the installed electronic control system operates under real-time conditions. Therefore, the simulator must also support real-time simulation. It is not the case that the simulated process must provide continuous output of all input variables, or that it must receive variables continuously from SUT outputs. However, the period between consecutive signal inputs to, and outputs from the simulated

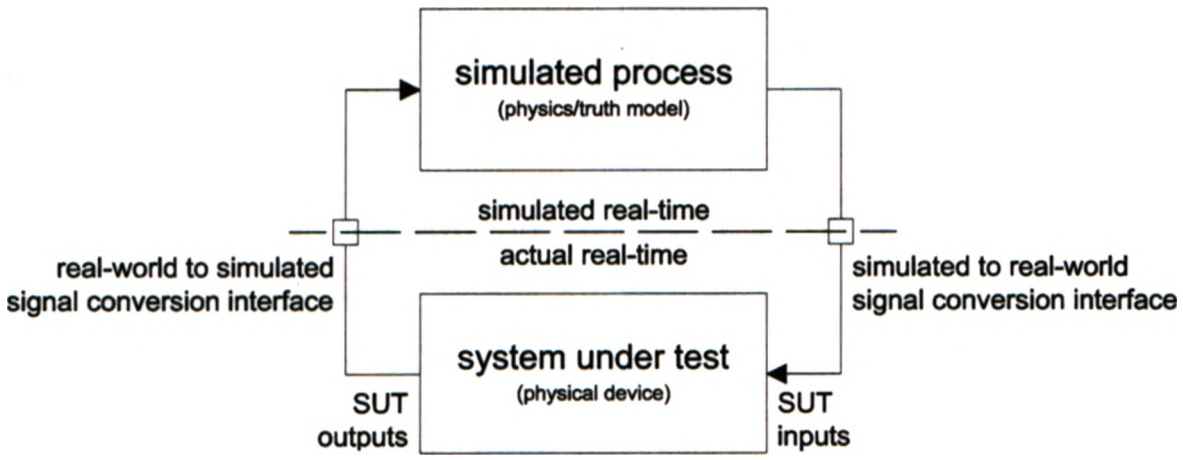

Figure 3.1: HIL simulation platform concept.

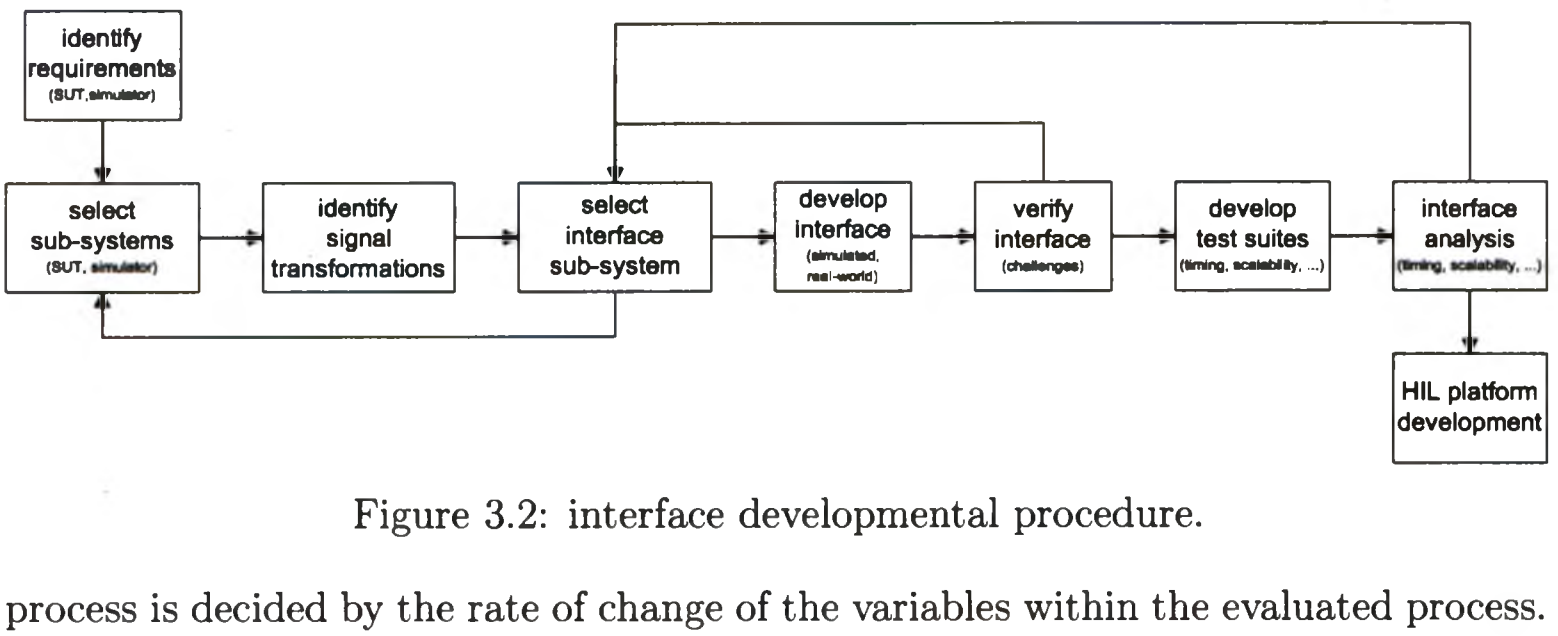

Figure 3.2: interface developmental procedure.

process is decided by the rate of change of the variables within the evaluated process. Capabilities of the SUT must also be considered. In general, an HIL simulation can achieve real-time operation if the transmission of the outputs and inputs satisfies the dynamics of the process being evaluated [71].

HIL simulation limitations are introduced by the hardware and software selected for the interface. For this reason, a procedure to develop, test and verify an interface is proposed. This procedure is illustrated in Figure 3.2. In the end, interface functionality is verified both prior to and following installation into an application specific simulation platform. If sufficient performance is not met prior to HIL simulation, the interface is re-developed and undergoes repeated verification procedures until acceptable interface performance is observed. A measure of acceptability for interface performance metrics is left to the discretion of the design and systems engineer. However, the acceptability of the interface performance between differing configurations is discussed during analysis in Chapter 4.

Within the procedure of Figure 3.2, the requirements of the process being evaluated and the purpose for performing HIL simulation are first identified. If already

chosen, SUT features and simulator model dynamics are analyzed and functional requirements for the interface are recorded. There are four likely scenarios when faced with the challenge of developing an interface.

- *Known process only* - no simulation model, or SUT have been selected. The process being evaluated specifies all requirements for interface and simulator development, as well as SUT selection.
- *Existing SUT* - the SUT has been selected as the electronic control system for the process being evaluated. However, the electronic control system requires verification through simulation. Therefore, simulator development is required.
- *Existing software simulator* - the functionality of an existing simulator is extended to include HIL simulation. Therefore, multiple SUTs can be evaluated and their performance compared.
- *Existing SUT and simulator* - both the existing SUT and simulator scenarios are satisfied. Simulator development and multiple SUT platform evaluations are not required.

In all cases, including the existing simulator scenario, simulation requirements rely most heavily on the process being evaluated. However, in the existing simulator scenario the process being evaluated has already been reviewed and its dynamic requirements accounted for during simulator development.

Regarding the SUT, two electronic control system specifications are considered. First, the number of input/output (I/O) supported by the system, $n_{I/O}$, is calculated. This determines the number of required electrical connections between the simulator and the interface. Also, the execution interval, $t_{SUT_{EI}}$, of the logic within the SUT must satisfy the pre-determined requirements of the process being evaluated. After analyzing the control process and calculating the number of I/O and the execution interval of the control logic, manufacturer supplied electronic control system specifications are reviewed to ensure compliance.

For the software simulator, thorough planning is performed. The rate of change of the variables within the controlled process is accurately reflected in the output and acquisition intervals of the simulator, t_{EI}. Fundamental sampling theory must be practised so that valid simulation and accurate reproduction of the real-world response can be performed. Further, the execution interval of the SUT $t_{SUT_{EI}}$, and

the asynchronous execution of the control logic within is addressed and accounted for within the interface.

During the following interface development the 'existing simulator scenario' is undertaken. Therefore, an SUT is not defined. However, a range of values for SUT execution intervals, $t_{SUT_{EI}}$, and I/O capacities, $n_{I/O}$, are investigated. Additionally, the dynamics of the process being evaluated are established in an existing NPP simulator and do not need to be derived.

The chosen NPP simulator has specific process variable output and acquisition intervals, t_{EI}, which range from 50 milliseconds to 2 seconds. A set of intervals within this range are implemented during testing and verification. It is recommended that a thorough examination of real-world process dynamics compared with the simulated dynamics is performed. However, this is beyond the scope of defined research. Therefore, the installed simulator is assumed to perform adequately for the process being evaluated.

Next, the simulator and SUT sub-systems are selected. It may be the case that the simulator sub-system already exists as in the 'existing simulator scenario'. This is the case with the NPP simulator installed in Chapter 5. It is common that the intended SUT or a set of potential SUT sub-systems have already been selected. For the developed interface the selection of the SUT and simulator sub-systems is flexible. The two are specified only by signal conversion requirements. Though an existing NPP simulator will be installed, any simulator with Ethernet communication and UDP/IP communication module support is compatible. Alternatively, standard industrial analog and digital hard-wired signals are specified for SUT compatibility.

Prior to interface development, signal transformations are identified. The current design includes transformation of Ethernet signals to and from 4-20 mA analog and 0-5, 10 and 24V digital signals. Therefore, the selected interface sub-system accommodates both process variable media.

The selected sub-system for the interface is a National Instruments (NI) computer based data acquisition (DAQ) system and custom NI LabVIEW Virtual Instrument (VI) developed using the G programming language. The interface supports both connectivity media. The NI LabVIEW G programming environment is extensive. Additionally, support for various DAQ hardware platforms, processors and operating systems allows for easy transition between interface sub-systems.

Following development of the interface, verification suites are defined and extensive analysis of the interface is recommended prior to performing HIL simulations. Primarily, the interface must enable communication between the simulator and the SUT. If a method for variable transfer cannot be established an alternate interface sub-system is required.

Acquiring variables from the simulator relies on the simulator architecture. If the simulator platform is known prior to interface development it is suggested that the runtime capabilities of the simulator be evaluated to determine if process variable extraction and real-time simulation is possible.

When the capability to transfer variables from the simulator to the SUT is established, verification and testing routines are developed. Analysis of the interface, without an SUT installed, is performed to identify timing requirements within a simulation platform. Identifying the limitations of the interface is not necessary, but assists in extending the usefulness of the interface beyond a single application. To proceed with application specific HIL simulation the interface must perform suitably during these verification procedures.

3.2 Hardware-in-the-loop Interface Development

The development platform is specifically used for interface development. It is designed with standardized units and is modular. The platform is flexible and enables the installation of a range of unique software simulators, I/O configurations, execution intervals and industry standard electronic control systems as SUTs. To establish modularity, the development platform is composed of three major components:

- *mock simulator* - Ethernet and UDP/IP capable computer platform,
- *Ethernet signal transceiver module* - A module capable of generating a sinusoidal waveform. The waveform simulates process variables normally generated by the model within a software simulator. The transceiver enables transmission of the generated signals through UDP/IP sockets and Ethernet media.
- *interface: physical adaptor and signal converter* - computer based NI DAQ workstation and NI LabVIEW VI which provide hardware interface and software conversion between Ethernet (engineering units) and hard-wired signals (analog and digital signals).

There is no SUT installed within the development platform. In the following descriptions an SUT is referenced. However, a simple hard-wired analog feed-back loop is implemented. The analog loop is installed between a single output channel and an input channel within the DAQ. No SUT is not installed so that delays associated with the interface are isolated from delays induced by the SUT.

3.2.1 Mock simulator

The installed simulator is installed on a Hewlett Packard Tru64 dual processor Unix server. No plant specific processes are modelled within the simulator, thus it is referred to as a mock simulator. The simulator is simply an Ethernet signal transceiver and data capturing device. The signal transceiver is a C program executed at a constant configurable interval (e.g. 200 ms) on the mock simulator. The program supports the generation of instantaneous sinusoidal waveform signals. Generated signals are transmitted through the HIL platform to observe potential simulation delays and reveal other limitations. An Ethernet controller is installed and operates in 100 Mbps (mega bits per second) full duplex mode to connect to the interface through Ethernet media.

Ethernet communication is accomplished through user datagram protocol/Internet protocol (UDP/IP) communications. UDP/IP is a standard for communication of data between electronic systems commonly used in the information technology industry. Another common protocol is the transmission control protocol/Internet protocol (TCP/IP). Due to the popularity of TCP/IP, the four basic differences between the two protocols and reasons for UDP/IP selection are explained below.

- *reliability* - UDP/IP transmissions do not have handshake, retransmit or timeout functions. This reduces communication reliability, but enables more deterministic transmission a valuable asset during simulation. Retransmitted variables introduce added delays, a problem which is reduced in UDP/IP communication. Also, simply monitoring the number of packets sent and comparing the total against the number received in post process analysis can provide insight into the reliability of the UDP/IP communication negating the need for the overhead processing of TCP/IP.
- *transmission order* - UDP/IP does not support the queueing of transmissions for receipt within the system. If two packets are transmitted in order, the order at

which they are received is not predictable. Within the interface, an identifier is implemented on the application layer of the virtual instrument and transceiver modules to identify process variables during transmission. Therefore, this function of TCP/IP is negated.

- *compactness* - UDP/IP transmissions are establish using efficient techniques in a compact transport layer with minimal overhead. Socket connections require little to no transmission for configuration. Also, congestion and traffic monitoring are eliminated.
- *whole packet architecture* - Finally, UDP/IP messages are guaranteed to be transmitted in whole packets. TCP/IP supports merging and splitting of packets which may interfere with the variable conversion procedures used during communication.

3.2.2 Ethernet signal transceiver

The Ethernet signal transceiver is a C program. The program enables communication between the simulator and the interface by opening and closing UDP/IP sockets, generating a sine waveform, transmitting and receiving instantaneous signal values through UDP/IP sockets. Properties of the generated sine wave are configured through three parameters, sampling frequency, wave frequency and number of samples. The amplitude of the sine wave is configured to cover the entire 4 to 20 mA range of industrial analog control signals. A sinusoidal signal is used as it emulates the continuous nature of process signals in a NPP. Sample sine waves are illustrated in Figure 3.3.

During development, the transfer of process variables between the interface and the mock simulator is controlled entirely by the C program. Therefore, a UDP/IP packet structure is designed to accommodate the bi-directional communication, to and from the SUT, required during HIL simulation.

3.2.2.1 Sub-system Synchronization

A unique feature of the proposed HIL simulation platform is the full control of the transmission of process variables. The ability to control the scheduling of transmissions between the SUT and the simulator provides the benefit of not having to poll through received variables. For example, assume n variables are being transferred.

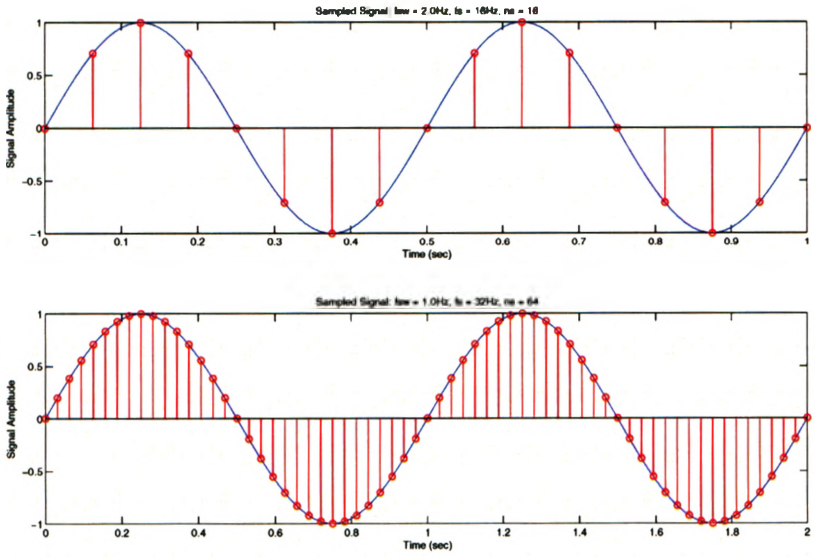

Figure 3.3: Sample sine wave generation with parameters: *top - waveform frequency = 2.0Hz, sampling frequency = 16Hz, number of samples = 17; bottom - waveform frequency = 1.0Hz, sampling frequency = 32Hz, number of samples = 65.*

Transmission to the SUT is easily scheduled by the mock simulator. During every execution interval, all n variables are transmitted. However, the receipt of variables is not synchronized with the simulator execution interval. If controlled by either the interface, or the SUT, a portion, m, of the n signals could be returned in one execution interval, and another portion, $n - m$, in the next execution interval. This non-deterministic performance can affect the accuracy of the HIL simulation. Proper simulation requires that the simulator receives current state SUT outputs for the set of most previously transmitted SUT inputs. Therefore, a delay is executed between SUT input and output routines. This delay is referred to as the SUT delay, t_{SUT}.

In an ordinary HIL simulation platform, process models are executed on a sub-system which supports direct electrical signal production. In this case, control of variable transmission relies only on the synchronization of the simulator and the SUT. However, it is common that no synchronization is implemented. With no synchronization, the longest delay required between SUT input and output routine execution occurs if the SUT input variables are transmitted immediately following the execution of input signal acquisition routines on the SUT. In this case, the current SUT execution interval must finish executing and also the entire next SUT execution interval must also execute to be assured proper SUT output values. Assuming the SUT execution interval has no variance, an SUT delay of $2 \times t_{SUT_{EI}}$ must be implemented

where $t_{SUT_{EI}}$ is the execution interval for the control logic of the SUT

Introducing and additional interfacing device within the HIL simulation platform, as with the developed interface, complicates the variable transmission procedure. Assuming the interface runs asynchronously with a constant period of $0.25 \times t_{SUT_{EI}}$. The delay,t_{SUT}, required for the SUT to execute properly would have to be extended. The worst case scenario with the three systems executing asynchronously occurs when the SUT input UDP/IP packet arrives at the interface immediately following UDP/IP receipt routines. Again, the eventual generation of an electrical signal at the SUT input occurs immediately following the execution of input signal acquisition on the SUT. The SUT executes and SUT output values are produced. However, due to variances in timing, the return of the variables experience similar delays as the SUT input transmission. The result is $2 \times 0.25 \times t_{SUT_{EI}}$ for electrical SUT input generation, $2 \times t_{SUT_{EI}}$ for SUT control logic execution and $2 \times 0.25 \times t_{SUT_{EI}}$ for conversion from electrical to software variable and acquisition through UDP/IP by the software simulator. Therefore, a total delay of $4 \times t_{SUT_{EI}}$ is necessary. However, by using a return request routine, interface synchronization is achieved and the variables can be returned in a manner similar to the ordinary HIL simulation platform discussed previously.

The requirements for performing two system asynchronous HIL simulations are simple. First, the frequency of consecutive control system outputs for the specified process are identified. Then the capability of the SUT to produce outputs for current state SUT inputs at twice the identified output frequency is determined. By using a return request packet, SUT output transmissions can be scheduled through the interface. Therefore, specified SUT outputs and the deterministic operation of the HIL simulations are guaranteed. Further, variables received by the simulator can be identified and validated against the requested variable identified in the return request packet.

3.2.2.2 UDP/IP Packet Structure

To enable flexible and scalable transmission of controlled variables within the HIL simulation platform a unique UDP/IP HIL packet structure is adopted. The structure includes four components:

- *Controlled variable identifying integer (2 digits)*- responsible for identifying the process variable as a specific input, output or auxiliary variable in relation to the I/O of the SUT and the configuration of the interface. (range = 00 - 99)
- *Controlled variable divisor (4 digits)*- required for analog signal scaling conversions between engineering units (eu) and real-world analog electrical values (4-20 mA). The controlled variable divisor is derived as follows;

$$cvd = \frac{cv_{max} - cv_{min}}{20mA - 4mA} = \frac{cv_{range}}{16mA}, \tag{3.1}$$

where cv_{max} and cv_{min} are respectively the maximum and minimum expected value of the controlled variable and *cvd* is the controlled variable divisor. For digital signals the controlled variable divisor is equal to 1.000.

- *Controlled variable intercept (4 digits)*- required for translating signal values between engineering units and real-world analog electrical values. Similar to the divisor, the intercept is derived as follows;

$$cvi = cv_{min}, \tag{3.2}$$

where cv_{min} is the minimum expected value of an analog controlled variable and *cvi* is the controlled variable intercept. For digital signals, the controlled variable intercept is used to define the logical conversion between simulator and SUT binary, TRUE and FALSE, definitions.

- *Controlled variable value (8 digits)*- the instantaneous sinusoidal waveform value. The decimal point occupies one of the eight character positions. This method provides a maximum resolution of one ten-millionth of a controlled variable engineering unit. Resolution should be reviewed for all controlled variable engineering units (eu) to determine if it is adequate for specific applications. For example, with a 12-bit SUT, cv_{max} = 6eu and cv_{min} = 2eu, eu could represent kg/s, m, °C, etc., and the following would hold:

$$cvd = \frac{cv_{max} - cv_{min}}{20mA - 4mA} = \frac{6-2}{16mA} = 0.25\text{eu}/mA, \tag{3.3}$$

$$cvi = cv_{min} = 2. \tag{3.4}$$

The generic structure for all UDP/IP packets is illustrated in Table 3.1. All

ID (2)		**divisor (4)**					**intercept (4)**					**value (8)**							
0	1	0	.	1	2	3	0	1	.	2	3	0	1	2	3	4	5	6	7

Table 3.1: Generic (SUT input) UDP/IP packet structure.

ID (2)		**divisor (4)**					**intercept (4)**				
0	1	0	.	1	2	3	0	1	.	2	3

Table 3.2: Return request UDP/IP packet structure.

parameters within the packet are stored in a character array buffer and then transmitted through UDP/IP using the `sendto()` C function. An improved approach to transferring variables is suggested in Chapter 6.

Within the packet structure, the controlled variable identifying integer has three purposes. The first is to identify a controlled variable as either an SUT input or an output. The second is to verify proper requested output signal transfer. Finally, the identifying integer can be used to configure the interface through UDP/IP packets. This eliminates the requirement for external hardware programmers.

SUT input UDP/IP packets are transferred using the structure illustrated in Table 3.1. An SUT input is identified by a pre-configured block of identifying integers (e.g. 00-49) within the entire (00-99) range. If a packet is identified as an SUT input variable, a unique input terminal, or physical channel, on the SUT is accessed and the analog or digital electrical signal is generated after being converted using the accompanying divisor and the intercept.

When an SUT output is to be returned a return request packet is issued. The return request packet structure is illustrated in Table 3.2. This request packet enables the transmission of I/O throughout the interface controlled by the simulator. The return request packet initiates the acquisition of an SUT output from an analog or digital output terminal of the SUT. Similar to input variables, the output terminal is specified by the identifying integer. The divisor and intercept are included within the packet as the interface are used to convert the analog and digital signal values back to engineering units and compatible logic values. The output is then packaged for transmission, and returned to the simulator.

Finally, the returned UDP/IP packet structure is illustrated in Table 3.3. There

ID (2)		**value (8)**							
0	1	0	1	2	3	4	5	6	7

Table 3.3: Returned (SUT output) UDP/IP packet structure.

is no divisor or intercept included in the returned packet as signal conversion has already taken place. However, the identifying integer is included in the return packet to verify that the returned process variable matches the requested return process variable.

With the UDP/IP packet structure defined, the simplified process for transmitting and receiving signals using the Ethernet signal transceiver is as follows:

1. An instantaneous sinusoidal value is updated within the mock simulator.
2. The sinusoidal value is formatted to interface UDP/IP specifications.
3. The UDP/IP packet containing the id, value, divisor and intercept is transmitted to the interface from the mock simulator.
4. Various delays are implemented to satisfy SUT timing requirements.
5. A value return request packet containing id, divisor and intercept is transmitted to the interface.
6. Additional delays are implemented to satisfy interface timing requirements.
7. An id and value are received at the mock simulator.

3.2.2.3 Limitations

The 12-bit resolution of the SUT analog to digital converter imparts a limitation to the resolution of the engineering units and controlled variables res_{cv}. The resolution is calculated in the following;

$$res_{analog} = \frac{16mA}{2^{n_{bit}}} = \frac{16mA}{2^{12}} = 0.00390625mA, \tag{3.5}$$

where n_{bit} is the number of bits available for analog to digital conversion. Subsequently, the observed controlled variable resolution is calculated as follows;

$$res_{cv} = 0.00390625mA \times \frac{0.25\text{eu}}{mA} = 0.00146484\text{eu}. \tag{3.6}$$

The controlled variable resolution constrained by the 12-bit analog to digital conversion process within the SUT is evaluated against the constraints of the UDP/IP packet structure:

$$0.00146484\text{eu} \gg 0.000001\text{eu}. \tag{3.7}$$

To summarize, the following relationship assures adequate resolution within the UDP/IP packet structure.

$$\frac{cv_{range}}{2^{n_{bit}}} \gg res_{\text{UDP/IP}}, \tag{3.8}$$

where $res_{\text{UDP/IP}}$ is the resolution imposed by the UDP/IP packet structure.

The UDP/IP packet structure enables an extensive controlled variable range $(cv_{max} - cv_{min})$. Maximum and minimum ranges are presented in Table 3.4. The limits are calculated for 4-20 mA signals at a resolution of 12-bits. Each of the rows in Table 3.4 represents a decimal point position within the eight digit controller variable value. The minimum limit for controlled variable range is found from (3.8), and is presented in the following;

$$\min(cv_{range}) = res_{\text{UDP/IP}} \times 2^{n_{bit}} \tag{3.9}$$

Maintaining a margin between the implemented minimum and this calculated minimum is recommended. In Table 3.4, all maximum ranges exceed the minimum ranges of the controlled variable with a lower resolution. For example, with a resolution of 00.00001 the maximum range is 99.99999. However, with the resolution 000.0001 the minimum range is 0.4096. Therefore, a simulation that includes a range from 0.0004096 to 100,000,000 is possible, with at least seven significant digits.

The intercept of the process variable is restricted to a constant maximum of 100eu and a minimum of -10eu. Further, the divisor and the intercept have resolutions of one one-thousandth and one one-hundredth respectively. However, these limitations can be easily modified for alternate applications. Future work includes expanding the range of the divisor, intercept and controlled variable value using byte-wise UDP/IP transmission packets.

The interface is expected to respond on demand to UDP/IP packet transmissions. For the interface to perform similar to standard HIL simulations with built in

Decimal Position	Minimum Range	Maximum Range
XXXXXXXX	4096	100,000,000
XXXXXXX.	4096	10,000,000
XXXXXX.X	409.6	1,000,000
XXXXX.XX	40.96	100,000
XXXX.XXX	4.096	10,000
XXX.XXXX	0.4096	1,000
XX.XXXXX	0.04096	100
X.XXXXXX	0.004096	10
.XXXXXXX	0.0004096	1

Table 3.4: Controlled variable ranges by decimal position for 16 mA analog signals at a 12bit resolution.

I/O capabilities, the amount of time for the interface to respond to a return request packet must be minimized. This time will be evaluated during the included analysis performed in Chapter 4.

During interface analysis, data collecting and timing routines are added to the signal transceiver module within the mock simulator. A timer is initiated prior to transmitting process variables. The elapsed time is then calculated and captured following receipt of the instantaneous sinusoidal signal. The entire time for all signals to be transmitted is captured. Additionally, the data collection routine includes the capability to capture events both prior to, during and following signal transmission. Therefore, SUT inputs, outputs and other variables within a single execution interval can be captured. The data is stored in a comma separated value (CSV) database on the hard drive of the simulator.

3.2.3 Hardware-in-the-loop interface device

To convert electrical signals to and from software variables NI PCI-6704 and PCI-6071E DAQ cards are installed on a computer workstation and a NI LabVIEW VI is created. The PCI-6704 DAQ card supports 16 voltage outputs, 16 current outputs and eight (5V TTL/CMOS) digital I/O lines, whereas the PCI-6071E DAQ card supports 64 analog voltage/current input, two analog outputs and eight digital (5V TTL/CMOS) I/O lines. To communicate through UDP/IP with the mock simulator, an Ethernet controller is used. Similar to the mock simulator, this Ethernet connection supports 100 Mbps full duplex operation. Further, to minimize network delays, com-

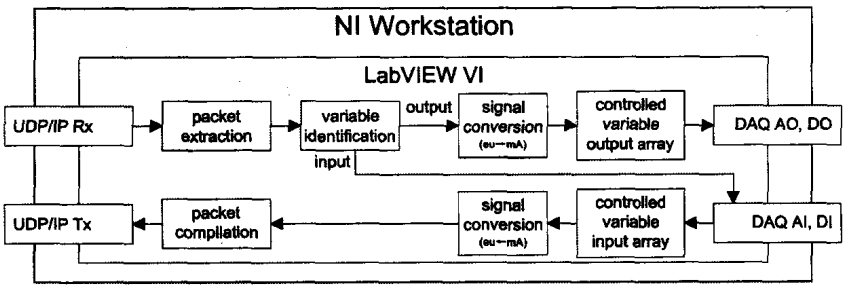

Figure 3.4: LabVIEW virtual instrument process flow.

munication between the mock simulator and the NI workstation is achieved through a wired crossover point to point topology. Point to point topology eliminates the delays associated with interfering network traffic.

The process of transferring and converting signals within the interface VI is illustrated in Figure 3.4. This process is consistent over all HIL simulations including development routines. However, slight modifications for a number of I/O must be made. Within the VI, UDP/IP packets are extracted to a string using standard LabVIEW, G programming language communication blocks. The string is then segmented into process variable identifying integer, divisor, intercept and signal value. The signal values are then converted using the accompanying divisor and intercept (Figure 3.4: signal conversion) from engineering units to analog signal values (4-20 mA). The conversion is demonstrated in the following equation;

$$cv_{mA} = \frac{cv_{\text{eu}} - cvi}{cvd} + 4mA \tag{3.10}$$

Again, digital signals are converted using the unitary divisor 1.000. Further, for digital signals the intercept identifies the conversion between TRUE and FALSE logic. For example, the simulator may require a TRUE value of 0 and a FALSE value of -1. However, the SUT recognizes TRUE as +1 and FALSE as 0. Therefore, the digital value is converted to the correct value of TRUE or FALSE within the VI. Following conversion, the values are inserted into the correct index within the controlled variable array. Each index within the array specifies the unique I/O hard-wired physical channel upon which the controlled variable will be generated. The array is output to the PCI-6704 DAQ card. Analog and digital signals are generated and acquired by the SUT on appropriate physical input channels.

The reverse process is performed for SUT outputs when signals are returned to the mock simulator. However, the acquisition of the controlled variable from the SUT

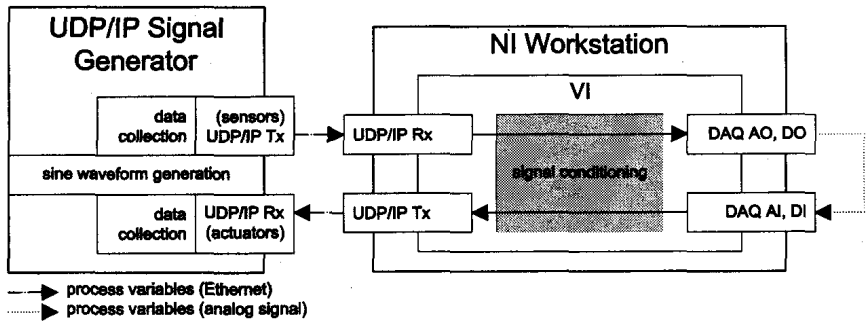

Figure 3.5: HIL simulation evaluation platform.

must be requested as per the UDP/IP return request packet procedure. Following a delay to allow SUT execution, a controlled variable return request is issued by the mock simulator and the appropriate SUT output signal is acquired by one of the DAQ cards. The controlled variable is then converted to engineering units or simulator compatible logic values. This conversion is presented in (3.11). Following conversion the value and variable identification are transmitted through UDP/IP to the simulator within the returned variable UDP/IP packet structure.

$$cv_{\text{eu}} = (cv_{mA} - 4mA) \times cvd + cvi \tag{3.11}$$

3.2.4 Hardware-in-the-loop analog loop simulation platform

The path for communication between each of the components within the interface development platform is illustrated in Figure 3.5. The arrows represent signal transmission from the software signal transceiver through the interface, including the analog feed-back circuit and the return path to the mock simulator. This analog loop-back is used to study the effects that the developed interface introduces during HIL simulations. In Chapter 4, two studies are performed using the analog loop-back circuit. In both studies, the availability, timing, signal bias and other interface performance metrics are evaluated.

Chapter 4
Hardware-in-the-loop Interface Analysis

The developed interface device must be evaluated prior to being installed in an HIL simulation platform. Therefore, two studies are performed. The first study, or timing study, evaluates the effects of pre-determined execution intervals, SUT delays and return request timeouts, or HIL timeouts, on the performance of the interface. In evaluating these three characteristics, a guarantee of the accuracy of the HIL simulations, and proper signal transfer, is made.

The second study, or scalability study, reveals the effects of various I/O configurations on the interface. Also, the study evaluates the performance of the interface over a range of SUT delays and introduces an additional delay to allow for multiple variable transmissions. A set of equal numbers of I/O as well as maximum achievable input and output configurations are investigated. In performing this study a range of acceptable I/O configurations is determined.

4.1 Hardware-in-the-loop Interface Timing Analysis

4.1.1 Method

The previously outlined Ethernet signal transceiver module is modified to automate the analysis of the developed interface. During timing analysis interface response is evaluated against a pre-defined set of values for three parameters. The three parameters, described below, include execution interval, the delay required to execute SUT control logic appropriately and the delay required for the interface to respond to a return request. One C module is developed for each simulator execution interval.

1. *execution interval, t_{EI} (200, 100, 50 ms)* - the period of time between subsequent C program executions on the software simulator. The execution interval

is equivalent to the period of time between consecutive SUT input or transmissions or SUT output receipt requests.

2. *SUT delay, t_{SUT} (0, 10, 20 ms)* - the amount of time required for the SUT to generate a set of outputs for the most previously transmitted inputs. Further, the t_{SUT} encompasses all transmission and signal settling delays. For example, when an SUT is installed within the analog loop-back circuit, t_{SUT} enables proper execution of the SUT control logic prior to issuing an SUT output return request.
3. *HIL timeout, t_{HIL} (1, 2, 3, 4, 5, 10 ms)* - the maximum amount of time to wait for a UDP/IP packet to return from the interface after a return request is issued. This parameter corresponds to the timeout parameter of the `poll()` function common to multiplexing input events during UDP/IP communications in Unix. When a packet is received before the timeout, the program exits the polling routine and continues processing the received data. If the `poll()` function expires no data is received and the interface is considered unavailable.

A single I/O pair is configured for the timing study. Execution interval, t_{EI}, HIL timeout, t_{HIL}, and SUT delay, t_{SUT}, are evaluated against each other and 54 sets of data are produced. Each of the sets of data includes 3000 transmissions or 10, 5 and 2.5 minutes of simulation for 200, 100 and 50 ms execution intervals respectively. Parameters t_{EI} , t_{HIL} and t_{SUT} are measured in thousandths of a second and have a resolution of approximately 1 ms. Further, the evaluation is performed over multiple iterations to confirm data accuracy and the ability to replicate the simulation.

It is expected that timing criteria for packet transmission and basic limitations of the interface can be observed and discussed in evaluating the interface over a range of execution intervals t_{EI} , t_{HIL} and t_{SUT} values. The interface requires time to transfer variables to and from the SUT. Without acknowledging and compensating for the delays introduced by the interface, simulation results are misleading and ultimately invalid.

The source code of the timing analysis C module is provided in Appendix C. For immediate and simplified reference, the general flow of the modified Ethernet signal transceiver module is presented in pseudo code below.

begin

for: *each* t_{EI} - C module is loaded (200, 100, 50 ms),

- *Initialize global variables* - socket connection, data collection, etc...,
- *Open UDP/IP sockets* - outgoing/incoming,
- *Generate sine waveform* -

$$n_s = 1000, f_s = 1\text{kHz}, f_{sw} = 1\text{Hz}, \tag{4.1}$$

where n_s is the number of samples, f_s is the sampling frequency and f_{sw} is the frequency of the sine waveform,

for: *each* t_{HIL} - loop (1, 2 ,3, 4, 5, 10 ms),

for: *each* t_{SUT} - loop for 3000 samples (0, 10, 20 ms),

- *Initialize local variables* - packet buffer, controlled variable, etc...,
- *Sine waveform* - obtain current sample and increment pointer,
- *Start timer* - capture current timer value,
- *Controlled variable Tx* - format and transmit the controlled variable, id, divisor and intercept through outgoing UDP/IP socket,
- *Controlled variable store* - capture the transmitted controlled variable,
- *Sleep* - current t_{SUT},
- *Flush UDP/IP socket* - clear the incoming UDP/IP socket,
- *Request controlled variable return* - transmit id, divisor and intercept through outgoing UDP/IP socket,
- *Wait for interface* - poll incoming UDP/IP socket (current t_{HIL}),

if: *UDP/IP packet detected*

- *Controlled variable Rx* - receive data on incoming UDP/IP socket and extract controlled variable and id,

if: *received ID equal to requested ID*

- *Controlled variable store* - capture the received controlled variable,

else: *Invalid ID* - capture invalid ID flag,

else if: *poll timeout* - capture timeout flag,

- *Stop timer* - capture current timer value, store timer difference and increment data-set pointer,

end for

end for

- *Output collected data* - write to CSV file with all captured data.
- *Wait* - perform other simulation tasks prior to next t_{EI}.

end for

end

4.1.2 Assumptions

During the timing analysis no SUT is installed. This is intentional. Yes, SUT delays are imposed to observe the effect an SUT has on the interface. However, the evaluation should be independent of the delays that may be present with an SUT installed. When an SUT and software simulator sub-platform are selected, the verification procedure can be repeated to identify any discrepancies the SUT installation may cause.

Throughout all studies it is assumed that the workstation on which NI DAQ system and NI LabVIEW VI are installed does not experience computational disturbances. Some measures have been taken to assure no external disturbances, including Microsoft Windows XP tasks, occur. These measures are necessary as tasks on the workstation can significantly affect the performance of the interface. Best practise includes disabling all scheduled background activities on the Microsoft Windows XP operating system (OS). Further, the LabVIEW VI has been configured to run at the highest priority. However, during observation, computational disturbances may be referenced as the cause of specific performance deviations. A thorough analysis will improve understanding of the influence that the Microsoft Windows XP OS has on HIL simulation when used as a platform for an HIL interfacing device.

Finally, during analysis the complete reproduction of the transmitted sinusoidal signal on a physical channel is not necessary. Therefore, the frequency of transmission and the frequency of the transmitted signal do not need to satisfy sampling theory fundamentals. Further, timing of the digital to analog and analog to digital converters within the NI DAQ system are assumed negligible. The instantaneously generated

sinusoid values are only required to provide changing values in each execution interval or for each I/O throughout the entire timing and scaling analyses.

4.1.3 Analysis of results

Three properties are monitored during the timing analysis. These include; elapsed time, transmitted controlled variable value and received controlled variable value. Elapsed time is calculated as the difference in time between the transmission of the SUT input from the software simulator to the receipt of the SUT output within the software simulator. The resolution of the elapsed time measurement is 1 ms. Transmitted and received controlled variables do not require calculation. The SUT input variable value is stored within the transmitted controlled variable field of a database as it is received on the UDP/IP socket. The same is true for the SUT output variable which is stored in the received controlled variable field of the same database. The resolution of captured variables is limited by the UDP/IP protocol discussed in Section 3.2.2.

Two error flags are identified. The two flags indicate communication errors within the interface. The first error flag, or no data flag, is produced if during receipt of a controlled variable from the SUT there is no data received on the incoming UDP/IP socket. This error corresponds with a `poll()` timeout expiration. The second error flag, or invalid id flag, indicates discrepancy between the sent and received identifying integers. When this error occurs, the returned variable from the interface does not match the variable requested by the return request UDP/IP packet. In the event of either error flag, the received data is invalid. A measure of the availability of the interface during analysis provides assurance as to whether the simulation is compromised.

All properties and flags are stored within a CSV database. A database record is created for every SUT input variable that is transmitted. Therefore, one record for each of 3000 execution intervals within all SUT delay/HIL timeout combinations is created. All calculations are performed using both Matlab and Microsoft Excel tools. All plots are generated using MatLAB.

The following results provide analysis of the data collected during interface development. The figures generated present a summary of the data collected. Every effort is made to present relevant illustrations of the data which assist in determining

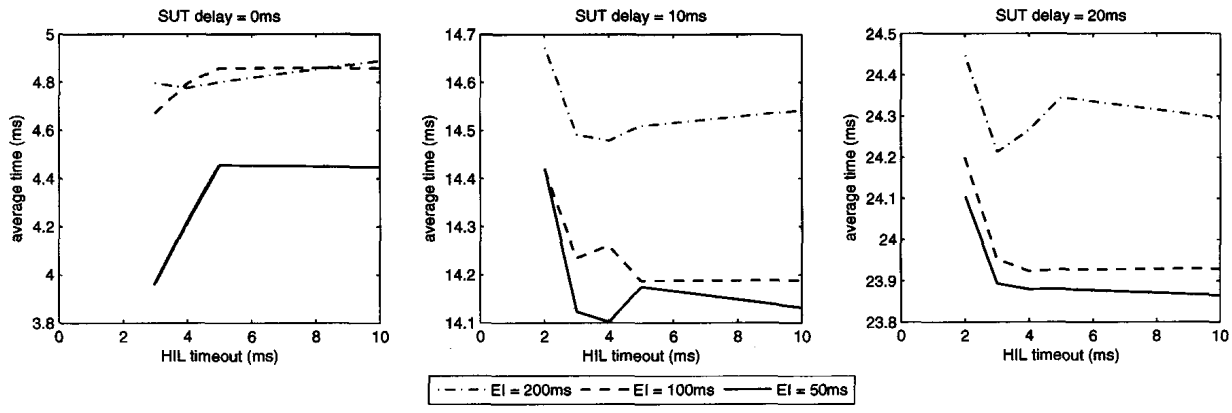

Figure 4.1: Average signal transmission time vs. HIL timeout per SUT delay.

the limitations of the developed interface. Following each figure, table or result is a discussion of the observations made from the material presented. Discussion includes; explanation of results; deduction, general application of the results; and hypothesis, a possible conclusion arising from the results.

Figure 4.1 includes three graphs. The graphs illustrate the average elapsed time for single controlled variable transmission. Each of the graphs corresponds to an SUT delay, from left to right, 0, 10 and 20 ms respectively. During every execution interval the time for the sinusoid signal to be transmitted through the entire interface, from SUT input to SUT output, is measured. The elapsed time is accumulated over all 3000 execution intervals. From this total, the average elapsed time is calculated. From Figure 4.1 the following is observed:

- A 50 ms execution interval results in the shortest average elapsed time for all configurations.
- A 200 ms execution interval results in the longest average elapsed time.
- A100 ms execution interval results in similar average elapse times as the 50 ms execution interval for a 10 and 20 ms SUT delay. However, for a 0 ms SUT delay, the average elapsed times are similar to the 200 ms execution interval.
- 5 ms and longer HIL timeouts result in consistent average elapsed times. However, HIL timeouts shorter than 5 ms results in varying average elapsed times.
- There is a strong relationship between SUT delay and the average elapsed time. However, for all configurations a 4-5 ms delay is observed beyond the value of SUT delay.

It is expected that average elapsed time will increase when a shorter execution interval is used. This expectation is due to an anticipated reduction in processing time available for transmission in the software simulator and in increased processing required by the interface. However, the opposite is observed. The 50 ms execution interval results in the shortest average elapsed time. Further, the 200 ms execution interval results in the longest average elapsed time. From literature, UDP/IP communication transmission performance is known to vary when the time between consecutive transmissions is modified [71]. Therefore, a shorter average elapsed time is the result of shorter execution intervals which keep the communication sockets poised for packet transmission, avoiding Ethernet controller power saving sleep and network maintenance routines. Also, an aspect of the 50 and 100 ms execution interval causes the distinctly shorter average elapsed time over 3000 execution intervals. The main difference between the evaluated values of execution interval is the time between execution of the controlled variables. Therefore, shorter time between subsequent transmissions may ultimately result in shorter transmission times.

Average elapsed time varies with SUT delay for a 100 ms execution interval. SUT delay is not expected to have any influence on the average elapsed time beyond the SUT delay itself. However, observations indicate that the 100 ms execution interval requires a distinctly longer average elapsed time with a 0 ms SUT delay than with larger SUT delays. With a longer SUT delay, the time between transmission of an SUT input and the request for an SUT output is longer. Therefore, a longer execution time is required by the C module. On the contrary, for a 0 ms SUT delay the total execution time required by the C module is minimal. When using a 100 ms execution interval to execute the required minimal execution time of the C module with 0 ms SUT delay, the execution of the C module may not be performed at as high a priority as with 20 ms SUT delay. This is assuming that the software simulator assigns priorities during module execution. A higher priority would be required when a 20 ms SUT delay is implemented with 100 ms execution interval as less time is available for additional processing within the execution interval. Therefore, slight variances in C module priority scheduling may result in varying average elapsed times.

A strong proportional relationship between the SUT delay and the average elapsed time is expected. The largest component of the average elapsed time is the SUT delay. Compared with this delay, UDP/IP transmission delays are negligible. Therefore, as the SUT delay is increased, the average elapsed time grows longer.

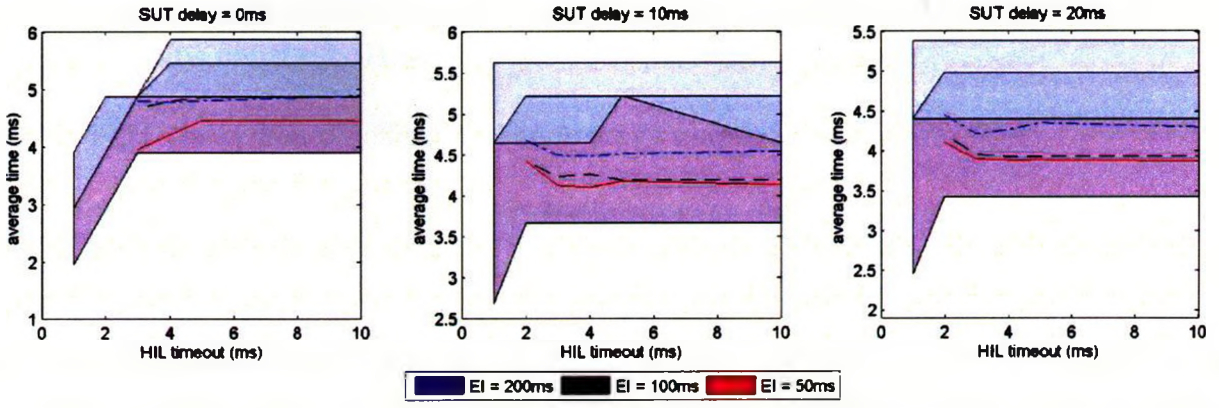

Figure 4.2: Average time with 95th percentile vs. HIL timeout per SUT delay.

Further, an additional 4-5 ms delay is observed. This delay is largely due to the HIL timeout. Though the HIL timeout is configured for 10 ms, when a UDP/IP packet is received the `poll()` function does not expire entirely. It is apparent from Figure 4.1 that the `poll()` function is satisfied within 4 to 5 ms on average. Therefore, the average attainable elapsed time for the developed interface occurs 4 to 5 ms greater than the SUT delay. If a shorter average elapsed time is required for a specific process, the interface must be redesigned.

For short HIL timeouts measured average times vary slightly. A short HIL timeout reduces the amount of time that the software simulator will wait for a requested variable to be returned. If the wait time, or HIL timeout, is reduced beyond a certain level, no SUT output variables will be received at the software simulator. This happens because the interface responds at an average rate to return request packets. For a short HIL timeout, less packets are successfully received. This can affect the average calculation of observed interface properties including average elapsed time.

The review of the elapsed time required by the developed interface is extended in Figure 4.2 which includes three graphs. The graphs illustrate the 95th percentile time for single controlled variable transmission. From Figure 4.2 the following is observed:

- The 95th percentile measurements are consistent over all valid average elapsed times, including those below 5 ms.
- Average elapsed times for each SUT delay beyond the SUT delay are within 1 ms of each other.

- For all execution intervals the lower 95th percentile of the average elapsed time remains constant. However, the 95th percentile increases.
- SUT delay has no effect on the average elapsed time range.

It is expected that inconsistencies in average elapsed times observed for shorter than 5 ms HIL timeouts are due to a reduced number of successful transfers through the interface. However, the inconsistencies could be caused by a varying range of average elapsed times due to SUT delay inaccuracies or other unknown factors. Evaluation the 95th percentile average elapse time helps identify the true cause. The 95th percentile are consistent over all valid HIL timeouts. Therefore, inconsistencies are not the result of an increased range of average elapsed times.

The 95th percentile average elapsed times are consistent over all HIL timeouts indicating that the variances observed between execution interval configurations and for HIL timeouts below 5 ms are not caused by extreme values. Actually, the variance between execution intervals is due to a larger range of measurements as the lower 95th percentiles are equal for all execution intervals. The relationship between the 95th percentile average elapsed time and the execution interval indicates that for a longer execution interval the range of average elapsed times is increased. This relationship is further evidence that processing priority is given to C modules. Those modules which include code requiring nearly the entire execution time provided in the execution interval are executed at higher priority. In previous discussion it was determined that the execution interval restricts the execution time of the C module. For 100 ms and 200 ms execution intervals a lower priority must result in interruption or a slight delay or offset during the SUT delay or UDP/IP packet polling process.

The time between 95th percentile for each SUT delay is equal. Therefore, the deterministic ability of the interface to respond within a given time does not rely on the SUT delay. Further, the average elapsed time for each of the three SUT delays are within 1 ms of each other. Therefore, timing requirements for the transmission of variables can be made from these observations. The minimum average time occurs when the SUT delay is 0 ms. In this configuration the average value is greater than 4 ms. Also, it is clear that the 95th percentile is covered within 6 ms of the SUT delay for each execution interval and HIL timeout. Therefore, an HIL timeout of at least 5 ms is required for the majority of transmissions to be completed successfully and 6 ms to cover the 95th percentile.

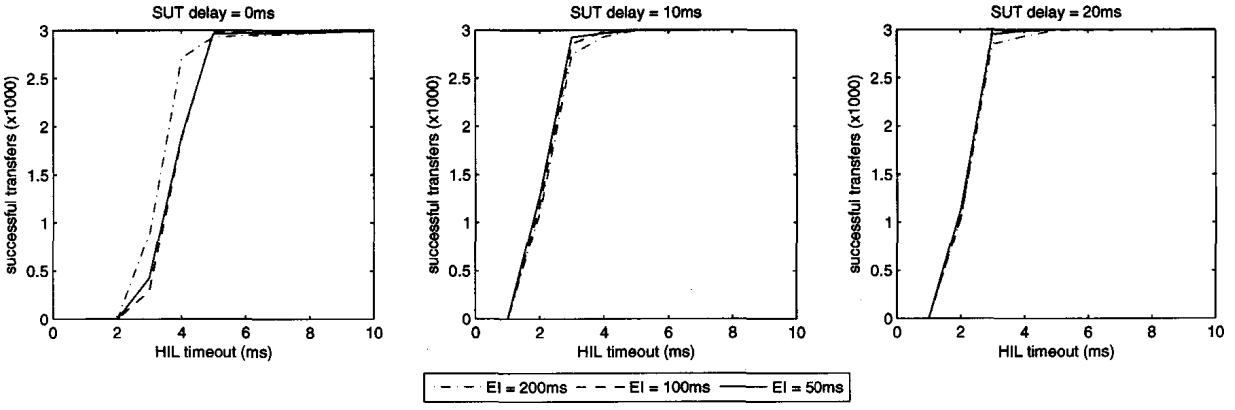

Figure 4.3: Successful transfers vs. HIL timeout per SUT delay.

Figure 4.3 presents the number of successful transmissions through the interface over 3000 execution intervals for each SUT delay. The number of successful transfers is the total number of transfers which do not produce either error flag. From Figure 4.3 the following is observed:

- Below a 3 ms HIL timeout the number of successful transfers drops rapidly.
- For all SUT delays a greater than 5 ms HIL timeout results in very high percentage of successful transfers.
- A 0 ms SUT delay results in a reduced number of successful transfers for low HIL timeout configurations.

The inconsistencies in the average elapsed times observed in Figure 4.1 are supported by the observed rapid drop in successful transfers. It is apparent that a reduced number of successful transfers results in a significant loss of accuracy in average elapsed time calculations. The observed drop in successful transfers is expect as a shorter HIL timeout will result in a reduction of incoming packet detection. In fact, for a 0 ms SUT delay there are no successful transfers below a 2 ms HIL timeout. Additionally, for 10 and 20 ms SUT delays there are no successful transfers at the 1 ms HIL timeout.

The very high successful transfer rates observed above an HIL timeout of 5 ms provide assurance that the interface is capable of performing adequately for single variable simulations. This observation is expected as with only a single variable transmitted, the interface will wait for a return request packet without being disrupted. Table 4.1 reveals the numerical percentage of successful transfers illustrated in Figure

		Successful Transfers (%)		
		EI		
SUT Delay	HIL Timeout	50 ms	100 ms	200 ms
0 ms	1 ms	0	0	0
	5 ms	98.8	99.4	97.7
	10 ms	99.7	99.9	100
10 ms	1 ms	0	0	0
	5 ms	99.8	99.9	99.9
	10 ms	100	99.9	100
20 ms	1 ms	0	0	0
	5 ms	99.9	99.9	99.8
	10 ms	100	100	99.9

Table 4.1: Percentage of successful transfers.

4.3. These percentages provide a measure of certainty for HIL simulation variable transmission.

For a 0 ms SUT delay, a reduction in successful transfers is observed. This reduction is of little concern as a 0 ms SUT delay will never be implemented when performing true HIL simulations. Also, given adequate HIL timeout, an SUT output is received at the UDP/IP socket and the success rate increases. With a 10 ms HIL timeout and 0 ms SUT delay, the percentage of successful transfers is greater than 99%. Therefore, the return receipt packet is received by the NI workstation immediately following receipt of the SUT input. However, the transmission of the SUT output is delayed. In addition, the success rates observed between the 10 and 20 ms SUT graphs are relatively constant. Therefore, increasing the SUT delay beyond 20 ms is not expected to provide higher success rates during single variable transmissions.

To achieve a near perfect successful transfer rate the HIL timeout can be set to 10 ms. However, if the `poll()` function is not satisfied the HIL timeout expires and may cause the simulation to become unstable. Therefore, it is important to determine best practise for HIL timeout selection. This is performed by monitoring the time required for the `poll()` function to be satisfied and comparing the actual time to the implemented HIL timeout. Further, other parameters of the interface should be examined. For example, if a small number of I/O are used, a larger HIL timeout may be satisfactory.

In Figure 4.4 the actual time the `poll()` function executes, or poll time, is

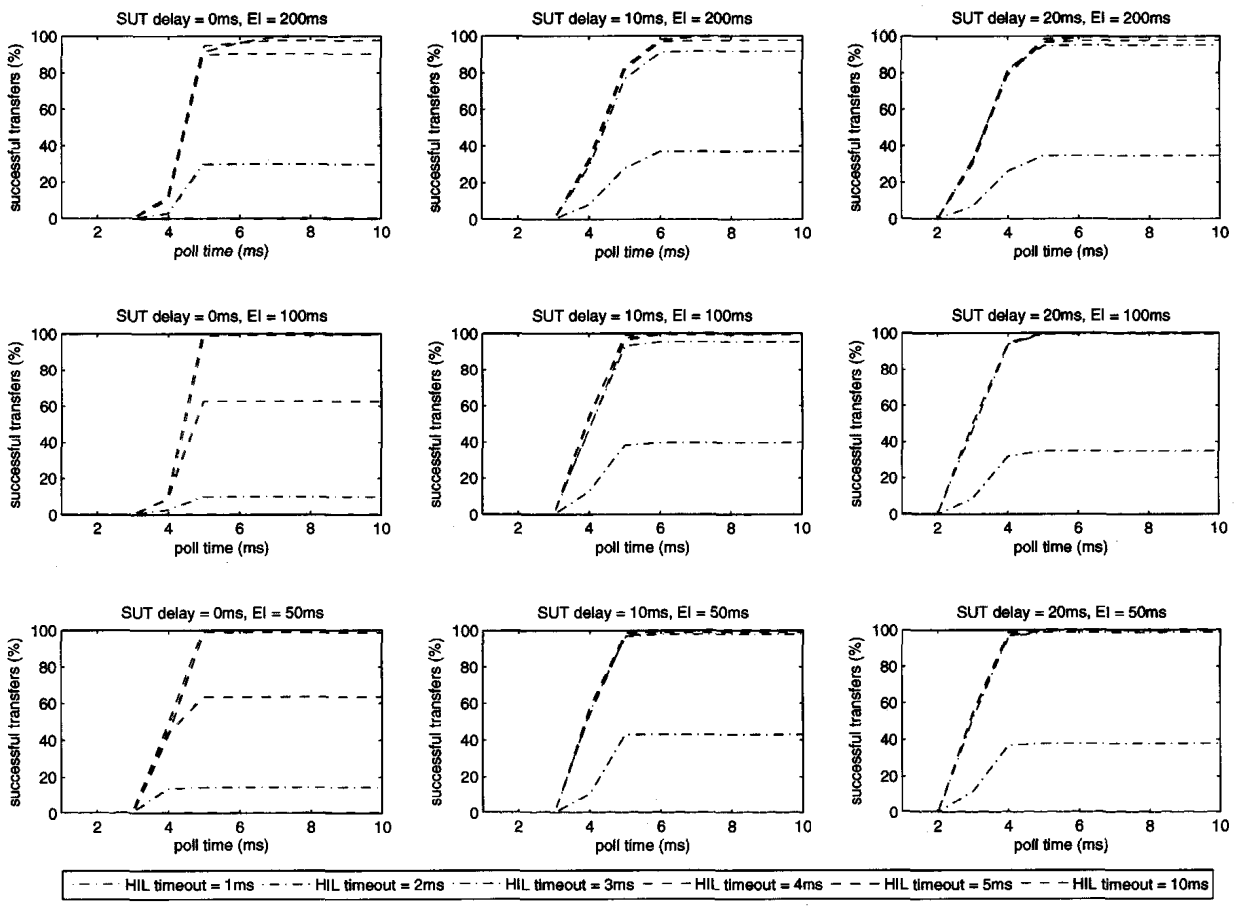

Figure 4.4: Cumulative successful transfers vs. poll time per SUT delay.

calculated by subtracting the SUT delay from the elapsed times. It is expected that a large portion of the remaining time is required by the `poll()` function. Figure 4.4 includes nine graphs which present the cumulative `poll()` successful transfers for each execution interval, SUT delay configuration. From Figure 4.4 the following is observed:

- For a 1 ms HIL timeout no successful transfers are recorded over all configurations. Consequently, there are no recorded successful transfers at the 1 or 2 ms poll time.
- A 0 ms SUT delay results in no successful transfers during a 2 ms HIL timeout. However, the 2 ms HIL timeout results in insufficient successful transfers over all configurations.
- A 0 ms SUT delay results in a reduced number of successful transfers for 3 and 4 ms HIL timeouts.

- All HIL timeouts continue to accumulate beyond their respective `poll()` function expiration. There are no successful transfers at 2 or 3 ms for the 2 or 3 ms HIL timeouts.
- An HIL timeout of greater than 5 ms performs adequately over all configurations.
- A shorter 50 ms execution interval results in a more consistent number of total successful transfers after 10 ms poll time.

It is clear that a 1 ms HIL timeout does not provide adequate time for the `poll()` function to detect a returning UDP/IP packet from the interface. Also, the 2 ms HIL timeout is very near to the critical point of signal receipt or detection. Similar to previous discussion, insufficient transfers are completed with a 2 ms HIL timeout for all SUT delays and 3 and 4 ms HIL timeouts for a 0 ms SUT delay. The 2, 3 and 4 ms HIL timeout plots plateau during some configurations due to the reduced number of successful transfers. With no SUT delay implemented, the interface does not respond to the return request packet and an error flag is generated as the `poll()` function is not satisfied. This is further evidence that an HIL timeout of greater than 5 ms is necessary to achieve adequate performance. In fact, using an HIL timeout of greater than 5 ms is observed to perform adequately over all execution intverals and SUT delay configurations.

UDP/IP packets can not be received after the `poll()` function has expired. The observed transfer accumulation following HIL timeout expiration is the result of the delays following the `poll()` function detecting the UDP/IP packet. Once detected, the packet must be obtained and parsed into identifying integer and returned variable value. Therefore, the poll time include delays beyond that of satisfying the `poll()` function. For example, The 1 ms HIL timeout for all SUT delays and 2 ms for the 0 ms SUT delay does not allow enough time for the `poll()` function to detect the incoming packet. However, with a 2 ms HIL timeout and 10 ms SUT delay transfers succeed beyond the 2 ms poll time expiration. This is evidence that the `poll()` function detects UDP/IP packet transmission but requires extra processing time. Other configurations exhibit similar performance. When a packet is detected before expiration there are extra delays that extend the actual receipt of the variable and the observed HIL transmission time by up to 4 ms.

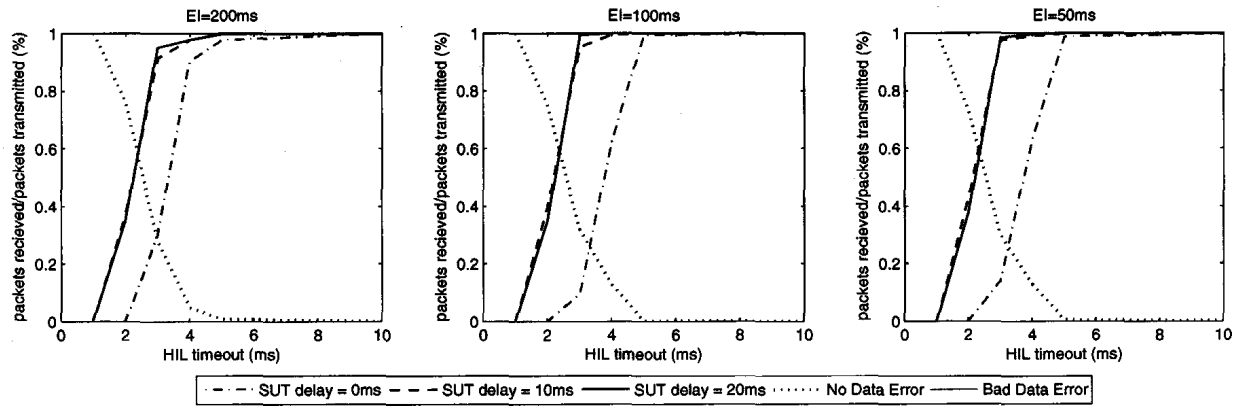

Figure 4.5: interface availability vs. HIL timeout per execution interval.

In previous discussion it was found that a 50 ms execution interval resulted in shorter average elapsed time and that shorter time between UDP/IP communication that results from a 50 ms allows packets to be transmitted more efficiently. When cumulative successful transfers are considered, the 50 ms execution interval again demonstrates efficient operation. The total number of successful transfers when a 50 ms execution interval is implemented is consistent over a larger range of HIL timeouts and SUT delays than other execution interval. Especially when compared to the performance of the 200 ms execution interval.

Error flags recorded during the study are used for analyzing the availability of the interface. This availability is calculated in the following equation and represents the percentage of properly received process variables through the HIL simulation platform;

$$av_{HIL} = \frac{n_{total} - n_{error}}{n_{total}}, \tag{4.2}$$

where n_{total} is the total number of attempts to transmit variables through the HIL simulation development platform, and n_{error} is the total number of error flags which are generated during the entire study. Further, the availability is simply the percentage of successful transfers illustrated in Figure 4.3. From Figure 4.5 the following is observed:

- It is very clear that a 0 ms SUT delay results in a reduced availability for low HIL timeout configurations.
- A 20 ms SUT delay produces slightly greater availability for medium range HIL timeouts than the 10 or 0 ms SUT delays.

- The unavailability observed with low HIL timeouts is almost entirely due to expiration of the `poll()` Function (no data flag).
- There are almost no instances where the returned variable does not match the requested variable (invalid id flag).

Similar to previous observations, Figure 4.5 reveals decreased availability when the SUT delay is 0 ms. The distinct difference between the availability of the 0 and 10 ms SUT delays indicates that the interface requires between 0 to 10 ms to execute signal transmission. Further, though a reduction in availability is observed, the availability during 0 ms SUT delay is only delayed by 1-2 ms. Therefore, the actual time required for the interface to respond at all to subsequent SUT input, SUT output requests is at least 2 ms over all configurations. However, the implementation of a larger HIL timeout resolves all transmission conflicts.

The availability of the interface is nearly perfect when the HIL timeout is greater than 5 ms. With adequate time to detect an incoming UDP/IP packet, very few transmission are lost due to the `poll()` function expiring. The percentages observed are documented in table 4.1 where it is evident that for shorter HIL timeouts availability is significantly affected by the no data flag as discussed previously. Further, for single variable transmission it is impossible for two variables to become mixed during transmission, therefore the number of invalid id flags reported is near 0%.

To evaluate the relationship between each SUT input and output value, the value of variables successfully transferred through the interface are captured. The capture procedure occurs prior to transmission and upon successful receipt of a variable and is intended to identify timing constraints regarding the settling time of the interface. Figure 4.6 includes three graphs which present the per unit average bias of the transmitted and received signal. The per unit value is derived in the following equation. Using this method, different sets of engineering units the measured biases are comparable;

$$\text{per unit} = \frac{cv_{eu_{tx}} - cv_{eu_{rx}}}{cv_{max} - cv_{min}}. \tag{4.3}$$

From Figure 4.6 the following is observed:

- A per unit average bias measurement of around 2.75×10^{-3} is achieved.
- For greater than 5 ms HIL timeouts the per unit average bias is consistent.

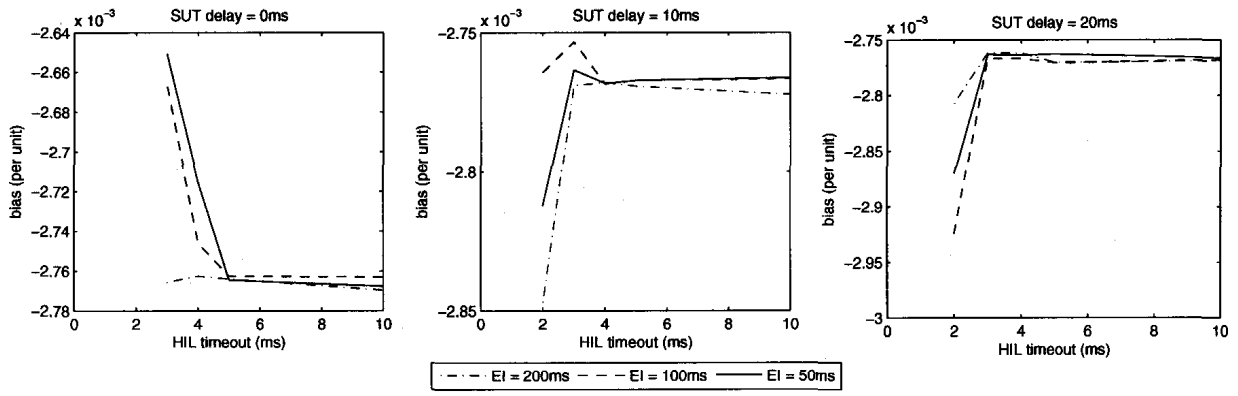

Figure 4.6: Average signal bias vs. HIL timeout per SUT delay.

- Even at HIL timeouts below 5 ms the average bias is acceptable. However, the average measured bias varies within these configurations.

The bias observed in Figure 4.6 is due to inaccuracies in component measurements, as well as calibration of the NI DAQ hardware. Biases can be compensated for prior to simulation if necessary. Inconsistent bias is only a concern if the inconsistencies exist between the evaluated configurations. If inconsistencies are observed, the SUT delay is not adequate for the analog or digital signal to settle. Therefore, the signal input to the SUT from the interface, or the signal input to the interface from the SUT may vary from the actual transmitted value. SUT or electronic control system documentation should be reviewed for expected analog signal settling time. This settling time should be taken into account when calculating the SUT delay t_{SUT} in the following;

$$t_{SUT} = t_{s_{HIL}} + 2 \times t_{SUT_{EI}} + t_{s_{SUT}}, \tag{4.4}$$

where $t_{s_{HIL}}$ and $t_{s_{SUT}}$ are the settling times for the interface and the SUT respectively, and $t_{SUT_{EI}}$ is the execution interval of the control logic on the SUT from Section 3.2.2.2. In Figure 4.6 the biases are very similar for different execution interval and SUT delay configurations.

In Figure 4.6 the per unit average bias for HIL timeout values greater than 5 ms are consistent for all SUT delays. The steady state bias is approximately -2.76×10^{-3}. The inconsistencies observed with lower HIL timeouts are not the result of insufficient settling time. Again, the cause is the reduced number of successful transfers observed in Figure 4.1 . The HIL timeout provides delay for interface response, and SUT output capture, conversion and transmission. The SUT output signal is captured

when the return request packet is issued prior to the HIL timeout delay. Therefore, the SUT delay is the only delay which is used to compensate for signal settling.

4.2 Hardware-in-the-loop Interface Scalability Analysis

4.2.1 Method

Similar to the timing analysis, the Ethernet signal transceiver module is again modified to automate the analysis of the developed interface. However, during scalability analysis HIL response is evaluated against a pre-defined set of values for two parameters. The two parameters, described below, include SUT delay and number of I/O. Also, an additional parameter and the maximum number of I/O ares reviewed at the end of the section . The reviewed parameter, also described below, is the transmit delay, or transmission delay (Tx delay). Many C modules are used throughout the evaluation. The C modules, or timing analysis programs are then executed automatically using a Perl scripts on the mock simulator.

1. *SUT delay (10, 25, 50, 100, 150 ms)* - the amount of time required for the SUT to generate a set of outputs for the most previously transmitted inputs. Further, the SUT delay encompasses all transmission and signal settling delays. During the scaling analysis the range of SUT delays is significantly larger than during timing analysis. These longer delays are used in real SUT installations and assist in evaluating interface I/O capacity limits. SUT delay is measured in thousandths of a second and has a resolution of approximately 1ms.
2. *Number of I/O (1, 5, 10, 25)* - during timing analysis only a single I/O combination was studied. Scalability analysis includes a range of I/O configurations. The number of I/O is the number of SUT inputs and outputs transferred through the interface. During scalability analysis the number of inputs is equal to the number of outputs. However, maximum input and output configurations are evaluated at the end of this section.
3. *Tx delay (0, 1, 5, 10 ms)* - during timing analysis, with 0 ms SUT delay and 1 or 2 ms HIL timeout no data is received. When the SUT delay is increased a 2

ms HIL timeout results in the receipt of data. It is apparent that there is processing time required by the interface to receive controlled variables. Therefore, when implementing multiple I/O the transmission delay is introduced between consecutive SUT input UDP/IP packet transmissions. Similar to the SUT delay, the transmission delay is also measured in thousandths of a second and has a resolution of approximately 1 ms.

SUT delay and number of I/O are evaluated against each other and 20 sets of data are produced. Each of the sets of data includes 2250 transmissions or 7.5 minutes of simulation with an execution interval of 200 ms. Again, tests are performed over multiple iterations to confirm data accuracy and the ability to replicate the simulation. For this study the HIL timeout and execution interval are constant throughout. An execution interval of 200 ms is implemented as it is the specified execution interval in the application study of Chapter 5. Further, a 10 ms HIL timeout is used. In Section 4.1.3 observation of the average and 95th percentile elapsed time revealed that a 10 ms HIL timeout demonstrated guaranteed coverage for SUT output transmission.

Two additional analysis are performed and are included at the end of this section. The first includes the implementation of the transmit delay into the timing routine. In this analysis transmission delay and number of I/O are evaluated against each other and an additional 20 sets of data are produced. These sets of data include 3000 transmissions, or 10 minutes of simulation with an execution interval of 200 ms. Again, tests are performed over multiple iterations to confirm data accuracy. For this study the HIL timeout, SUT delay and execution interval are constant throughout. The HIL timeout and execution interval are the same as used in the scaling analysis. The SUT delay remains constant at 25 ms. This time is slightly greater than the expected SUT execution interval implemented in Chapter 5.

The final analysis identifies stable operation for maximum SUT input only, and output only configurations. In this analysis either a single SUT input or a single SUT output is implemented for the maximum SUT output and input configurations respectively. The number of SUT inputs or outputs is then increased and transmission of a sinusoidal signal is performed over a period of five minutes. If the simulator does not abort the simulation process then the SUT input or output is considered a maximum. When subsequent larger SUT inputs or outputs succeed they are determined to be the maximum. This process is performed until the simulator aborts. Once more, tests are performed over multiple iterations to confirm data accuracy. For this

study the HIL timeout, SUT delay and execution interval are constant throughout and hold the same values as in the previous two studies. Further, no transmission delay is implemented.

It is expected that extended timing criteria and best practises, as well as connectivity limitations for the interface can be observed by evaluating the interface over a range of I/O configurations and SUT delays. Specific applications implemented in HIL simulations have different requirements for I/O capacity and SUT execution time. Without verifying the ability of the interface to support a range of I/O interfaces, including those used within the intended specific application, HIL simulation results are not reliable.

The entire source code for the scaling analysis C module is available in Appendix C. For immediate and simplified reference, the general flow of the Ethernet signal transceiver module is presented in pseudo code below. The presented pseudo-code is primarily used for the first scaling study identified by a :1 in the bullet. However, items that are unique to the transmission delay study include a :2 in the bullet. No pseudo code is included for the maximum I/O analysis.

begin

for:1 *each of the SUT delays* loop (10, 25, 50, 100, 150 ms),

for: *each of the I/O configurations* loop (1, 5, 10, 25I/O),

- *Initialize global variables* - socket connection, data collection, etc...,
- *Open UDP/IP sockets* - outgoing/incoming,
- *Generate sine waveform* -

$$ns = 1000, fs = 1\text{kHz}, fsw = 1\text{Hz}, \tag{4.5}$$

where ns is the number of samples, fs is the sampling frequency and fsw is the frequency of the sine waveform,

for:2 *each of the TX delays* loop (0, 1, 5, 10 ms),

- *Initialize local variables* - number of I/O, SUT delay, TX delay, packet buffer, controlled variable, etc...,
- *Start timer* - capture current timer value,

for: *current number of SUT inputs*

- *Sine waveform* - obtain current sample and increment pointer,
- *Controlled variable Tx* - format and transmit the controlled variable, id, divisor and intercept through outgoing UDP/IP socket,

if:1 Last SUT input

- *Controlled variable store* - capture the transmitted controlled variable,

if:2 Not last SUT input

- *Sleep* - current **TX delay**,

end for

- *Sleep* - current **SUT delay**,
- *Flush UDP/IP socket* - clear the incoming UDP/IP socket,

for: current number of SUT outputs

- *Request controlled variable return* - transmit id, divisor and intercept through outgoing UDP/IP socket,
- *Wait for interface* - poll incoming UDP/IP socket (HIL timeout = 10 ms),

if: UDP/IP packet detected

- *Controlled variable Rx* - receive data on incoming UDP/IP socket and extract controlled variable and id,

if: received ID equal to requested ID

if:1 First SUT output

- *Controlled variable store* - capture the received controlled variable,

else: Invalid ID - capture invalid ID flag,

else if: poll timeout - capture timeout flag,

if: Last SUT output

- *Stop timer* - capture current timer value, store timer difference and increment data-set pointer,

end for

- *Output collected data* - write to CSV file with all captured data.

- *Wait* - perform other simulation tasks prior to next execution interval.

end for

end:2

end

end:1

4.2.2 Assumptions

Assumptions include those described in the timing study, Section 4.1.2. Additionally, during the scalability analysis n process variables are transmitted and received. However, only one set of physical output and input terminals of the interface are used. Therefore, it is assumed that the delay associated with switching terminals on the interface is negligible. The interface is an on demand system and requests for varying physical terminals will not affect the measured elapsed times. In using a single terminal for transmitting a dynamic signal over n SUT inputs a comparison between the n^{th} SUT input and the 1^{st} SUT output can be made. From this measurement, analysis of the signal bias from output to input is performed for different I/O combinations. Writing to the same physical channel is assumed not to affect the calculation of the average bias as two consecutive signals within the same execution interval may have similar values.

4.2.3 Analysis of results

Similar to the timing analysis, three properties are monitored during the scalability analysis. These include the three parameters and two flags from the timing analysis. Again, elapsed time is calculated as the difference in time between the transmission of the SUT input from the software simulator to the receipt of the SUT output within the software simulator. The resolution of the elapsed time measurement is 1 ms. Also, the SUT input variable value is stored within the transmitted controlled variable field of a database. The same is true for the SUT output variable which is stored in the received controlled variable field of the same database. The resolution of captured variables is limited by the UDP/IP protocol discussed in Section 3.2.2.

The same two error flags are identified. However, error flag accumulation is added. For example, if 25 I/O are being transmitted in a single execution interval

and 13 transmissions result in invalid ID flags, the invalid ID flag is incremented 13 times and stored. The same is true for HIL timeouts, or no data flags. This enables accurate observation of the number of errors within the interface and assists in identifying invalid transfers which can compromise simulation results.

All properties and flags are again stored within a comma separated value (CSV) database. A database record is created for every set of SUT input variables transmitted. All variables transmitted within a single execution interval are accumulated in one record– one record for each of 2250 execution intervals within all SUT delay/HIL timeout combinations.

The following results provide analysis of the data collected during interface development. The figures generated present a summary of the data collected. Every effort is made to present relevant illustrations of the data which assist in determining the limitations of the developed interface. Following each figure, table or result is a discussion of the observations made from the material presented. Discussion includes; explanation of results; deduction, general application of the results; and hypothesis, a possible conclusion arising from the results.

Figure 4.7 illustrates the average total elapsed time for each I/O configuration including n controlled variable transmissions. In the figure the independent variable represents the number of inputs and outputs ($n_{inputs} = n_{outputs} = n_{I/O}$) within the I/O configuration. In every execution interval the total time for n instantaneous sinusoid signal values to be transmitted through the interface is measured. The total elapsed time is accumulated over all 2250 execution intervals. The average total elapsed time is calculated from this total. From Figure 4.7 the following is observed:

- The average total time measured with a single I/O and 10 ms SUT delay, 14.5 ms, is comparable with the results of the same configuration during the timing analysis, 14.54 ms.
- There is a strong relationship between the SUT delay and the average total elapsed time.
- There is a linear relationship between the number of I/O and the average total elapsed time beyond the time taken in the SUT delay.
- The 150 ms SUT delay reaches the limitations of the HIL platform with 10 I/O. Average total elapsed time is not measurable for 25 I/O.

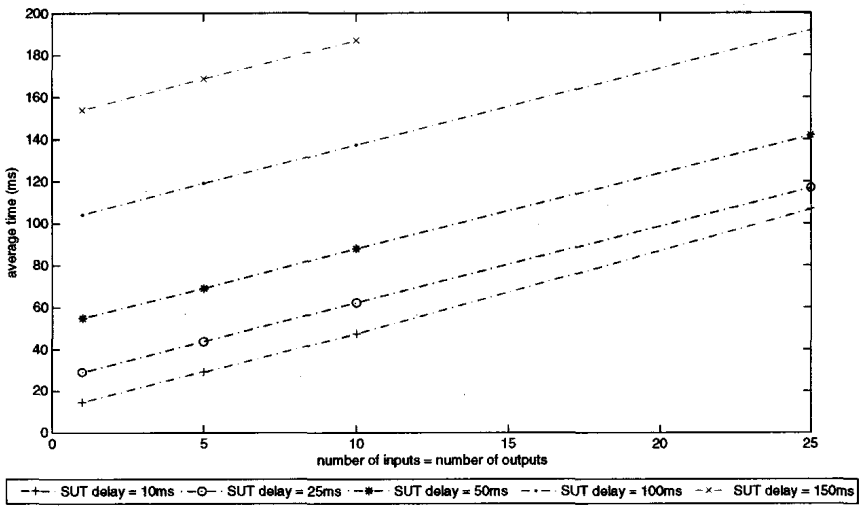

Figure 4.7: Average total elapsed time vs. I/O configuration.

A larger number of I/O is expected to result in longer average total elapsed times. Also, the proportion of the average total elapsed time to the SUT delay results in similar results to the timing analysis. Further, the average total elapsed times are approximately 4-5 ms greater than the SUT delay component of the total elapsed time. For a 10 ms SUT delay and 25 I/O an average total elapsed time of 4.33 ms is observed per successful I/O transmission. This is similar to the delay observed during single I/O transmission. Therefore, the additional delay when I/O are added is near constant per I/O for incrementing I/O configurations. The accumulation of HIL timeout delays for every SUT output requested is the cause of the increasing additional delay.

The linear relationship between the number of I/O and the average total elapsed time enables extrapolation. Therefore, the response of configurations not included in the analysis can be approximated and extended limitations of the interface can be revealed. The formula for extrapolation is presented in the following;

$$t_{te} = 3.698 \times n_{I/O} + 0.5514 + t_{SUT}, \tag{4.6}$$

where t_{te} is the expected total elapsed time, $n_{I/O}$ is the number of I/O and t_{SUT} is the SUT delay.

With 25 I/O and a 150 ms SUT delay the simulation fails to complete and the sinusoidal signal generator crashes. It is the purpose of this analysis to identify the limitations of the interface. Therefore, the eventual failure of a configuration is

expected. Further, similar limitation is nearly reached with a 100 ms SUT delay and 25 I/O reaching a maximum sustainable execution time of 191.9 ms. This limitation is expected as the execution interval of the simulator allows 200 ms for the C module to execute entirely.

Though it is not evident in Figure 4.7, all SUT delays failed stable transmission during tests with 50 I/O. The failure is evaluated in (4.7) using (4.6), where even the lowest SUT delay of 10 ms results in;

$$t_{te} = 3.698 \times n_{I/O} + 0.5514 + t_{SUT} = 3.698 \times 50 + 0.5514 + 10 = 195.45ms > 191.9ms. \quad (4.7)$$

The maximum equal number of I/O that would be captured for a 10 ms SUT delay reflects the maximum equal I/O capacity of the interface. A SUT delay of 10 ms is a minimal delay and is used for very fast responding electronic control systems. In (4.8) the maximum I/O configuration is approximated using the following equation;

$$n_{I/O} = (t_{te_{max}} - 0.5514 - t_{SUT})/3.698 = (191.9 - 0.5514 - 10)3.698 = 49. \quad (4.8)$$

The interface is designed to support a total number of 100 identifying integers for SUT I/O. Therefore, the UDP/IP protocol matches the capabilities of the interface for a very fast electronic control system. A total of 49 I and 49 O can be implemented simultaneously. When it is necessary to transmit more than 100 process variables, the interface would have to be redesigned, or two or more interfaces installed in parallel. Also, for many slower responding electronic control systems the limitations for the number of I/O that can be implemented is much lower.

It is important to identify that evaluating the scalability limitations of the interface is problematic. When the C program fails due to a `running to slowly` error the instantaneous signals are not captured and the simulation is aborted. This failure can be used to the advantage of the analysis. However, data collection must be well planned and failures should be expected. In some cases when the simulator aborts no data is captured. This is the case for each of the SUT delays with an I/O configuration of 50 inputs and 50 outputs.

The review of the total elapsed time required by the developed interface is extended in Figure 4.8 which illustrates the 95th percentile time for multiple controlled variable transmissions. From Figure 4.8 the following is observed:

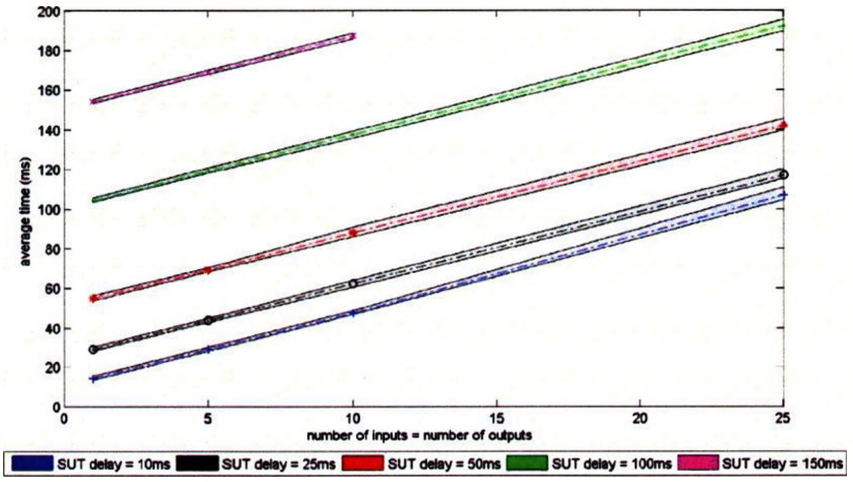

Figure 4.8: Average total elapsed time with 95th percentile vs. I/O configuration.

- The range of 95th percentile total elapsed times increases with the number of I/O.
- The range of 95th percentile total elapsed times is very narrow per transmitted I/O.

It is expected that as the number of I/O increases so does the range of the total elapsed times measured. During the timing analysis for the single I/O requested the 95th percentile had an extra 1.5 ms during receipt of the SUT output. This can accumulate over 25 I/O and can cause major delays in the interface. Therefore, analysis of the deterministic response of the HIL timeout during different I/O configurations is essential. The specific increase in range for each I/O configuration is not known and is difficult to discern in Figure 4.8. Therefore, further analysis is required. By subtracting the average times from the 95th percentile and then dividing by the number of I/O a better understanding of the two observations can be made. The total range deviation and the deviation per I/O can be observed and discussed regarding the deterministic response of the software simulator.

FIgure 4.9 includes two plots. Each plot illustrates the range of the 95th percentile for the evaluated set of I/O configurations. The top plot includes the total range deviation. This is calculated by subtracting the lower 95th percentile from the 95th percentile. The bottom plot includes the deviation per number of I/O. Though the accumulated plot is included, it is the per I/O plot which demonstrates the deterministic qualities of the return request packet and SUT output routine. In addition

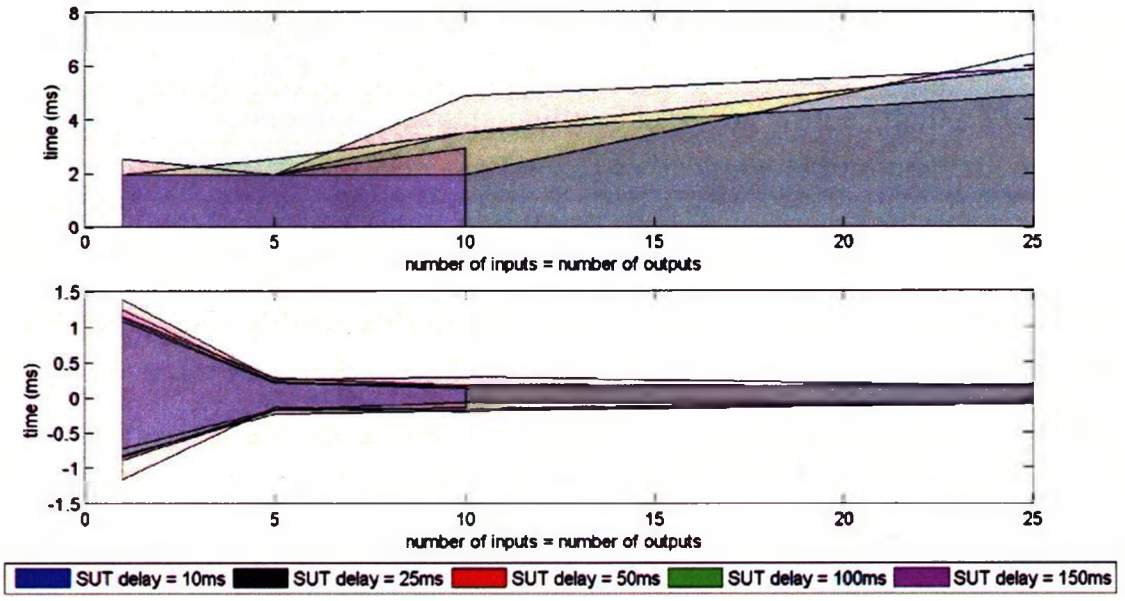

Figure 4.9: Average total elapsed time with 95th percentile vs. I/O configuration.

to the observations made from Figure 4.9, the following is observed in Figure 4.9:

- The increase in the range of the 95th percentiles only extends to approximately 7 ms from 2 ms for 24 additional I/O.
- When multiple I/O are transmitted the perceived range of elapsed times decreases from approximately 1.5 ms to less than 250us.

An increase in total range for multiple I/O is expected as the accumulation of HIL timeout and SUT delay variances will result in a greater overall variance. Similar to Figure 4.8, an increase in range is evident. However, in Figure 4.9, it is apparent that for an addition of 24 I/O, the increase in the range is less than 6 ms. Therefore, the total time added to the SUT output routine changes with number of I/O.

When deterministic operation of the interface is discussed, it is important to identify how deterministic operation affects the HIL platform as a whole. Any deterministic operation involves a specific sequence of events. In the HIL simulation platform both the SUT delay and HIL timeout for each SUT output must execute prior to the execution interval expiring. If a significant range in elapsed times is observed for large numbers of I/O the HIL platform cannot operate efficiently. For example, assume 1 ms above the average total elapsed time is required to assure the

95th percentile is captured for a single I/O. For 25 I/O, the execution interval would have to be 25 ms greater than the average total elapsed time to assure proper functionality. That would be 12.5 percent of wasted time occupied within the execution interval due to non-deterministic functionality. Therefore, with a smaller range of total elapsed times, the HIL platform can be said to be more deterministic in operation and can perform over a larger range of number of I/O.

During the timing analysis for all HIL timeouts, it was found that the deviation in range of the 95th percentile per I/O was within 1.5 ms of the average value. This remains the case for an increase in number of I/O. It is interesting to note that the range decreases with increasing I/O. Previously it was expected that the range would increase per number of I/O implemented. However, the decrease is evidence that the resolution of elapsed time measurement produces inaccurate results in the timing analysis. With maximum average elapsed times in the 0-30 ms range, the addition of a single milli-second during HIL-timeout, or the SUT delay can offset average calculations substantially. The timing analysis did reveal that with increasing HIL timeout the range remains relatively constant. However, an accurate illustration of the actual variance of the elapsed time for the interface is provided in Figure 4.9. It appears that having a larger number of I/O may result in more efficient timing within the HIL platform. This is not the case. Lower numbers of I/O perform efficiently just as the larger numbers of I/O. It is the measurement resolution of 1 ms which prevents an accurate representation of the range when using the single I/O configuration. Therefore, the software simulator responds more deterministically than previously expected with the entire range of total elapsed times occurring within 0.3 ms per I/O.

Figure 4.10 presents the number of successful transmissions through the interface over 2250 execution intervals for each I/O configuration. The number of successful transfers is the total number of transfers which do not produce either error flag. From Figure 4.10 the following is observed:

- For large numbers of I/O the number of successful transfers per I/O is reduced.

It is expected that the number of successful transfers will increase linearly with an increase in the number of I/O transmitted through the interface. However, it is difficult to compare the number of successful transfers between I/O configurations due to the large range of successful transfers covered. It is known that as the number

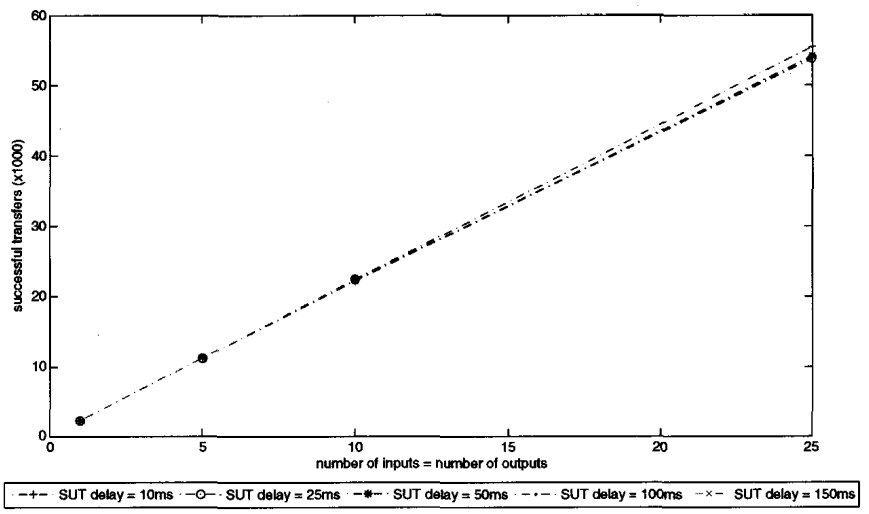

Figure 4.10: Successful transfers vs. I/O configuration.

of I/O increase, so does the chance that a UDP/IP packet fails to transmit properly. Increased traffic is known to reduce the capability of the interface [90]. For this reason, a slight deviation from linear performance is observed. This deviation is most evident for 25 I/O. At 25 I/O the expected number of successful transfers is $25 \times 2250 = 56250$. The reduction occurs for all but the 10 ms SUT delay.

Two explanations can justify the reduction in successful transfers. Both are the result of no data or invalid ID flags. The cause of the flags could either be network traffic, as mentioned, or the expiration of the execution interval. It is documented in the pseudo code for this analysis that if the execution interval were to expire during the HIL timeout a error flag would not be accumulated. Therefore, the deviation cannot be the result of an insufficient long execution interval. However, the observation illustrates a fundamental problem within the analysis. Every successful transmission is not documented unless an error flag is produced. Therefore, further analysis is required.

During the timing analysis the availability of the interface is expected to vary dramatically for different HIL timeouts. However, the effect of the number of I/O on the availability of the interface is not expected to be as dramatic. Acceptable availability is expected over the entire range I/O configurations. Further, in Figure 4.10 it is difficult to discern the percentage of successful transfers. Figure 4.11 presents these percentages. Further, the percentages of error flags recorded during the analysis are presented. From Figure 4.11 the following is observed:

- There is no obvious relationship between the number of I/O and the availability

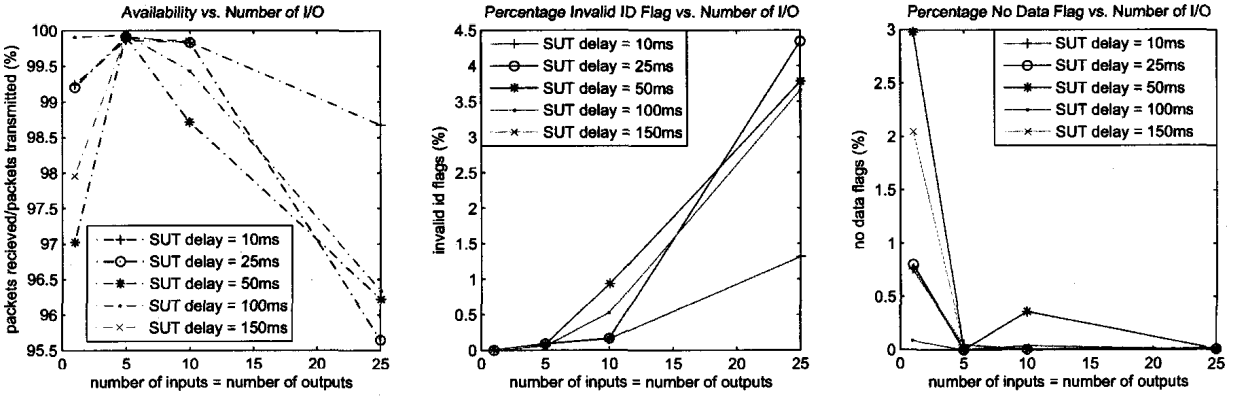

Figure 4.11: interface availability with errors vs. I/O configuration (Dataset 1).

of the interface.

- With 5 I/O the availability of the interface is similar for all SUT delays.
- The majority of errors observed with a low number of I/O are due to expiration of the `poll()` Function (no data flag).
- The majority of errors observed with a high number of I/O are due to the returned variable not matching the requested variable (invalid id flag).
- The 50 ms SUT delay results in a significantly reduced availability. The reduction is due to both error flags.

The availability of the interface for different I/O configurations is expected to be consistent. Though all availabilities are above 95.5 per cent, there is little to no relationship between the number of I/O and the measured availability. Actually, the single I/O configuration is removed from the trend. As mentioned previously when a larger number of I/O are transmitted there is a great chance for a transmission to fail. The data does not satisfy the expectations for interface performance. Therefore, a second set of data is produced. The data is presented in Figure 4.12.

Figure 4.12 is similar to Figure 4.11. However, the data presented is from a secondary simulation. Again, the percentages of error flags recorded during the analysis are presented. From Figure 4.12 the following is observed:

- Again, there is no obvious relationship between the number of I/O and the availability of the interface.

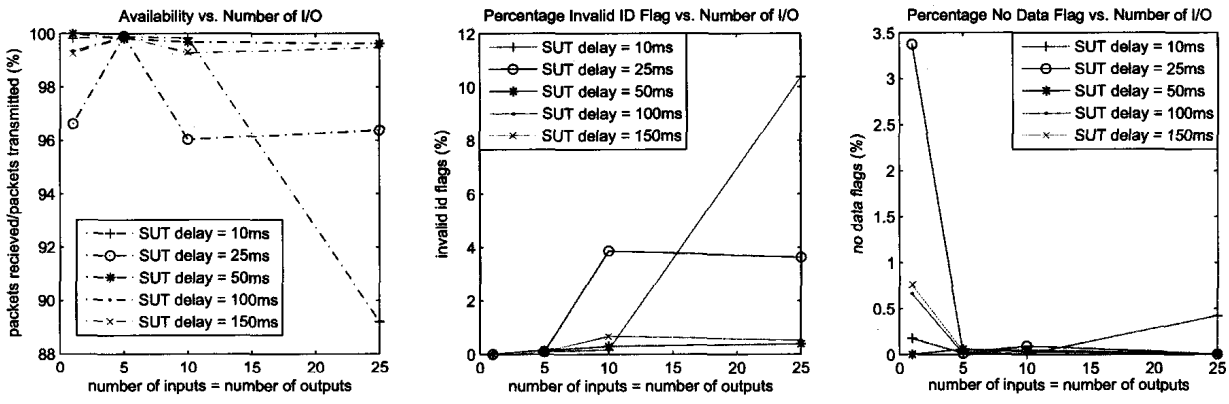

Figure 4.12: interface availability with errors vs. I/O configuration (Dataset 2).

- Also again, with 5 I/O the availability of the interface is similar for all SUT delays.
- The majority of errors observed with a low number of I/O are due to expiration of the `poll()` Function (no data flag).
- The majority of errors observed with a high number of I/O are due to the returned variable not matching the requested variable (invalid id flag). One extreme case is observed with a 10 ms SUT delay and 25 I/O.
- In comparison with the previous observation the 50 ms SUT delay performs very well with the highest availability over many configurations.

Again, there is little to no relationship between the number of I/O and the measured availability. The second set of data demonstrates less relationship between the number of I/O and the measured availability than the first. Comparison between the availability in the single I/O configuration and previous 10 ms SUT delay timing analysis results indicate that a reduction in the number of successful transfers has occurred. To determine the cause of the reduced availability, the right most sub plot of 4.12 reveals that no data errors occurred. There are two factors which may cause the observed reduction. The first is the time between subsequent UDP/IP transmissions. In Figure 4.1 it is observed that a 50 ms execution interval results in a reduced average transmission time than a 200 ms execution interval. This observation is explained through evaluation of UDP/IP protocol performance when the time between subsequent transmissions is changed. In Figures 4.11 and 4.12 having an I/O configuration of 5 provides a higher availability for almost all SUT delays than with a single

I/O. This and the reduced performance of during single I/O transmission could be the result of maintained UDP/IP transmission.

There is a definite decrease in availability beyond 5 I/Os for the first set of data. To better understand the cause of the reduction in availability a sub plot illustrating percentage invalid identifying integers is presented in the second sub plot for Figures 4.11 and 4.12. At configurations above 5 I/O invalid identifying integers account for nearly all lost controlled variables. At and below 5 I/O there are very few invalid identifying integers. During UDP/IP transmission, in order to reduce overhead and increase network speed, various hand-shaking and acknowledgment routines found in other networking protocols are removed. If a single packet is transmitted out of place or network traffic increases drastically, packets are more likely to be improperly communicated. Further, during the 10 ms SUT delay 100 per cent of no data flags resulted in subsequent invalid id flags within the same set of transmitted SUT output. Therefore, when an SUT output is not received at the software simulator, the UDP/IP socket is affected. From the second set of data a relationship between no data flags and invalid id flags is identified. Where a 25 ms SUT delay and a 10 ms SUT delay have maximum flags, they do for both flags.

The ability to predict the availability of the interface to respond to a return request packet comes in to question. However, over two evaluations performed at different times, reduced single variable availability is experienced. In the first set of data increased availability is observed in lower I/O configurations. The effects of the UDP/IP communication protocol are apparent. However, in the second set of data the availability remains very high for increasing numbers of I/O, unless a no data error occurs. The observed performance of the interface is unexpected. Therefore, it is recommended that the availability of the IL interface be observed over the length of the evaluated simulation prior to performing HIL simulation.

To evaluate the relationship between SUT input and output values, the value of the n^{th} variables successfully transferred through the interface and the 1^{st} variable received are captured. The capture procedure occurs at the n^{th} transmission of n variables and the 1^{st} receipt of n variables. Only a single physical terminal is used for analog signal loop-back. Therefore, comparison of SUT input and output can only occur between the last transmitted variable and the first received variable. Figure 4.13 presents the per unit average bias of the transmitted and received signal. The per unit value is again derived from (4.3). From Figure 4.13 the following is observed:

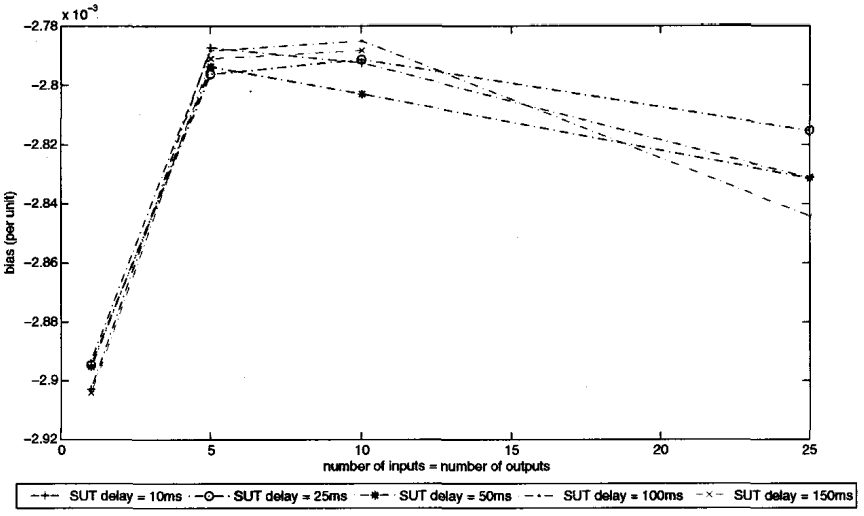

Figure 4.13: Average signal bias vs. I/O configuration.

- A per unit average bias measurement of around 2.8×10^{-3} is achieved.
- The average bias is not greatly affected by the SUT delay.
- For a single I/O the average bias is offset from the remainder of the I/O configurations.
- At higher I/O configurations the average bias is reduced slightly.

The captured signal biases are observed in Figure 4.13. Over all configurations the signal offset, or bias is relatively comparable, especially when comparing the bias between different SUT delays. Again, the settling time of the interface is not a concern when performing HIL simulations.

The fluctuation in signal bias for the higher I/O configurations and the single I/O configuration are very similar to the observations made during the timing study. In Figure 4.11 the availability of the interface is presented for the same simulation as Figure 4.13. Again, a reduction in the number of successful transfers results in a reduction in the accuracy of the calculated signal bias.

4.2.4 Transmission delay study

During the timing analysis, a 1-2 ms delay is identified for the interface to properly respond to a return request packet with a 0 ms SUT delay. This delay in interface

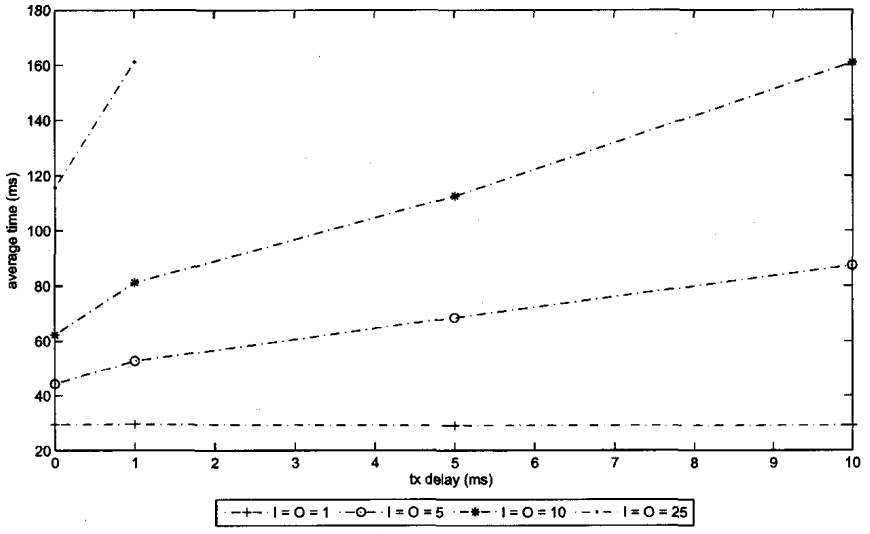

Figure 4.14: Average total elapsed time vs.transmission delay.

processing is expected due to signal conversion routines and electrical signal generation. For this reason a transmission delay is proposed. The transmission delay, or transmission delay, is inserted between subsequent SUT input transmissions.

If multiple SUT inputs are transmitted, the increased processing required by the interface is expected to result in lower availability. The ability of the interface to respond to multiple SUT inputs, when a transmission delay is included, is necessary. Further, limitations resulting from the transmission delay are evaluated. Figure 4.14 illustrates the average total elapsed time of the interface for a 25 ms SUT delay for multiple I/O configurations and transmission delays. From Figure 4.14 the following is observed:

- The average total time measured for all but the single I/O configurations is increased.
- With 25 I/O the average time increases beyond the HIL capacity at the 2 ms TX delay.
- For 1 ms and greater tranmission delays there is a linear relationship between the transmission delay and the average time beyond the time taken in the SUT delay.

The effect that the transmission delay has on the average total elapsed time required by the interface is predictable. When performing n SUT input transmissions

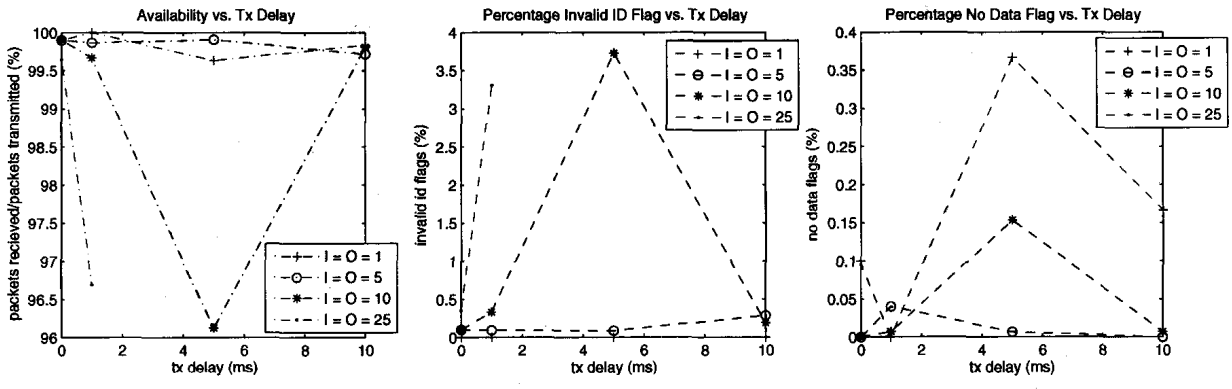

Figure 4.15: interface availability and error flags vs. tranmission delay.

with a transmission delay of t_x, a total additional delay of $n \times t_x$ is expected to be observed. From Figure 4.14 it is apparent that the interface responds as expected. However, different limitations to the number of I/O supported by the interface are observed. Even with a minimal transmission delay of 1 ms a significant difference in total elapsed time is observed. The introduction of the transmission delay must demonstrate increased interface performance to justify reduced numbers of I/O.

The transmission delay is implemented to evaluate the availability of the interface. The delay increases the time between consecutive SUT input transmissions. Therefore, the increased delay is expected to reduce interface processing requirements and increase availability. Figure 4.15 illustrates the availability of the interface for a 25 ms SUT delay over multiple I/O configurations and transmission delays. Further, the percentage of error flags which occurred during the simulation is presented. From Figure 4.15 the following is observed:

- There is no obvious relationship between the transmission delay and the availability.

During the timing analysis a delay required for interface processing was observed. Further, during scalability analysis, no relationship between number of I/O and availability could be made. Both of these observations were expected to be resolved with the implementation of a transmission delay. However, to justify the reduction in number of I/O supported by the interface, increased availability should be demonstrated. From Figure 4.15 it is clear that including a delay between consecutive SUT input transmissions does not increase interface availability. The interface demonstrates similar availability for 0 and 10 ms transmission delays. Therefore, the transmission delay is not necessary for proper system functionality.

SUT Delay	Maximum SUT Inputs	Maximum SUT Outputs
150	99	10
10	99	49

Table 4.2: Maximum stable number of input only and output only configurations.

4.2.5 Maximum system under test input/output study

During the scalability analysis only equal numbers of inputs and outputs are evaluated. The evaluation of equal numbers of I/O provides a reasonable amount of confidence in various system configurations. However, it is difficult to discern which signal, input or output is responsible for the majority of the required transmission time. Through extended analysis, a measure of the confidence in having unbalanced I/O can be made.

To better understand the delays that are present during simulation the interface is automatically configured with an increasing number of I/O until failure. During the maximum input analysis, a single output is configured. The opposite is true for the maximum output analysis, where a single input is configured. A failure is registered when the software signal generator aborts operation due to a `running to slowly` error. For a configuration to be considered successful it has to maintain operation for a period of five minutes. Only the 200 ms execution interval is investigated. Also, two SUT delays and a 10 ms HIL timeout is implemented over all evaluations. Table 4.2 presents the results of the maximum input and output analysis.

From Table 4.2, a restricted maximum was not observed when the SUT inputs were incremented. The limitation of the UDP/IP packet identifying integer, 99, is the limit for both SUT delays. Therefore, the transmission of an SUT input must require less than $(200 - 150ms)/99$ $0.5ms$. However, the maximum stable number of SUT outputs was recorded as 10 and 49 outputs for $150ms$ and $10ms$ SUT delays respectively. Therefore, the transmission of SUT outputs must require less than $(200 - 10ms)/49 = 3.88$ or $(200 - 150ms)/10 = 5$. Although it was found that the number of I/O has a limited effect on the average total elapsed time. For very large SUT delays the ability of the simulator to maintain operation is reduced. Further, (4.8) successfully predicted the 49 SUT outputs required for a 10 ms SUT delay. This justifies the minimal time required for SUT input transmission.

Chapter 5
Hardware-in-the-loop Shutdown System Simulation

In Chapter 3 the purpose of performing HIL simulation is described as extending beyond the development and testing of a interface device. Therefore, though a interface device is developed, various modifications to the HIL simulation platform are required to enable the proper simulation of NPP processes.

The task of interfacing simulated signals with existing sub-systems remains. This chapter focuses on the installation of an existing software simulator as the physics/truth model and a specific electronic control system as SUT within an HIL simulation platform. Transmission of controlled variables within the HIL simulation platform is performed by the interface device developed in Chapter 3. First, the HIL simulation platform is verified to assure proper transmission of controlled variables between the installed software simulator and SUT prior to HIL simulation. Following verification, the HIL simulation of a safety critical NPP process is performed and the performance of the SUT and the interface is analyzed.

5.1 NPP Hardware-in-the-loop Simulation Platform Development Procedure

The development of an HIL simulation platform involves physical system connections, connectivity supporting software development and platform verification. Figure 5.1 illustrates the procedure for installing the interface within an HIL simulation platform. The procedure begins with the development of a interface which is performed in Chapter 3.

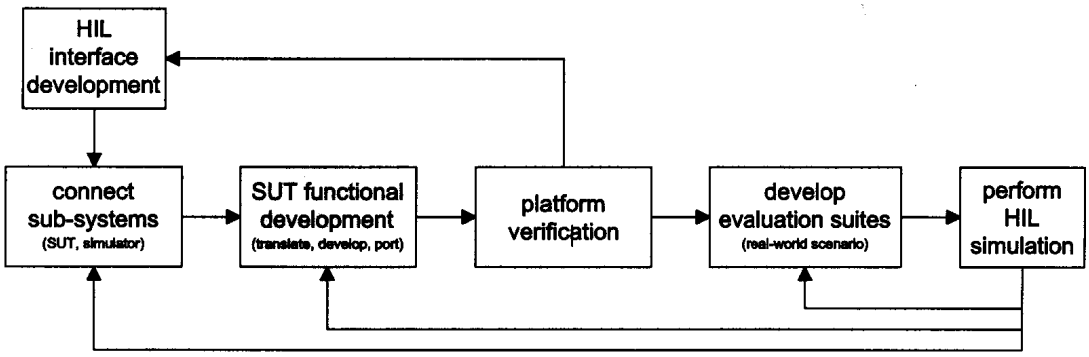

Figure 5.1: Application specific HIL simulation procedure.

Establishing simple physical connections between the interface with a software simulator and a electronic control system is not adequate for performing HIL simulations. Therefore, the proposed procedure is executed. The result is a complete HIL simulation platform capable of accurate NPP process simulation.

Firstly, software simulator connectivity is established. This requires communication module development within the software simulator. Also, control logic is installed on the SUT and anticipated operational occurrence related to the process being evaluated are identified. These tasks require control logic translation and programming as well as extensive evaluation of the the process being evaluated. The interface is then configured for the number of I/O required by the process being evaluated and the timing requirements identified in Chapter 3. Finally, prior to HIL simulation, the HIL simulation platform is evaluated to assure proper variable transmission.

During the interface development procedure of Chapter 3, the requirements of the process being evaluated and the purpose for performing HIL simulation are identified. Four likely scenarios are presented and challenges within each for interface development are identified. Similarly, the challenges related to each scenario can be identified for HIL simulation platform development.

- *Known process* - no physics/truth model, or SUT have been selected. A simulator is developed through process modelling, assuring that a method for connectivity with the interface exists. Potential electronic control systems are selected and control logic developed and installed according to the process being evaluated.
- *Existing SUT* - the SUT has been selected as the electronic control system for

the process being evaluated. Similar to the 'known process' scenario a simulator is developed. Further, control logic is developed and installed on the electronic control system.

- *Existing software simulator* - the functionality of an existing software simulator is extended to include HIL simulation. Again, similar to the 'known process' scenario, potential electronic control systems are selected and control logic developed and installed according to the process being evaluated. Further, controlled variable transceiver modules are developed for the software simulator.
- *Existing SUT and software simulator* - both the existing SUT and software simulator scenarios are satisfied. Controlled variable transceiver modules are developed for the software simulator.

To perform any HIL simulation, a software simulator and a electronic control system must be connected to the interface. For this reason the interface introduced in Chapter 3 includes flexible connectivity specifications. Though flexible, the connectivity specifications are selected to assure connection to the existing NPP training simulator and the Tricon v9 PLC installed in this chapter. The NPP training simulator supports both Ethernet communication and UDP/IP communication module execution. Further, the Tricon v9 PLC is a standardized electronic control system and supports common industrial analog and digital signal levels.

During interface development no electronic control system is connected. Instead, an analog feed-back loop is implemented. The installation of an SUT into an HIL simulation platform involves interface terminal to SUT terminal wiring for each controlled variable and Ethernet cabling to the software simulator. Details of these connections are not covered within this thesis. It is recommended that sub-platform manuals are reviewed prior to physical connection. Once the two sub-systems are connected to the interface, the HIL simulation platform illustrated in Figure 3.1 is established.

An Ethernet signal transceiver is installed to simulate a physics/truth model during interface development. This signal transceiver is designed to mimic the structure of the C program modules installed on the NPP training simulator. Therefore, the signal acquiring capabilities that are experienced during HIL simulation are identical to the sinusoidal signal packet transmissions evaluated in Chapter 3. This allows

for fluent transition between HIL simulation and interface testing, verification and development. In fact, the task of modifying the controlled variable transceiver module to satisfy the requirements of the software simulator includes only the configuration of the proper process variables for control.

Before HIL simulation is performed control logic is installed on the electronic control system. The application being evaluated within this chapter is the SGLL trip condition of SDS1. Therefore, CANDU SDS1 SGLL logic is translated from original source code to a language supported by the SUT. This process is known as porting. The ported logic is installed on the SUT and if necessary the equivalent logic is removed from or disabled within the software simulator.

HIL simulation can be executed at this point. However, it is recommended that verification of the timing within the completed HIL simulation platform is performed. The timing parameters from Chapter 3 including SUT delay, HIL timeout and transmission delay are applied to the interface. HIL simulation platform verification identifies parameter values which facilitate maximal availability of the interface. This is achieved by comparing expected SUT outputs with current SUT outputs based on the current state of SUT inputs. Initial configured timing parameters are derived from from the equations and methods of Chapter 3. Also, practical limitations for interface delays are observed for higher fidelity processes. Restrictions introduced by the execution interval of the process may require lower availabilities and consequently simulation accuracy may be reduced. If a timing configuration can not be verified, the interface is not suitable for the process being evaluated and should be re-developed.

The final step prior to HIL simulation is the replication of design basis events, or suites for the process being evaluated within the software simulator. Proper identification of the operational states occupied by the electronic control system during an anticipated operational occurrence is imperative for accurate HIL simulation. Therefore, input sequences are developed which result in design basis event and eventually SGLL trip conditions. To evaluate the functionality of SDS1 to trip for a SGLL condition, for example, specific events must occur in a specific sequence within the NPP. Most importantly the reactor power should be reduced to a shutdown level. Extensive evaluation of the process being evaluated is required to assure that the proper functionality of the electronic control system can be identified independent of auxiliary NPP systems. The method of identifying input sequences for proper process simulation is not included within. However, the resulting shutdown system scenarios

evaluated are described in detail.

Finally, evaluation of the SUT within the HIL simulation platform is enabled. Executing the evaluation requires repetitive tasks. Therefore, the development of a script for automating the state of the software simulator and the SUT is recommended. Again, this procedure varies significantly between software simulator platforms and is not covered within. Also, acquiring data from an actual physics/truth model and electronic control system introduces various problems. Therefore, some adaptation to HIL simulation timing methods and connectivity are revealed while executing the proposed procedure with the selected sub-systems.

5.2 NPP Hardware-in-the-loop Simulation Platform Development

The NPP HIL simulation platform utilizes the interface developed in Chapter 3. A NPP training simulator and the Tricon v9 are installed as the physics/truth model and SUT respectively. Further, control logic is developed for the Tricon v9 to replicate the electronic control system control system emulated in the NPP training simulator. Therefore, the HIL simulation platform for SDS1 evaluation is comprised of five major components:

- *physics/truth models* - Darlington NPP training simulator (DarlSIM) and computer platform,
- *interface: controlled variable Ethernet transceiver* - provides similar function to the Ethernet signal generator. However, signals are not generated but are acquired from a simulated NPP process. This module executes during run-time within DarlSIM,
- *interface: physical adaptor and signal converter* - conversion between Ethernet (engineering units) and hard-wired signals (analog and digital signals). The developed interface of Chapter 3 is installed as interface for all HIL simulation routines.
- *System under test (SUT): hardware* - Invensys Triconex Tricon v9 safety PLC,
- *System under test (SUT): control logic* - replicated Darlington NPP SDS1 SGLL logic. This logic is ported from Fortran source code available within DarlSIM.

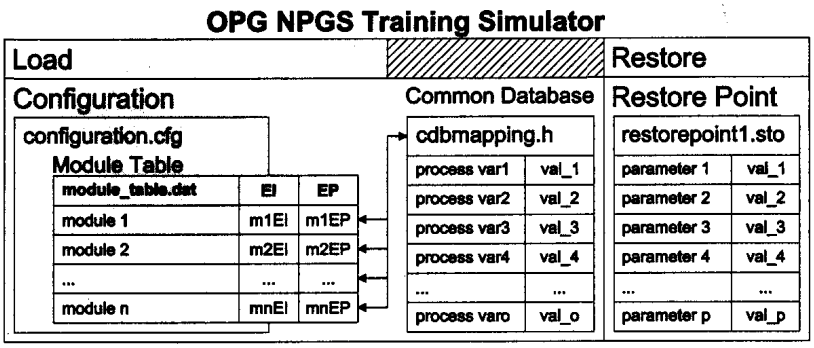

Figure 5.2: DarlSIM component interconnections.

5.2.1 Darlington NPP training simulator

OPG maintains a Unix based training simulator, referred to within this paper as DarlSIM. DarlSIM simulates the operation of the entire Darlington NPP in real-time. Therefore, DarlSIM encompasses an extensive collection of NPP plant models including related control loops. The simulated control loops access plant specific process variables and emulate the physical electronic control systems within the plant. In establishing an interface between the software simulated process variables and external physical hardware, HIL simulation is performed. The main components of DarlSIM relevant to performing HIL simulation include modules, a module table, configurations, restore points and a common database. These components are described below. The relationship between the four components of DarlSIM is illustrated in Figure 5.2.

Modules are compiled simulation models and auxiliary functions which represent unique NPP processes within DarlSIM. They are catalogued and scheduled for execution within a module table. For a given configuration a module table is attached. The module table specifies the processes to be executed within the connected configuration. Modules are included for all major systems within the NPP and are developed using C or Fortran programming languages. Therefore, modules provide the core functionality of the simulated NPP. Further, modules access a CDB. The CDB is a memory bank where DarlSIM process, temporary and related variables are stored during run-time. Three modules, for example, are responsible for SDS1 logic execution. One module for each of channels D, E, and F. The modules access the CDB in a shared memory configuration to communicate process variables, inputs, outputs and other data between modules.

The module table is a list of the modules required for proper functionality of the NPP. It is the primary method for scheduling module execution. This list

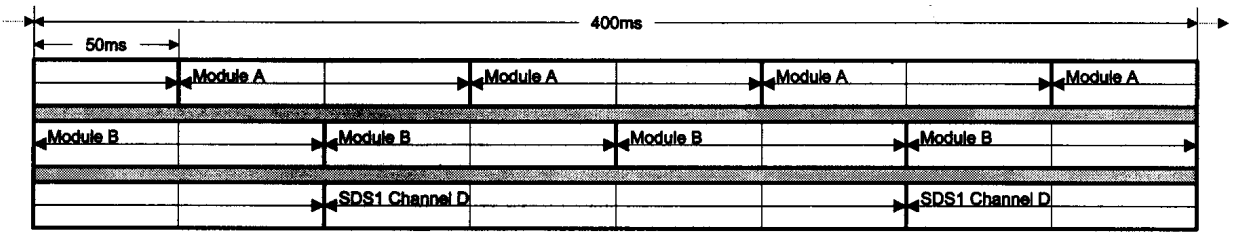

Figure 5.3: Module execution interval and execution phase.

is complicated by two factors, execution interval and execution phase. Execution intervals exist for 50ms, 100ms, 200ms, up to 2s. The execution interval specifies the period over which the module must completely execute. In the case that a module is too complex, or requires too much time to execute properly within the interval, the simulator will produce a `'running to slowly'` error, fail and abort. Channel specific SDS1 logic modules have an execution interval of 200 ms and are therefore executed every 200 ms.

Module execution phase is slightly more complicated. Execution phases are available at multiples of 50 ms over the entire period of the execution interval for any given module. Illustrated in Figure 5.3, the execution intervals of Module A and Module B are 100 ms. However the execution phases are 50 and 0 ms respectively. The time of execution of the two modules is offset by 50 ms. The third example illustrates the configuration of the channelized SDS1 modules implemented during the HIL simulation performed in this chapter. Channel D has an execution interval of 200 ms and an execution phase of 100 ms. This is the same for channels E and F. Further, execution intervals initialization (INIT) and termination (TERM) exist for those modules which should only execute on either initialization or termination of the simulator.

The configuration is the simplest but most crucial of the four components. The configuration includes links to all compiled modules referenced within a specific module table. In other words, configurations and module tables are one in the same in terms of included modules. The configuration simply compiles all of the required modules and schedules module execution according to the attached module table. The configuration creates a configuration file and directory. The configuration file is the means of loading the simulator with a specific set of modules. Therefore, there are different configurations for simulation. The platform for SDS1 bench-mark simulation used in Section 5.4.3, for example, is named 'SDS1 BM.cfg'. However,

the configuration for SDS1 HIL simulation is named 'SDS1 HIL.cfg'. The differences between the two configurations are outlined in experimental setup later in Sections 5.4.3 and 5.4 respectively.

Finally, DarlSIM restore points provide access to common NPP operating modes and conditions. They represent instances in time during the operation of the NPP and facilitate repetitive scenario simulation with automated initial conditions. During simulator operation, the NPP within the simulator is manipulated and processes altered. When a certain state is achieve, the simulator can be paused, or frozen. The paused simulator instance may then be stored to a 'restore point'.

In using an external script the NPP simulator is automatically restored to the instance captured within a restore point. Using a script enables the repetition of a scenario over a large number of iterations. To evaluate SDS1, for example, malfunctions which would induce a shutdown procedure are instantiated, the simulator is frozen, and a restore point created. The restore point is then reloaded multiple times over a specified time interval from within a Unix terminal using a Perl script. However, it is important that the time interval dedicated to the simulation is sufficient to allow the entire scenario to complete.

5.2.2 Controlled variable transceiver module

During interface development a C program is developed to generate, transmit and receive signals. Conveniently DarlSIM supports modules developed using the C programming language. Therefore, the source code used in the interface development for Ethernet signal transceiver can be modified to transmit and receive process variables from DarlSIM to an SUT.

The task of transmitting variables is divided into four functions. Each function plays a specific role in enabling UDP/IP communication and variable transmission between DarlSIM and the external SUT. The following functions are used during all HIL simulations including verification and process evaluation procedures.

1. open UDP/IP connection,
2. SUT input transfer,
3. SUT output transfer, and
4. close UDP/IP connection.

Source code of all functions is included in Appendix C as reference.

The open and close UDP/IP connection functions are scheduled in the initialization (INIT) and termination (TERM) execution intervals of the module table. Therefore the UDP/IP sockets remain open for the duration of the simulation and do not impose additional time and system resources during signal transfer and training simulator processing.

To enable HIL simulation, process variables transmitted to the SUT must be accessible by the the SUT input function. A process variable that is an input or output of the SUT is referred to as a controlled variable throughout evaluations performed using HIL simulation. Within DarlSIM process variable access is supported through shared memory, or the CDB. By using shared memory, either a DarlSIM NPP related module or one of the functions listed above can read or write to the CDB at any time. Therefore, the same technique which is used to update and transfer variables between training simulator models is used to extract process variables. This is not possible with all Ethernet based simulators. In some cases a program may have to be developed to provide external access to the process variables.

The functions required for accessing the CDB are included with the training simulator. Variable acquisition is achieved using memory pointers. For example, two of the data access functions are:

`CdbFloatPointer(variable_label)`, and
`CdbBytePointer(variable_label)`.

Passing a process variable label with one of the two data access functions will return the current value of the specified variable.

Transferring controlled variables between the training simulator and the SUT is controlled completely by the SUT input and SUT output transfer functions. However, various delays are required to achieve proper signal transfer. The process sequence is as follows.

SUT input: read controlled variables from the CDB

SUT input: transmit controlled variables to interface (transmit delay)

SUT output: sleep (wait for SUT to execute)

SUT output: transmit controlled variable request packet to interface

SUT output: poll UDP/IP socket

SUT output: receive controlled variables from interface

SUT output: write controlled variable to the CDB

The interface responds on demand to UDP/IP packet transmissions. It is expected that the capability of the interface will be reduced if too many variables are transferred through the interface. However, this is not observed during the interface development and analysis in Chapter 3. Therefore, the communication module can maintain synchronization between the interface and DarlSIM. The only exception from synchronization is the execution of the control logic within the SUT. Though deterministic, the execution of the Tricon v9 PLC is asynchronous to the execution of the modules within DarlSIM.

There are two delays and a timeout associated with variable transmission. The first delay enables the interface to respond to consecutive SUT inputs. It is demonstrated that the transmit delay does not affect the probability of the interface to receive SUT input UDP/IP packets in Chapter 3. However, the delay is included when the number of I/O required for simulation is relatively small.

A secondary delay is included to enable the proper and timely execution of the SUT control logic and interface and SUT signal settling times. This delay provides a guarantee that SUT outputs are not captured until the current inputs have been processed by the SUT.

Finally, a timeout is required to enable the interface to respond when a variable is requested. Again, without this delay, either the requested variable will not be returned to the software simulator, or the UDP/IP packets received will include invalid data.

5.2.3 Hardware-in-the-loop interface

The interface of Chapter 3 is designed to require minimal configuration when installed into HIL simulation platforms for various applications. Firstly, the configuration of Ethernet communication parameters including IP address and communication ports is required. Other changes to interface configuration include; wiring the interface DAQ cards to the Tricon v9 PLC analog and digital I/O terminals; and selection of appropriate I/O configurations within the interface VI. For SGLL trip, the Ethernet parameters are configured according to the NPP software simulator configuration.

Also, four analog inputs and a single digital output are wired to the Tricon v9 PLC and configured within the VI.

5.2.4 System under test: Tricon v9 PLC

The SUT implemented during the HIL simulation is the Tricon v9 PLC. Tricon v9 has been certified by the USNRC [69] [17] to meet IEEE Class 1E and 603-1991 standards and was recently selected for the replacement of SDS1 controllers at Point Lepreau NPP in New Brunswick. Therefore, the Tricon v9 PLC system is being installed during the current refurbishment at Point Lepreau. The Tricon v9 PLC provides complete triple redundancy from the input to output terminal. Extended review of the Tricon v9 PLC system is provided in Appendix D.

The Tricon v9 PLC installed within the HIL simulation platform includes; triplicated 3008 Tricon enhanced main processors; a 4351 Tricon communication module; 32 points 3503/E discrete input 24V; 32 points 3604/E discrete output 24V; 32 points 3700/A analog input 5V; and 8 points 3805/E analog output 4-20 mA [50]. Tricon v9 can be configured to execute over a range of execution intervals. However, the requirements of the control loops within this study require only a 25 ms execution interval. Other execution intervals are utilized for verification purposes and are stated when used.

Of the Tricon v9 PLC hardware modules, four of the 32 points available on the analog input module are used for the steam generator level measurements. The steam generator level signals are industry standard 4-20 mA simulated differential pressure level sensors. One of the 32 points of discrete outputs is used for SGLL trip signal. The output is a 24VDC signal. However, this voltage is converted to a 5VDC signal for interface compatibility. In the actual plant this 24VDC signal would be connected to the SDS1 trip circuit reviewed in Chapter 2. From the trip circuit, a voltage triggered relay would de-energize the clutching mechanism releasing two banks of shut-off rods into the reactor core.

5.2.5 System under test: trip detection logic

Current SDS1 trip logic is comprised mostly of programmable digital comparators. SGLL trip detection is no exception. The purpose of the SGLL trip detection is to detect a low level in any of the four steam generators at the Darlington NPP. The

SGLL trip detection logic includes a condition based on neutronic power sensor readings. If the average of the highest 16 neutronic power sensors is less than a specific percentage of the total reactor capacity, a reduced threshold is utilized. This feature of the SGLL detection logic has been removed for the following HIL simulations. The response of the NPP at low power is not required to evaluate proper SGLL trip detection. The logic also omits manual SGLL conditioning and log N rate neutronic power conditioning. Finally, modified thresholds are implemented for SGLL detection. The process of determining SGLL trip is otherwise identical.

Reference SGLL logic is an emulation of the code that exists within the programmable digital comparators at Darlington NPP. Similar Fortran code is executed within DarlSIM. The logic is ported from Fortran source to block schematic diagrams and pseudo-code. Block diagrams and pseudo-code are developed to provide simplified reference for programming the SUT. The SGLL trip detection logic is developed using Tristation 1131 Developers Workbench (1131DW). Though Tricon v9 PLC supports ladder logic diagrams; structured text; and cause and effect programming language editor; function block diagrams (FBDs) are preferred for the following reasons:

- Tricon v9 SDS1 logic at Point Lepreau NPP will incorporate this method;
- proven performance of Wolsong 2, 3 and 4 and Qinshan 1 and 2 NPPs which utilize a similar graphical engineering software approach or integrated approach (IA), and;
- similarities in concepts and functions between existing IA function block language and the available IEC61131-3 FBDs [39].

The implemented SGLL trip detection logic is illustrated in Figure 5.4. Once 4-20 mA signals are converted to digital signals and filtered within the PLC they undergo validity range checks and comparison to SGLL trip thresholds. Thresholds are chosen to mitigate physical damage to the steam generator units.

When all four steam generator levels are in normal operating range they are greater than the SGLL hysteresis threshold. In this state the input of the select function is low and the trip condition is false. However, the resulting trip output is high. The shut-off rod clutch mechanism is energized and the reactor operates in a non-shutdown mode.

If any of the four steam generator levels drops below the hysteresis threshold the input of the select function becomes high. If the trip status is currently in a

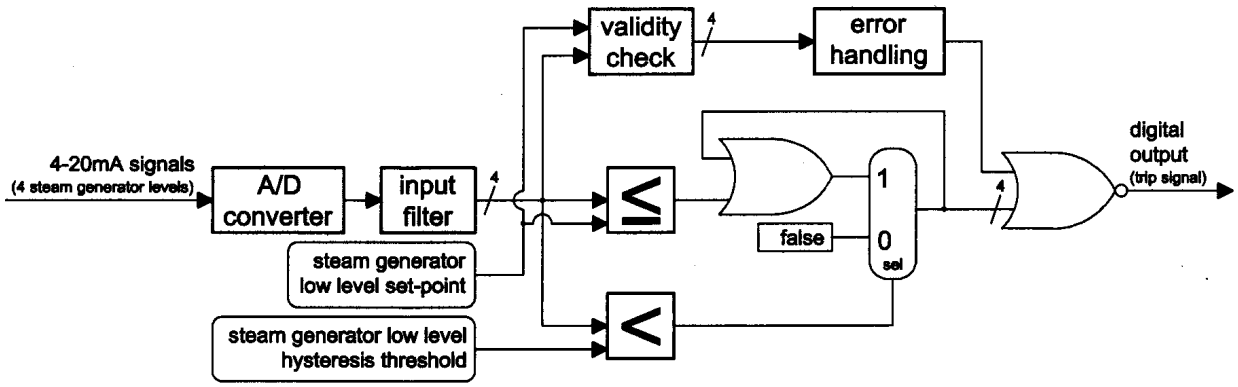

Figure 5.4: SGLL trip logic.

tripped state it remains tripped via a feed-back 'or' function. Therefore, a SGLL trip condition will remain until the steam generator level rises above the hysteresis threshold.

Standard hysteresis threshold level is approximately 10cm above the trip threshold. If the trip condition was previously low the trip condition remains low. However, the steam generator level is approaching the trip threshold. In this state, the input of the select function is high and the trip status is dependent on the second condition ($\leq$). If the trip signal then drops below the threshold, the ($\leq$) condition becomes high. The resulting trip condition is high the shut-off rod clutch mechanism is de-energized and the reactor enters shutdown mode.

In the NPP the decision-making unit for SDS1 is directly wired to sensors and actuators. The analog and digital input signals that are received by Tricon v9 PLC analog and digital input modules are conditioned by the interface to replicate real-world connectivity. Therefore the SGLL trip detection logic believes that it is installed in the actual NPP. Validity checks and error handling are performed to ensure that safety critical controlled variables are within expected boundaries. Also, SGLL logic is configured to execute at an interval of 25 ms on the Tricon v9 PLC.

5.3 Hardware-in-the-loop Simulation Platform Timing Verification

5.3.1 Method

Before performing an HIL simulation the HIL simulation platform, including the interface must be verified. The performance of the interface is proven and a certainty as to the accuracy of the system is observed in Chapter 3. However, installation of an SUT can induce additional delay requirements and timing routine modifications.

Previous HIL simulation results reveal discrepancies in the received SDS1 output process variables from DarlSIM and the SUT output signals received from the Tricon v9 PLC [73]. It is expected that the discrepancies are the result of a delay that occurs during controlled variable transmission. The delay does not impact the macroscopic evaluation of a shutdown scenario. In fact, the SUT appears to respond as expected. However, without accurate, reliable controlled variable communication between the simulator and the external hardware, the credibility of the HIL simulation platform as a basis for SUT selection is diminished.

To resolve the identified delays, the proposed procedure for HIL simulation platform development includes a timing verification and optimization routine. Verification is performed after the SUT is installed within the HIL simulation platform. Therefore, signal transmission throughout the entire platform can be verified. The verification procedure produces a measure of availability of the interface within the developed HIL simulation platform.

Recommendations for the timing of the interface are proposed in Chapter 3. These recommendations remain valid. However, some influences of the installed SUT on the transmission of controlled variables may result in modifications to the timing requirements within the HIL simulation platform. Without these modifications the performance of the interface cannot be guaranteed.

When verifying interface functionality within the HIL simulation platform at least two unique SUT output states are required. Proper transmission of signals through the HIL simulation platform is analyzed by alternating SUT input signals automatically every execution interval. It is important that the SUT input signals induce unique identifiable SUT outputs. Then, by analyzing SUT input to SUT output correlation the availability of the interface is calculated.

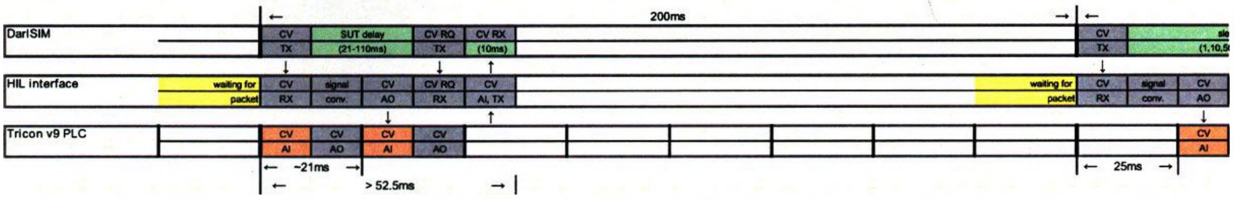

Figure 5.5: Expected sequence of events during HIL simulation platform verification for a 21 ms SUT execution interval.

The Tricon v9 PLC is programmed with the SGLL trip detection logic reviewed in Section 5.2.5. Function block diagrams are used to program the logic within the PLC and are available in Appendix E. Also, the Tricon v9 PLC is physically connected to the interface through four analog SUT input channels and one digital SUT output channel.

From Section 5.2.5, when one of four analog input signals drops below the trip set-point, or threshold, the Tricon v9 PLC will generate a trip signal on the digital output terminal. Alternatively, when the analog input signal rises above the specified hysteresis threshold, the trip signal is removed. For verification purposes, one of the four steam generator analog input signals is alternated between a non-trip state and a trip state. The North-West steam generator level, for example, is alternated between 1 and 5m every execution interval, a period of 200 ms. The alternating process is repeated over 6000 iterations, or 20min.

For non-SGLL, or generic applications, a simple digital loop back circuit including SUT digital input and SUT digital output will provide adequate signal alternating for interface availability analysis. However, if this method is performed the loop back function should be scheduled following all other control routines which are scheduled within the SUT for the process being evaluated. Therefore, a measure of the availability of the system can be obtained for the final control routine within the execution interval of the SUT. The unavailability of the interface is equivalent to the inability of the interface to produce the expected SUT output for given SUT inputs within the allotted SUT delay.

The expected Tricon v9 PLC control logic execution interval during verification procedures is 21 ms as illustrated in Figure 5.5. Figure 5.5 includes a single control variable analog input (CV/AI) block. From Chapter 3 the recommended SUT delay is recommended to be twice as long as the SUT execution interval. Equivalently, $t_{SUT} = 2 \times t_{SUT_{EI}}$.

In Figure 5.5 the expected worse case scenario for controlled variable transmission is illustrated. The SUT does not accept the SUT input immediately upon generation by the interface. A maximum delay of 21 ms passes before the SUT CV/AI function block executes appropriately. The SUT execution interval continues executing and SUT outputs are generated. SUT outputs are requested by the software simulator and returned through the interface before the 10 ms timeout. In the worst case scenario the maximum time for accurate HIL simulation transmission is slightly greater than 50 ms or $2 \times t_{SUT_{EI}}$. It is necessary for the delay to be slightly larger than 50 ms to allow for variance in all scheduled events within the HIL simulation platform.

5.3.2 Assumptions

DarlSIM has a minimum execution interval of 50 ms. However, the SDS1 execution interval is 200 ms. Therefore, for HIL simulation verification purposes, an execution interval of 200 ms is implemented. It is assumed that the dynamics of the evaluated process variable within DarlSIM are not updated during this interval.

The execution of the Tricon v9 PLC control logic is assumed to completely execute at least once within two of SUT execution intervals. Control logic should execute at least once within a 50 ms interval, for example, for a 25 ms execution interval.

Consecutively alternating SUT outputs are expected throughout the verification procedure. If the SUT output signals are delayed by an amount of time equal to the transmission of an even number of variables the signals could be misidentified as correlating with the expected output. If the output set [0,1,0,1,0,1] is expected, for example, and the output set [-,-,0,1,0,1] is observed, the two sets correlate for the set three through six. It is assumed that this does not occur for SUT delays which are larger than the SUT execution interval.

5.3.3 Analysis of results

The results of the verification procedure are presented in Figure 5.6. The correlation between the expected output and the actual output is illustrated. The correlation is presented for a range of SUT delays and two SUT execution intervals.

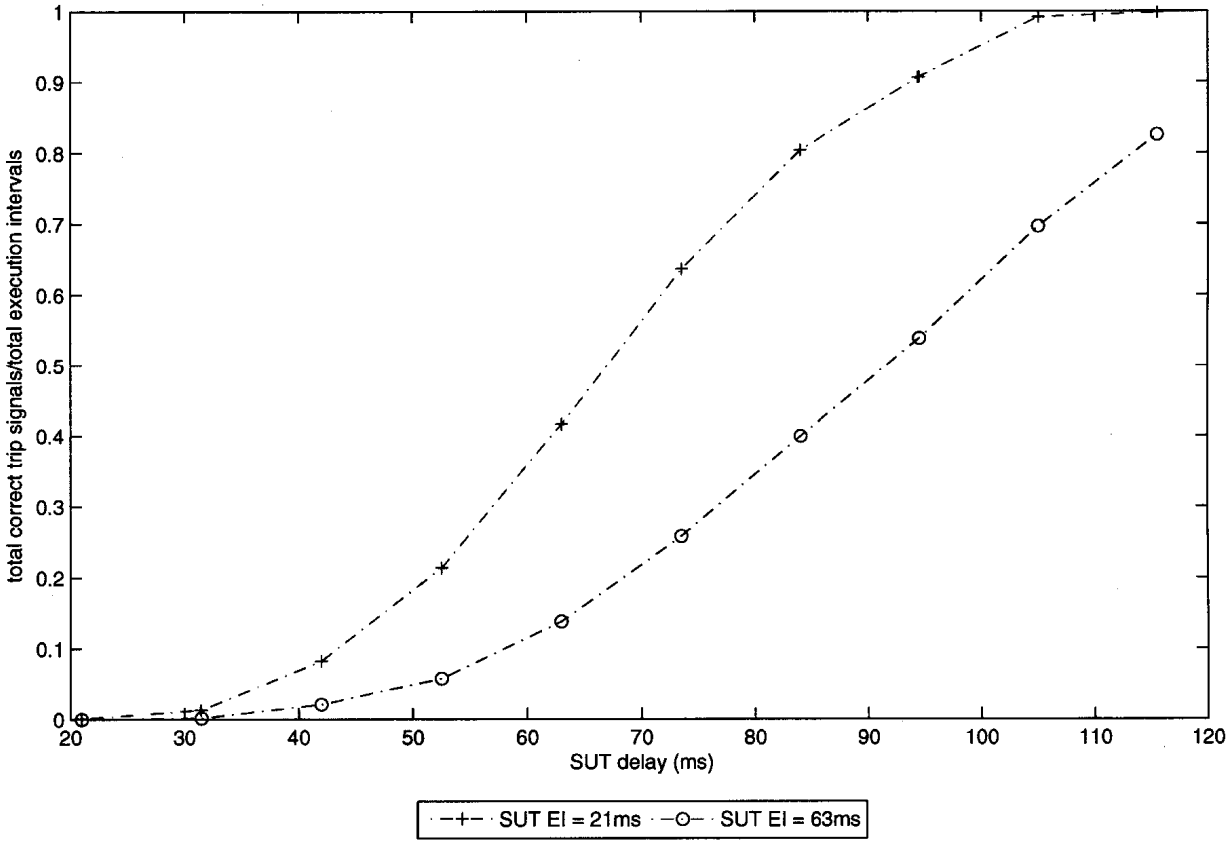

Figure 5.6: NPP HIL simulation platform availability vs. SUT delay with Tricon v9 PLC installed.

With the Tricon v9 PLC execution interval configured to 21 ms, to achieve 80 per cent availability the required SUT delay is approximately 83 ms. Further, 100 per cent availability is not achieved until 105 ms is implemented as SUT delay. An approximation for an SUT delay to achieve 80 per cent availability and 100 per cent availability are included in the following;

$$t_{SUT_{100\%}} = 5 \times t_{SUT_{EI}}, \tag{5.1}$$

$$t_{SUT_{80\%}} = 4 \times t_{SUT_{EI}}. \tag{5.2}$$

Similarly, with the Tricon v9 PLC execution interval of 63 ms the required time to 80 per cent is 113 ms. The 100 per cent availability of the Tricon v9 PLC for this SUT execution interval is not observed during the verification procedure. However, 100 per cent availability for the 63 ms SUT execution interval is not expected until

126 ms. An approximation for an SUT delay to achieve 80 per cent availability is the following;

$$t_{SUT_{80\%}} = 1.8 \times t_{SUT_{EI}}. \tag{5.3}$$

The resulting availabilities demonstrate significant overhead required for proper Tricon v9 PLC execution. From Figure 5.6, it is expected that at twice the SUT execution interval, all SUT output signals should accurately reflect the current SUT input signals. However, only 80 per cent of the output signals correlate with the input signals at almost four times the execution interval.

The ability of the HIL simulation platform comes into question when extensive delays are observed. Especially when these delay occur during simple system simulation. However, inspection of the entire sequence of events is required to determine the cause of the increased delay requirements.

The interface has been proven in Chapter 3 to be capable of replicating process variables transmitted through Ethernet using the developed UDP/IP structure. In fact, the signals generated at the output terminals of the interface demonstrated a maximum delay of 2 ms including the time required for the signal to return to the software simulator. For this reason, it is not acceptable to suggest that the cause of the above delays is the interface device.

A better understanding of the sequence of events occurring during the HIL simulation verification procedure is found when reviewing the operation of the Tricon v9 PLC. A review of the Tricon v9 PLC is included in Appendix D. The process within the Tricon v9 is as follows. The Tricon v9 PLC system contains three main processor modules to control three separate legs of I/O within the system. Each main processor operates in parallel with the other two main processors, as a member of a triad. A dedicated communication processor on each main processor manages the data exchanged between the main processors and the I/O modules.

As each input module is polled, the new input data is transmitted to the main processor over the appropriate leg of the I/O bus. Synchronization of the main processor is performed at the beginning of each scan. Each main processor sends its data to its upstream and downstream neighbours. One of two functions is then performed.

- *Transfer of data only* - for I/O, diagnostic and communication data.

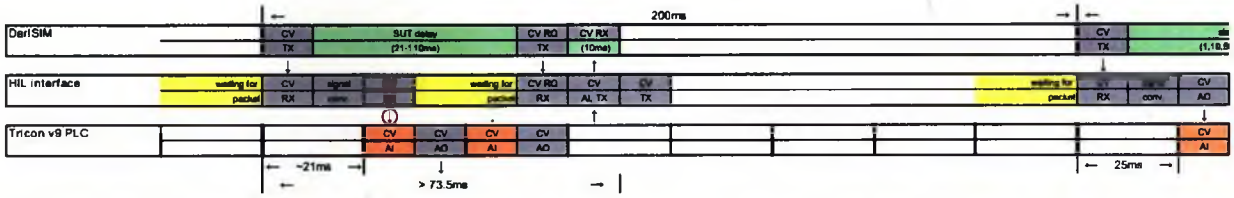

Figure 5.7: Sequence of events during HIL simulation platform verification for a 21 ms SUT execution interval.

- *Comparing data and flagging disagreements* -for previous scans output data and memory of user written application.

The input data from each input module is assembled into a table in the main processor and stored in memory for use in the hardware voting process. The individual input table in each main processor is transferred to its neighbouring main processors. During this transfer, hardware voting takes place. If a disagreement is discovered, the signal value found in two out of three tables prevails, and the third table is corrected accordingly. After the transfer and input data voting have corrected the input values, these corrected values are used by the main processors as input to the control program. The 32-bit main microprocessor and a math co-processor execute the control program in parallel with the neighbouring main processor modules [72].

As the control program executes, a table of output values is generated. Using the table of output values, the I/O processor on each main processor generates smaller tables, each corresponding to an individual output module in the system. Each small table is transmitted to the appropriate leg of the corresponding output module over the I/O bus. The transmittal of output data has priority over the routine, scanning of all I/O modules.

It is apparent that the Tricon v9 PLC is a very complicated fault tolerant electronic control system. There is significant overhead built into the PLC to assure that all decisions are accurate. Further, the Tricon v9 PLC requires extensive voting and redundant functionality to satisfy the strict requirements of the nuclear safety critical control industry. For this reason, manufacturers recommended avoiding short execution intervals within the Tricon v9 PLC. If the execution interval configured on the Tricon v9 PLC is not adequate for overhead processing, communications, voting and other tasks, the deterministic operation of the PLC is not guaranteed.

Figure 5.7 demonstrates hypothetical timing characteristics of the HIL simulation platform when an inadequate execution interval for the implemented control

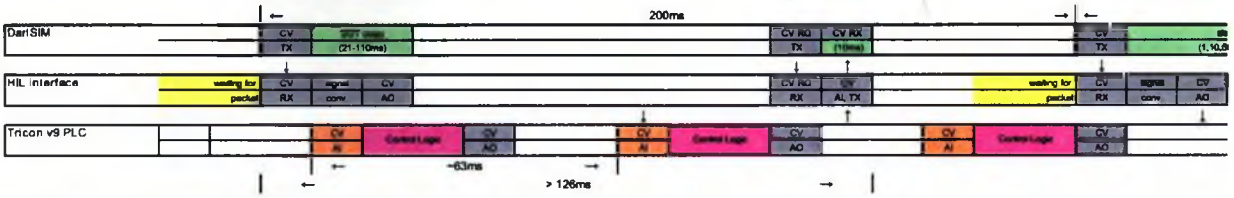

Figure 5.8: Sequence of events during HIL simulation platform verification for a 63 ms SUT execution interval.

logic is configured. The expected Tricon v9 PLC control logic execution interval in Figure 5.6 is 21 ms. Figure 5.7 also includes the 21 ms execution interval of the Tricon v9 PLC. However, after observing interface availability within the HIL simulation platform, the expected response illustrated in Figure 5.6 is not accurate. It is expected that this is due to the configuration of an execution interval that does not satisfy the proper operation of the triplicated Tricon v9 PLC central processing units (CPUs).

The exact operation of the Tricon v9 PLC is not known. However, Figure 5.7 illustrates a sequence of events which could result from unsynchronized Tricon v9 PLC CPUs, inadequate time for voting, or other potential issues caused by lack of processing time within the configured execution interval.

The Tricon v9 PLC does not accept the SUT input immediately upon generation, indicated by the circled arrow. A maximum delay of 21 ms passes before the SUT CV/AI function block executes. The SUT input signal is not acquired due to misaligned voting among the CPUs. On the next execution interval the SUT inputs are acquired successfully as indicated by the green arrow. The SUT execution interval continues to execute the control logic and the SUT outputs are generated. SUT outputs are requested by the software simulator and returned through the interface before the 10 ms timeout. Therefore, in the sample scenario the maximum time for accurate HIL simulation transmission is greater than 70 ms. This sequence of events would produce results similar to the availability observed with a 21 ms SUT execution interval.

For comparison, Figure 5.8 illustrates the worst case scenario for the operation of the Tricon v9 PLC configured with a 63 ms SUT execution interval. The additional time allotted for control logic execution is essential for the Tricon v9 PLC to operate properly. Voting mechanisms are executed properly and synchronization is maintained between the three CPUs with the Tricon v9 PLC.

Again, the exact operation of the Tricon v9 PLC is not known. However, Figure 5.8 illustrates a probable scenario when synchronization between Tricon v9 PLC CPUs, voting routines, or other redundant functions are operating properly.

The Tricon v9 PLC CV/AI function executes immediately prior to SUT input signal generation from the interface. Therefore, the SUT input signal is acquired in the next execution interval after a maximum delay of $\leq$ 63 ms. Voting among the 3 CPUs commences. The SUT execution interval then continues to execute the control logic and the SUT outputs are generated. The SUT outputs are requested by the software simulator and returned through the interface before the 10 ms timeout. Therefore, in the sample scenario the maximum time for accurate HIL simulation transmission is greater than 120 ms. The expected time satisfies the observed operation of the interface when a 63 ms SUT execution interval is configured. Again, the expectation for an SUT delay of twice the SUT execution interval is due to the asynchronous operation of the SUT and the software simulator, as covered in Section 3.2.2.2.

5.4 NPP Shutdown System Hardware-in-the-loop Simulation

5.4.1 Method

With all preliminary development and verification tasks complete, a NPP shutdown system HIL simulation is performed in this chapter. The control system being evaluated during the HIL simulation is SDS1 of CANDU NPPs. More specifically, the SGLL logic of SDS1 is replicated in the installed SUT. SUT performance is then evaluated during NPP design basis events and shutdown scenarios.

A Tricon v9 PLC is programmed with the SGLL trip detection logic reviewed in Section 5.2.5. Function block diagrams are used to program the logic within the PLC. Also, the Tricon v9 PLC is physically connected to the interface through four analog SUT input channels and one digital SUT output channel. The same configuration used during HIL simulation platform verification. For reference, PLC source function block diagrams are available in Appendix E.

With the Tricon v9 PLC connected in the loop, the HIL simulation platform is complete. The performance of the Tricon v9 PLC within the NPP HIL simulation

platform is evaluated against bench-mark SDS1 simulation. Bench-mark response to the shutdown scenario is generated through an entirely software simulation.

5.4.1.1 Shutdown scenario

The shutdown scenario selected for the current study involves both the SCFW and a valve position malfunction. SDS1 shutdown scenarios are developed according to the individual trip parameters presented in Figure 2.9. A set of design base event exist for each of the trip parameters. The SCFW initiates one such design base event , loss of secondary side heat removal [74]. This design base event directly affects the steam generator level. Upon initiating SCFW within the feed-water system of any of the four steam generators, the corresponding steam generator level will decrease. If the steam generator level decreases to an unsafe level the reactor will trip upon SGLL detection.

For all SGLL simulations, the cause of SDS1 channel D trip is the spurious closure of LCV101 within the feed-water system illustrated in Figure 5.9. LCV101, 102 and 103 control flow of light water coolant to the North-West (NW) steam generator from the feed-heating system. When a spurious closure is detected on any of the valves (e.g. LCV101), the steam generator level controller opens a parallel valve (e.g. LCV103). The redundant design of NPPs is essential to avoid single points of failure resulting in costly reactor shutdown procedures. However, if this redundant valve (LCV103) fails, the NPP experiences a design base event and the design is no longer redundant. Therefore, to induce the design base event , LCV103 is restricted to only open partially. With the steam generator feed-water flow below sustainable level, LCV102 could be opened. However, the redundancy of the system has been compromised, the reactor must shutdown and repair procedures initiated.

The restore point associated with the design base event restores DarlSIM to the instant that the SCFW is initiated. Both the partial opening and the spurious closure occur the instant the simulator is removed from a frozen state. The performance of the emulated SDS1 logic within DarlSIM and the response of the NPP to the SGLL trip is performed over multiple iterations.

For proper identification of SDS1 channelized trip signals, channel E and F steam generator levels are biased -0.2m and +1.00m respectively. Biases are presented in Table 5.1 and are included to force the Tricon v9 PLC, or the emulated channel D electronic control system, to act as the general coincident trip signal for SDS1 2oo3

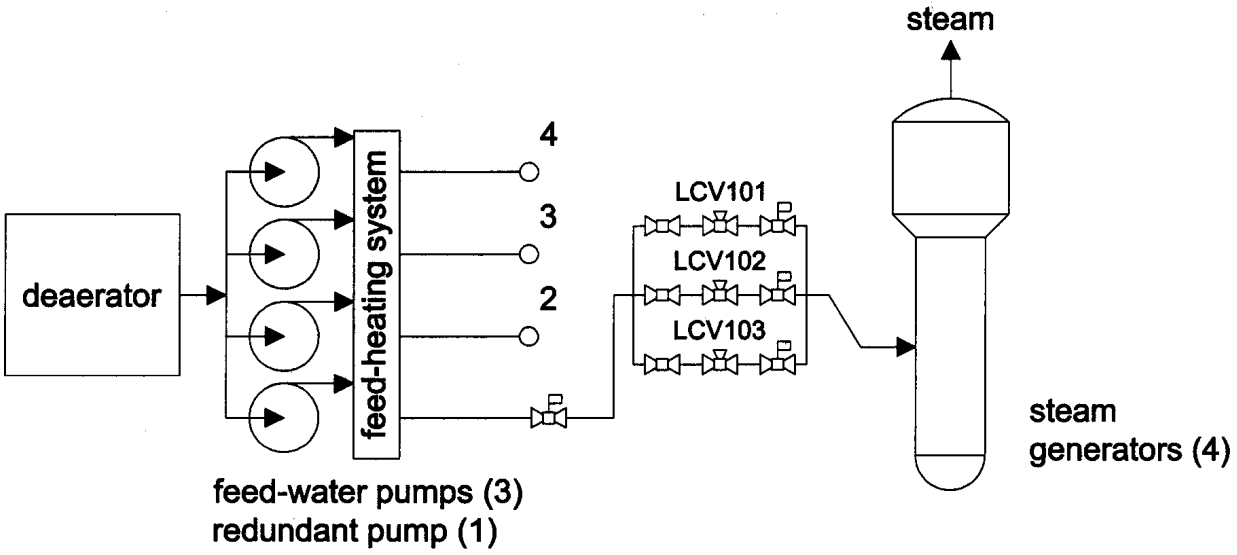

Figure 5.9: Steam generator feed-water system.

channel	channel D	ch-E	ch-F
	Tricon v9	DarlSIM SDS1	DarlSIM SDS1
bias	0.00m	-0.20m	+0.95m
trip set-point	2.00m	2.20m	1.05m

Table 5.1: Channelized steam generator level biases and low level trip thresholds.

relay trip logic. Thus, the Tricon v9 PLC initiates the reactor shutdown procedure and the release of the shut-off rods.

The eleven properties listed below are monitored during all SGLL simulations. The properties are essential in demonstrating proper functionality of SDS1 during the initiated design base event and are time stamped upon acquisition.

- channel D NW steam generator level (m)
- channel D NE steam generator level (m)
- channel D SW steam generator level (m)
- channel D SE steam generator level (m)
- reactor power (%)
- level control valve 101 position (%)
- level control valve 102 position (%)

- level control valve 103 position (%)
- channel D trip signal (high/low)
- channel E trip signal (high/low)
- channel F trip signal (high/low)

All properties are stored within a CSV database. A database record is created for each execution interval. Therefore, one record is collected for every 200 ms during each simulation. Complete observation of the shutdown scenario is performed over 120 seconds for a total of 600 execution intervals or equivalently 600 records. All calculations are performed using both MatLAB and Microsoft Excel tools. All plots are generated using MatLAB.

5.4.2 Assumptions

In performing bench-mark and HIL simulations of SGLL trip logic, it is assumed that the emulated logic within the bench-mark simulation is equivalent to the logic programmed within the SUT. The logic from DarlSIM is translated to compatible SUT control logic. However, some functions are not easily translated between electronic control system platforms.

No physical interconnections or simulations of the delays between simulated sensors and actuators and the terminals of the emulated electronic control system in DarlSIM. Within the simulator, engineering units are converted to milli-volt/milli-amp values. However, this conversion does not account for all dead and settling times experienced during actual electrical signal transmission in a NPP. In evaluating the effects of the interface on SDS1, the delays induced by the interface and the Ethernet network are assumed to be much larger than the propagation delays associated with electrical signal transmissions.

5.4.3 Bench-mark SDS1 SGLL simulation

Figure 5.10 illustrates the bench-mark simulation functions within DarlSIM. The data collection routines from the Ethernet signal transceiver module developed in Section 3.2.2 are installed as the illustrated data collection modules. The collected data is referenced as the norm when evaluating SUT functionality.

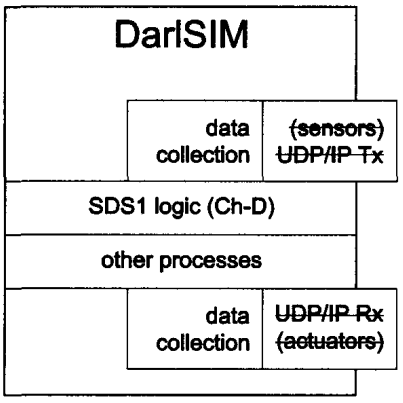

Figure 5.10: Simulation platform for bench-mark SDS1 evaluation.

The SDS1 logic executed within DarlSIM during bench-mark simulation is an emulation of the original Fortran source code programmed into the SDS1 electronic control system at Darlington NPP. During DarlSIM development the Fortran source code is modified by OPG training simulation personnel to allow interfacing with the remainder of the training simulator modules. It is assumed that these modifications have not compromised core SDS1 logic or simulator functionality.

Channel D SDS1 logic is compiled as the `dla113d` module and is included in the normal DarlSIM configuration module table. The module is identified in the module table by the corresponding label, 'SDS 1 CH D TRIP COMPUTER'. The module is responsible for the execution of the entire logic for channel D of SDS1. However, only a segment of the logic is evaluated in the following bench-mark and HIL simulations. The logic evaluated in the following is the SGLL trip logic illustrated in Figure 5.4. The execution interval is configured for 200 ms with an execution phase of 100 ms from the module table.

Two data collection modules are added to the module table. The SDS1 input collecting routines is scheduled prior to execution of the channel D SDS1 logic module. To capture the corresponding outputs produced by SDS1 a second data collection routine is scheduled to execute immediately after the channel D SDS1 logic module.

The following results provide analysis of the data collected during bench-mark simulation. The figures generated present a summary of the data collected. Every effort is made to present relevant illustrations of the data which assist in determining the proper operation of the NPP SDS1 resulting from a design base event. Following each figure, table or result is a discussion of the observations made from the material presented.

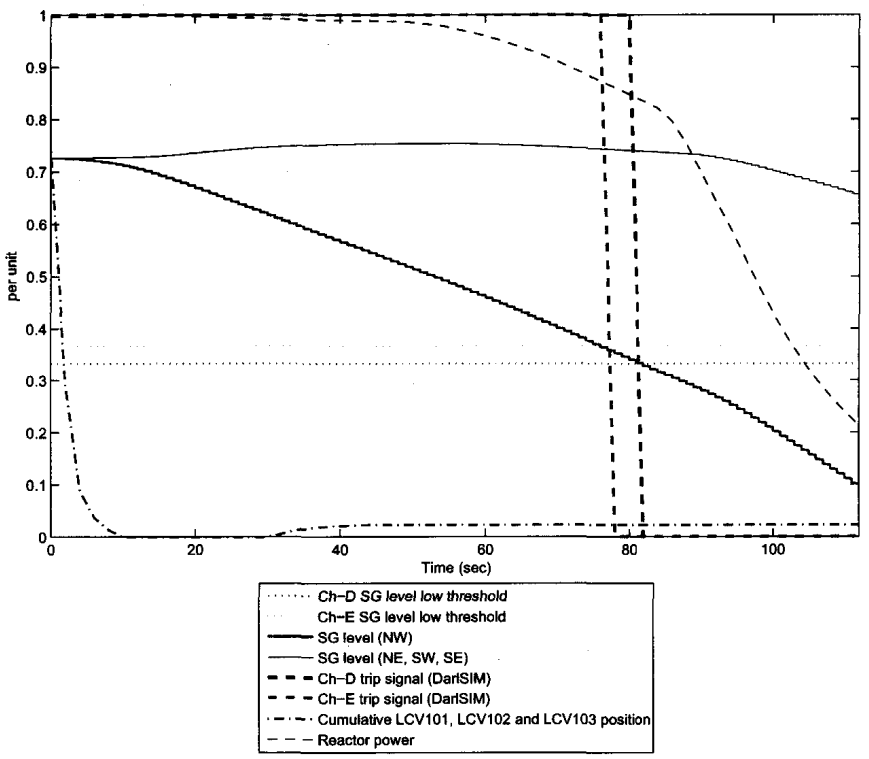

Figure 5.11: Bench-mark SGLL trip detection, simulation of Darlington NPP response to loss of secondary side heat removal design base event.

Figure 5.11 illustrates the response of SDS1 to the shutdown scenario. All eleven parameters are plotted. However, to simplify the plot, non-affected steam generator levels are averaged and valve percentages are summed. Further, SGLL thresholds are included. Each of the parameters is plotted using per unit measurements. The per unit value of each variable is calculated using the following;

$$\text{per unit} = \frac{cv_i - cv_{min}}{cv_{max} - cv_{min}}, \tag{5.4}$$

where cv_i is the current controlled variable value and cv_{max} and cv_{min} are the maximum and minimum controlled variable values. The parameters cv_{max} and cv_{min} for the parameters illustrated in Figure 5.11 are included in Table 5.2.

The initial conditions for all SCFW simulations are identified in Figure 5.11 at time 0 seconds. A reactor power of near 100 per cent, a steam generator level of approximately 4.35m and a cumulative LCV position of approximately 74 per cent are observed. LCV101 is the only operating valve at the inception of the scenario. This valve experiences a spurious closure immediately after the scenario begins.

The spontaneous closure of LCV101 causes the level of the affected steam gen-

Parameter	cv_{max}	cv_{min}
Ch-D steam generator level low threshold	6 meters	0meters
Ch-E steam generator level low threshold	6 meters	0meters
SG level (NW)	6 meters	0 meters
SG level (NE, SW, SE)	6 meters	0 meters
Ch-D trip signal (PLC)	high	low
Ch-E trip signal (DarlSIM)	high	low
Cumulative LCV101, LCV102...	33%	0%
Reactor power	100%	0%

Table 5.2: Controlled variable values.

erator to begin decreasing (~15 seconds). This decrease occurs because feed-water flow has been reduced from ~$308kg/s$ to $0kg/s$. However, the three additional steam generators maintain feed-water flow and do not exhibit level reduction.

After 30 seconds the redundant valve (LCV103) opens. However, the backup valve fails at 2.43 per cent, representing a 97.57 per cent blockage or a defective valve (~40 seconds). Feed-water requirements cannot be achieved, the steam generator level continues decreasing and the reactor power begins decreasing (~60 seconds).

The steam generator level continues decreasing toward the SGLL threshold. Reactor power begins decreasing linearly (70-80 seconds). When the steam generator level drops below the channel E steam generator level threshold (2.20meters) the channel E trip signal de-energizes (~78 seconds). The general coincident two-out-of-three (2oo3) relay voting logic has not been satisfied. The reactor power and steam generator level continue to decrease.

When the steam generator level drops below the channel D steam generator level threshold (2.00meters) the channel D trip signal de-energizes (~82 seconds). The 2oo3 relay voting logic has now been satisfied as illustrated in Figure 5.12 and the reactor shutdown procedure initiates.

After approximately 2 seconds the reactor power decreases rapidly. This response indicates proper shut-off rod deployment into the reactor core. The brief delay following the trip is expected during the shut-off rod insertion process. Though the rods are released very quickly following trip activation, the process of complete insertion and the absorption of neutrons sufficient for power reduction requires additional time.

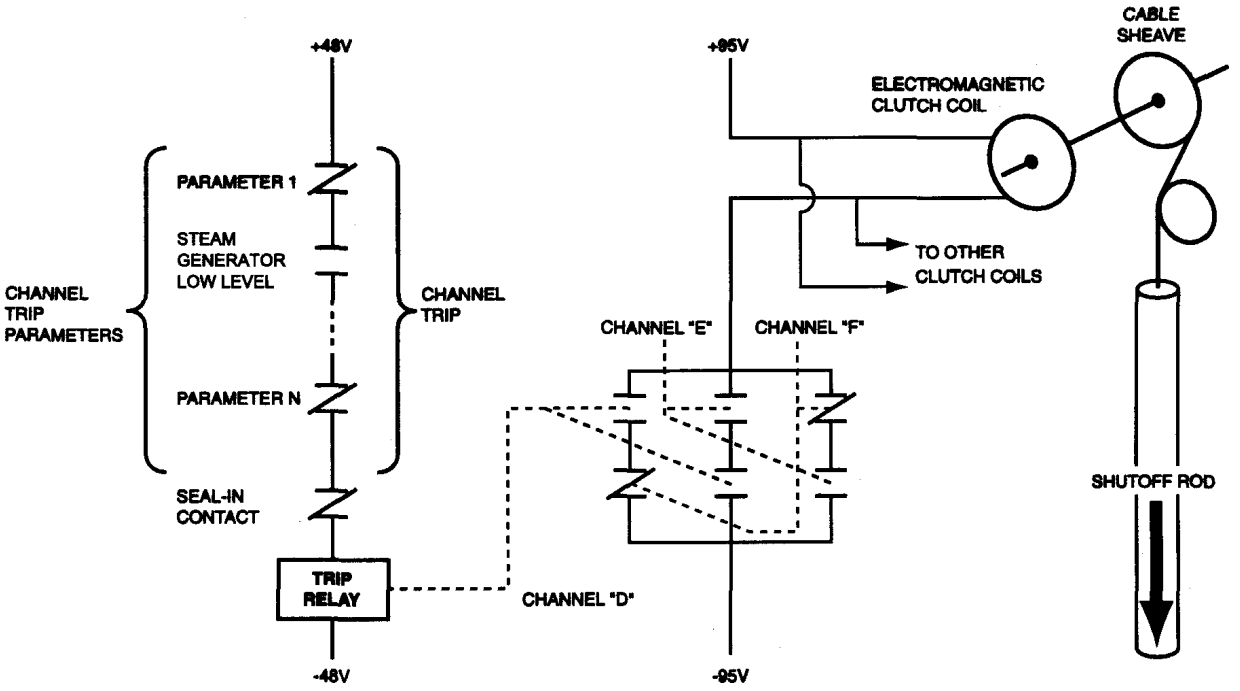

Figure 5.12: General co-incidence trip logic satisfied by channel D and E SGLL trip parameters.

The shutdown scenario is executed over 50 instances. All instances are reviewed to ensure proper system response and repeatability. Four of the instances are illustrated in Figure 5.13. All instances depict proper shutdown signal production by the simulated electronic control systems. Only slight variances in the times of the redundant valves opening and the time of channel E and D trip de-energization are observed.

5.4.4 Hardware-in-the-loop SDS1 SGLL simulation

Figure 5.14 illustrates the signal transmission functions within the HIL simulation platform. Modified routines from the Ethernet signal transceiver module developed in Section 3.2.2 are installed, including data collection and process variable acquisition and transmission. The module accesses the steam generator level sensors and trip contacts within the simulated NPP and communicates the variables between the interface.

The SDS1 logic executed within the Tricon v9 PLC during HIL simulation is also an emulation of the original Fortran source code programmed into the SDS1 electronic control system at Darlington NPP. During Tricon v9 PLC development the Fortran

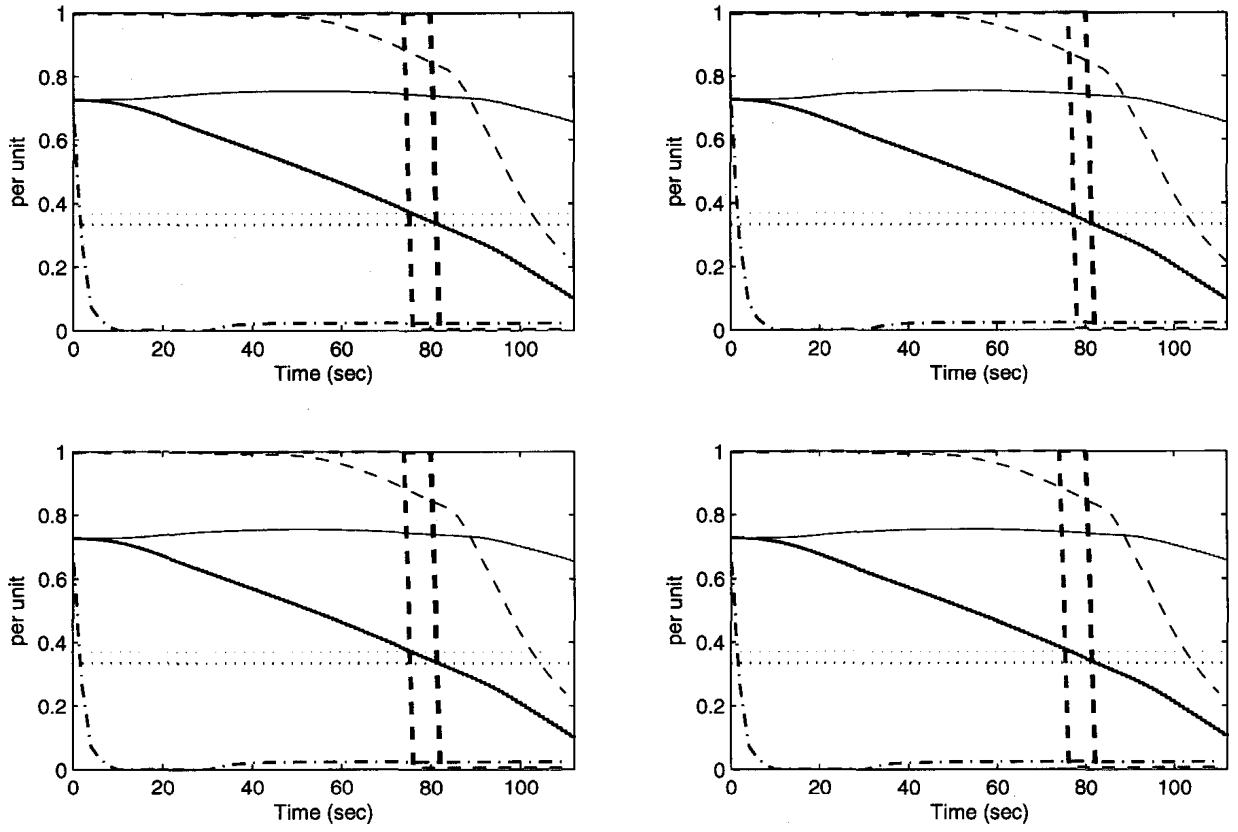

Figure 5.13: Four consecutive bench-mark simulation shutdown scenarios.

source code was translated from the source code used with DarlSIM. Again, only a segment of the SDS1 logic is evaluated in the following HIL simulations. The logic evaluated is the SGLL trip logic illustrated in Figure 5.4. SGLL logic is translated to function block diagrams and programmed onto the Tricon v9 PLC using the Invensys Tristation 1131 Developer's Workbench. It is assumed that logic modifications have not compromised the core SDS1 or SGLL logic.

To enable the replacement of the SDS1 controller from within DarlSIM the channel D SDS1 logic module is disabled within the HIL simulation configuration and module table. Disabling the SDS1 module removes the watchdog event from the simulated NPP safety systems. To replicate the shutdown system watchdog signal required by auxiliary safety systems, the SUT input and SUT output modules are modified to produced alternating watchdog signals. Therefore, DarlSIM can run without the SDS1 module enabled.

The two data collection modules added to the module table for bench-mark simulation remain. However, now the SDS1 data collecting routines are scheduled

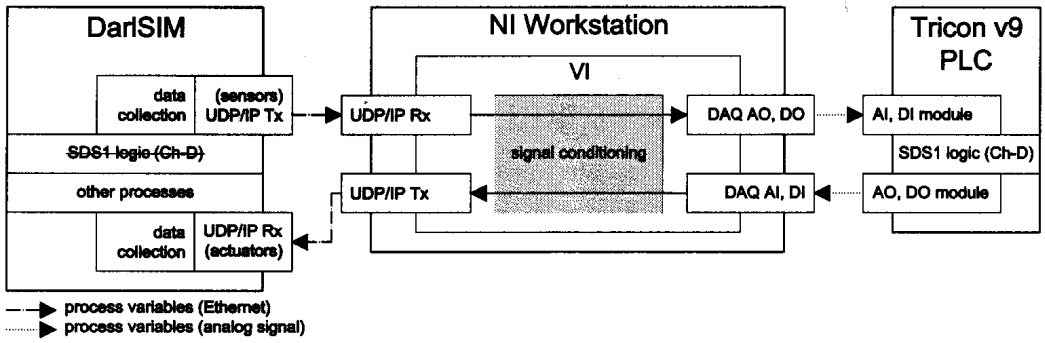

Figure 5.14: HIL simulation platform for SDS1 evaluation using the Tricon v9 PLC.

prior to the disabled channel D SDS1 logic module. Both data collection modules are placed within a single module includes signal transmission and delay routines. With channel D SDS1 logic removed, the controlled variable transceiver is compiled as the `dlacvt` module and is included in the HIL simulation configuration module table. The module is responsible for the transmission of controlled variables throughout the HIL simulation platform. The procedure executed within the module is very similar to the Ethernet signal transceiver in Section 4.1.1 and Section 4.2.1 . The adapted procedure follows:

begin

- *Initialize global variables* - socket connection, data collection, etc...,
- *Open UDP/IP sockets* - outgoing/incoming,

for: *execution interval* t_{EI} - C module is loaded (200 ms),

- *Initialize local variables* - packet buffer, controlled variable, etc...,
- *Controlled variable capture* - acquire steam generator level sensor values from CDB

for: *current number of SUT inputs*

- *Controlled variable Tx* - format and transmit the controlled variable, id, divisor and intercept through outgoing UDP/IP socket,

end for

- *Sleep* - t_{SUT} (100 ms),
- *Flush UDP/IP socket* - clear the incoming UDP/IP socket,

- *Request controlled variable return* - transmit id, divisor and intercept through outgoing UDP/IP socket,
- *Wait for interface* - poll incoming UDP/IP socket (t_{HIL}. 10 ms),

if: UDP/IP packet detected

- *Controlled variable Rx* - receive data on incoming UDP/IP socket and extract controlled variable and id,

if: received ID equal to transmitted ID

- *Controlled variable store* - capture the received controlled variable and store to CDB,

end for

end

The execution interval is configured for 200 ms with an execution phase of 100 ms from the module table. To achieve high interface availability an SUT execution interval of 40 ms is utilized as well as an SUT delay of 100 ms and an HIL timeout of 10 ms.

The following results provide analysis of the data collected during bench-mark simulation. The figures generated present a summary of the data collected. Every effort is made to present relevant illustrations of the data which assist in determining the proper operation of the NPP SDS1 resulting from a design base event. Following each figure, table or result is a discussion of the observations made from the material presented.

Figure 5.15 illustrates the response of SDS1 to the shutdown scenario. All eleven parameters are plotted. Again however, to simplify the plot, non-affected steam generator levels are averaged and valve percentages are summed. Further, SGLL thresholds are included. Also, each of the parameters is plotted using per unit measurements. The per unit value of each variable is calculated using (5.4). The cv_{max} and cv_{min} for the parameters illustrated in Figure 5.15 are included in Table 5.2.

The response of the NPP to the SCFW and resulting design base event is almost exactly as the bench-mark simulation. A reactor power of near 100 per cent, a steam generator level of approximately 4.35m and a cumulative LCV position of approximately 74 per cent are observed. LCV101 is the only operating valve at the

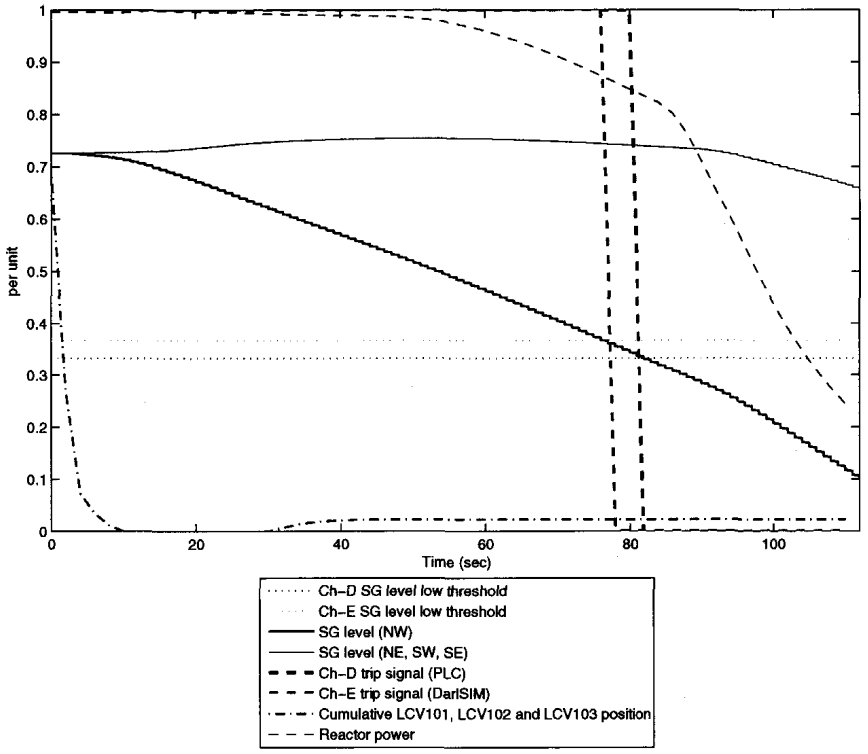

Figure 5.15: Hardware-in-the-loop SGLL trip detection, Tricon v9 PLC induced response to loss of secondary side heat removal design base event.

inception of the scenario. This valve experiences a spurious closure immediately after the scenario begins.

The remaining events are all the same as within the bench-mark, including the spontaneous closure of LCV101 causes the level of the affected steam generator to begin decreasing (~15 seconds). The redundant valve opening after 30 seconds. The backup valve failing at 2.43 per cent, representing a defective valve (~40 seconds). The steam generator level continues decreasing and the reactor power begins decreasing (~60 seconds).

Similarly, the steam generator level continues decreasing toward the SGLL threshold. Reactor power begins decreasing linearly (70-80 seconds). When the steam generator level drops below the channel E steam generator level threshold (2.20meters) the channel E trip signal de-energizes (~78 seconds). This signal is generated within DarlSIM. The channel E SDS1 logic is still enabled within the module table. The general coincident two-out-of-three (2oo3) relay voting logic has not been satisfied. Therefore, the reactor power and steam generator level continue to decrease.

When the steam generator level drops below the channel D steam generator

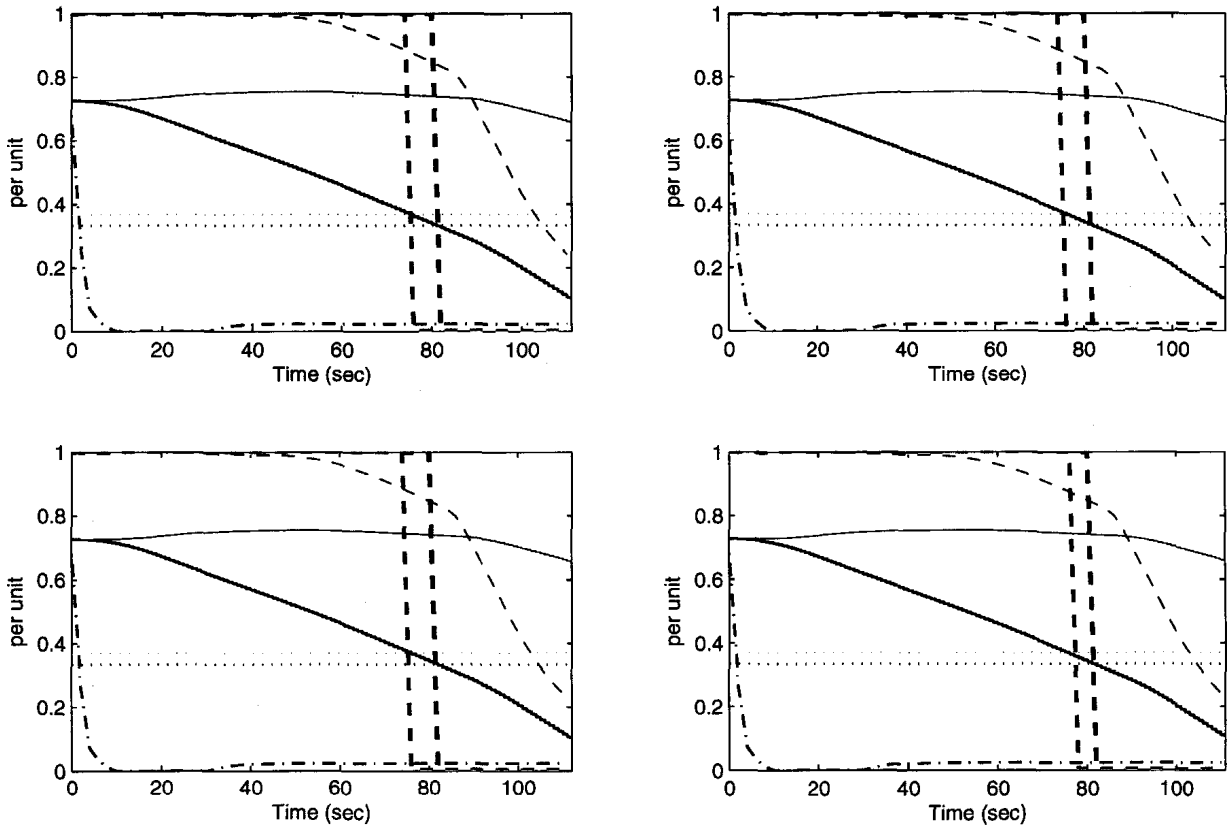

Figure 5.16: Four consecutive hardware-in-the-loop simulation shutdown scenarios.

level threshold (2.00meters) the channel D trip signal de-energizes (~82 seconds). However, this signal is not generated within DarlSIM. The Ch-D trip signal (PLC) is generated by the Tricon v9 PLC. The signal is transmitted back to the software simulator within the same execution interval as occurrence of the trip condition. In other words, the steam generator level which causes the trip is transmitted within the same execution interval as the Ch-D SDS1 SUT trip output, or Tricon v9 PLC digital output. The 2oo3 relay voting logic has now been satisfied and the reactor shutdown procedure initiates. Proper shut-off rod deployment into the reactor core is performed after approximately 2 seconds.

The shutdown scenario is executed over 50 instances. All instances are reviewed to ensure proper system response and repeatability. Four of the instances are illustrated in Figure 5.16. All instances depict proper shutdown signal production by the simulated electronic control systems and the Tricon v9 PLC. Again, only slight variances in the times of the redundant valves opening and the time of channel E and D trip de-energization are observed.

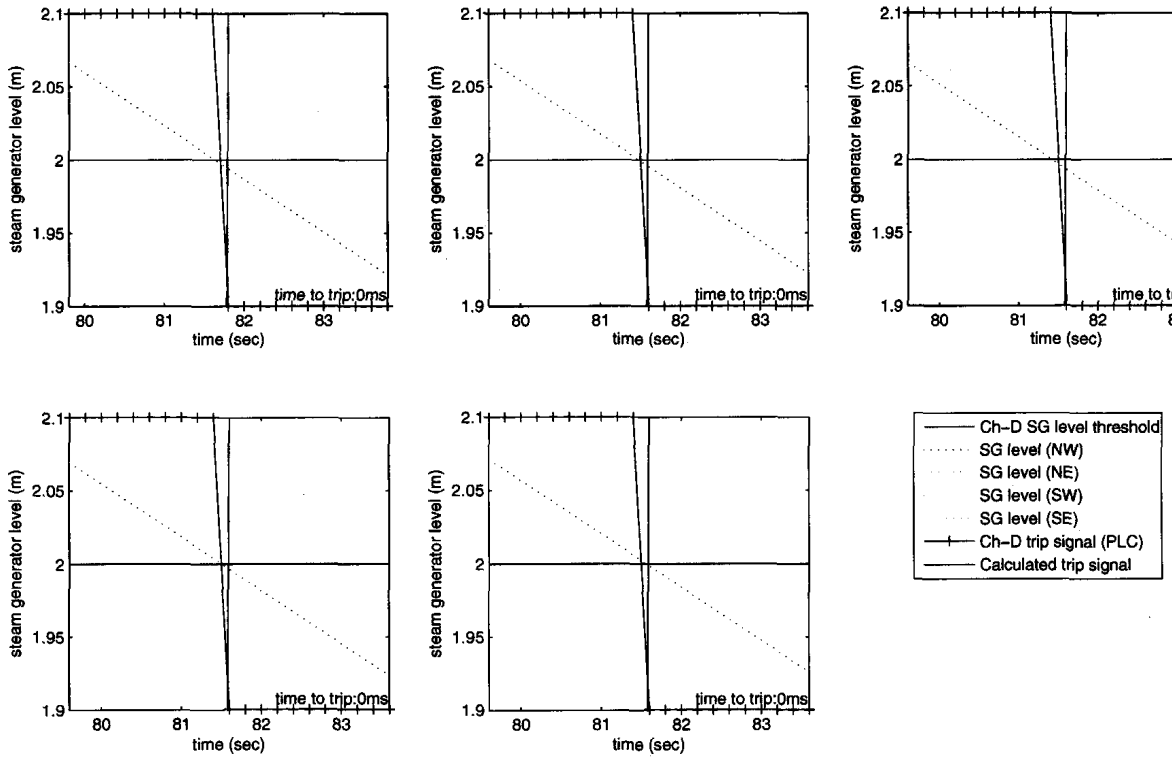

Figure 5.17: Hardware-in-the-loop SGLL trip detection, detailed analysis of Tricon v9 PLC induced response to loss of secondary side heat removal design base event.

In Section 5.3.1 previous discrepancies in HIL simulation results are discussed. The discrepancies are the result of inadequate delays within the HIL simulation platform. The purpose of the verification procedure of Section 5.3.1 is to assure that the transmission of SUT inputs and the resulting SUT outputs occurs within the same execution interval.

Figure 5.17 illustrates the detailed response of SDS1 to the shutdown scenario. Seven parameters are plotted. Each parameter is plotted for steam generator level and SGLL trip analysis. It is apparent in all five consecutive shutdown scenarios that the Tricon v9 PLC trip signal is produced within the same execution interval (200 ms) as the calculated trip signal.

Chapter 6
Conclusions

The research performed and presented in this thesis focuses on the application of HIL simulations in NPP safety control system replacement routines and shutdown system control logic evaluation. The research began with the proposal of a procedure for developing an interface. The procedure included developmental testing and verification routines. Ultimately, the proposed procedure resulted in the development of an interface. The HIL interface was then verified for proper connection and transmission capabilities. The interface was designed to be compatible with a range of electronic control systems and software simulators. Extensive observations and discussion were included to determine the proper timing of process variables during HIL simulation and physical limitations of the developed interface.

The remainder of the thesis included a procedure for installing the developed HIL interface into an HIL simulation platform. The procedure was executed for NPP safety critical systems. More specifically, a Tricon v9 safety PLC is programmed to replicate the SGLL logic of CANDU SDS1. The HIL simulation platform was verified to assure proper signal transmission capabilities. Finally, a bench-mark response of the simulated Darlington NPP to a design basis event with SDS1 initiated by the simulated electronic control system within the simulator was compared with the response of the NPP to a design basis event with SDS1 trip logic installed with the Tricon v9 PLC.

6.1 Summary of contributions

The major contributions of the work presented in this thesis included the following:

- The functionality of an existing software-based NPP training simulator was extended to include HIL simulation capabilities. Through the developmental procedure proposed and executed in Chapter 3, an interface device was developed

which supports the existing Unix NPP training simulator. Though communication with the simulator by external hardware was accomplished prior to HIL interface device development, only simple communications including few I/O transmission were performed. The introduced interface development procedure provides a method of assuring that proper connectivity and signal transmissions occur for a range of evaluated processes and I/O configurations on the software simulator. Therefore, electronic control system verification was enabled and a measure level of confidence in the simulations performed was established.

- A generic procedure for developing HIL interface devices was proposed. Execution of the proposed procedure included a variable transmission verification procedure. Verifying an interface performance prior to performing HIL simulation revealed the expected timing requirements within HIL controlled variable transmission routines. Further, signal reproduction biases and an interface availability are measured. These parameters are essential in performing accurate HIL simulations. In a secondary study, limitations for multiple variable transmissions were revealed. The procedure can be applied to the development of any interfacing device.
- A flexible interface device, including customized NI LabVIEW VI, was created. In following the developed procedure, an interface device is developed. The device was not limited to the original application of NPP training simulator HIL simulation. Throughout the developmental procedure, the device was designed to be flexible. Therefore, the requirement for connectivity to the device included Ethernet communication media and industry standard analog and digital signals.
- A Tricon v9 safety PLC was installed along with the NPP training simulator and the interface device to perform HIL simulations. Chapter 5 included a procedure for installing sub-systems within an HIL simulation platform. The procedure included verification of interface transmission capabilities following interface installation. During HIL simulation platform development, the timing requirements for performing actual HIL simulations were revealed. The requirements differ from those found during interface analysis due to the installation of an SUT. Within this procedure a measure of the availability of the platform for a specified application was observed. This availability is essential to ensuring

accurate HIL simulation results.

- The Tricon v9 PLC was programmed as a CANDU SDS1 Ch-D control logic and installed within the NPP HIL simulation environment. Again, the Tricon v9 PLC integration was verified. An availability of over 99 per cent was achieved for the PLC using the developed HIL interface device. However, this availability was only realized at above expected delays within the HIL simulation platform. Even so, the delays are suitable for a range of applications.
- A NPP shutdown scenario is illustrated, where Ch-D SGLL trip is induced by the Tricon v9 PLC. The response of the NPP was compared against a benchmark response and determined to be nearly identical. Therefore, it was demonstrated that the HIL simulation environment is appropriate for evaluation of NPP processes. Upon closer inspection the Ch-D trip signal acquired by the software simulator occurred exactly when the software simulator would normally realize an internal trip.

In the end, all research objectives were successfully met. A method for hardware connectivity to the software NPP simulator was achieved, modules were developed for the acquisition of process variables and an interface that supports connection to the Tricon v9 PLC was developed. Further, the procedures proposed within this thesis can be applied in many industries, and the functionality of many existing simulators can be extended to include HIL simulation.

The three distinct motivations behind the research performed in this thesis are satisfied to some degree. The preliminary steps have been established in the need for both a process and a platform for the qualification of shutdown system hardware and software based on relevant standards through HIL simulation. Further, the replacement of obsolete digital and analog shutdown system hardware and software was enabled. The HIL simulation platform supports the installation of potential electronic control system for evaluation, and provides a measure availability or assurance for the accuracy of the HIL simulation. However, the dynamics of the process evaluated should be reviewed. Finally, though the integration of the enhanced functionalities was only performed by installing an advanced PLC for safety critical control, the implementation of enhanced functions was supported and HIL simulation can be performed.

6.2 Suggestions for future work

The following subjects are suggested for the continuation of work on HIL platform development:

- The highest frequency process supported by the HIL interface was not revealed within. The frequency does depend on number of I/O. However, it would be useful to understand the capabilities of the platform for average HIL simulations.
- The developed HIL interface device was installed and executed within a Microsoft Windows OS. It is recommended that the interface be converted to an independent OS platform.
- Parallel application of the developed HIL interface should be performed and analyzed. Understanding the limits of the interface includes examining the capabilities of the connectivity media.

The following subjects are suggested for the continuation of work on HIL simulation of NPP processes:

- Within the Tricon v9 PLC, the entire SDS1 logic should be replicated. Only SGLL was implemented within this thesis.
- The enhanced capabilities of modern controllers should be evaluated further. In this thesis a modern controller was installed. However, the enhanced capabilities of the controller were not evaluated.
- A modified UDP/IP structure including byte-wise transmission capabilities for process variables should be developed.
- A comparison of the performance of the Tricon v9 PLC against other NPP qualified safety critical controllers (Teleperm XS (T3000), Common Q, etc.) should be performed.

The suggested future work is an extension of the research performed within this thesis. The above are problems that are the result of the work performed to solve previously identified problems. However, the two major problems identified prior to the research presented in this thesis were satisfied. The development of an HIL simulation platform for NPP process evaluation was extensively covered and included assurances for simulation performance. Also, the problem of simulating

a NPP process, or more specifically the SDS1 logic of a CANDU NPP, has been addressed. Though there are obsolete systems in NPPs, the developed platform can be used to minimize the amount of work required to replace these systems.

References

[1] International Atomic Energy Agency. Factors relevant to the recycling or reuse of components arising from the decommissioning and refurbishment of nuclear facilities. (293), Feb. 1989.

[2] International Atomic Energy Agency. Guidebook on spent fuel store. (240), Apr. 1991.

[3] International Atomic Energy Agency. Principle for intervention for protection of the public in a radiological emergency. (63):22p, 1991.

[4] International Atomic Energy Agency. Modern instrumentation and control for nuclear power plants: A guidebook. (387):629p, Sep. 1999.

[5] International Atomic Energy Agency. Specification of requirements for upgrades using digital instrumentation and control systems, Jan. 1999.

[6] International Atomic Energy Agency. Verification and validation of software related to nuclear power plant instrumentation and control. page 126p, May 1999.

[7] International Atomic Energy Agency. Assessment and management of ageing of major nuclear power plant components important to safety: Candu reactor assemblies. (1197):54p, Feb. 2001.

[8] International Atomic Energy Agency. Instrumentation and control systems important to safety in nuclear power plants safety guide. page 91p, Apr. 2002.

[9] International Atomic Energy Agency. Evaluation of seismic hazards for nuclear power plants safety guide. (NS-G-3.3):31p, Mar. 2003.

[10] International Atomic Energy Agency. Managing change in the nuclear industry: The effects on safety. (18):12p, Feb. 2003.

[11] International Atomic Energy Agency. Solutions for cost effective assessment of software based instrumentation and control systems in nuclear power plants. page 136p, Jan. 2003.

[12] International Atomic Energy Agency. Use of control room simulators for training of nuclear power plant personnel. (1411):101p, Sep. 2004.

[13] International Atomic Energy Agency. Assessment of defence in depth for nuclear power plants. (46):120p, Apr. 2005.

[14] International Atomic Energy Agency. Fundamental safety principles. page 21p, Nov. 2006.

[15] International Atomic Energy Agency. Guidelines for upgrade and modernization of nuclear power plant training simulators. (1500):89p, Aug. 2006.

[16] International Atomic Energy Agency. Energy, electricity and nuclear power estimates for the period up to 2030. (1):56p, July 2007.

[17] International Atomic Energy Agency. On-line monitoring for improving performance of nuclear power plants: Instrument channel monitoring. page 109p, Oct. 2008.

[18] International Atomic Energy Agency. The role of instrumentation and control systems in power uprating projects for nuclear power plants. (NP-T-1.3), Dec. 2008.

[19] International Energy Agency. World energy outlook 2007 executive summary: China and india insights. page 18p, 2007.

[20] Nuclear Energy Agency. Nuclear energy today. page 112p, 2005.

[21] Nuclear Energy Agency. Nea annual report 2007. page 48p, 2008.

[22] Canadian Nuclear Association. Canada's nuclear energy: Reliable, affordable and clean electricity. page 31p, 2008.

[23] Canadian Nuclear Association. Candu nuclear reactor performance 2007. page 1p, 2008.

[24] V.V. Balashov, A.G. Bakhmurov, M.V. Chistolinov, R.L. Smeliansky, D.Yu. Volkanov, and N.V. Youshchenko. A hardware-in-the-loop simulation environment for real-time systems development and architecture evaluation. *Dependability of Computer Systems, 2008. DepCos-RELCOMEX '08. Third International Conference on*, pages 80–86, June 2008.

[25] I.D. Baxter and M. Mehlich. Reverse engineering is reverse forward engineering. *Reverse Engineering, 1997. Proceedings of the Fourth Working Conference on*, pages 104–113, Oct. 1997.

[26] G. Bereznai. Heat transport. page 19p, Nov. 1996.

[27] G. Bereznai. Overall unit. page 31p, Nov. 1996.

[28] G. Bereznai. Reactor and moderator. page 19p, Nov. 1996.

[29] G. Bereznai. Special safety systems. page 35p, Nov. 1996.

[30] G. Bereznai. Steam, turbine and feedwater. page 19p, Nov. 1996.

[31] Atomic Energy Control Board. Requirements for shutdown systems for candu nuclear power plants, Feb. 1991.

[32] F.E. Celier and L.C. Schooley. Computer-aided design of intelligent controllers: Challenge of the nineties. *Recent Advances in Computer-Aided Control Systems*, pages p53–77, 1992.

[33] K.C. Chang and C.A. Lomasney. Obsolete integrated circuit replacement methodology using advanced electronic design automation technology. *Circuits and Systems, 1999. ISCAS '99. Proceedings of the 1999 IEEE International Symposium on*, Vol. 1:400–403, July 1999.

[34] D. M. Chapin. Digital instrumentation and control systems in nuclear power plants safety and reliability issues, 1997.

[35] S.W. Cheon, J.S. Lee, K.C. Kwon, D.H. Kim, and H. Kim. The software verification and validation process for a plc-based engineered safety features-component control system in nuclear power plants. *Industrial Electronics Society, 2004. IECON 2004. 30th Annual Conference of IEEE*, Vol. 1:827–831, Nov. 2004.

[36] Canadian Nuclear Safety Commission. Trip parameter acceptance criteria for the safety analysis of candu nuclear power plants. page 14p, 2006.

[37] Ontario Power Generation Review Committee. Pickering. pages p47–71, Mar. 2004.

[38] Hewlett-Packard Company. Alphaserver ds25 owner's guide. page 281p, Apr. 2003.

[39] D. Craigen, S. Gerhart, and T. Ralston. Case study: Darlington nuclear generating station [software-driven shutdown systems]. *Software, IEEE*, Vol. 11(1):30–32, Jan. 1994.

[40] L.E. Crossley, J.R. Finlay, and N.R. Pillai. The development of a programmable controller system for use in ontario hydro nuclear power plants. *IEEE Transactions on Power Apparatus and Systems*, Vol. PAS-102(1):123–126, Jan. 1983.

[41] R. M. Edwards. Testbed for nuclear plant instrumentation and control validation. Proc. American Nuclear Society Int. Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies, May 1996.

[42] L. Ferrarini and R. Ciancimino. Modular simulation for logic, distributed, real-time control systems. *Proceedings of the IEEE/IFAC Joint Symposium on Computer-Aided*, pages p225–30, Mar. 1994.

[43] H. Gall. Functional safety iec 61508 / iec 61511 the impact to certification and the user. *Computer Systems and Applications, 2008. AICCSA 2008. IEEE/ACS International Conference on*, pages 1027–1031, Apr. 2008.

[44] W. J. Garland. Design basis accidents. page 5p, 1998.

[45] Ontario Power Generation. Energy balance. pages p128–135, Apr. 1993.

[46] Ontario Power Generation. Major components of candu reactors. pages p58–73, Apr. 1993.

[47] Ontario Power Generation. Unit operational control. pages p136–138, Apr. 1993.

[48] O. Glockler. Testing the dynamics of shutdown systems instrumentation in reactor trip measurements. Vol. 43:p91–96, 2003.

[49] M. Gomez. Hardware-in-the-loop simulation. Nov. 2001.

[50] J. de Grosbois, G. R. Mitchel, and R. Brown. Application-specific qualification of digital i&c products in a safety-related nuclear context. page 28p, Nov. 2007.

[51] J.M Hopwood and P.J. Allen. Candu 6 evolution. *Proceedings of the Canadian Nuclear Association Annual Conference*, Vol. 1:2p, June 1997.

[52] N. M. Ichiyen, D. Chan, and P. D. Thompson. Point lepreau refurbishment project programmable digital comparison (pdc) replacement for sds1 and sds2. *Proceedings of the 24th Annual Canadian Nuclear Society Conference*, June 2003.

[53] J. S. Janosy. Simulator-aided instrumentation and control system refurbishment at paks nuclear power plant. *Modelling & Simulation, 2007. AMS '07. First Asia International Conference on*, pages 53–58, Mar. 2007.

[54] M.N. Khajavi, M.B. Menhaj, and A.A. Suratgar. Fuzzy adaptive robust optimal controller to increase load following capability of nuclear reactors. *Power System Technology, 2000. Proceedings. PowerCon 2000. International Conference on*, Vol. 1:115–120, 2000.

[55] M.N. Khajavi, M.B. Menhaj, and A.A. Suratgar. Comparison of three modern controllers for wide range power regulation of nuclear reactors. *Fuzzy Systems, 2001. The 10th IEEE International Conference on*, Vol. 3:1235–1238, 2001.

[56] H. Klee. Simulation and design of a digital control system with tutsim. *Education, IEEE Transactions on*, Vol. 34(1):76–82, Feb. 1991.

[57] K. C. Kwon and C. S. Ham. Proposed plan for the development of advanced instrumentation and control technology in korea. Proceedings of the IAEA Technical Committee Meeting on Advanced C&I Systems in NPPs, June 1994.

[58] K. C. Kwon, S. J. Song, M. N. Park, and S. P. Lyu. The real-time functional test facility for advanced instrumentation and control in nuclear power plants. *Nuclear Science, IEEE Transactions on*, Vol. 46(2):92–99, Apr. 1999.

[59] Zhen Li, M. Kyte, and B. Johnson. Hardware-in-the-loop real-time simulation interface software design. *Intelligent Transportation Systems, 2004. Proceedings. The 7th International IEEE Conference on*, pages 1012–1017, Oct. 2004.

[60] L. Lu and G. Lewis. Reliability evaluation of standby safety systems due to independent and common cause failures. *Automation Science and Engineering, 2006. CASE '06. IEEE International Conference on*, pages 264–269, Oct. 2006.

[61] C. J. Luxat. Safety analysis technology: Evolution, revolution and the drive to re-establish margins. page 19p, Sep. 2000.

[62] D.A. Meneley and Y.Q. Ruan. Comparison of phwr and pwr. page 25p, Sep. 1998.

[63] D.A. Meneley and Y.Q. Ruan. Moderator, hts, heavy water. page 15p, Sep. 1998.

[64] D.A. Meneley and Y.Q. Ruan. Reactor and fuel handling. page 12p, Sep. 1998.

[65] A. Meyer, L. Pretorius, and J.H.C. Pretorius. A model using an obsolescence mitigation timeline for managing component obsolescence of complex or long life systems. *Engineering Management Conference, 2004. Proceedings. 2004 IEEE International*, Vol. 3:1310–1313, Oct. 2004.

[66] A.I. Miller and H.M. van Alstyne. Heavy water: A distinctive and essential component of candu. (10962):11p, June 1994.

[67] J.F. Miller and R.W. McDowell. Significance of analog instrumentation-design philosophy of replacement dump arrest unit at pickering station candu reactor. *Nuclear Science, IEEE Transactions on*, Vol. 44(3):1081–1083, June 1997.

[68] B. Mishra and R. Harrington. Hybrid simulation and digital controller design for a nuclear propulsion plant. *OCEANS*, Vol. 7:629–633, Sep. 1975.

[69] J. Murray. Qualification lifecycle and method of obsolescence management of the invensys tricon. *Technical Meeting on Impact of Modern Technology on Instrumentation and Control in Nuclear Power Plants*, page 9p, Sep. 2005.

[70] Institute of Electrical and Electronics Engineers. Ieee standard for qualification of class 1e static battery chargers and inverters for nuclear power generating stations. *ANSI/IEEE Std 650-1979*, Oct. 1979.

[71] V. Okol'nishnikov and A. Zenzin. Use of simulation for development of process control system. *Computational Technologies in Electrical and Electronics Engineering, 2008. SIBIRCON 2008. IEEE Region 8 International Conference on*, pages 248–251, July 2008.

[72] Drew J. Rankin. Tricon and shutdown system one (sds1), June 2007.

[73] Drew J. Rankin. Hardware-in-the-loop nuclear power plant training simulation platform design and validation, June 2008.

[74] Drew J. Rankin, Jingke She, and Jin Jiang. Evaluation of safety plcs and fpgas for shutdown systems in candu nuclear power plants, Sep. 2008.

[75] B. Rouben. Introduction to reactor physics. page 54p, Sep. 2002.

[76] J.J. Sammarco. Programmable electronic and hardwired emergency shutdown systems: A quantified safety analysis. *Industry Applications, IEEE Transactions on*, Vol. 43(4):1061–1068, July 2007.

[77] M.H. Sanwarwalla and A.J. Alsammarae. Program for life extension and preserving existing resources for motor control center components [nuclear plants]. *Nuclear Science, IEEE Transactions on*, Vol. 42(4):1000–1004, Aug. 1995.

[78] C.K. Scott, R. Jeppesen, A.R. McKenzie, M.A. Petrilli, and P.D. Thompson. Integrated safety review for the point lepreau refurbishment for life extension. page 8p, June 2002.

[79] S.M. Shah and M. Irfan. Embedded hardware/software verification and validation using hardware-in-the-loop simulation. *Emerging Technologies, 2005. Proceedings of the IEEE Symposium on*, pages 494–498, Sep. 2005.

[80] V. G. Snell. Safety of candu nuclear power stations. (6329):17p, Nov. 1978.

[81] Ki Chang Son, Chong Son Chun, Se Do Sohn, Byung Chai Lee, Byeong Joo Lee, and B.R. Whittall. A study on software verification for shutdown system no.1 of wolsong 2, 3 and 4 in candu. *Proceedings of the 1996 American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Control and Human-Machine Interface Technologies, NPIC&HMIT*, (2):p1259–64, May 1996.

[82] CANDU 6 Program Team. Candu 6 technical summary. page 62p, June 2005.

[83] P. Thomas. Safety regulation and the not-so-level playing field. *Energy Trading and Risk Management, 2005. The IEE International Conference on (Ref. No. 2005/11245)*, pages 9–12, Nov. 2005.

[84] Invensys Triconex. Communication and installation guide for tricon systems. June 2005.

[85] Invensys Triconex. Field terminations guide for tricon systems. June 2005.

[86] Invensys Triconex. Tristation 1131 developer's guide for tricon systems. June 2005.

[87] L. Tsoukalas, R.C. Berkan, and A. Ikonomopoulos. Uncertainty modelling in anticipatory systems [nuclear reactor operator emulation]. *Uncertainty Modelling and Analysis, 1990. Proceedings., First International Symposium on*, pages 366–371, Dec. 1990.

[88] A. G. Wikjord. The disposal of canada's nuclear fuel waste. July 1996.

[89] X. Wu, H. Figueroa, and A. Monti. Testing of digital controllers using real-time hardware in the loop simulation. *Power Electronics Specialists Conference, 2004. PESC 04. 2004 IEEE 35th Annual*, Vol. 5:3622–3627, June 2004.

[90] G. Yan and J. V. R. L'Archeveque. On distributed control and instrumentation systems for future nuclear power plants. *Nuclear Science, IEEE Transactions on*, Vol. 23(1):431–435, Feb. 1976.

Appendix A
Darlington Nuclear Power Plant Design Specifications

Parameter	Value
General	
Owner, operator	Ontario Power Generation
Designers	Ontario Hydro
Number of units	Four
Rated output per unit	Generator output 935 MW(e)
Self-consumption	54 MW(e)
Net electrical	881 MW(e)
Overall net efficiency	31.70%
Fuel	Natural Uranium Dioxide (UO2)
Moderator	Deuterium Oxide (D2O-heavy water)
Coolant	Pressurized heavy water
Type	Horizontal pressure tube
Construction	Start of construction late 1977
In-service dates:	Unit 1-1990, Unit 3-1991, Unit 2-1990, Unit 4-1992
Reactor Core	
Pressure Tubes	
Quantity	480 tubes (24 x 24 array)
Core Radius	3 532 mm
Core Length	5 944 mm
Fuel load	6 240 bundles (108 Mg U)
Maximum channel power	6.4 MW
Fuel Elements	
Type	37 element bundles, 495 mm long
No. per channel	13 bundles
Total weight of bundle	23.5 kg
Maximum bundle power	787 kW (time averaged)

Table A.1: Darlington NPP general and reactor core design specifications.

Parameter	Value
Reactor Control	
Reactor control	164 MW.h per kg. U
Reactivity control Units	2 digital computers per unit
Control system - liquid zone control units	
Purpose	suppress unwanted changes in neutron flux distribution
Quantity	14 separate zones
Type	Light water compartments
Control system - control absorber rods	
Purpose	Inserted if reactivity rate of change of Liquid Zone Control system is inadequate
Quantity	4 rods
Control system - adjuster rods	
Purpose	Normally inserted to limit fuel power; can be withdrawn to provide increased eactivity
Quantity	24 rods
Type	Stainless steel
Reactor Shutdown	
Shutdown systems - shut-off rods	
Purpose	Safety devices to quickly terminate reactor operation
Quantity	32 rods
Type	Stainless steel - cadmium - stainless steel sandwich in the form of a tube
Orientation	Vertical through lattice
Reactivity worth	-49 mk within 2.0 sec.
Drive mechanism	Winch and cable driven via an electro magnetic friction clutch by a constant speed induction motor
Shutdown system - liquid injection	
Purpose	Safety system to terminate reactor operation
Type	Gadolinium nitrate solution
Reactivity worth	-55 mk
Injection method	Pressurized helium to drive solution into bulk moderator through nozzles

Table A.2: Darlington NPP reactivity control and shutdown system specifications, continued.

Appendix B
Coverage of Process Failures by Shutdown Systems no. 1 and 2

No.	Process Failure	Trip Parameter	Alternative Trip Parameter
	Loss of Regulation from High Power:		
1.	Fast	High Rate Neutron Power	High Neutron Power/High Heat Transport Pressure
	Slow	High Neutron Power	High Heat Transport Pressure/Manual[1]
	Loss of Regulation from Decay Power Levels:		
2.	Pressurized/Pumps On		
	Fast	High Rate Neutron Power	High Heat Transport Pressure
	Slow	High Heat Transport Pressure	High Neutron Power[1] Manual[1]
3.	Pressurized/Pumps Off		
	Fast	High Rate Neutron Power	Low Gross Coolant Flow [2] [6]
	Slow	Low Gross Coolant Flow [2] [6]	High Heat Transport Pressure
4.	Reduced Pressure/Pumps Off		
	Fast	High Rate Neutron Power [2] [6]	Low Heat Transport Pressure/Low Gross Coolant Flow[2] [6]
	Slow	Low Gross Coolant Flow [2] [6]	Low Heat Transport Pressure [2] [6]/Manual[1]
5.	Reduced Pressure/Pumps On		
	Fast	High Rate Neutron Power	Low Heat Transport Pressure[2] [6]
	Slow	Low Heat Transport Pressure(2) (6)	Manual
6.	Loss of Class IV Power	Low Gross Coolant Flow[2]	High Heat Transport Pressure

Table B.1: Coverage of process failures by shutdown system no. 1 and independently by shutdown system no. 2.

No.	Process Failure	Trip Parameter	Alternative Trip Parameter
	Loss of Coolant Into Containment:		
7.	Large	High Rate Neutron Power	High Neutron Power/High reactor building Pressure
8.	Intermediate	High Neutron Power	High reactor building Pressure
9.	Small		
	With Regulation	High reactor building Pressure	Low Heat Transport Pressure[2]/Low Pressurizer Level
	With Regulation - Pressurizer Isolated	High reactor building Pressure	Low Heat Transport Pressure
	Without Regulation	High reactor building Pressure	High Neutron Power
10.	Very Small		
	With Regulation	Low Heat Transport Pressure[2]	Low Pressurizer Level/ Manual
	With Regulation- Pressurizer Isolated	Low Heat Transport Pressure[2]	Manual
	Without Regulation	High Neutron Power	Manual[1]
11.	Loss-of-Coolant Into Calandria		
	With Regulation	Low Heat Transport Pressure[2] Moderator High Level	Low Pressurizer Level/ Manual[1]
	With Regulation - Pressurizer Isolated	Low Heat Transport Pressure(2) Moderator High Level	Manual[1]
	Without Regulation	High Neutron Power Moderator High Level	Manual[1]

Table B.2: Coverage of process failures by shutdown system no. 1 and independently by shutdown system no. 2, continued.

No.	Process Failure	Trip Parameter	Alternative Trip Parameter
	Loss of Coolant Into Containment:		
7.	Large	High Rate Neutron Power	High Neutron Power/High reactor building Pressure
8.	Intermediate	High Neutron Power	High reactor building Pressure
9.	Small		
	With Regulation	High reactor building Pressure	Low Heat Transport Pressure[2]/Low Pressurizer Level
	With Regulation - Pressurizer Isolated	High reactor building Pressure	Low Heat Transport Pressure
	Without Regulation	High reactor building Pressure	High Neutron Power
10.	Very Small		
	With Regulation	Low Heat Transport Pressure[2]	Low Pressurizer Level/ Manual
	With Regulation- Pressurizer Isolated	Low Heat Transport Pressure[2]	Manual
	Without Regulation	High Neutron Power	Manual[1]
11.	Loss-of-Coolant Into Calandria		
	With Regulation	Low Heat Transport Pressure[2] Moderator High Level	Low Pressurizer Level/ Manual[1]
	With Regulation - Pressurizer Isolated	Low Heat Transport Pressure(2) Moderator High Level	Manual[1]
	Without Regulation	High Neutron Power Moderator High Level	Manual[1]

Table B.3: Coverage of process failures by shutdown system no. 1 and independently by shutdown system no. 2, continued.

No.	Process Failure	Trip Parameter	Alternative Trip Parameter
15.	Loss of Feedwater Control (e.g., Closure of Feedwater Valves to a steam generator)	Low steam generator Level	High Heat Transport Pressure[3] [1] /Manual[3]
16.	Feedwater Pumps Trip	Low Steam generator Level	High Heat Transport Pressure [3] [1] / Low steam generator Feedline Pressure [1] [4]
17.	Loss of Moderator Cooling	High Moderator Level	Manual
18.	Moderator Pipe Break	Low Moderator Level	Manual/High Neutron Power

Notes:

(1) Alternative trip parameters which provide trip coverage over a limited range of event scale (e.g., break size).

(2) The low coolant flow and low heat transport system pressure trips are conditioned out on log power < 0.3 percent.

(3) If 4 percent feedwater flow is available after trip.

(4) The low steam generator feedline pressure trip is conditioned out when log power < 10 percent.

(5) Feedline pressure may precede steam generator low level.

(6) Trip acts as high power trip in effect since power is increasing - this instigates a trip by removing the conditioned-out state.

Table B.4: Coverage of process failures by shutdown system no. 1 and independently by shutdown system no. 2, continued.

Appendix C
C Program Modules for Simulation and Development Analysis

Timing Analysis C Module (1000)

```
1000 #include <sys/types.h>
1001 #include <sys/socket.h>
1002 #include <netinet/in.h>
1003 #include <arpa/inet.h>
1004 #include <netdb.h>
1005 #include <unistd.h>
1006 #include <signal.h>
1007 #include <stdio.h>
1008 #include <fcntl.h>
1009 #include <errno.h>
1010 #include <sys/time.h>
1011 #include <stdlib.h>
1012 #include <memory.h>
1013 #include <math.h>
1014 #include <sys/poll.h>
1015 #include <stdio.h>
1016 #include <stdlib.h>
1017 #include "/usr/mainlc/sim/types.h"
1018 #include "/usr/release/com20050705/include/cdbmapping.h"
1019
1020 /* Define data collection parameters */
1021 #define MAXDATA   100000      /* Number of data samples to
        capture */
```

```
1022 /* Define the sine wave parameters */
1023 #define Fs  1000.0      /* sampling frequency in hertz */
1024 #define Fo  1.0       /* sampling wave frequency in hertz */
1025 #define NUM 1000      /* generate number of sample */
1026 #define sineWave_OFF 3  /* sine wave offset translation */
1027 #define sineWave_AMP 3   /* sine wave amplitude */
1028 /* Sine wave "on/off" control macros */
1029 #define sineOn(sampleNum, sineAmplitude) { \
1030     if (n == sampleNum) xImpulse = sineAmplitude;\
1031     if (n == sampleNum + 1) xImpulse = 0.0;\
1032 }
1033 #define sineOff(sampleNum) if (n == sampleNum) U[0] = U[1] =
        0;
1034
1035 /* Define socket address structures and variables */
1036     struct sockaddr_in sinai;
1037     struct sockaddr_in sinbo;
1038     int si;
1039     struct sockaddr_in sinao;
1040     struct sockaddr_in sinbi;
1041     int so;
1042 /* Define data collection variables */
1043     long int time_array [MAXDATA];
1044     int idx_time_array = 0;
1045     float tx_array [MAXDATA];
1046     float rx_array [MAXDATA];
1047 /* Define sineWave generator variables */
1048     int sineSpot = 0;
1049     double    sineWave[NUM];
1050     int    delay_HIL;
1051     int    delay_SUT;
1052 /* Open and initialize UDP connection between Alpha server
        and NI workstation */
1053 void dla1udpo (void) {
```

```
1054    /* UDP/IP variables */
1055    int error_in;
1056    int error_out;
1057    /* SineWave variables */
1058    double   U[3],
1059    xImpulse = 0.0;
1060    int      n;
1061    const double C1 = 2.0, C2 = 1.0, D2 = 1.0;
1062    const double pi = 4.0 * atan(1.0);
1063    double D1, K, B, M, N;
1064    idx_time_array = 0;
1065    /* Alpha server connection configuration (input) */
1066    sinai.sin_family = AF_INET;
1067    sinai.sin_port = htons(9999);
1068    sinai.sin_addr.s_addr = inet_addr("192.168.1.1");
1069    /*sinai.sin_addr.s_addr = inet_addr("###.###.###.###");*/
1070    /* NI workstation connection configuration (output) */
1071    sinbo.sin_family = AF_INET;
1072    sinbo.sin_port = htons(10001);
1073    sinbo.sin_addr.s_addr = inet_addr("192.168.1.2");
1074    /*sinbo.sin_addr.s_addr = inet_addr("###.###.###.###");*/
1075    si = socket(AF_INET, SOCK_DGRAM, 0);
1076    /* Alpha server connection configuration (output) */
1077    sinao.sin_family = AF_INET;
1078    sinao.sin_port = htons(9998);
1079    sinao.sin_addr.s_addr = inet_addr("192.168.1.1");
1080    /*sinao.sin_addr.s_addr = inet_addr("###.###.###.###");*/
1081    /* NI workstation connection configuration (input) */
1082    sinbi.sin_family = AF_INET;
1083    sinbi.sin_port = htons(10000);
1084    sinbi.sin_addr.s_addr = inet_addr("192.168.1.2");
1085    /*sinbi.sin_addr.s_addr = inet_addr("###.###.###.###");*/
1086    so = socket(AF_INET, SOCK_DGRAM, 0);
1087
```

```
1088    /* Detect input error */
1089    error_in = bind (si, (struct sockaddr *)&sinai, sizeof(
           sinai));
1090    if (error_in < 0) {
1091       close(si);
1092       printf("UDP In Bind Error code = %d\n", error_in);
1093       exit(2);
1094    }
1095
1096    /* Detect output error */
1097    error_out = bind (so, (struct sockaddr *)&sinao, sizeof(
           sinao));
1098    if ( error_out < 0)  {
1099       puts("UDP Out Bind Error");
1100       printf("Error code = %d", error_out);
1101       close(so);
1102       exit(2);
1103    }
1104
1105    /* SineWave Generator */
1106    /* Calculate variable parameters */
1107    B = (Fs/(pi*Fo))*tan(pi*Fo/Fs);
1108    M = 2.0*pi*Fo*B;
1109    N = 2.0*Fs;
1110
1111    /* Synthesis coefficients */
1112    D1 = 2.0*(M*M-N*N)/(M*M+N*N);
1113    K = M*B*Fs/(M*M+N*N);
1114
1115    /* Generate NUM samples */
1116    for (n=0;n<NUM;++n)  {
1117       sineOn(1,sineWave_AMP);
1118       sineOff(NUM);
1119       /* Sinewave synthesis */
```

```
1120        /* Biquad IIR filter */
1121        U[2] = xImpulse-D1*U[0]-D2*U[1];
1122        sineWave[n] = K*(U[2]+C1*U[0]+C2*U[1]) + sineWave_OFF;
1123        /* Update registers */
1124        U[1] = U[0];
1125        U[0] = U[2];
1126    }
1127 }
1128
1129 /* Output variable to UPD/IP socket */
1130 void SDS1_HIL (void) {
1131 /* Define process variable pointers */
1132 #define NUM_PROCVAR 1
1133 #define NUM_RECVAR 1
1134
1135    int       data_mult = 600;
1136    long int        time_diff=0;
1137    struct timespec    time_send;
1138    struct timespec    time_receive;
1139    int stat, i, j, rcv_data;
1140    float delay_loop;
1141    float PROCVAR[NUM_PROCVAR];
1142    float RECVAR[NUM_RECVAR];
1143    time_t rawtime;
1144    /* Define receive polling variables */
1145    int rv;
1146    struct pollfd ufds[2];
1147    /* Define transmit message pointer */
1148    char msg_out [18];
1149    /* Define receive message pointer */
1150    char *msg_in;
1151    char *msg_in_id;
1152    unsigned int sin_length;
1153    /* Configure input timing requirements */
```

```
1154    struct timeval timeout;
1155    fd_set read_handles;
1156    int number_data = 0;
1157    struct tm * timeinfo;
1158    time ( &rawtime );
1159    timeinfo = localtime ( &rawtime );
1160
1161    if (idx_time_array%(5*data_mult)==0){
1162       if (idx_time_array < 5*data_mult) {
1163          delay_HIL = 1; delay_SUT = 0;
1164       }
1165       else if (idx_time_array >= 5*data_mult &&
             idx_time_array < 10*data_mult) {
1166          delay_HIL = 1; delay_SUT = 10000;
1167       }
1168       else if (idx_time_array >= 10*data_mult &&
             idx_time_array < 15*data_mult) {
1169          delay_HIL = 1; delay_SUT = 20000;
1170       }
1171       else if (idx_time_array >= 15*data_mult &&
             idx_time_array < 20*data_mult) {
1172          delay_HIL = 2; delay_SUT = 0;
1173       }
1174       else if (idx_time_array >= 20*data_mult &&
             idx_time_array < 25*data_mult) {
1175          delay_HIL = 2; delay_SUT = 10000;
1176       }
1177       else if (idx_time_array >= 25*data_mult &&
             idx_time_array < 30*data_mult) {
1178          delay_HIL = 2; delay_SUT = 20000;
1179       }
1180       else if (idx_time_array >= 30*data_mult &&
             idx_time_array < 35*data_mult) {
1181          delay_HIL = 3; delay_SUT = 0;
```

```
1182        }
1183        else if (idx_time_array >= 35*data_mult &&
               idx_time_array < 40*data_mult) {
1184           delay_HIL = 3; delay_SUT = 10000;
1185        }
1186        else if (idx_time_array >= 40*data_mult &&
               idx_time_array < 45*data_mult) {
1187           delay_HIL = 3; delay_SUT = 20000;
1188        }
1189        else if (idx_time_array >= 45*data_mult &&
               idx_time_array < 50*data_mult) {
1190           delay_HIL = 4; delay_SUT = 0;
1191        }
1192        else if (idx_time_array >= 50*data_mult &&
               idx_time_array < 55*data_mult) {
1193           delay_HIL = 4; delay_SUT = 10000;
1194        }
1195        else if (idx_time_array >= 55*data_mult &&
               idx_time_array < 60*data_mult) {
1196           delay_HIL = 4; delay_SUT = 20000;
1197        }
1198        else if (idx_time_array >= 60*data_mult &&
               idx_time_array < 65*data_mult) {
1199           delay_HIL = 5; delay_SUT = 0;
1200        }
1201        else if (idx_time_array >= 65*data_mult &&
               idx_time_array < 70*data_mult) {
1202           delay_HIL = 5; delay_SUT = 10000;
1203        }
1204        else if (idx_time_array >= 70*data_mult &&
               idx_time_array < 75*data_mult) {
1205           delay_HIL = 5; delay_SUT = 20000;
1206        }
```

```
1207        else if (idx_time_array >= 75*data_mult &&
                idx_time_array < 80*data_mult) {
1208            delay_HIL = 10; delay_SUT = 0;
1209        }
1210        else if (idx_time_array >= 80*data_mult &&
                idx_time_array < 85*data_mult) {
1211            delay_HIL = 10; delay_SUT = 10000;
1212        }
1213        else if (idx_time_array >= 85*data_mult &&
                idx_time_array < 90*data_mult) {
1214            delay_HIL = 10; delay_SUT = 20000;
1215        }
1216        else {
1217            delay_HIL = 0; delay_SUT = 0;
1218        }
1219        printf("H%ld , S%ld\n" ,delay_HIL , delay_SUT);
1220    }
1221
1222    PROCVAR[0] = sineWave[sineSpot];
1223    sineSpot++;
1224    if (sineSpot >= NUM) sineSpot =0;
1225
1226    /* Start HIL interface timer */
1227    stat = clock_gettime(CLOCK_REALTIME, &time_send);
1228
1229    /* Transmit process variables */
1230    for (i = 0; i < NUM_PROCVAR; i++)   {
1231        sprintf(msg_out, "%02d0375%012.6f" ,i ,PROCVAR[i]);
1232        tx_array[idx_time_array] = PROCVAR[i];
1233        sendto(so, msg_out, strlen(msg_out)+1, 0, (struct
                sockaddr *)&sinbi, sizeof(sinbi));
1234    }
1235
1236    /* Delay for HIL interface and SUT */
```

```
1237    usleep(delay_SUT);
1238    FD_ZERO(&read_handles);
1239    FD_SET(si, &read_handles);
1240    timeout.tv_sec =  0;
1241    timeout.tv_usec = 1;
1242
1243    /* Set the value of input-independant variables */
1244    msg_in = "12123412345678";
1245    msg_in_id = "12";
1246    rcv_data=0;
1247
1248    /* Flush incoming UDP/IP packets before return request */
1249    while ( select(si+1, &read_handles, NULL, NULL, &timeout)
          > 0) {
1250       recvfrom(si, msg_in, 10, 0, (struct sockaddr *)&sinbo,
             &sin_length);
1251    }
1252
1253    for (i = 50; i < 50+NUM_RECVAR; i++)   {
1254       /* Request process variables return */
1255       sprintf(msg_out,"%02d0375000000000000",i);
1256       sendto(so, msg_out, strlen(msg_out)+1, 0, (struct
             sockaddr *)&sinbi, sizeof(sinbi));
1257
1258       ufds[0].fd = si;
1259       ufds[0].events = POLLIN;
1260       ufds[1].fd = so;
1261       ufds[1].events = POLLIN;
1262       rv = poll(ufds, 2, delay_HIL);
1263
1264       if (rv == -1) { perror("poll")    }
1265       else if (rv == 0) {
1266          printf("Timeout occurred!\n");
1267       }
```

```
1268        else {
1269            /*printf("Data.\n");*/
1270            /* Receive returned variables */
1271            recvfrom(si, msg_in, 10, 0, (struct sockaddr *)&
                    sinbo, &sin_length);
1272            /*printf("%02d %s on %s\n",i,msg_in,asctime(timeinfo
                    ));*/
1273            number_data++;
1274        }
1275        if (number_data == 0)    {
1276            printf ( "Warning: No Data (%d) on %s",i, asctime (
                    timeinfo) );
1277            rx_array[idx_time_array] = 1001;
1278        }
1279        else {
1280            /* Remove identifier */
1281            /* printf("Entire:%s\n", msg_in);*/
1282            strncpy(msg_in_id,msg_in,2);
1283            strncpy(msg_in_id + 2,"\0",2);
1284            strncpy(msg_in,msg_in + 2,8);
1285            strncpy(msg_in + 8,"\0",2);
1286            /*printf("Variable:%s, Value:%s\n", msg_in_id,
                    msg_in);*/
1287
1288            /* Verify identifier */
1289            if (i == atoi(msg_in_id))  {
1290                /* Verify identifier */
1291                RECVAR[i-50]=atof(msg_in);
1292                rx_array[idx_time_array] = RECVAR[i-50];
1293            }
1294            else  {
1295                printf ( "Warning: Invalid identifier (%d) on %s"
                        ,i, asctime (timeinfo) );
1296                rx_array[idx_time_array] = 1002;
```

```
1297            }
1298        }
1299        stat = clock_gettime(CLOCK_REALTIME, &time_receive);
1300        if (time_receive.tv_nsec < time_send.tv_nsec) {
1301            time_diff = 1000000000 - (time_send.tv_nsec -
                    time_receive.tv_nsec);
1302            /*printf("%ld, %ld",time_send.tv_nsec,time_receive.
                    tv_nsec);*/
1303        }
1304        else {
1305            time_diff = time_receive.tv_nsec - time_send.tv_nsec
                    ;
1306        }
1307        time_array[idx_time_array] = time_diff;
1308        idx_time_array++;
1309        /*printf("Time Difference: %ld\n", time_diff);*/
1310    }
1311 }
1312 /* Close UDP connection */
1313 void dla1udpc (void) {
1314    int i;
1315    char filename;
1316    FILE * hFile;
1317    hFile = fopen("HIL_timing.out", "w");
1318    for (i = 0; i < idx_time_array; i++)   {
1319        fprintf(hFile,"%ld, %f, %f\n",time_array[i],tx_array[i
                ],rx_array[i]);
1320    }
1321    fclose(hFile);
1322    close(so);
1323    close(si);
1324 }
```

Scalability Analysis C Module (2000)

```
2000 #include <sys/types.h>
2001 #include <sys/socket.h>
2002 #include <netinet/in.h>
2003 #include <arpa/inet.h>
2004 #include <netdb.h>
2005 #include <unistd.h>
2006 #include <signal.h>
2007 #include <stdio.h>
2008 #include <fcntl.h>
2009 #include <errno.h>
2010 #include <sys/time.h>
2011 #include <stdlib.h>
2012 #include <memory.h>
2013 #include <math.h>
2014 #include <sys/poll.h>
2015 #include <stdio.h>
2016 #include <stdlib.h>
2017 #include "/usr/mainlc/sim/types.h"
2018 #include "/usr/release/com20050705/include/cdbmapping.h"
2019
2020 /* Define data collection parameters */
2021 #define MAXDATA 100000          /* Number of data samples to
        capture */
2022 /* Define the sine wave parameters */
2023 #define Fs      1000.0          /* sampling frequency in
        hertz */
2024 #define Fo      1.0             /* sampling wave frequency in
         hertz */
2025 #define NUM     1000            /* generate number of sample
        */
2026 #define sineWave_OFF    3       /* sine wave offset
        translation */
```

```
2027 #define sineWave_AMP     3    /* sine wave amplitude */
2028
2029 /* Sine wave "on/off" control macros */
2030 #define sineOn(sampleNum, sineAmplitude) {\
2031         if (n == sampleNum) xImpulse = sineAmplitude;\
2032         if (n == sampleNum + 1) xImpulse = 0.0;\
2033 }
2034 #define sineOff(sampleNum) if (n == sampleNum) U[0] = U[1] =
        0;
2035
2036 /* Define socket address structures and variables */
2037         struct sockaddr_in sinai;
2038         struct sockaddr_in sinbo;
2039         int si;
2040         struct sockaddr_in sinao;
2041         struct sockaddr_in sinbi;
2042         int so;
2043 /* Define data collection variables */
2044         long int time_array [MAXDATA];
2045         int idx_time_array;
2046         float tx_array [MAXDATA];
2047         float rx_array [MAXDATA];
2048         long int          nodata_array [MAXDATA];
2049         long int          invalid_array [MAXDATA];
2050 /* Define sineWave generator variables */
2051         int sineSpot = 0;
2052         double  sineWave[NUM];
2053         int       NUM_PROC_VAR;
2054         int       NUM_REC_VAR;
2055         int                 delay_SUT;
2056                 int write_i;
2057         char filename;
2058         FILE * hFile;
2059
```

```
2060 /* Open and initialize UDP connection between Alpha server
        and NI workstation */
2061 void dla1udpo (void)    {
2062         /* UDP/IP variables */
2063         int error_in;
2064         int error_out;
2065         /* SineWave variables */
2066         double  U[3],
2067         xImpulse = 0.0;
2068         int              n;
2069         const    double C1 = 2.0, C2 = 1.0, D2 = 1.0;
2070         const    double pi = 4.0 * atan(1.0);
2071         double D1, K, B, M, N;
2072         idx_time_array = 0;
2073         /* Alpha server connection configuration (input) */
2074         sinai.sin_family = AF_INET;
2075         sinai.sin_port = htons(9999);
2076         sinai.sin_addr.s_addr = inet_addr("192.168.1.1");
2077         /*sinai.sin_addr.s_addr = inet_addr
              ("###.###.###.###");*/
2078         /* NI workstation connection configuration (output)
              */
2079         sinbo.sin_family = AF_INET;
2080         sinbo.sin_port = htons(10001);
2081         sinbo.sin_addr.s_addr = inet_addr("192.168.1.2");
2082         /*sinbo.sin_addr.s_addr = inet_addr
              ("###.###.###.###");*/
2083         si = socket(AF_INET, SOCK_DGRAM, 0);
2084         /* Alpha server connection configuration (output) */
2085         sinao.sin_family = AF_INET;
2086         sinao.sin_port = htons(9998);
2087         sinao.sin_addr.s_addr = inet_addr("192.168.1.1");
2088         /*sinao.sin_addr.s_addr = inet_addr
              ("###.###.###.###");*/
```

```
2089        /* NI workstation connection configuration (input) */
2090        sinbi.sin_family = AF_INET;
2091        sinbi.sin_port = htons(10000);
2092        sinbi.sin_addr.s_addr = inet_addr("192.168.1.2");
2093        /*sinbi.sin_addr.s_addr = inet_addr
              ("###.###.###.###");*/
2094        so = socket(AF_INET, SOCK_DGRAM, 0);
2095
2096        /* Detect input error */
2097        error_in = bind (si, (struct sockaddr *)&sinai,
              sizeof(sinai));
2098        if (error_in < 0)        {
2099              close(si);
2100              printf("UDP In Bind Error code = %d\n",
                    error_in);
2101              exit(2);
2102        }
2103
2104        /* Detect output error */
2105        error_out = bind (so, (struct sockaddr *)&sinao,
              sizeof(sinao));
2106        if ( error_out < 0)      {
2107              puts("UDP Out Bind Error");
2108              printf("Error code = %d", error_out);
2109              close(so);
2110              exit(2);
2111        }
2112
2113        /* SineWave Generator */
2114        /* Calculate variable parameters */
2115        B = (Fs/(pi*Fo))*tan(pi*Fo/Fs);
2116        M = 2.0*pi*Fo*B;
2117        N = 2.0*Fs;
2118        /* Synthesis coefficients */
```

```
2119          D1 = 2.0*(M*M-N*N)/(M*M+N*N);
2120          K = M*B*Fs/(M*M+N*N);
2121
2122          /* Generate NUM samples */
2123          for (n=0;n<NUM;++n)       {
2124                    sineOn(1,sineWave_AMP);
2125                    sineOff(NUM);=
2126                    /* Sinewave synthesis */
2127                    /* Biquad IIR filter */
2128                    U[2] = xImpulse-D1*U[0]-D2*U[1];
2129                    sineWave[n] = K*(U[2]+C1*U[0]+C2*U[1]) +
                           sineWave_OFF;
2130                    /* Update registers */
2131                    U[1] = U[0];
2132                    U[0] = U[2];
2133          }
2134          hFile = fopen("HIL_scaling.out", "w");
2135 }
2136
2137 /* Output variable to UPD/IP socket */
2138 void SDS1_HIL (void)     {
2139 #define delay_HIL 10
2140          /* Define process variable pointers */
2141          int               data_mult = 3;
2142          long int                    time_diff=0;
2143          struct timespec             time_send;
2144          struct timespec             time_receive;
2145          int stat, i, j, rcv_data;
2146          float delay_loop;
2147          float PROCVAR[50];
2148          float RECVAR[50];
2149          /* Define receive polling variables */
2150          int rv;
2151          struct pollfd ufds[2];
```

```
2152        time_t rawtime;
2153        /* Define transmit message pointer */
2154        char msg_out [18];
2155        /* Define receive message pointer */
2156        char *msg_in;
2157        char *msg_in_id;
2158        unsigned int sin_length;
2159        /* Configure input timing requirements */
2160        struct timeval timeout;
2161        fd_set read_handles;
2162        int number_data = 0;
2163        struct tm * timeinfo;
2164        time ( &rawtime );
2165        timeinfo = localtime ( &rawtime );
2166
2167        if (idx_time_array%(5*data_mult)==0)     {
2168                if (idx_time_array < 5*data_mult) {
2169                        NUM_PROC_VAR = 1; NUM_REC_VAR = 1;
                                delay_SUT = 10000;
2170                }
2171                else if (idx_time_array >= 5*data_mult &&
                           idx_time_array < 10*data_mult) {
                           NUM_PROC_VAR = 1; NUM_REC_VAR = 1; delay_SUT
                            = 25000;
2172                }
2173                else if (idx_time_array >= 10*data_mult &&
                           idx_time_array < 15*data_mult) {
2174                        NUM_PROC_VAR = 1; NUM_REC_VAR = 1;
                                delay_SUT = 50000;
2175                }
2176                else if (idx_time_array >= 15*data_mult &&
                           idx_time_array < 20*data_mult) {
2177                        NUM_PROC_VAR = 1; NUM_REC_VAR = 1;
                                delay_SUT = 100000;
```

```
2178            }
2179            else if (idx_time_array >= 20*data_mult &&
                   idx_time_array < 25*data_mult) {
2180                 NUM_PROC_VAR = 1; NUM_REC_VAR = 1;
                        delay_SUT = 150000;
2181            }
2182            else if (idx_time_array >= 25*data_mult &&
                   idx_time_array < 30*data_mult) {
2183                 NUM_PROC_VAR = 3; NUM_REC_VAR = 3;
                        delay_SUT = 10000;
2184            }
2185            else if (idx_time_array >= 30*data_mult &&
                   idx_time_array < 35*data_mult) {
2186                 NUM_PROC_VAR = 3; NUM_REC_VAR = 3;
                        delay_SUT = 25000;
2187            }
2188            else if (idx_time_array >= 35*data_mult &&
                   idx_time_array < 40*data_mult) {
2189                 NUM_PROC_VAR = 3; NUM_REC_VAR = 3;
                        delay_SUT = 50000;
2190            }
2191            else if (idx_time_array >= 40*data_mult &&
                   idx_time_array < 45*data_mult) {
2192                 NUM_PROC_VAR = 3; NUM_REC_VAR = 3;
                        delay_SUT = 100000;
2193            }
2194            else if (idx_time_array >= 45*data_mult &&
                   idx_time_array < 50*data_mult) {
2195                 NUM_PROC_VAR = 3; NUM_REC_VAR = 3;
                        delay_SUT = 150000;
2196            }
2197            else if (idx_time_array >= 50*data_mult &&
                   idx_time_array < 55*data_mult) {
```

```
2198                    NUM_PROC_VAR = 5; NUM_REC_VAR = 5;
                            delay_SUT = 10000;
2199            }
2200            else if (idx_time_array >= 55*data_mult &&
                   idx_time_array < 60*data_mult) {
2201                    NUM_PROC_VAR = 5; NUM_REC_VAR = 5;
                            delay_SUT = 25000;
2202            }
2203            else if (idx_time_array >= 60*data_mult &&
                   idx_time_array < 65*data_mult) {
2204                    NUM_PROC_VAR = 5; NUM_REC_VAR = 5;
                            delay_SUT = 50000;
2205            }
2206            else if (idx_time_array >= 65*data_mult &&
                   idx_time_array < 70*data_mult) {
2207                    NUM_PROC_VAR = 5; NUM_REC_VAR = 5;
                            delay_SUT = 100000;
2208            }
2209            else if (idx_time_array >= 70*data_mult &&
                   idx_time_array < 75*data_mult) {
2210                    NUM_PROC_VAR = 5; NUM_REC_VAR = 5;
                            delay_SUT = 150000;
2211            }
2212            else if (idx_time_array >= 75*data_mult &&
                   idx_time_array < 80*data_mult) {
2213                    NUM_PROC_VAR = 10; NUM_REC_VAR = 10;
                            delay_SUT = 10000;
2214            }
2215            else if (idx_time_array >= 80*data_mult &&
                   idx_time_array < 85*data_mult) {
2216                    NUM_PROC_VAR = 10; NUM_REC_VAR = 10;
                            delay_SUT = 25000;
2217            }
```

```
2218                else if (idx_time_array >= 85*data_mult &&
                       idx_time_array < 90*data_mult) {
2219                     NUM_PROC_VAR = 10; NUM_REC_VAR = 10;
                            delay_SUT = 50000;
2220                }
2221                else if (idx_time_array >= 90*data_mult &&
                       idx_time_array < 95*data_mult) {
2222                     NUM_PROC_VAR = 10; NUM_REC_VAR = 10;
                            delay_SUT = 100000;
2223                }
2224                else if (idx_time_array >= 95*data_mult &&
                       idx_time_array < 100*data_mult) {
2225                     NUM_PROC_VAR = 10; NUM_REC_VAR = 10;
                            delay_SUT = 150000;
2226                }
2227                else if (idx_time_array >= 100*data_mult &&
                       idx_time_array < 105*data_mult) {
2228                     NUM_PROC_VAR = 25; NUM_REC_VAR = 25;
                            delay_SUT = 10000;
2229                }
2230                else if (idx_time_array >= 105*data_mult &&
                       idx_time_array < 110*data_mult) {
2231                     NUM_PROC_VAR = 25; NUM_REC_VAR = 25;
                            delay_SUT = 25000;
2232                }
2233                else if (idx_time_array >= 110*data_mult &&
                       idx_time_array < 115*data_mult) {
2234                     NUM_PROC_VAR = 25; NUM_REC_VAR = 25;
                            delay_SUT = 50000;
2235                }
2236                else if (idx_time_array >= 115*data_mult &&
                       idx_time_array < 120*data_mult) {
2237                     NUM_PROC_VAR = 25; NUM_REC_VAR = 25;
                            delay_SUT = 100000;
```

```
2238                }
2239                else if (idx_time_array >= 120*data_mult &&
                       idx_time_array < 125*data_mult) {
2240                     NUM_PROC_VAR = 25; NUM_REC_VAR = 25;
                            delay_SUT = 150000;
2241                }
2242                else if (idx_time_array >= 125*data_mult &&
                       idx_time_array < 130*data_mult) {
2243                     NUM_PROC_VAR = 50; NUM_REC_VAR = 50;
                            delay_SUT = 10000;
2244                }
2245                else if (idx_time_array >= 130*data_mult &&
                       idx_time_array < 135*data_mult) {
2246                     NUM_PROC_VAR = 50; NUM_REC_VAR = 50;
                            delay_SUT = 25000;
2247                }
2248                else if (idx_time_array >= 135*data_mult &&
                       idx_time_array < 140*data_mult) {
2249                     NUM_PROC_VAR = 50; NUM_REC_VAR = 50;
                            delay_SUT = 50000;
2250                }
2251                else if (idx_time_array >= 140*data_mult &&
                       idx_time_array < 145*data_mult) {
2252                     NUM_PROC_VAR = 50; NUM_REC_VAR = 50;
                            delay_SUT = 100000;
2253                }
2254                else if (idx_time_array >= 145*data_mult &&
                       idx_time_array < 150*data_mult) {
2255                     NUM_PROC_VAR = 50; NUM_REC_VAR = 50;
                            delay_SUT = 150000;
2256                }
2257                else {
2258                     NUM_PROC_VAR = 1; NUM_REC_VAR =1;
                            delay_SUT = 0;
```

```
2259                }
2260                printf("H%ld , S%ld\n" ,delay_HIL , delay_SUT);
2261        }
2262
2263        /* Start HIL interface timer */
2264        stat = clock_gettime(CLOCK_REALTIME, &time_send);
2265
2266        /* Transmit process variables */
2267        for (i = 0; i < NUM_PROCVAR; i++)        {
2268                PROCVAR[i] = sineWave[sineSpot];
2269                sineSpot++;
2270                /*printf("%f\n",PROCVAR[0]);*/
2271                if (sineSpot >= NUM) sineSpot =0;
2272                sprintf(msg_out, "%02d0375%012.6f",i ,PROCVAR[
                        i]);
2273                if (i==NUM_PROCVAR-1)    tx_array[
                        idx_time_array] = PROCVAR[i];
2274                sendto(so, msg_out, strlen(msg_out)+1, 0, (
                        struct sockaddr *)&sinbi, sizeof(sinbi));
2275        }
2276
2277        /* Delay for HIL interface and SUT */
2278        usleep(delay_SUT);
2279
2280        FD_ZERO(&read_handles);
2281        FD_SET(si, &read_handles);
2282        timeout.tv_sec =  0;
2283        timeout.tv_usec = 1;
2284
2285        /* Set the value of input-independant variables */
2286        msg_in = "12123412345678";
2287        msg_in_id = "12";
2288        rcv_data=0;
2289
```

```
2290        /* Flush incoming UDP/IP packets before return
               request */
2291        while ( select(si+1, &read_handles, NULL, NULL, &
               timeout) > 0)  {
2292              recvfrom(si, msg_in, 10, 0, (struct sockaddr
                     *)&sinbo, &sin_length);
2293        }
2294
2295        for (i = 50; i < 50+NUM_RECVAR; i++)     {
2296              /* Request process variables return */
2297              sprintf(msg_out,"%02d0375000000000000",i);
2298              sendto(so, msg_out, strlen(msg_out)+1, 0, (
                     struct sockaddr *)&sinbi, sizeof(sinbi));
2299
2300              ufds[0].fd = si;
2301              ufds[0].events = POLLIN;
2302              ufds[1].fd = so;
2303              ufds[1].events = POLLIN;
2304              rv = poll(ufds, 2, delay_HIL);
2305
2306              if (rv == -1) {
2307                    perror("poll");
2308              }
2309              else if (rv == 0)        {
2310                    printf("Timeout occurred!\n");
2311              }
2312              else {
2313                    /*printf("Data.\n");*/
2314                    /* Receive returned variables */
2315                    recvfrom(si, msg_in, 10, 0, (struct
                           sockaddr *)&sinbo, &sin_length);
2316                    /*printf("%02d %s on %s\n",i,msg_in,
                           asctime(timeinfo));*/
2317                    number_data++;
```

```
2318                }
2319                if (number_data == 0)    {
2320                        printf ( "Warning: No Data (%d) on %s
                              ",i, asctime (timeinfo) );
2321                        if (i==50)        rx_array[
                              idx_time_array] = 1001;
2322                        nodata_array[idx_time_array]++;
2323                }
2324                else     {
2325                        /* Remove identifier */
2326                        /* printf("Entire:%s\n", msg_in);*/
2327                        strncpy(msg_in_id,msg_in,2);
2328                        strncpy(msg_in_id + 2,"\0",2);
2329                        strncpy(msg_in,msg_in + 2,8);
2330                        strncpy(msg_in + 8,"\0",2);
2331                        /*printf("Variable:%s, Value:%s\n",
                              msg_in_id, msg_in);*/
2332
2333                        /* Verify identifier */
2334                        if (i == atoi(msg_in_id))        {
2335                                /* Verify identifier */
2336                                RECVAR[i-50]=atof(msg_in);
2337                                if (i==50)        rx_array[
                                      idx_time_array] = RECVAR[i
                                      -50];
2338                        }
2339                        else     {
2340                                printf ( "Warning: Invalid
                                      identifier (%d) on %s",i,
                                      asctime (timeinfo) );
2341                                if (i==50)        rx_array[
                                      idx_time_array] = 1002;
2342                                invalid_array[idx_time_array
                                      ]++;
```

```
2343                        }
2344                    }
2345                    if (i==50+NUM_RECVAR-1) {
2346                        stat = clock_gettime(CLOCK_REALTIME,
                              &time_receive);
2347                        if (time_receive.tv_nsec < time_send.
                              tv_nsec)    {
2348                            time_diff = 1000000000 - (
                                  time_send.tv_nsec -
                                  time_receive.tv_nsec);
2349                            /*printf("%ld, %ld",time_send
                                  .tv_nsec,time_receive.
                                  tv_nsec);*/
2350                        }
2351                        else    {
2352                            time_diff = time_receive.
                                  tv_nsec - time_send.
                                  tv_nsec;
2353                        }
2354                        time_array[idx_time_array] =
                              time_diff;
2355                        idx_time_array++;
2356                    }
2357                    /*printf("Time Difference: %ld\n", time_diff)
                          ;*/
2358            }
2359 }
2360
2361 void datawri (void)     {
2362            for (write_i = 0; write_i < idx_time_array; write_i
                  ++) {
2363                fprintf(hFile,"%ld,%f,%f,%ld,%ld\n",
                      time_array[write_i], tx_array[write_i],
                      rx_array[write_i], nodata_array[write_i],
```

```
                        invalid_array[write_i]);
2364            }
2365    }
2366
2367    /* Close UDP connection */
2368    void dla1udpc (void)    {
2369            fclose(hFile);
2370            close(so);
2371            close(si);
2372    }
```

Controlled Variable Transceiver Module (3000)

```
3000    #include <sys/types.h>
3001    #include <sys/socket.h>
3002    #include <netinet/in.h>
3003    #include <arpa/inet.h>
3004    #include <netdb.h>
3005    #include <unistd.h>
3006    #include <signal.h>
3007    #include <stdio.h>
3008    #include <fcntl.h>
3009    #include <errno.h>
3010    #include <sys/time.h>
3011    #include <stdlib.h>
3012    #include <memory.h>
3013    #include <math.h>
3014    #include <sys/poll.h>
3015    #include <stdio.h>
3016    #include <stdlib.h>
3017    #include "/usr/mainlc/sim/types.h"
3018    #include "/usr/release/com20050705/include/cdbmapping.h"
3019
3020    /* Define data collection parameters */
```

```
3021 #define MAXDATA 100000          /* Number of data samples to
        capture */
3022 /* Define socket address structures and variables */
3023    struct sockaddr_in sinai;
3024    struct sockaddr_in sinbo;
3025    int si;
3026    struct sockaddr_in sinao;
3027    struct sockaddr_in sinbi;
3028    int so;
3029 /* Define data collection variables */
3030         double time_array [MAXDATA];
3031         int idx_time_array;
3032         long int        nodata_array [MAXDATA];
3033         long int        invalid_array [MAXDATA];
3034         float sgl1_array [MAXDATA];
3035         float sgl2_array [MAXDATA];
3036         float sgl3_array [MAXDATA];
3037         float sgl4_array [MAXDATA];
3038         float rp_array [MAXDATA];
3039         float lcv101_array [MAXDATA];
3040         float lcv102_array [MAXDATA];
3041         float lcv103_array [MAXDATA];
3042         float flow_array [MAXDATA];
3043         char SDS1_tripD [MAXDATA];
3044         char SDS1_tripE [MAXDATA];
3045         char SDS1_tripF [MAXDATA];
3046
3047 /* Open and initialize UDP connection between Alpha server
        and NI workstation */
3048 void dla1udpo (void) {
3049         /* UDP/IP variables */
3050         int error_in;
3051         int error_out;
3052         idx_time_array = 0;
```

```
3053        /* Alpha server connection configuration (input) */
3054        sinai.sin_family = AF_INET;
3055        sinai.sin_port = htons(9999);
3056        sinai.sin_addr.s_addr = inet_addr("192.168.1.1");
3057        /*sinai.sin_addr.s_addr = inet_addr
              ("###.###.###.###");*/
3058        /* NI workstation connection configuration (output)
              */
3059        sinbo.sin_family = AF_INET;
3060        sinbo.sin_port = htons(10001);
3061        sinbo.sin_addr.s_addr = inet_addr("192.168.1.2");
3062        /*sinbo.sin_addr.s_addr = inet_addr
              ("###.###.###.###");*/
3063        si = socket(AF_INET, SOCK_DGRAM, 0);
3064        /* Alpha server connection configuration (output) */
3065        sinao.sin_family = AF_INET;
3066        sinao.sin_port = htons(9998);
3067        sinao.sin_addr.s_addr = inet_addr("192.168.1.1");
3068        /*sinao.sin_addr.s_addr = inet_addr
              ("###.###.###.###");*/
3069        /* NI workstation connection configuration (input) */
3070        sinbi.sin_family = AF_INET;
3071        sinbi.sin_port = htons(10000);
3072        sinbi.sin_addr.s_addr = inet_addr("192.168.1.2");
3073        /*sinbi.sin_addr.s_addr = inet_addr
              ("###.###.###.###");*/
3074        so = socket(AF_INET, SOCK_DGRAM, 0);
3075
3076        /* Detect input error */
3077        error_in = bind (si, (struct sockaddr *)&sinai,
              sizeof(sinai));
3078        if (error_in < 0)        {
3079              close(si);
```

```
3080                printf("UDP In Bind Error code = %d\n",
                       error_in);
3081                exit(2);
3082        }
3083
3084        /* Detect output error */
3085        error_out = bind (so, (struct sockaddr *)&sinao,
              sizeof(sinao));
3086        if ( error_out < 0)      {
3087                puts("UDP Out Bind Error");
3088                printf("Error code = %d", error_out);
3089                close(so);
3090                exit(2);
3091        }
3092 }
3093
3094 /* Output variable to UPD/IP socket */
3095 void SDS1_HIL (void)     {
3096 #define NUM_PROCVAR 4
3097 #define NUM_MONVAR 8
3098 #define NUM_RECVAR 1
3099 #define delay_HIL 10
3100 #define delay_SUT 115500
3101 #define delay_TX 1000
3102
3103        /* Define process variable pointers */
3104        long int                time_diff=0;
3105        struct timespec         time_send;
3106        struct timespec         time_receive;
3107        int stat, i, j, rcv_data;
3108        float delay_loop;
3109        float PROCVAR[NUM_PROCVAR];
3110        float RECVAR[NUM_RECVAR];
3111        /* Define controlled variable pointers SUT INPUTS*/
```

```
3112        float *PROCVAR_T1;
3113        float *PROCVAR_T2;
3114        float *PROCVAR_T3;
3115        float *PROCVAR_T4;
3116        /* Define monitored variable pointers */
3117        float *MONVAR_T1;
3118        float *MONVAR_T2;
3119        float *MONVAR_T3;
3120        float *MONVAR_T4;
3121        float *MONVAR_T5;
3122        char *MONVAR_T6;
3123        char *MONVAR_T7;
3124        char *MONVAR_T8;
3125        /* Define process variable pointers INPUT */
3126        char *TRIPOUTPUT_T;
3127        /* Define auxillary variable pointers WATCHDOG */
3128        char *WDWG;
3129        char RWDWG;
3130        /* Define receive polling variables */
3131        int rv;
3132        struct pollfd ufds[2];
3133        time_t rawtime;
3134        /* Define transmit message pointer */
3135        char msg_out [18];
3136        /* Define receive message pointer */
3137        char *msg_in;
3138        char *msg_in_id;
3139        unsigned int sin_length;
3140        /* Configure input timing requirements */
3141        struct timeval timeout;
3142        fd_set read_handles;
3143        int number_data = 0;
3144        struct tm * timeinfo;
3145        time ( &rawtime );
```

```
3146        timeinfo = localtime ( &rawtime );
3147
3148        /* Attach process variables to CDB location in memory
                */
3149        PROCVAR_T1 = CdbFloatPointer("L3DAI042", NULL);
3150        PROCVAR_T2 = CdbFloatPointer("L3DAI043", NULL);
3151        PROCVAR_T3 = CdbFloatPointer("L3DAI044", NULL);
3152        PROCVAR_T4 = CdbFloatPointer("L3DAI045", NULL);
3153        MONVAR_T1 = CdbFloatPointer("LJIVPTHP", NULL);
3154        MONVAR_T2 = CdbFloatPointer("fwacv101", NULL);
3155        MONVAR_T3 = CdbFloatPointer("fwacv102", NULL);
3156        MONVAR_T4 = CdbFloatPointer("fwacv103", NULL);
3157        MONVAR_T5 = CdbFloatPointer("FFWB", NULL);
3158        MONVAR_T6 = CdbBytePointer("L3DDO008", NULL);
3159        MONVAR_T7 = CdbBytePointer("L3EDO008", NULL);
3160        MONVAR_T8 = CdbBytePointer("L3FDO008", NULL);
3161        /* Create controlled variable array */
3162        PROCVAR[0] = *PROCVAR_T1;
3163        PROCVAR[1] = *PROCVAR_T2;
3164        PROCVAR[2] = *PROCVAR_T3;
3165        PROCVAR[3] = *PROCVAR_T4;
3166        /* Create monitored variable array */
3167        sgl1_array[idx_time_array] = *PROCVAR_T1;
3168        sgl2_array[idx_time_array] = *PROCVAR_T2;
3169        sgl3_array[idx_time_array] = *PROCVAR_T3;
3170        sgl4_array[idx_time_array] = *PROCVAR_T4;
3171        rp_array[idx_time_array] = *MONVAR_T1;
3172        lcv101_array[idx_time_array] = *MONVAR_T2;
3173        lcv102_array[idx_time_array] = *MONVAR_T3;
3174        lcv103_array[idx_time_array] = *MONVAR_T4;
3175        flow_array[idx_time_array] = *MONVAR_T5;
3176
3177        /* Transmit process variables */
3178        for (i = 0; i < NUM_PROCVAR; i++)        {
```

```
3179                    sprintf(msg_out, "%02d0375%012.6f",i,PROCVAR[
                            i]);
3180                    sendto(so, msg_out, strlen(msg_out)+1, 0, (
                            struct sockaddr *)&sinbi, sizeof(sinbi));
3181                    /* Delay for HIL interface and SUT */
3182                    if (i < NUM_PROCVAR-1) usleep(delay_TX);
3183            }
3184            /* Delay for HIL interface and SUT */
3185            usleep(delay_SUT);
3186            FD_ZERO(&read_handles);
3187            FD_SET(si, &read_handles);
3188            timeout.tv_sec =  0;
3189            timeout.tv_usec = 1;
3190            /* Set the value of input-independant variables */
3191            msg_in = "12123412345678";
3192            msg_in_id = "12";
3193            rcv_data=0;
3194            /* Attach process variables to CDB location in memory
                    */
3195    TRIPOUTPUT_T = CdbBytePointer("L3DDO016", NULL);
3196    WDWG = CdbBytePointer("L3DDO005", NULL);
3197    RWDWG = *WDWG;
3198    if (RWDWG == 1)*WDWG = 2;
3199    else *WDWG = 1;
3200            /* Flush incoming UDP/IP packets before return
                    request */
3201            while ( select(si+1, &read_handles, NULL, NULL, &
                    timeout) > 0)  {
3202                    recvfrom(si, msg_in, 10, 0, (struct sockaddr
                            *)&sinbo, &sin_length);
3203            }
3204            for (i = 50; i < 50+NUM_RECVAR; i++)    {
3205                    /* Request process variables return */
3206                    sprintf(msg_out,"%02d0375000000000000",i);
```

```
3207                sendto(so, msg_out, strlen(msg_out)+1, 0, (
                       struct sockaddr *)&sinbi, sizeof(sinbi));
3208                ufds[0].fd = si;
3209                ufds[0].events = POLLIN;
3210                ufds[1].fd = so;
3211                ufds[1].events = POLLIN;
3212                rv = poll(ufds, 2, delay_HIL);
3213                if (rv == -1) {
3214                        perror("poll");
3215                }
3216                else if (rv == 0) {
3217                        printf("Timeout occurred!\n");
3218                }
3219                else {
3220                        /*printf("Data.\n");*/
3221                        /* Receive returned variables */
3222                        recvfrom(si, msg_in, 10, 0, (struct
                           sockaddr *)&sinbo, &sin_length);
3223                        /*printf("%02d %s on %s\n",i,msg_in,
                           asctime(timeinfo));*/
3224                        number_data++;
3225                }
3226                if (number_data == 0)    {
3227                        printf ( "Warning: No Data (%d) on %s
                           ",i, asctime (timeinfo) );
3228                        nodata_array[idx_time_array]++;
3229                }
3230                else     {
3231                        /* Remove identifier */
3232                        /* printf("Entire:%s\n", msg_in);*/
3233                        strncpy(msg_in_id,msg_in,2);
3234                        strncpy(msg_in_id + 2,"\0",2);
3235                        strncpy(msg_in,msg_in + 2,8);
3236                        strncpy(msg_in + 8,"\0",2);
```

```
3237                            /*printf("Variable:%s, Value:%s\n",
                                    msg_in_id, msg_in);*/
3238
3239                            /* Verify identifier */
3240                            if (i == atoi(msg_in_id))        {
3241                                    /* Verify identifier */
3242                                    RECVAR[i-50]=atof(msg_in);
3243                                    *TRIPOUTPUT_T = RECVAR[0];
3244                            }
3245                            else    {
3246                                    printf ( "Warning: Invalid
                                           identifier (%d) on %s",i,
                                           asctime (timeinfo) );
3247                                    invalid_array[idx_time_array
                                           ]++;
3248                            }
3249                    }
3250                    if (i==50+NUM_RECVAR-1) {
3251                            idx_time_array++;
3252                    }
3253                    /*printf("Time Difference: %ld\n", time_diff)
                           ;*/
3254            }
3255            stat = clock_gettime(CLOCK_REALTIME, &time_send);
3256            sprintf(msg_out,"%ld.%09ld",time_send.tv_sec,
                   time_send.tv_nsec);
3257            time_array[idx_time_array-1] = atof(msg_out);
3258            SDS1_tripD[idx_time_array-1] = RECVAR[0];
3259            SDS1_tripE[idx_time_array-1] = *MONVAR_T7;
3260            SDS1_tripF[idx_time_array-1] = *MONVAR_T8;
3261    }
3262
3263    /* Close UDP connection */
3264    void dla1udpc (void)    {
```

```
3265        int i;
3266        char filename;
3267        FILE * hFile;
3268        hFile = fopen("HIL.out", "w");
3269        for (i = 0; i < idx_time_array; i++)    {
3270                fprintf(hFile,"%f, %f, %f, %f, %f, , , , , , ,
                    %f, %f, %f, %f, %f, %c, %c, %c, , %ld, %ld
                    \n", time_array[i], sgl1_array[i],
                    sgl2_array[i], sgl3_array[i], sgl4_array[i
                    ], rp_array[i], lcv101_array[i],
                    lcv102_array[i], lcv103_array[i],
                    flow_array[i], SDS1_tripD[i], SDS1_tripE[i
                    ], SDS1_tripF[i], nodata_array[i],
                    invalid_array[i]);
3271        }
3272        fclose(hFile);
3273        close(so);
3274        close(si);
3275  }
```

Appendix D
Invensys Tricon v9 Safety Programmable Logic Controller and Shutdown System no. 1

Rankin, Drew J. (June 3-6 2007) Tricon and Shutdown System One (SDS1). Proceedings of the 31st Canadian Nuclear Society Student Conference. Saint John, New Brunswick.

Overview

Tricon v9 fault tolerant control system is described by Invensys as their most trusted safety controller. It is also the first controller developed by Invensys to be completely triple redundant and industrially ruggedized. Tricon v9 is a state-of-the-art controller based on a triple modular redundant (TMR) architecture. TMR provides three isolated control systems that work in parallel. Additionally the controller boasts extensive diagnostics integrated into a single system and there is no single point of failure within Tricon which would cause the system to respond abnormally. High integrity process operation is provided through two out of three voting procedures for all decisional logic and signal verification.

Tricon provides simplified application development for a TMR system as it operates as a single control system from the user's point of view. The diagnostic capabilities, as within most advanced controllers is transparent to the programmer, who needs only access various related flags, status bytes and system variables.

Tricon controllers have been installed in over fifty countries in various applications. A summarized list of the most current applications includes; emergency shutdown systems (ESD), burner management systems (BMS), fire and gas systems (F&G), critical turbo-machinery control, railway switching, semiconductor life safety systems (SEMI S2) and nuclear 1E safety systems.

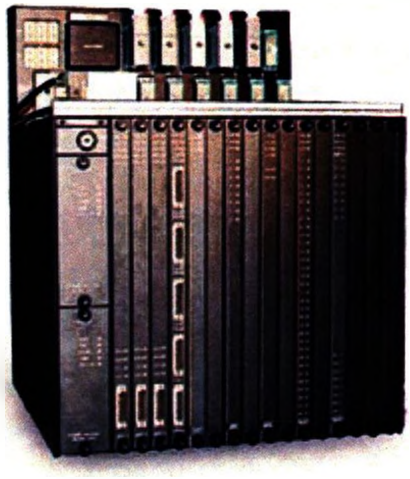

Figure D.1: Tricon v9 triple modular redundant (TMR) controller

Tricon v9 controller provides safety critical control capabilities at the highest levels of certification. Designing an advanced controller to meet the stringent standards of current Nuclear regulations requires rigorous testing and satisfying all requirements takes years of development. The following is intended to reveal the major functionalities Tricon v9 controller possesses that will enhance SDS1 control procedures. The review of nuclear instrumentation and controls standards or qualifying techniques is not included in this assessment.

General system characteristics

Overview of the general characteristics of Tricon controller is provided in Table D.1 as a precursor to extended functionality assessment.

Redundancy

Tricon v9 controller is based on extensive TMR architecture which ensures system functionality in the probable scenario that a single channel may become faulty within the systems expect useful lifetime. TMR architecture ensures fault tolerance and provides error-free, uninterrupted control in the presence of either hard failures of components or transient faults from internal or external sources [D1]. The redundancy

Feature	Invensys Tricon v9
Main processor	Motorola MPC860, 32-bit, 50 MHz
Memory	16 MB DRAM (without battery back-up) 32 KB SRAM (with battery back-up) 6 MB Flash PROM
Clock	Typical drift: 2 seconds/day Maximum drift: 8.6 seconds/day
System bus	25 megabits/second 32-bit CRC-protected 32-bit DMA, fully isolated
Communication processor Communication interface	Motorola MPC860, 32-bit, 50 MHz Protocol: RS-485 Baud rate: 2Mbit/sec
I/O interface	Protocol: RS-485 Baud rate: 375kbit/sec Logic power: 10W

Table D.1: General characteristic of Tricon v9 controller

of the entire system is important as relying on a common components can significantly decrease likelihood of proper safety system response during abnormal conditions.

The main processor of Tricon v9 is triplicated, each processor controlling in parallel one of three channels. Dedicated I/O processors with each of the three main processors manage the transfer of data to and form the I/O modules. This communication is performed via a triplicated I/O bus located on the backplane of the chassis. Extending this bus to additional chassis is achieved using I/O bus cables. These cables maintain triplication capabilities to additional chassis.

There are three independent channels within every I/O module (analog input, analog output, digital input, digital output, thermocouple, etc.) of Tricon v9. Each channel I/O processor processes channel specific data in parallel and concurrently send the information to the channel specific main processor. Through rigorous triplication of the communication channels and I/O modules the probability of complete system malfunction on single point of failure is unrealizable. Through proper maintenance, monitoring and testing procedures Tricon v9 provides excellent candidacy for safety critical control application.

In addition to the process specific redundancy, each Tricon v9 controller contains

two power modules arranged in dual-redundant configuration. The dual-redundant power is available throughout the backplane and additional chassis. Individual I/O module channels obtain power from independent power regulator connecting to each of the power channels. The power modules are designed to support the entire system in case of complete loss of the alternate power module. Tricon v9 controller also contains dual-redundant batteries to provide memory backup in case of a complete power failure of the controller. In the absence of field power, a sole battery can sustain the control program in the main processor RAM for a cumulative time period of six months.

Expandable modular system and online replacement

Tricon v9 system extends redundancy with modular capabilities. Varying combinations of up to 118 modules on 15 chassis may be utilized within a system. Providing the flexibility to incorporate even the most intricate safety critical systems. Additionally, there are two available slots for each I/O module. If a fault is detected on one module, control is switched to the functioning healthy module through bump-less transition. To facilitate maximum controller run-time, the modules can be replaced while the controller is online. This replacement allows for module repair and substitution routines and uninterrupted system functionality.

Alarms and status indicators

Status indicators (Pass, Fault and Active) in the form of light emitting diodes (LED) are provided on the front panel of every module. These common status indicators provide module specific status, however additional status indicators which relay both internal and external module information may be present. All indications are generated through internal diagnostic and alarm status data. This data is available through isolated network interfacing for remote logging and report generation. There are two general levels of indicator within Tricon v9; fault indicators which identify potentially serious module specific problems and alarm conditions which identify abnormal field conditions such as loss of power or loss of communication [D2].

The type of indicators included in Tricon v9 include:

- Status indicators which identify the processing state of the module. Each module includes a pass, fault, and active indicator.
- Field power and power load indicators which identify whether a power problem has occurred. (Only on some I/O modules.)
- Communication indicators which identify the type of communication occurring. (Only on main processor and communication modules.)
- Points indicators which identify whether the point is energized. (Not on analog input, analog output, or thermocouple modules.)

Programming and simulation environments

The programming and configuration environment for Tricon v9 controller is provided through Tristation 1131 Developers Workbench. The workbench provides designers with tools required for generating the configurations and applications for execution in Tricon controllers. Programming the controller on the workbench can be accomplished by function block diagrams, ladder logic diagrams or structured text as defined per the IEC 61131-3 standard (libraries included). Developers may also develop their own libraries and import them to other TriStation projects, or use Triconex standard libraries which are designed for compressor surge control, turbine governor control, automatic voltage regulation, burner management, and fire and gas applications. Additional to the three programming methods, the workbench also includes a cause and effect programming language editor (CEMPLE) to support the use of cause and effect matrix methodology. Overall the software suite is very intuitive and provides all functions within a well organized user friendly application.

Tricon v9 simulation environment is provided by TriSim. System models and data can be configured as would be implemented into real controllers and simulated on a PC prior to build. The troubleshooting capabilities available through simulation are endless. Further steady-state design, operational analysis, dynamic simulation, operator training, performance monitoring, and real-time optimization are all possible through the simulation engine SIM4ME. The following can be realized through predevelopment with Tricon simulation platform.

- Reduction of time to commission and startup
- Superior design quality verification in real-time control software
- Analysis and troubleshooting prior to integration
- Operator training capabilities
- Excellent offline simulation platform to facilitate rapid development

National and international standards

The following standardizations have been achieved by the current Tricon v9 system. The following list is not a complete review of all the certifications of Tricon v9 controller, however a summarized list of those certifications related to either nuclear application or Canadian standards. Related regulatory bodies and complying standards follow.

Canadian Standards Association (CSA) has certified Tricon v9 controller compliant with the following summarized international electrical safety standards.

- CAN/CSA-C22.2 No.0-M91 - General Requirements-Canadian Electrical Code, Part II
- CSA Std C22.2 No.0.4-M1982 - Bonding and Grounding of Electrical Equipment (Protective Grounding)
- CSA Std C22.2 No. 142-M1987 - Process Control Equipment
- UL Std No. 508 - Industrial Control Equipment

Certification from TV is critical to comply with controller integration into the following industries, applications and jurisdictions. This list is adapted from [D3].

- Emergency safety shut-down or other critical control applications requiring SIL 1-3 certification per the functional safety requirements of IEC 61508 9 (only Tricon v9.6 or later)

- Emergency safety shut-down or other critical control applications requiring AK 1-AK6 3 certification per the functional safety requirements of DIN V 19250 and DIN V VDE 0801

- All applications for use in European Union or other jurisdictions requiring compliance with the EMC Directive No. 89/336/EEC and Low Voltage Equipment Directive No. 72/23/EEC

Finally for direct implementation into the nuclear industry, though not by Canadian standards the United States Nuclear Regulatory Commission (NRC) has certified that Tricon v9 controller is suitable for use in nuclear 1E applications within the limitations and guidelines referenced in the NRC Safety Evaluation Report (SER) ML013470433, Review of Triconex Corporation Topical Reports 7286-545, "Qualification Summary Report" and 7286-546, "Amendment 1 To Qualification Summary Report," Revision 1. This qualification was based upon EPRI TR-107330, Generic Requirements Specification for Qualifying a Commercially Available PLC for Safety-Related Applications in Nuclear Power Plants [D4].

Tricon v9 controller would not be considered an advanced controller if it did not include extensive dynamic functional capabilities. Though the static characteristics provide a basis for safety controller design, it is the intelligence within the controller that can be utilized to significantly increase safety assurance within the shutdown system cycle. Table D.2 presented the dynamic self checking capabilities of the Data General MP/100 and the conformance of Tricon v9. The following will describe the beneficial assets related to SDS1 of Tricon v9.

Two-out-of-three (2oo3) voting

At a rate of once per scan, synchronization between the main processors is performed over the TriBus. TriBus is a proprietary high speed bus system specifically for communication between the triple redundant controllers. Digital input data from all input modules, through analog to digital conversion or directly from digital inputs is voted on prior to arriving at the main processor. Following control process execution, the data is sent to the output modules. The data is then voted on again and evaluated to assure no errors have been introduced prior to being output. This is performed as

close to the output terminal as possible to reduce the probability of internally caused faults.

The following is provided for better understanding of the input voting procedures. Each digital input modules has three independent channels for independent processing. The microprocessor for each channel scans every input point and transmits the data to the main channel specific processor upon demand. Once the data has been requested by the main processor, the data is voted on (compared with the other two independent processor input values), and if two out of three are similar, that two similar signals are considered valid. The third signal is simply ignored.

The voting at analog output modules is performed following I/O processor receiving the output signals from the main processors on each of three channels. Each set of data is voted on similarly to the input method (two out of three) and the decidedly valid data is driven to the output terminals.

Runtime-diagnostics, self check and self testing

There are extensive diagnostic procedures built into the system to evaluate the functionality of each main processor, I/O module and communication channel. Main processor diagnostics include: verifying fixed-program memory and the static portion of RAM, testing all basic processor instructions and operating modes as well as basic floating-point processor instructions and verifying the shared memory interface with I/O processors and each I/0 processor, communication processor, local memory, shared memory access, and loop-back of RS-485 transceivers. Further verification of the TriClock and TriBus interface are also performed as well as comparative checks between processors via TriBus.

Power modules have diagnostics built-in to check for out of range voltages and over temperature conditions. In the case of a short on a channel, the power regulator for that channel and module is disabled if possible rather than the entire power bus. Isolation of the system from external systems is also assured and diagnosed to prevent against ground faults.

The built-in capabilities of controllers to detect internal malfunction and perform signal monitoring and verification provide major benefit for advanced controller integration. Verifying system functionality and correct operation during runtime is critical in safety applications. If the controller fails to react during abnormal plant

Function	MP/100 Description	Tricon v9 Description
Watchdog Timer	Failure to update and external device periodically, within a specified time period, causes digital outputs to be placed in the safe state.	Independent timers verify the timely execution of the I/O module and main processor firmware and diagnostics. If a problem is identified, the faulty processors are disabled and control is transferred to the health processors.
Sequence Check	The actual module execution sequence is checked against the expected sequence.	If a main processor fails to report the proper sequence of execution, the I/O processor causes the main processor to enter the fail-safe state.
Checksums	A calculated checksum for a block of ROM, and a block of secure RAM, is compared to the expected.	Verification of fixed program memory
RAM Read-/Write	A word of RAM is tested with various bit patterns every pass, and then is restored.	Verification of static RAM, processor instructions and operating modes (specific routines not available).
Analog I/O Loopback	Checks the values read from the self-check A/Is to those written to the corresponding self-check A/Os. The A/O values are ramped up and down through the operating range.	Provided internal to controller (details in assessment)
Digital I/O Loopback	Compares values read from the self-check D/Is to those written to the self-check D/Os. The D/Os are toggled periodically.	Provided internal to controller (details in assessment)

Table D.2: Comparison of self checking capabilities across two controllers

operation the consequences could be devastating and possibly catastrophic. The basic self checking functions performed by controllers have not changed greatly through the development of digital control systems. Table D.2 demonstrates the self checking capabilities provided in the Darlington NGS trip computer software design in 1995.

Function	MP/100 Description	Tricon v9 Description
A/I and A/O Device Ready	If the applicable hardware is not ready (response time), the fatal error handler is invoked.	Verification of all I/O processors is performed
D/I and D/O Device Ready	If the applicable hardware is not ready (response time), the fatal error handler is invoked	Verification of all I/O processors is performed
Real-time Clock Tests	The number of real-time clock interrupts is tested against the range of the expected number of interrupts.	Verification of TriClock interface
Receive/Transmit Data Checks	Data is checked for validity. A fatal error occurs if the amount of receive and transmit data exceeds an upper limit.	All communication processors are verified and data are time checked and verified as well as two of three voted.
Range Checks	These are application checks on critical values immediately before their use to determine if they fall within the specified valid range.	All process variables will trigger indicators or alarms if they are outside of the designated range.
Spread Checks	Process inputs and calculations are periodically transmitted to an external computer to be checked for reasonableness.	Performed between triplicated processors via TriBus

Table D.3: Comparison of self checking capabilities across two controllers

Included in Table D.2 are the same characteristics as performed by Tricon v9.

The are two versions of self-test features which continuously verify the ability of Tricon v9 controller to detect the transition of a circuit. The versions include the following characteristics:

- Circuit stuck-on self-test feature that verifies the ability to detect transitions from a normally energized circuit to the off state
- Circuit stuck-on or stuck-off self-test feature that verifies the ability of a Tricon controller to detect transitions to the opposite state, either from on to off or from off to on.

Loop-back capabilities - Digital output modules perform output voter diagnostic (OVD). The purpose of this feature is to distinguish multiple fault scenarios from normal controller operation. OVD execution momentarily reverses the commanded state of each point within an output module. Integrated loop-back within the module allows the I/O processor to read the output value and determine if a fault exists at the output circuit.

Additional digital output modules are available and provide both voltage and current loop-back. The fault coverage performed by this function is to allow for both energized to trip and de-energized to trip conditions. Further, the module performs a continuous circuit continuity check and annunciates any loss of field load by the module.

Similarly, each channel on the analog output module has a current loop-back circuit which verifies the accuracy and presence of analog signals independent of load presence or channel selection [D5]. This function identifies a malfunction if current does not flow to one or more outputs because of an open loop.

Watchdog timer - Every I/O module and main processor is protected by an independent watchdog timer. The timer verifies the execution of I/O module firmware and diagnostics in a timely manner. A fail-safe state is entered if the execution order is not correct or the watchdog timer (500ms) does not reset. In fail-safe state all outputs for the faulty channel are disabled, and control is passed to the remaining healthy channels. Appropriate alarms and indicators will trigger accordingly.

Sequence of events - Communications modules allow for implementation of the sequence of events (SOE) procedures in Tricon v9. SOE performs inspection of designated variables for changes of state (events). When an event occurs the main processors save the current state of the variable along with a time stamp in memory buffer. Triconex SOE Recorder software may then be installed on a workstation and utilized to track the events that are performed by any given controller.

Summary

Through static and dynamic assessment, Tricon v9 controller appears to provide all of the capabilities apparent in the previously implemented systems. Comparative review to previous controller specifications is provided and demonstrates the compliance and

extension of Tricon to previous technologies. Though the information regarding the previous systems is not extensive in this documentation, it is apparent that the enhanced processing power of Tricon controller allows for extended functionality within the simple logic of SDS1.

Evaluating Tricon v9 system for implementation into SDS1 requires further attention. This paper simply overviews the functionalities that are available and could be utilized within SDS1. The process of implementing Tricon v9 controller into real-time simulation of the Darlington NPP is necessary. Current efforts at the University of Western Ontario have achieved communication between real-time NPP simulation and DCS hardware. The immediate future will see the integration of Tricon v9 controller among other control systems into the NPP environment for process performance and power generation indices evaluation. Once this is accomplished, detailed results as to enhanced methods for NPP control, both safety and non-safety critical, may be tabulated and presented.

References

[$D1$] Rooney, J.P., "Aging in Electronic Systems", Proceedings Annual Reliability and Maintainability Symposium, 1999, pp.293-299

[$D2$] de Grosbois, J., Hepburn, G.A., Olmstead R., "Qualification of Programmable Electronic System (PES) Equipment Based on International Nuclear I&C Standards,", American Nuclear Society Annual Meeting 2006 Proceedings, 2006

[$D3$] O'Connor, T., "Instrumentation, Control, and Human-Machine Interface to Support DOE Advanced Nuclear Energy Programs", Idaho National Laboratory, 2007

[$D4$] "Modern Instrumentation and Control for Nuclear Power Plants: A Guidebook", International Atomic Energy Agency, Vienna, 1999

[$D5$] "Planning and Installation Guide for Tricon v9 Systems", Invensys Triconex, 2004

Appendix E
Invensys Tricon v9 PLC Function Block Diagrams

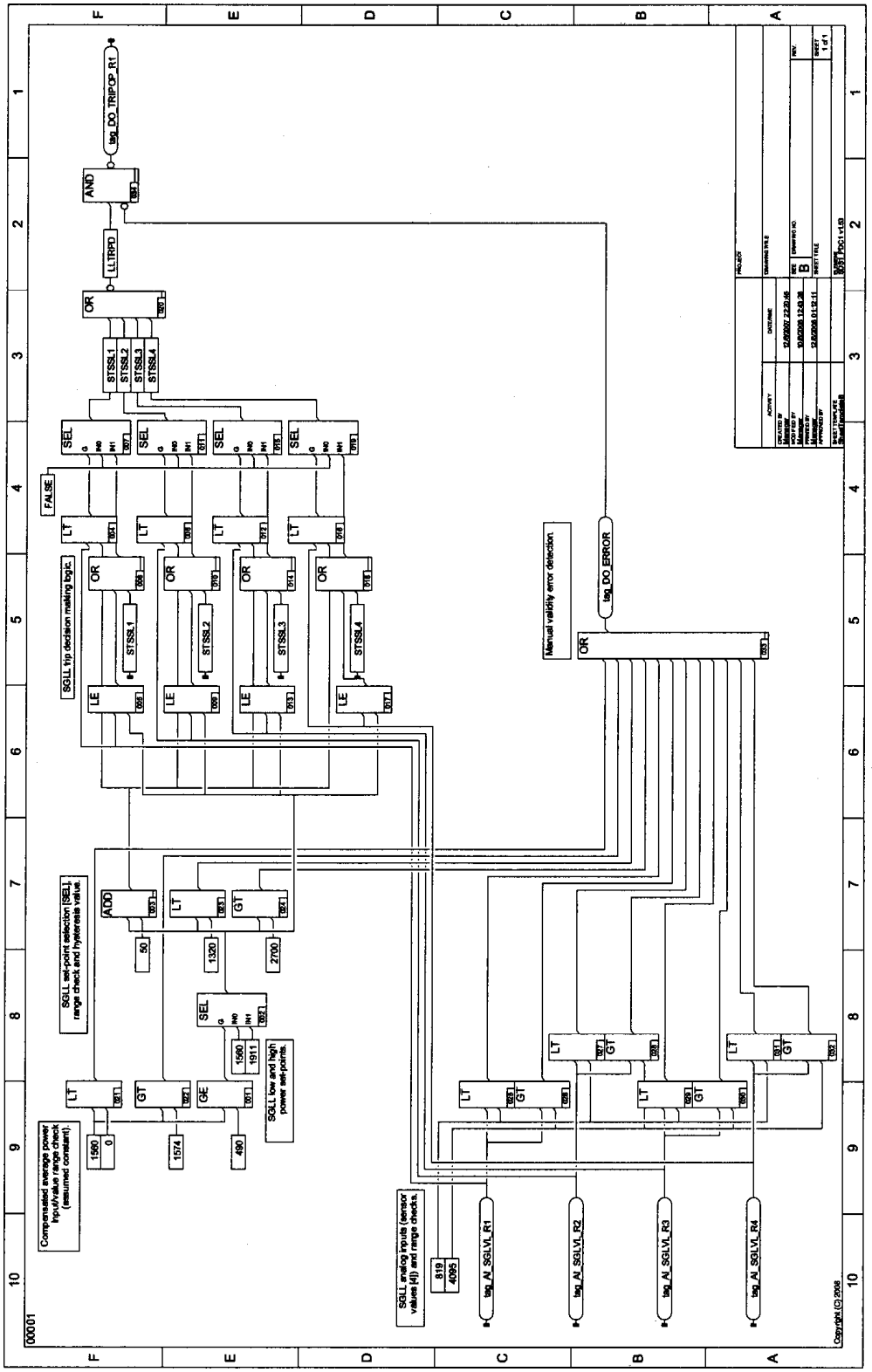

Figure E.1: Invensys Tricon v9 safety PLC steam generator level low function block diagram logic (Tristation 1131 Developer's Workbench)

Appendix F
National Intruments LabVIEW HIL Interface Virtual Instrument

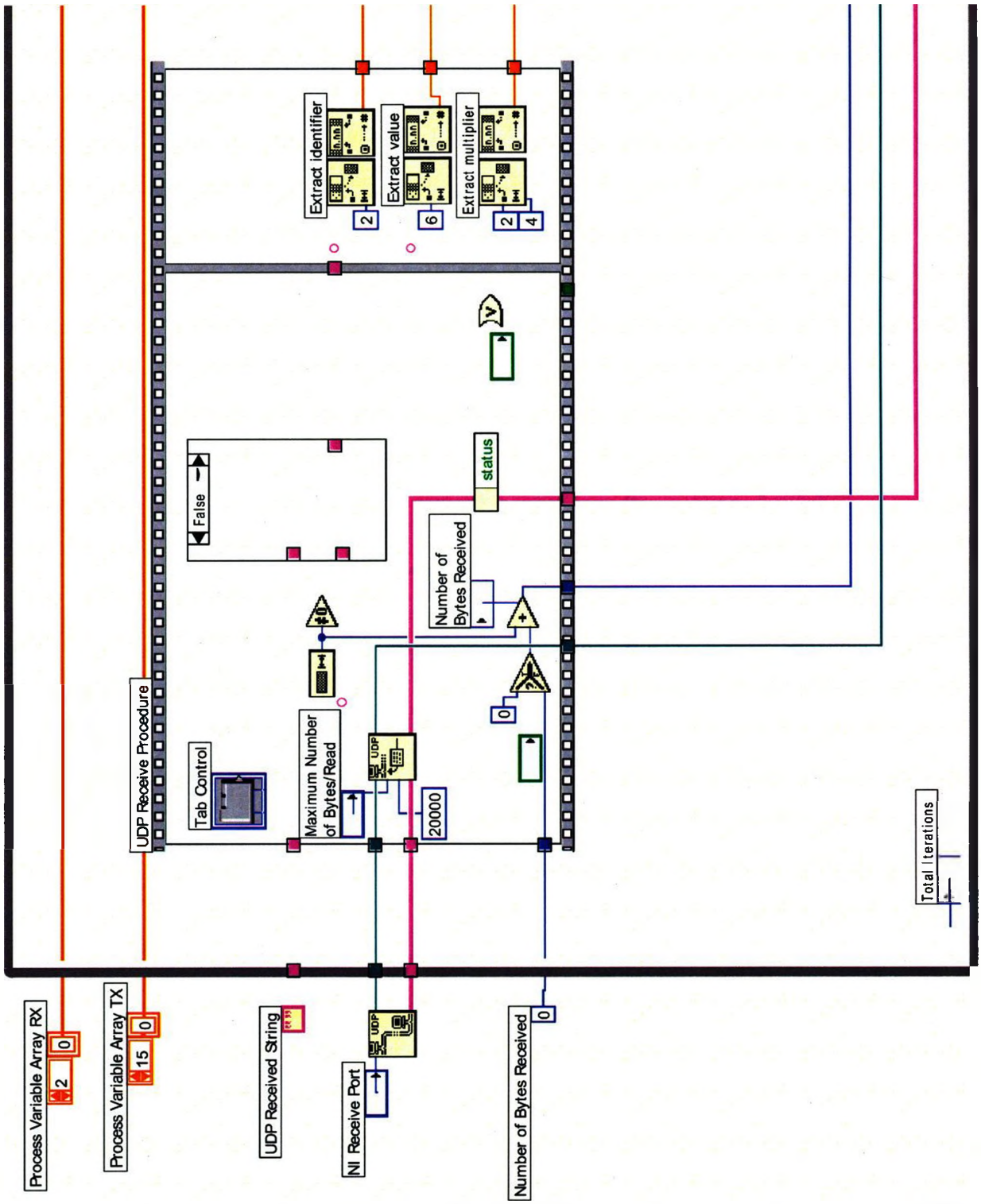

Figure F.1: National Intruments LabVIEW HIL interface virtual instrument (G programming language)

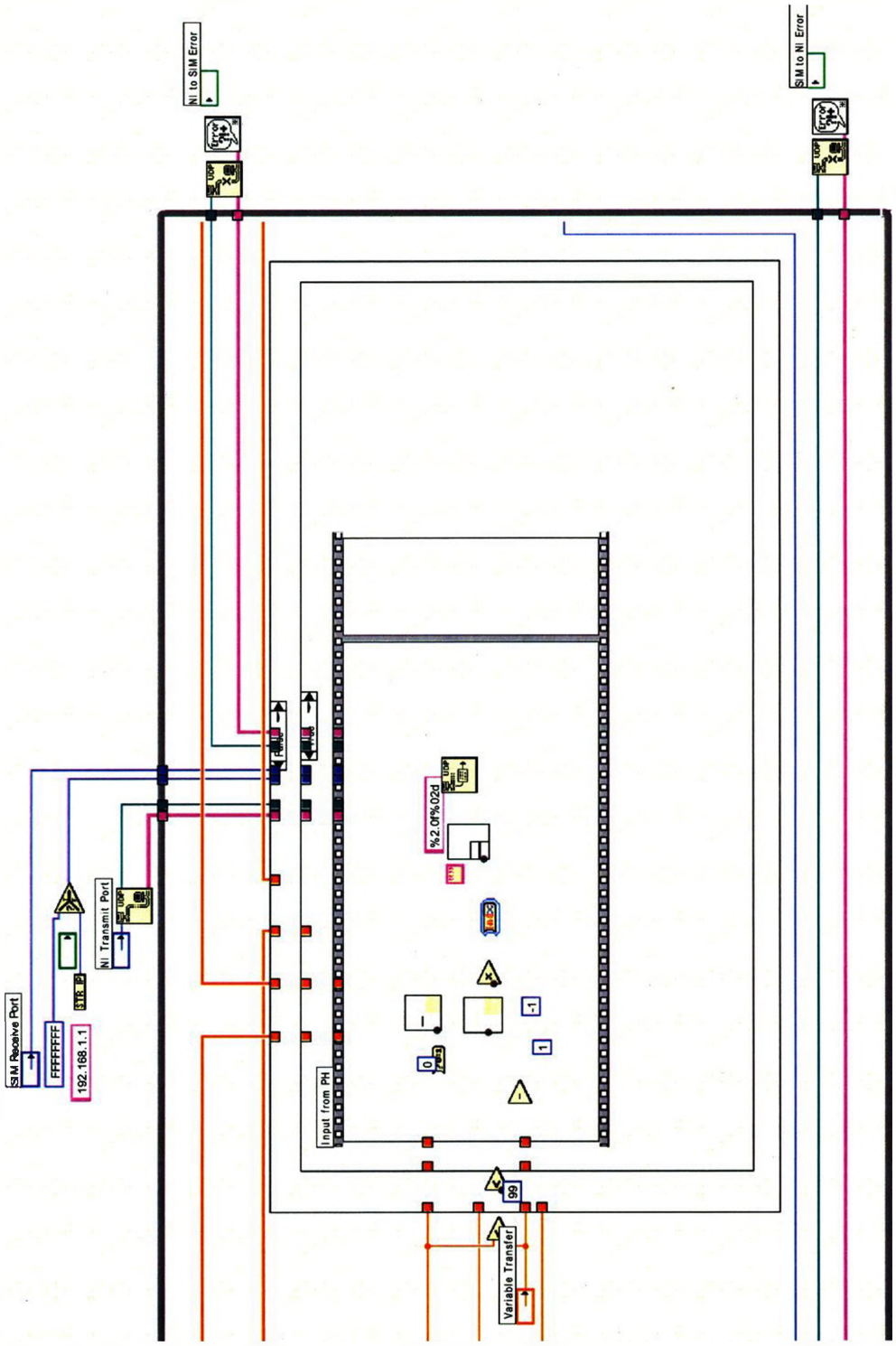

Figure F.2: National Intruments LabVIEW HIL interface virtual instrument (G programming language), continued

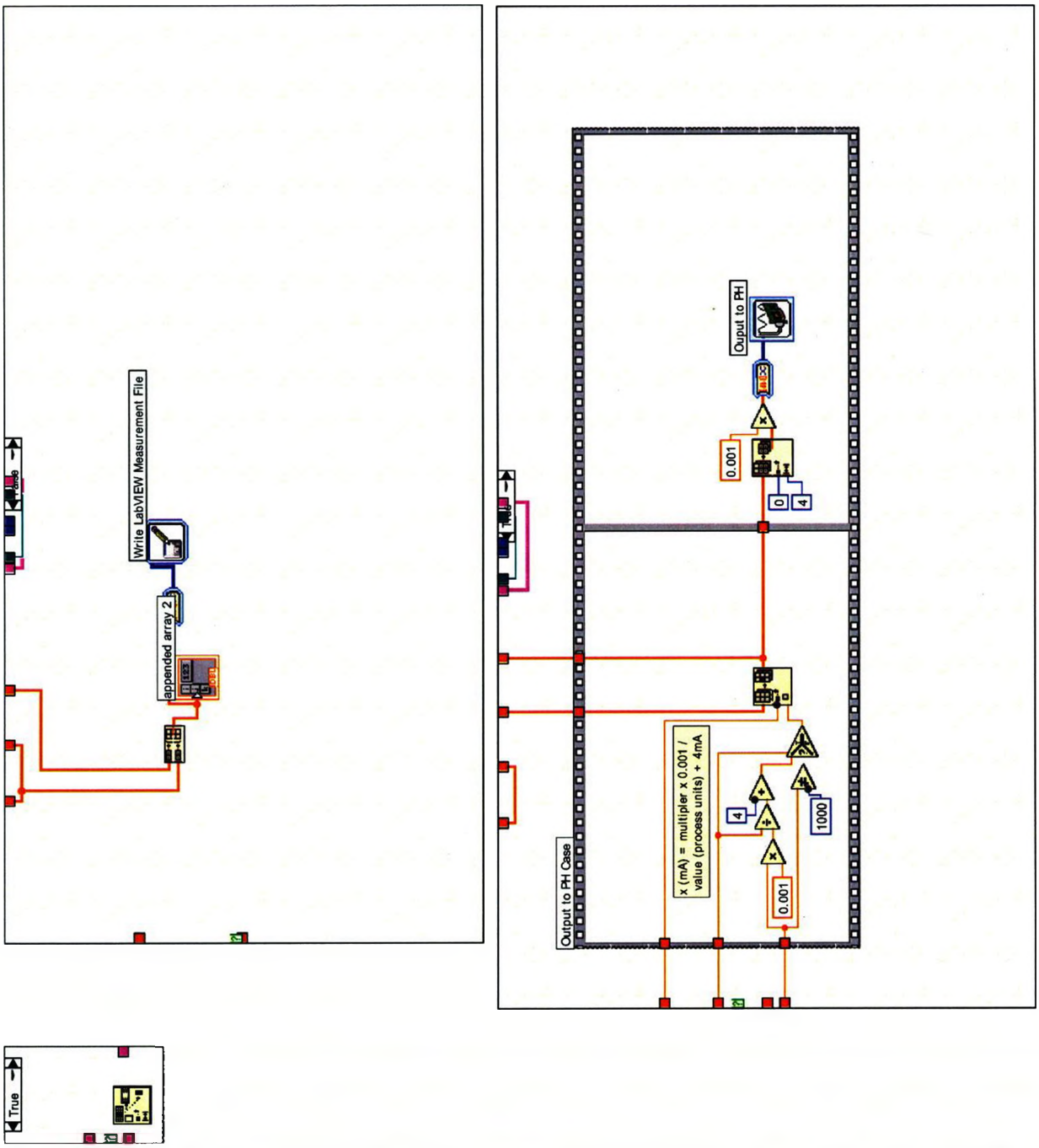

Figure F.3: National Intruments LabVIEW HIL interface virtual instrument (G programming language), continued