NUREG/CR-6465

Development of Tools for Safety Analysis of Control Software in Advanced Reactors

Manuscript Completed: March 1996
Date Published: April 1996

Prepared by
S. Guarro, M. Yau, M. Motamed

ASCA, Inc.
2250 East Imperial Highway, Suite 200
El Segundo, CA 90245-3547

L. Beltracchi, NRC Project Manager

Prepared for
Division of Systems Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001
NRC Job Code W6157

ABSTRACT

Software based control systems have gained a pervasive presence in a wide variety of applications, including nuclear power plant control and protection systems which are within the oversight and licensing responsibility of the U.S. Nuclear Regulatory Commission. While the cost effectiveness and flexibility of software based plant process control is widely recognized, it is very difficult to achieve and prove high levels of demonstrated dependability and safety assurance for the functions performed by process control software, due to the very flexibility and potential complexity of the software itself.

The development of tools to model, analyze and test software design and implementations in the context of the system that the software is designed to control can greatly assist the task of providing higher levels of assurance than those obtainable by software testing alone. This report presents and discusses the development of the Dynamic Flowgraph Methodology (DFM) and its application in the dependability and assurance analysis of software-based control systems. The features of the methodology and full-scale examples of application to both generic process and nuclear power plant control systems are presented and discussed in detail. The features of a workstation software tool developed to assist users in the application of DFM are also described.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

The recent years have witnessed an explosive growth in the use of software to control the functions of complex technological processes and systems. Software-controlled systems can be defined as systems in which the functions of mechanical and physical devices are controlled and managed by digital processors and computers which, in turn, execute software routines to implement specific control functions and strategies. Software-controlled systems have gained a pervasive presence in all types of applications, from the defense and aerospace to the medical, manufacturing, and energy fields. This report documents the results of a project entitled "Development of Tools for Safety Analysis of Control Software in Advanced Reactors", funded by the U.S. Nuclear Regulatory Commission with the objective of defining and developing a methodology suited to analyze and assess the safety of nuclear power plant software-based control systems. This objective included also the development of implementation tools, in the form of a self-contained software package, to facilitate the application of the analytical procedures of the methodology.

Software based process control has found increased use in the nuclear industry, including in the safety-related areas that are of most direct concern to the U.S. Nuclear Regulatory Commission. Reactor Protection System (RPS) algorithms and logic are software-implemented in Combustion Engineering nuclear power plants, as well as in many of the CANDU Canadian reactors. Current designs for the latest generation of nuclear power plants -- such as the Westinghouse AP600, the General Electric ABWR and SBWR and the CANDU 3 -- rely on the use of digital computers and associated software to accomplish a wide variety of process control tasks, such as continuous regulation of key plant physical parameters, component status monitoring and diagnosis, process/operator interfaces, and emergency shutdown. The great advantage of the concept of using computers as process controllers is in the almost unlimited flexibility afforded by the software implementation of system control functions, given the computational power and speed of the modern microprocessors and computers. As a result, very sophisticated and complex logic can be executed by relatively inexpensive processors loaded with the appropriate software instructions. If necessary or desirable, the originally implemented logic can also be modified during the life of the system by uploading new software instructions, without hardware changeovers.

While the cost-effectiveness and flexibility of the digital process control solution is almost universally accepted and recognized, it is also increasingly recognized that software flexibility may also result in greater software function complexity, by which logic errors of design or coding may find their way into a critically important software routine and cripple the operation of a whole system. The task of providing high assurance of the dependability and safety of the functions performed by process control software is thus becoming quite difficult to accomplish, due to the very flexible and complex nature of the software itself. In nuclear applications, the task of software qualification for safety related systems, which is largely based on testing, is estimated to require a year to complete (Petrella, et al., 1991).

To reduce the level of effort spent in testing without reducing the level of assurance, many experts have been calling for more stringent and formal practices to be applied in the process of defining the software specifications for critical systems (Parnas, et al., 1991). While the enforcement of a more disciplined and structured process of critical software specification and development is certainly a must, it should also be accompanied by the development and use of tools to model, analyze and test control system software design specifications and implementations in the context of the system within which the software is meant to operate. This in fact allows the system designer to achieve a higher level of assurance that the system and software specifications and realization do not leave the door open for unanticipated, unwanted and unsafe system behavior, and permit the identification of a reference envelope of "system safe behavior" against which actual implemented code executions and actual system dynamic behavior can be tested and verified. The project documented in this report has produced the formulation of a methodology, called Dynamic Flowgraph Methodology (DFM), and corresponding software tools, which provide the control software engineer with analytical capability designed specifically to permit the achievement of the objectives just described. DFM can thus be added to the limited set of analytical tools which are presently available for purposes of critical software dependability and safety assurance. For a brief review of pre-existing tools and further discussion on the rationale at the core of the DFM development, the reader is referred to the report introduction contained in Chapter 1.

The principal accomplishment of this research are summarized below. A concise description and discussion of the technical issues and findings associated with these accomplishments is also provided in the remainder of this executive summary.

- The features of the DFM analytical approach have been developed and defined in all necessary detail. The approach is articulated in two steps, which involve, respectively, system modeling and system assurance analysis. These steps can be integrated with, and facilitate, the traditional step of system assurance-testing. The detailed discussion of this part of the work is contained in Chapter 2 of this report.

- An integrated analytical software package, which implements the DFM procedures and algorithms, has been developed. This software runs on PC workstations under the Windows™ environment and relies on graphic models and user interfaces for data input and output, which directly reflect the "directed graph" representation at the base of DFM. A detailed presentation of the DFM Software Toolset is given in Chapter 3 of this report.

- An "interim test case" (ITC) was developed and used to aid the development and finalization of DFM. The ITC consisted of a simple fluid storage process controlled by a software logic and its features and functions were modeled and analyzed with the help of development versions of the DFM technique and software. The capabilities of the DFM analysis were tested by analyzing both unfaulted and faulted versions of the ITC system control software and by using DFM to identify the injected faults. The experience gained in the execution of this test-case analysis was used to identify areas of the DFM technique and tools which needed further development or improvement. A detailed discussion of the Interim Test Case is presented Chapter 4 of this report.

- A "demonstration test case" (DTC) was developed and used to prove the viability of the DFM methods and tools, as finalized in a baseline version at the end of the development phase concluded with the ITC exercise. The DTC consisted of a Pressurized Water Reactor (PWR) steam generator level and main feedwater control system, which was defined in a software-implemented configuration. For obvious reasons, the DTC was exercised in a simulated version, which involved the development of full dynamic models of the steam generator thermal-hydraulic behavior, as well as of the associated control elements and software. The DFM techniques and tools were successfully applied to the DTC and control software faults injected in the system were correctly identified. Important insights were collected from this application, which can be advantageously used to provide guidance for future utilization of the methodology. A detailed discussion of the Demonstration Test Case is presented Chapter 5 of this report. The insights gained from the DTC (and from the ITC) with respect to the application of DFM modeling and analysis are discussed in Chapter 6. Chapter 7 contains, in addition, conclusions and considerations regarding the future direction of DFM application and development.

The **Dynamic Flowgraph Methodology (DFM)** (Garrett, et al., 1995a, Yau, et al., 1995 and Garrett, et al., 1995b) is a methodology specifically developed for modeling and analyzing software-controlled systems. DFM is also based on a systems approach towards this objective, i.e., on the principle that a thorough system assurance analysis can only be performed effectively if both the software and hardware portions of the system are modeled and analyzed, with a well integrated understanding and representation of the overall system functions and interfaces. The basic execution of a DFM analysis requires a two-step process, which is also typically associated with certain software test procedures. The basic two steps are as follows:

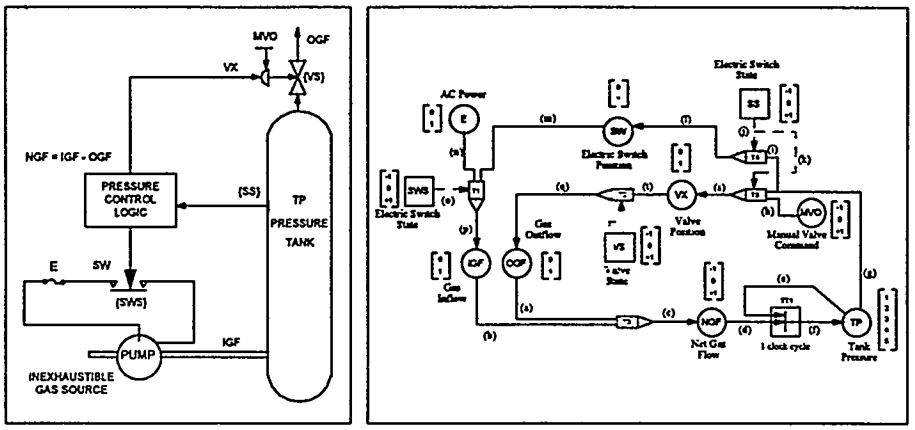
Step 1: Build a model of the digital control system for which a safety analysis is required. The model encompasses both the controlling software and the system being controlled.

Step 2: Using the model constructed in Step 1, systematically identify the modes by which specific system and process failure states may occur (this part of the process has been fully automated in the DFM Software Toolset described in Chapter 3).

If DFM-aided system-integration testing is also sought, a third step will be executed:

Step 3: Verify by integration testing, based on the results of the DFM analysis, that the digital control system exhibits the behavior predicted by its DFM model and, if corrections are applied to eliminate software failure modes, that the corrected digital control system behaves as desired.

A **DFM model** expresses the logical and dynamic behavior of a generic system. If this system is a digital control system, both the system being controlled and the controlling software are represented in the DFM model. A DFM model is a network built by using detailed multi-state representations of the cause-and-effect and time-varying relationships which exist among the key system and software parameters. The functional mappings for different combinations of these parameters are presented as decision tables. The decision tables can be constructed from empirical knowledge of the system, from physical equations which govern the system behavior, or from available software code and/or pseudo code. Figure ES.1 provides a relatively simple example of DFM model. More specifically, Figure ES.1b shows the DFM representation of the simple gas storage and pressure control system represented in Figure ES.1a. This system is a modified version of the system used in NUREG-0492,"Fault Tree Handbook" to provide an example of the fault tree analysis technique. For further details on the system represented here the reader is referred to the description given in Chapter 2.



(a) System (b) DFM Model

Figure ES.1 : A Simple Digital Control System and its DFM Model

The **analysis of a DFM system model** is conducted by tracing sequences of events deductively through the model structure, to identify the paths by which combinations of hardware and software conditions can propagate through the system to produce system events of interest. Deductive DFM analysis shares thus key conceptual features of fault tree analysis, but DFM uses a multi-valued logic (MVL), rather than binary, representation of system and parameter conditions. The top event of a DFM analysis can still be expressed in disjunctive form in terms of **prime implicants**, which can be considered as the MVL equivalent of the minimal cut sets encountered in binary fault trees. A prime implicant is a conjunction of primary events which is sufficient to cause the top event and which does not contain any shorter conjunction of the same events which is also sufficient to cause the top event. The prime implicants are unique and finite. However, finding them is a more challenging task than finding binary-logic minimal cut sets. Methods have been developed to obtain system prime implicants from component decision tables for static representations of systems. In DFM, the procedure for generating prime implicants has been extended to carry out deductive analysis across time transitions, so that dynamic representations of systems can be analyzed. DFM, therefore, represents a significant advancement beyond conventional fault tree analysis, as it is capable of producing, from one network model, MVL and time-dependent prime implicants, called "timed prime implicants" (TPIs), for a very large number of possible top events. The series of intermediate transition tables generated in the analysis show the time dependent sequence of events leading from the TPIs to the top event. Table I shows the seven prime implicants identified by the DFM analytical procedures for the top-event "tank pressure very high" in the system presented in Figure ES.1. Chapter 2 contains a detailed description of the analytical process and logic operation algorithms applied to carry out a DFM analysis.

Table ES.I : Prime Implicants for the Top Event "Tank Pressure Very High"

Number	Prime Implicant		
1	Electric switch was normal	@ t = -1	AND
	*Power was available	@ t = -1	AND
	Outlet valve was normal	@ t = -1	AND
	*No manual valve command	@ t = -1	AND
	*Sensor failed low	@ t = -1	AND
	*Tank pressure was high	@ t = -1	
2	Electric switch was normal	@ t = -1	AND
	*Power was available	@ t = -1	AND
	Outlet valve was normal	@ t = -1	AND
	*Valve closed manually	@ t = -1	AND
	*Sensor failed low	@ t = -1	AND
	*Tank pressure was high	@ t = -1	
3	Electric switch was normal	@ t = -1	AND
	*Power was available	@ t = -1	AND
	*Outlet valve failed closed	@ t = -1	AND
	*Sensor failed low	@ t = -1	AND
	*Tank pressure was high	@ t = -1	
4	*Electric switch failed closed	@ t = -1	AND
	*Power was available	@ t = -1	AND
	Outlet valve was normal	@ t = -1	AND
	*No Manual valve command	@ t = -1	AND
	*Sensor failed low	@ t = -1	AND
	*Tank pressure was high	@ t = -1	
5	*Electric switch failed closed	@ t = -1	AND
	*Power was available	@ t = -1	AND
	Outlet valve was normal	@ t = -1	AND
	*Valve closed manually	@ t = -1	AND
	*Tank pressure was high	@ t = -1	
6	*Electric switch failed closed	@ t = -1	AND
	*Power was available	@ t = -1	AND
	*Outlet valve failed closed	@ t = -1	AND
	*Tank pressure was high	@ t = -1	
7	*Tank pressure was very high	@ t = -1	

The **DFM Software Toolset** is an integrated set of software tools for implementing the model construction and analysis procedures of DFM. This software toolset is a Microsoft Windows™ application, and is developed as an integration of two principal modules: the Model Editor and the Model Analyzer. A detailed discussion of the features of both modules can be found in Chapter 3.

The **DFM Model Editor** is a graphical model building tool with which the user can create and edit DFM models. It converts the graphic representation of the DFM models into a set of data that can be stored in a database, and later used by the Model Analyzer. The Model Editor consists thus of a graphic model building environment for the user to create DFM models and a database structure to store information about the model created. The graphic model building environment provides a toolbox of icons representing DFM modeling elements with which the user can build a DFM model. The user defines the structure of a DFM model by picking the modeling elements from the icon menu and placing them on the screen. Dialog boxes are provided for the user to define the attributes of the model. These model attributes are stored in the form of a "B-trieve" database structure. This database structure consists of two major classes of data. One class characterizes the graphic attributes of the model, the other class characterizes the structure attributes of the model. The graphic attributes specify the placement of objects on the screen, while the structure attributes define the structure of the DFM model so that the Model Analyzer can backtrack the model correctly through the network and time transitions.

The function of the **DFM Model Analyzer** is to deductively analyze a DFM model to produce timed event sequences / fault trees and timed prime implicants for top events defined by the user. The Model Analyzer consists of the user interfaces and the analysis engine. The analysis engine is the part of the Model Analyzer which performs the deductive, "backtracking" steps. It carries out the steps of expanding the DFM model decision tables to form event-sequence intermediate transition tables, applying logic and dynamic consistency rules to remove inconsistent rows from the intermediate transition tables, simplifying the intermediate transition tables to obtain the "critical transition table", and finally applying the logic algorithm which generates the timed prime implicants from the critical transition table. The user interfaces facilitate the definition of the top events and the consistency rules by the user, and the display to the user of the intermediate transition tables and the timed prime implicants.

The testing and demonstration of the DFM modeling and analytical approach has been executed by applying the technique in two realistic test cases, which are referred to as the **"Interim Test Case" (ITC)** and the **"Demonstration Test Case" (DTC)**. The DTC, which is discussed in detail in Chapter 5, refers to the analysis of a PWR (Pressurized Water Reactor) steam generator level control system, the logic and algorithms of which are implemented via software. The DFM demonstration task required the development of a detailed thermal hydraulic simulator of the steam generator portion of the system, which in turn was recognized from the beginning as being a relatively lengthy and complex task. Thus, a simpler interim test case, i.e. the ITC, was conceived and constructed as a methodology test and development tool that would not require itself as much effort to construct as the DTC.

The ITC was constructed to represent a realistic system that could conceivably exist and be used in an actual industrial application. The system was to be defined in such a way as to be easy to model and simulate in terms of its physical hardware behavior, yet to include a digital control system with logic and functional characteristics of a relatively high degree of complexity so that it would provide a true test for the DFM application and generate feedback on how DFM may need modifications and/or improvements. The system is made up of a tank level and flow control system, as shown in Figure ES.2. The key components and features of this system are summarized below:

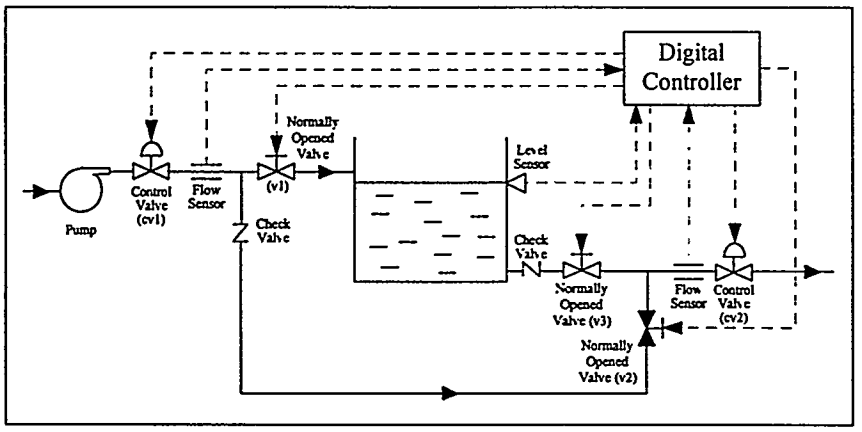


Figure ES.2 : ITC Digital Tank Level and Flow Control System

- A water tank, fed by water pump on the inflow pipe and regulated by control and stop valves on the inflow and outflow pipes.
- A 3-element (level sensor, inflow sensor, outflow sensor) tank flow and level control system, with control logic implemented in a software-driven controller.
- A tank bypass is allowed for emergency mode of operation (e.g., tank overflow). In this mode, the inflow and outflow pipes are directly connected and the tank is isolated via the actuation of the three stop valves located on the inlet and outlet sides of the tank piping.
- Stop-valve actuation and control logic selection implemented within the digital controller software.

As explained in detail in the discussion of DFM features provided in Chapter 2, different levels of detailed representation can be chosen when executing a DFM application. A more qualitative and high level representation

may be entirely sufficient for a DFM analysis that is conducted at the system specification level, when only a first-tier, preliminary definition and knowledge of the system design is available. On the other hand, in the assurance or safety analysis of a system that is either already operational, or that has at least been defined and designed to its detailed component level, the key features of the expected cause and effect and dynamic behavior of the system being modeled need to be known and represented in detail by the analyst, if the DFM analysis of the system is itself to produce results at a high degree of completeness and fidelity. Because the ITC exercise was intended to test the more complete set of DFM capabilities, the latter type of DFM modeling and analysis was sought, and a behavior simulation model of the ITC tank and piping system was developed. This simulation model was used to generate quantitative information on the static and dynamic behavior of the system, which in turn were used in the construction of the DFM model of the system.

The DFM model of the ITC tank level and flow control system is shown in Figure ES.3. The digital controller model is shown as a black box in this figure, but is expanded in full detail in Figure ES.4. The description of the variables that appear in the model as DFM "nodes" can be found in Table ES.II. In the **ITC DFM Model Analysis**, which was executed to test the capability of DFM in a system and software assurance mode of application, several analyses were conducted to better understand and refine DFM features. Particularly significant among these was the one conducted for a situation in which a fault was intentionally injected in the control software. The fault was placed in the module of the software code which sets the position of the control valves and the stop valves when the measured water level is above the high-high set-point. Under that condition, the digital controller should normally (i.e., when no fault is present) close the stop valve v1, open the stop valves v2 and v3, close the upstream control valve to the minimum position (5%) and open the downstream control valve to the maximum position (100%). The fault has the effect of closing the downstream control valve to 5%. The reader should note that this branch of the code will not be executed unless the level is above the high-high set-point. This requires some additional hardware failure to have also occurred. and makes the discovery of the fault during integrated system test in the actual system unlikely, since it would be difficult to exercise the actual operational software under such a faulted condition. The fault could of course be discovered if it existed also in "off-line" copies of the software, and one of these were tested for compliance with specified behavior.

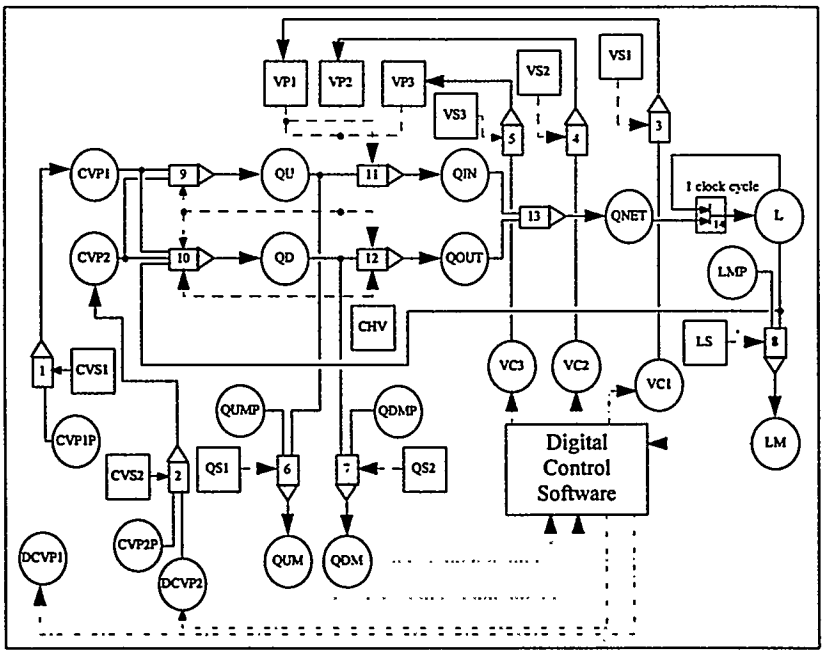


Figure ES.3 : DFM Model of the ITC Tank Level and Flow Control System

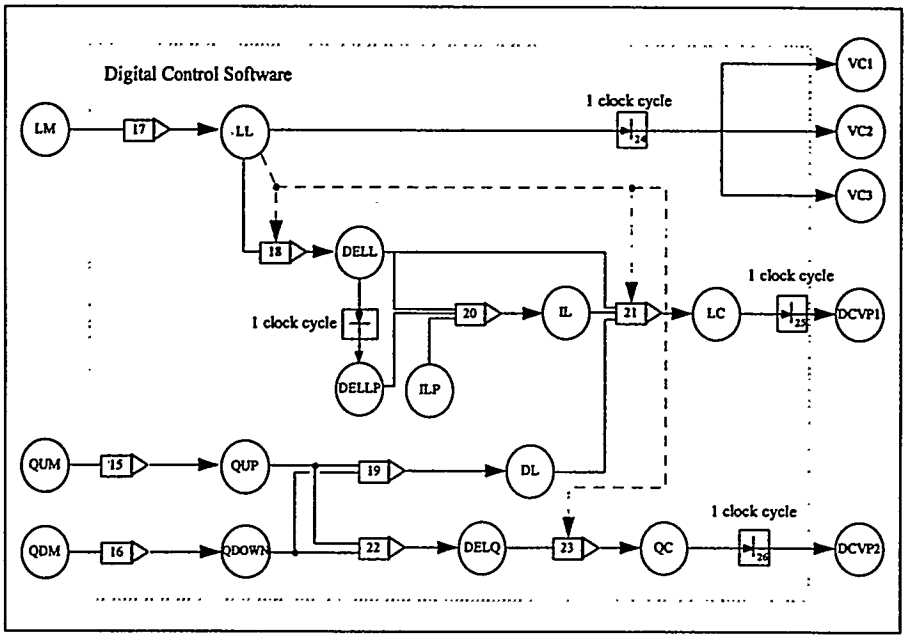


Figure ES.4 : DFM Model of the ITC Digital Controller

The ITC DFM model was constructed without using any prior knowledge of the software error, since the DFM decision tables are built directly by "testing" the individual modules of the digital control software. The system failure was defined as the tank "overflowing". This translated into a definition of the states of the pertinent DFM nodes as:

$$\{ (L = 5 @ t = 0) \text{ AND } (L = 5 @ t = -1) \text{ AND } (QNET = +1 @ t = 0) \}.$$

The meaning of the above definition is that the tank level is very high in both the current and the previous time step and that there is a net inflow of water into the tank. In the course of the various ITC analyses that were carried out, it was discovered that defining the top event as specifically as possible, such as using a combination of several conditions across different time steps to describe the tank overflowing, would enable the analysis to be performed more efficiently. Defining a top event in very precise terms ensures that the DFM Model Analyzer software needs less computer memory to store the intermediate transition tables developed during the analysis and spends less computing time tracing events which are irrelevant. As a comparison, when the top event was defined more simplistically as $\{ L = 5 @ t = 0 \}$ (the level is high at the current time step), the Model Analyzer ran out of memory before the analysis was completed. The care that has to be exercised in a specific and precise definition of the possible top events of interest is one of the key findings of our test cases and specific discussion of this point can be found in Chapter 6. Properly defined dynamic consistency rules are also important in constraining the prime implicant search to the domain of true significance. The dynamic consistency rules for the ITC analysis were mostly defined to reflect the assumption that if any sensor or valve fails in the system, it is expected to remain in the original failure state.

The ITC analysis of the injected fault described above was carried out for one step backward in the reference time frame. A "reduced form" of the prime implicant which was correspondingly identified is shown in Table ES.III. The software error that causes the tank to overflow is identified via its immediate effect, that is the command issued to the downstream control valve to its minimum position (software condition), AND the failure of the check valve (external condition). The reduced form of the prime implicant was obtained by the Model Analyzer from the full form initially identified, by deleting from the list of conditions in the prime implicant all those conditions which identify the states of sensors, control valves and stop valves related to the event sequence of interest as being normal, i.e. none of these components are failed.

Table ES.II : Description of the Variables in the ITC DFM Model

Variable	Description
CHV	State of the check valve
CVP1	Position of the upstream control valve cv1
CVP1P	Position of the upstream control valve cv1 in the previous cycle
CVP2	Position of the downstream control valve cv2
CVP2P	Position of the downstream control valve cv2 in the previous cycle
CVS1	State of the upstream control valve cv1
CVS2	State of the downstream control valve cv2
DCVP1	Change in position of the upstream control valve cv1
DCVP2	Change in position of the downstream control valve cv2
DELL	Level error term in the software
DELLP	Level error term in the software in the previous cycle
DELQ	Downstream flowrate error term in the software
DL	Mismatch between upstream flowrate and downstream flowrate
IL	Integral control term for level in the software
ILP	Integral control term for level in the previous cycle
L	Water Level in the Tank
LC	Upstream valve position command
LL	Software representation of the water level in the tank
LM	Measurement of the water level in the tank
LMP	Measurement of the water level in the tank in the previous cycle
LS	State of the water level sensor
QC	Downstream valve position command
QD	Downstream flowrate
QDM	Measurement of the downstream flowrate
QDMP	Measurement of the downstream flowrate in the previous cycle
QDOWN	Software representation of the downstream flowrate
QIN	Flowrate into the tank through the inlet
QNET	Net flowrate into the tank
QOUT	Flowrate out of the tank through the outlet
QS1	State of the upstream flowrate sensor
QS2	State of the downstream flowrate sensor
QU	Upstream flowrate
QUM	Measurement of the upstream flowrate
QUMP	Measurement of the upstream flowrate in the previous cycle
QUP	Software representation of the upstream flowrate
VC1	Command to stop valve v1
VC2	Command to stop valve v2
VC3	Command to stop valve v3
VP1	Position of stop valve v1
VP2	Position of stop valve v2
VP3	Position of stop valve v3
VS1	State of stop valve v1
VS2	State of stop valve v2
VS3	State of stop valve v3

In general, in a multi-state, non-coherent system representation such as that used in DFM, a parameter state can be always classified as "faulted" or "normal" only for the model parameters expressly set up to represent hardware failure / non-failure states. A reduced form of prime implicant can thus be obtained by not including in it the listing of normal states of this type of parameters. The states of process variables, on the other hand, are not definable a priori to be always "good" or "bad", and consequently are always listed even in the reduced prime implicant. This is because a process parameter state which is "good" in a certain type of situation may become "bad" when the

situation changes. For example, in the prime implicant in Table ES.III, the state of the upstream control valve is "good" (the valve is trying to reduce the tank inflow to a minimum in the presence of a potential overflow situation), whereas the state of the downstream valve is "bad" (since this valve is trying to reduce outflow). This classification of good and bad, however, would be completely reversed if we were in an opposite situation in which the tank water level had fallen below the minimum allowable. In essence, the state of being "commanded to close to its minimum" cannot be determined for either valve to be good or bad until the context within which this happens has been identified. This and other key features of multi-state non-coherent logic representation are encountered in the DFM application examples presented in Chapters 2, 4 and 5, and are further discussed in Chapter 6 (Section 6.3.2.3).

Table ES.III : Prime Implicant for "Tank Overflows" Event

	Prime Implicant		
1	check valve failed open	@ t = 0	AND
	Upstream control valve commanded to close to its minimum	@ t = -1	AND
	Downstream control valve commanded to close to its minimum	@ t = -1	AND
	Tank level was very high	@ t = -1	

The reader should note that the prime implicant in Table ES.III is not the only cause for the tank to overflow. In fact, many other prime implicants can lead to the same top event, one of which is, for example, the failure of the level sensor in the "stuck low" mode. The prime implicant in Table ES.III, however, is the only one containing a software error as a contributor to producing the top event. The fact that the non-software-related prime implicants were not produced by the DFM analysis which uncovered this particular time implicant is due to the application of logic rules in the search which required the DFM Model Analyzer to only look for event sequences that did not correspond to a specified control system behavior. For the ITC system the system specification requires the downstream valve to be always commanded open when the tank level is at "high" or "high-high". The effect of the application of these rules is to narrow the analysis into searching for a particular class of errors. Appropriate use of the rules allows the analyst to focus on particular failure paths, if he/she so desires, and to make more efficient use of the computational resources available for the analysis.

The **Demonstration Test Case (DTC)** is discussed in detail in Chapter 5 of this report and refers to the analysis of a Pressurized Water Reactor (PWR) steam generator level control system, the logic and algorithms of which are implemented via software. The case study called for a detailed analysis of the steam generator digital control system, which in turn required a detailed understanding of the dynamic behavior of the whole steam generator system. Thus, the development of a detailed thermal hydraulic simulator of the steam generator portion of the system was included as part of the task.

The configuration chosen for the dynamic simulation model is that of a vertical U-tube steam generator (SG) typical of a two loop Combustion Engineering Pressurized Water Reactor (PWR). The simulation model includes the Steam Generator, the Main Feedwater and Auxiliary Feedwater Systems, the Steam Header, the SG Pressure Control System, and the SG Level Control System. These systems are modeled in a considerable amount of detail, both from the thermal-hydraulic point of view (e.g., the SG model includes non-equilibrium conditions and a "level swell" model) and from the point of view of the equipment included in the simulation (e.g., sensors, actuators, emergency control devices such as the SG safety valves, etc.). This level of fidelity was sought to generate high quality information for the generation of the DFM models and the ensuing execution of the DFM analysis. The simulation model is implemented in a simulation code written in FORTRAN.

The principal purpose of the DTC was to demonstrate the application of DFM in the analysis of a nuclear power plant digital control system of realistic functionality and complexity. This was accomplished by creating, as part of the SG system simulation effort outlined above, a digital software version of a Combustion Engineering SG level control system, based on the actual detailed design of an existing plant analog control system. The function of the steam generator level control system is to maintain the water level at a pre-defined set-point. The system, in its simulated representation, consists of sensors, digital/analog (D/A) and analog/digital (A/D) converters, a processor running the control software and actuators which regulate the position of the main feed valve. The feedback control algorithm implemented by the software is based on a three element (proportional-integral-derivative, or PID) control

concept. To ensure that the integrated SG simulator (including all the subsystems identified above) behaves like an actual steam generator, and thus can be used as a realistic case study for DFM, the simulation code was exercised under five different scenarios, representing common conditions encountered by an actual steam generator control system:

1. Steady state
2. Turbine trip
3. Level sensor failure
4. Step power reduction
5. Ramp power reduction

The results of the simulation were fully consistent with the expected behavior of the actual system.

The SG simulator was used to produce "transfer functions" between key system parameters to be included as "nodes" of the DFM model. In the DFM model, these transfer functions would then be transformed into DFM decision table mappings, as explained in detail in Chapters 2 and 5. The DFM model of the DTC system is shown in Figure ES.5. The control software is shown as a black box in this figure, but is expanded in full detail in Figure ES.6. From the figures the reader can see that the DTC model is relatively complex. For this reason, and in order not to burden this summary with lengthy explanatory narrative, the reader is referred to Chapter 5, Table 5.III for the definition of the variables that appear in the model as DFM "nodes" and Section 5.2 for full details on the construction and features of the DTC DFM model.

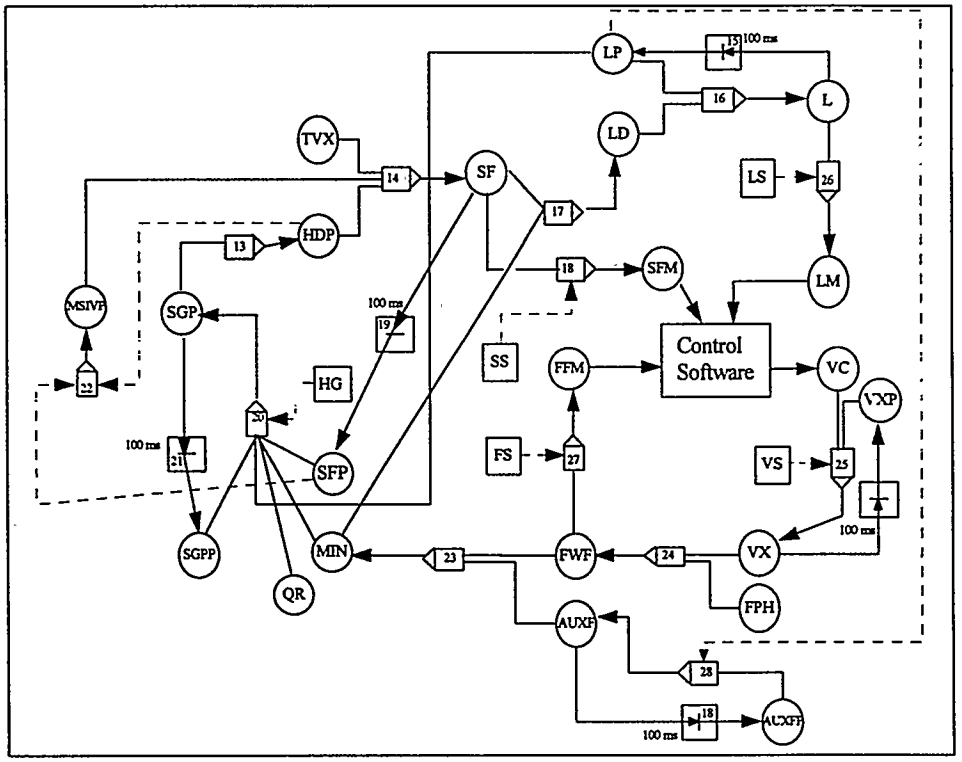


Figure ES.5 : DFM Model of the DTC Steam Generator Level Control System

The demonstration exercise carried out with the DTC followed the same steps as in the earlier analyses carried out on the ITC model. This time, however, two faulted conditions were analyzed instead on one, after also analyzing many unfaulted conditions for model validation purposes. The first faulted-case analysis involved a software specification error, while the second one involved a programming error. In both cases, and just as for the ITC fault-condition analysis, the DFM models were constructed without using any prior knowledge of the software faults. This was

again possible because the parameter-mapping decision tables were built directly by executing the modules of the digital control software, by a process similar in execution to "software module testing". Dynamic consistency rules and search boundary conditions were defined as appropriate (details are given in Chapter 5), and the DFM models were analyzed using the Model Analyzer. In the first faulted-case analysis, the software specification error could be identified in a timed prime implicant for the top event "Steam Generator Overflowing" after backtracking through the model in one time step. In the second faulted-case analysis, the software programming error could be identified in a timed prime implicant for the top event "Steam Generator Empty" after backtracking through the model for five time steps. Because of the relative complexity of these analyses, and the fact that the key features of a typical DFM backtracking analysis have already been discussed in this executive summary in relation to the ITC, the interested reader is encouraged to seek in Chapter 5 (Section 5.3) the details concerning the two "faulted-case" analyses of the DTC model.

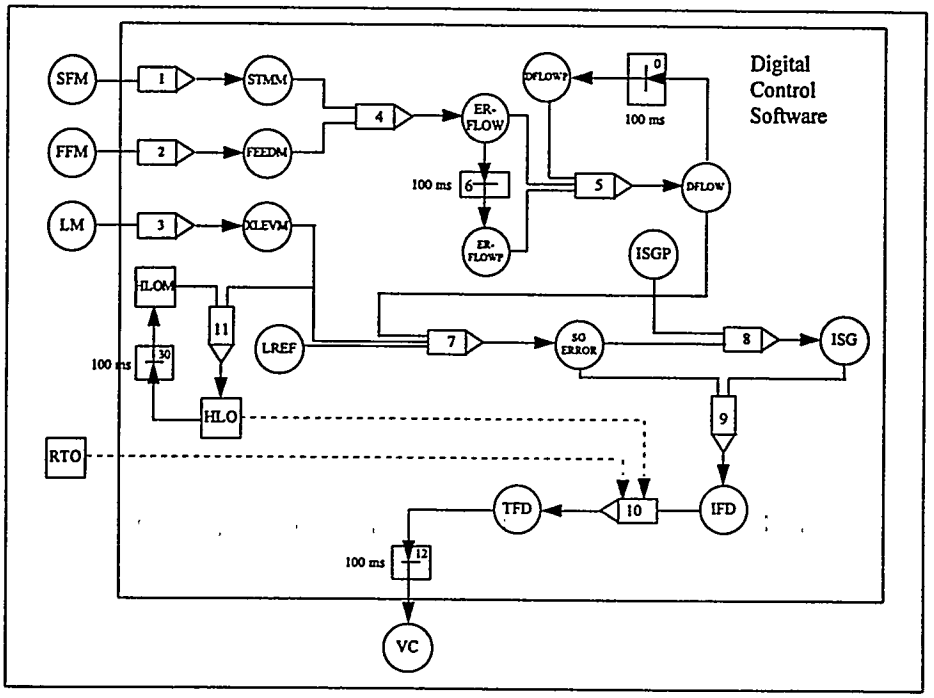


Figure ES.6 : DFM Model of the DTC Control Software

In the execution of this research, and especially in the modeling and analysis activities associated with the two test cases, many important insights were gained in several areas of interest for the future application of the analytical techniques that we have developed and discussed. With respect to **possible expanded objectives and uses of a DFM analysis**, the principal insight is that an extension of the DFM analytical procedures to include inductive (i.e., marching forward, rather than backtracking, in time and cause-effect sequences) would be very useful for software specification and design verification purposes. The development of this analytical capability would make in fact easier to test specifically for whether the control software and the associated system follow a certain type of desired behavior which is specified in advance (e.g., whether the system reaches a controlled and stable state starting from certain specified initial conditions). A possible mode of execution of this type of analysis that can be applied without changing the current form and features of the DFM models is illustrated with a simple application example in Chapter 6 (Section 6.1.1).

With respect to the **applicability of DFM to other types of systems**, i.e. systems other than those containing software exercising closed-loop continuous control actions, a point of considerable interest for nuclear safety applications, is whether the DFM technique is well suited for analyzing open loop control systems and software which implement plant safety and protection logic. Unlike closed-loop control systems that constantly apply mathematical manipulation of monitored parameters to provide continuous control adjustments, open loop control systems usually control one-shot, discrete actions associated with pre-defined system conditions which are used as

discontinuous trigger-points for the actions. Typical nuclear plant examples of such systems are the safety injection system and the reactor trip control system. An answer to this question can be given at a broad level by observing that the basic elements of DFM can be used to model any causality driven behavior. Thus, DFM can be applied to analyze a broad variety of systems, including open loop control systems, be they implemented in software or not. Indeed, the choice of closed loop systems as the principal focus of this study was made because, due to their dynamic characteristics, they are more difficult to analyze from the behavior-assurance point of view than open loop systems. Open loop protection systems were, to a degree, directly addressed in this study, since the test cases discussed in Chapters 4 and 5 actually dealt with systems in which open loop logic was intermixed to closed loop feedback control (e.g., the mode-of-control switch and trip logic included with the control of the water tank system discussed in Chapter 4). Indeed, two of the three "faulted-software" analyses carried out in the study deal with situations in which a "discrete software switch" is incorrectly triggered or actuated. The experience of modeling these systems provides practical evidence, and confidence, that applications of the DFM methodology to the verification and safety assurance of complex open loop control and safety should not pose any unexpected difficulties. If anything, the multi-valued and time-dependent logic modeling capability of DFM should provide an advantage over the traditional binary logic analytical tools presently used, in the modeling and analysis of those open loop control systems that, because of issues of relative timing of triggers a/o actions or because different actions may be associated with different "trigger-ranges" of a process parameter (or combinations of parameters), present greater complexity. A discussion of these issues can be found in Section 6.2.1. A further observation which appears relevant is that the potential DFM capability for inductive analysis, i.e., the analytical mode of application that generates and verifies forward-transition relations, as briefly presented and discussed in Section 6.1.1, can be used to verify that an open loop system will do what it is designed to do. That is, an automated inductive analysis of the DFM model of a reactor trip control system can be used to generate transition relations for all the possible execution paths and check that the execution sequences are followed exactly as desired and specified.

Insights have also been obtained regarding the **possible modes of optimization of the DFM modeling, analysis and testing procedures**. These insights are discussed in Section 6.3 and regard technical issues that are of interest to users of DFM at the application level.

In the area of **modeling**, it was observed that DFM "templates" (i.e., standard DFM model mini-modules) can be applied for certain control and hardware components that appear in systems in a recurring fashion (e.g., the elements of a PID-logic controller).

In the area of **analysis procedures**, an important insight was that the top-event of a deductive DFM search can be more advantageously defined as a combination of parameter conditions (if appropriate to express it in this fashion), rather than as a condition expressed in term of a single parameter. This in fact may save considerable amounts of memory and execution time resources to the DFM Model Analyzer software. Another very important insight regards the way in which the results of a DFM analysis may be presented and interpreted. More specifically, it is important to note that many software faults are identifiable not directly as basic conditions that are part of a "prime implicant" logic definition, but only via the observation of the actual sequence and logic path to the top event associated with the prime implicant itself. This is because, unlike for hardware failure modes, it is impractical to pre-define software faults as states of independent software parameters. On the contrary, most software faults are represented by "faulted" cause-effect mappings between software parameter states, which are, when considered by themselves, neither "good" nor "bad." This requires the analyst to examine carefully the sequences originated by a prime-implicant condition, and compare them with a reference model of "good behavior" in order to pinpoint the fault.

In the area of **testing procedures** the insights regard the relation of "module testing" and "system integration testing" with the modeling and analysis phases of a DFM application, respectively. More specifically, the equivalent of software **module testing** is practically carried out during the construction of the DFM decision tables that provide the software parameter state mappings needed for the full definition of the "transfer boxes" and "transition boxes" which appear in the DFM model of a functional software unit. Thus, in a software assurance activity involving the application of DFM, the process of DFM software model building can cover software module testing activities without additional effort. In fact, while exercising individual software modules in order to obtain the input - output parameter mappings needed for the construction of the DFM decision tables, the analyst can also compare these mappings with the existing module specifications and verify their correspondence to the latter, which is what

"module-testing" essentially consists of. **System integration testing**, on the other hand, can greatly benefit from the information provided by the DFM deductive analyses, since these can identify faulted conditions that result from the combination of both system hardware and software states, as well as from software dynamic conditions that could be missed in setting up the integration test envelopes that are to be executed in the assurance activity. When the DFM analysis identifies prime implicants corresponding to these conditions, this information can be used to set up tests that can reproduce them or simulate them, for both the purpose of confirming their existence and defining more precisely the corresponding range of system parameter values (in their actual continuous domain, rather than in its discrete DFM approximation).

In conclusion, at the end of the research and studies documented in this report, the DFM methodology has been developed to the level of being applicable to software-driven control systems of considerable complexity. This results both from the successful demonstration of its basic features and capabilities in two realistic, application-scale test cases and from the development of engineering-workstation software that implements and partially automates the execution of a desired analysis. Further refinement of the DFM tools will be conditional upon user feedback from field applications.

ACKNOWLEDGMENTS

The authors wish to acknowledge the assistance provided by the NRC Technical Project Monitor, Mr. Leo Beltracchi, with suggestions and comments which were very valuable to us towards the successful execution of this project. We also wish to thank Mr. Robert Brill, Mr. John Gallagher and Mr. Michael Waterman, all with the NRC Staff and Mr. Jim Lyle of the National Institute of Standards and Technology for feedback and suggestions provided to us at the mid-term presentation of our project at the NRC offices in April 1995.

LIST OF ACRONYMS

ABWR	Advanced Boiling Water Reactor
A/D	Analog to Digital
AFWS	Auxiliary Feedwater System
BDD	Binary Decision Diagram
D/A	Digital to Analog
DFM	Dynamic Flowgraph Methodology
DTC	Demonstration Test Case
FMEA	Failure Mode and Effect Analysis
FTA	Fault Tree Analysis
GUI	Graphic User Interface
HLO	High Level Override
ILI	Intermediate-Level Implicants
ITC	Interim Test Case
LIS	Laser Isotope Separation
MEPI	Multi-Event Prime Implicant
MFWS	Main Feedwater System
MSIV	Main Steam Isolation Valve
MVL	Multi-valued Logic
PD	Proportional Derivative
PI	Proportional Integral
PID	Proportional, Integral and Derivative
PWR	Pressurized Water Reactor
RPS	Reactor Protection System
RTO	Reactor Trip Override
SBWR	Small Boiling Water Reactor
SEPI	Single Event Prime Implicant
SG	Steam Generator
SV	Safety Valve
TED-Mode	Top-Event Decomposition Mode
TFT	Timed Fault Tree
TFTD-Mode	Timed Fault Tree Derivation Mode
TPI	Timed Prime Implicant

1 INTRODUCTION

This report presents a new methodology, the Dynamic Flowgraph Methodology (DFM), and the software tools for implementing DFM that were developed to address the requirement of tools for safety analysis of digital control software which will be used in advanced reactors. The methodology and software development was carried out as the NRC SBIR Phase II project titled "Development of Tools for Safety Analysis of Control Software in Advanced Reactors".

The ensuing sections in this chapter present the background that leads to the requirement of tools for safety analysis of control software and the objectives pursued in the development of DFM. The rest of the report presents the development of the methodology, the development of the software and the findings and insights gained in the Phase II research.

1.1 Issues Associated with the Use of Digital Control Systems

Digital control systems can be defined as systems in which mechanical and physical devices are controlled and managed by dedicated digital processors and computers. These latter devices, in turn, execute software routines (often of considerable complexity) to implement specific control functions and strategies. When the computer takes the form of a microprocessor which is physically associated with the remainder of the system, the term "embedded system" is also used (although it should be noted that very often the distinction between the term "digital control system" and the term "embedded system" disappears altogether in the day-to-day technical language usage). Digital control systems have gained a pervasive presence in all types of applications, from the defense and aerospace to the medical, manufacturing, and energy fields. The great advantage of using computers as process controllers is in the almost unlimited flexibility provided by the software implementation of system control functions and by the computational power and speed of the modern microprocessor devices. As a result, very sophisticated and complex logic can be executed by relatively inexpensive microprocessors loaded with the appropriate software instructions. The originally implemented logic can also be modified at any point in the life of the system it is designed to control by uploading new software instructions.

Software based process control, after a slow start, is finding increased use in the nuclear industry, even in the safety-related areas that are of most direct concern to a regulatory agency like the U.S. Nuclear Regulatory Commission. Reactor Protection System (RPS) algorithms and logic are software-implemented in Combustion Engineering nuclear power plants, as well as in many of the CANDU Canadian reactors. Current designs for the latest generation of nuclear power plants -- such as the Westinghouse AP600, the General Electric ABWR and SBWR and the CANDU 3 -- and for advanced nuclear enrichment processes -- such as the Laser Isotope Separation (LIS) demonstration plant -- rely on the use of digital computers and associated software to accomplish a wide variety of process control tasks, such as continuous regulation of key plant physical parameters, component status monitoring and diagnosis, process/operator interfaces, and emergency shutdown (Parnas, et al., 1991, Taylor and Sun, 1990, Vijuk and Bruschi, 1988, Petrella, et al., 1991 and Rippon, 1990).

While the cost-effectiveness and flexibility of the digital process control solution is almost universally accepted and recognized, it is also increasingly recognized that the task of providing high assurance of the dependability and safety of the functions performed by process control software is becoming quite difficult to accomplish, due to the very complicated and flexible nature of the software itself. In nuclear applications, the task of software qualification for safety related systems is estimated to require a year to complete (Petrella, et al., 1991). Even with such a level of effort, all potentially serious errors may not be identified by the current industry practices, based almost entirely on testing, so that several experts are calling for more stringent practices to be applied in the process of defining the software specifications for critical systems (Parnas, et al., 1991). The problem is considered serious enough at the higher levels of U.S. nuclear safety policy making that the Advisory Committee on Reactor Safeguards (ACRS) of the U.S. Nuclear Regulatory Commission has formed a special subcommittee to understand what the safety policy implications of this issue may be and what policy making recommendations should, accordingly, be made (Boenhert, 1990).

A sobering reminder of how serious digital process control software problems can be in terms of critical system safety was given by the incident which occurred at the Canadian Bruce-4 CANDU reactor in January 1990, which was the result of a programming error in the software used to control a reactor refueling machine (Boenhert, 1990). Because of this error, the control computer, after suspending execution of the main refueling machine positioning control subroutine while executing a fault-handling subroutine triggered by a minor fault condition detected elsewhere during the refueling process, returned to execution in the wrong segment of the main subroutine. Because of this software error, the refueling machine, which was at the time connected to one of the fuel channels of the pressure-tube reactor, released its brake and dropped its refueling assembly by about three feet, producing serious damage to the refueling assembly itself and to the fuel channel, and causing loss of coolant from the fuel channel.

In essence it must be recognized that the flexibility and power of control logic implemented in digital control system software has a dangerous back-side. Great software complexity means that logic errors of design or coding may find their way into a critically important software routine and cripple the operation of a whole system. While the enforcement of a more disciplined and structured process of software specification is certainly a must for safety-critical systems (Parnas, et al, 1991), this must also be accompanied by the development of tools to model, analyze and test digital control system software design and implementation in the context of the system within which the software is meant to operate. This will allow the system designer to achieve a sufficient level of assurance that the system and software being developed and integrated do not leave the door open for unwanted and unsafe system behavior, and would permit the identification, at a verified system specification and software specification level, of a reference envelope of "system safe behavior" against which actual implemented code executions and actual system dynamic behavior can be tested and verified. Analyzing and predicting digital control system behavior is especially important in light of the "discontinuous" nature of software errors, that is, the unforgiving attribute by which very "low level" software errors, such as the misreading of a single digit in a data structure, may produce large and catastrophic errors in the computer output used to drive and control the interfacing system hardware. Besides the nuclear plant incident cited above, far more serious failures in digitally controlled systems, caused by obscure software errors originating at very low programming or logic design levels, have indeed occurred, with consequences ranging from the very large financial losses produced by the half-collapse of a continental U.S. telephone network to lives lost because of the radiation overdoses meted out by the faulty control system of a medical cancer therapy machine (Neumann, 1985). These very serious occurrences have produced a growing awareness that in today's digital control systems the issues of reliability and safety for software can no longer be treated as if they played a secondary role with respect to issues concerning hardware reliability and safety.

1.2 Current Practices in Ensuring Safety of Digital Control Systems

Although the recognition is growing that it would be very desirable, for reliability and safety assurance purposes, to integrate in one process the modeling and analysis of the hardware and software components of a digital control system (Guarro, et al., 1990), the current state of the art does not offer practically implementable blueprints for such an approach. The approaches that have been proposed and/or developed in the past generally follow the philosophy of separating the hardware and software portions of the assurance analysis. The hardware reliability and safety analysts evaluate the hardware portion of the problem under the artificial assumption of perfect software behavior. The software analysts, on the other hand, usually attempt to verify or test the correctness of the logic implemented and executed by the software against a given set of design specifications, but do not have any means to verify the adequacy of these specifications against unusual circumstances developing on the hardware side of the overall system, including hardware fault scenarios and conditions not explicitly envisioned by the software designer.

Currently, digital control system software assurance is not treated much differently from that of any other type of software for real-time applications (such as communications software). Three principal types of software assurance philosophies can be recognized in the published literature, which we briefly attempt to describe and discuss below.

Assurance by testing, with or without the aid of reliability growth models is the most common approach. Testing is often performed by feeding random inputs into the software and observing the produced output to discover incorrect behavior. Software reliability models have been proposed to aid the testing strategies (Goel, 1985), although the applicability to software of reliability models extrapolated from the hardware reliability realm is seriously

questioned, even from within the software reliability research community itself (Littlewood and Miller, 1990). Software reliability models have not had a great impact so far in reducing the quantity and cost of software testing necessary to achieve reasonable assurance of correct behavior.

Formal verification is another approach to software assurance that applies logic and mathematical theorems to prove that certain abstract representations of software, in the form of logic statements and assertions, are consistent with the specifications expressing the desired software behavior. Recent work has been directed at developing varieties of this type of technique specifically for the handling of timing and concurrency problems (Narayana and Aaby, 1988 and Razouk and Gorlick, 1989). The abstract nature of the formalisms adopted in formal verification make this approach rather difficult to use properly by practitioners with non-specialized mathematical backgrounds. This practical difficulty is compounded by the growth in complexity and size of the process control software of the present generation. Finally, the issue of modeling and representation of hardware/software interaction, which we consider an important open issue in digital control system assurance analysis, does not appear to have surfaced as one of the current objectives of formal verification research.

The third type of approach to software assurance is one that analyzes the timing and logic characteristics of software executions by means of **discrete state simulation models**, such as queue networks and Petri-nets (IEEE Computer Society, 1985, Morgan and Razouk, 1987, Murata, 1989 and Leveson and Stolzy, 1987). Simulated executions are analyzed to discover undesirable execution paths. Although this approach can be extended to model combined hardware/software behavior (since the hardware behavior can in principle be approximated in terms of transitions within a set of pre-defined discrete states), difficulties arise from the "march-forward" nature (in time and causality) of this type of analysis, which forces the analyst to assume knowledge of the initial conditions from which a system simulation can be started. In large systems, many combinations of initial states may exist and the solution space may become unmanageable. A different approach, which reverses the search logic by using fault trees to trace backward from undesirable outcomes to possible cause conditions, offers an interesting solution to this problem, but encounters difficulties due to limitations in its ability to represent dynamic effects, and to the fact that a separate model needs to be constructed for each software state whose initiating causes are to be identified (Leveson and Harvey, 1983 and Cha, et al., 1988).

All the methods discussed above have merit, but none direct a special effort toward the philosophy of developing a "systems approach" to tackle the central issue of integrated hardware-software analysis in digital control system assurance. Useful elements of this philosophy can be found in Leveson and Harvey, 1983 and Jaffe, et al., 1989. In Phase I of the research, the authors had outlined an approach embracing this philosophy which combines features of an existing technique, namely the Logic Flowgraph Methodology (LFM) (Guarro and Okrent, 1984), with discrete state transition models, thereby solving the problem of providing an inductive (i.e., reverse causality backtracking) analysis capability while at the same time also providing the ability to keep track of the complex dynamic effects associated with sequential and time dependent software executions and digital control system behavior. In research conducted over the past decade, the principal investigator, working in cooperation with a UCLA research team in the Mechanical, Aerospace and Nuclear Engineering Department, has successfully demonstrated the usefulness of LFM as a tool for computer-automated failure and diagnostic analysis which shows broader potential applicability and efficiency than most other approaches that have been proposed for such objectives. As part of the LFM research effort, models of nuclear power plants and space-systems (Guarro, 1988 and Ting, 1990) have been derived; in addition, procedures to be applied in an expert system capable of assisting an analyst in the construction of LFM models have been identified (Guarro, 1987).

1.3 Objectives Pursued in the Development of DFM

The ultimate goal of this research is the development of a modeling environment and analytical framework that will enable the execution of a practically implementable process of verification and validation for software that is devoted to critical process control and safety functions. Verification and validation of critical software functions is an issue of great relevance for the approval and licensing of the new advanced designs that are being proposed for the next generation of nuclear power plants, as well as for the approval of digital upgrades that are presently being proposed and implemented in the control systems of existing plants. The principal objectives that were pursued in the Phase II

research are the development a set of implementation tools for this approach which will include application procedures and guidelines, as well as a self-contained software package embodying these procedures and the functionality/productivity features needed to make possible and simplify the use of the approach. After such an approach and associated "application package" are developed, they would be very useful as a means of assuring the dependability and safety of nuclear plants and installations with respect to the new set of problems posed by the extensive use of software in process control and supervision tasks, both in the commercial and government sections of the U.S. nuclear enterprise. Thus, we expect that the "application package", implemented as self-contained software, could be fully commercialized in Phase III.

2 THE DYNAMIC FLOWGRAPH METHODOLOGY (DFM)

This chapter describes, in detail, the formulation of the Dynamic Flowgraph Methodology (DFM) as it was developed under the Phase II research effort. The discussion begins with a presentation of the methodology's basic features, and then proceeds to describe the fundamental elements of the DFM modeling framework and how these modeling elements can be applied in the safety assurance of digital control systems. A tutorial example of DFM application to a simple control system is given in Section 2.5 to illustrate how the individual modeling and analytical features of the methodology work together and can be used in an actual task. While more complete and realistic applications are presented and discussed later in the chapters dedicated to the description of the two "test cases" that were selected for this project, the example in Section 2.5 is intended specifically for the readers who are not yet familiar with the DFM methodology and may find it useful to trace the conceptual and practical steps of a typical DFM analysis without the complication introduced by the larger amounts of detailed information and details generated in the analysis of more complex systems.

2.1 Overview of DFM

The DFM approach (Garrett, et al., 1995a, Yau, et al., 1995, Garrett, et al., 1995b) is essentially based on representing the system which is the object of the analysis in a "digraph" (directed graph) model, which is enriched with the explicit identification of the cause-and-effect and timing correspondences among the significant states of the parameters that are best suited to describe the system behavior. Once such a model has been produced, automated deductive or inductive algorithms that are built into the methodology can be applied to it. The deductive procedures that are discussed later in this chapter are applied to identify how system level states -- which may represent specific conditions of interest, be they success, anomaly or failure states -- can be produced by any combinations and sequences of basic component states. Conversely, inductive procedures can be applied to the same model, to determine how a particular basic component state can produce various possible sequences and system-level states. Thus, DFM can provide the multi-state and time-dependent equivalent of both fault tree analysis (FTA) and failure mode and effect analysis (FMEA), with the advantage that, once the DFM model of a system has been developed, the DFM system model already contains all the information necessary for the automated execution of these analyses for any system condition of possible interest. This can be compared, for example, with the execution of FTA, in which each system "top event" requires a separate manual analysis.

It is also worthwhile noting that, although the focus of this study is the application of DFM modeling and analysis to digital control systems, DFM, as a modeling and analytical tool, is very general in nature and can be applied to any kind of causality-driven system, whether such a system contains software subsystems or not.

The scope of the work discussed in this report is limited to the development and demonstration of DFM deductive analysis, as applied to digital control systems assurance. In this context, the application of DFM is typically a two-step process or, if DFM-aided system-integration testing is also sought, a three-step process, as follows:

Step 1: Build a model of the digital control system for which a safety analysis is required. The model encompasses both the controlling software and the system being controlled.

Step 2: Using the model constructed in Step 1, search for the manner in which specific system and process failure states may occur as the result of the propagation through the system of perturbations produced by basic "root cause" events (such as system component faults or manifestations of process-control logic errors).

Step 3: Verify by integration testing, based on the results of the DFM analysis, that the digital control system exhibits the behavior predicted by its DFM model and, if corrections are applied to eliminate software failure modes, that the corrected digital control system behaves as desired (the latter may require DFM analysis iteration to obtain predictions of corrected software behavior before the second cycle of testing).

As mentioned above, the first step consists of building a model of the digital control system that encompasses both the representation of the controlling software and the representation of the physical system being controlled. The model expresses the principal time dependent aspects of the system behavior and the functional relationships among the physical and software variables. The second step uses the model developed in the first step to identify logical combinations of "root cause" events (expressed in terms of hardware and/or software conditions) that cause certain specific system states of interest for which the analysis has been targeted, and the time sequences according to which these conditions come about. The system states for which the root causes are sought can be desirable or undesirable, depending on the objective of the analysis. This is accomplished by backtracking through the DFM model of the digital control system in a systematic, specified manner (which has been fully automated in the DFM Software Toolset described in Chapter 3), and by expressing the conditions that cause the system events of interest in the form of timed prime-implicants and timed fault (or success) trees. It should be noted that once a DFM system model is constructed, it can be analyzed to produce many timed fault trees; that is, the same model can be used repeatedly to check many different system states of interest. The information contained in the fault trees that describe the hardware and software conditions that can lead to system states of interest can be used to uncover undesirable or unanticipated software/hardware interactions, thereby allowing improvement of the system design by eliminating unsafe software-execution paths. This same information can be used in the third step to guide functional and system-integration testing to focus on particular domains of inputs and system conditions that are identified by the DFM analysis as potentially leading to undesired system behavior.

A discussion of some of the conceptual underpinning of DFM-assisted testing is given in Section 2.4. It is useful, however, to note up front that the application of DFM to systems containing software is inherently tied to software testing and that DFM is intended to assist testing by intermixing the testing steps with analysis steps that allow a more efficient use of the limited resources available for testing within a specific project. As will become apparent later through the discussion of the test cases in Chapters 4 and 5, a key step in the construction of a detailed DFM software model involves testing the individual modules of the software. In particular, the specific elements of a DFM model that describe the behavior of system software modules (these elements in the DFM nomenclature usually consist of "transfer boxes" and/or "transition boxes", as described in Sections 2.2.1.3 and 2.2.1.6) can be defined by test-executing those software modules. Thus, when modeling a system that includes actual software, "module testing" -- which itself constitutes the basic first step of standard software testing procedures -- becomes an integral part of the above mentioned first step of DFM application (i.e., the "system modeling" step). After the second, "system analysis" step is completed, the DFM analysis results provide the information necessary for integrating these module testing results with the model of the operating environment of the software (i.e., the model of the "hardware" and "external world interface" portion of the system), so that intelligent boundary conditions are identified by this analysis to test the integrated system. In fact, testing the software under all possible system conditions is practically impossible, but testing it only under the standard operating conditions is most likely not enough to guarantee the reliability and safety of the software. DFM provides thus a way to systematically identify the boundaries and the exception conditions for which system-level testing is most needed, and where the testing resources can be best applied. This issue will be further elaborated in the discussion of DFM-driven testing later in this chapter (Section 2.4) and in Section 6.3.3, which discusses findings and insights derived from the application of DFM to the two test-cases presented in Chapters 4 and 5.

2.2 Framework for Model Construction (Step 1)

A DFM model expresses the logical and dynamic behavior of a generic system. If this system is a digital control system, both the physical system controlled by the software and the controlling software itself are represented in the DFM model. A DFM model is an integration of a "time-transition network", a "causality network" and a "conditioning network", which is built by using detailed multi-state representations of the cause-and-effect and time-varying relationships that exist among the key system and software parameters. Figure 2.1 shows a simple gas-storage system with its associated pressure control system, which we can assume for the sake of our discussion to be implemented by a simple digital control module, although for the introductory nature of the discussion of DFM features that is sought in this chapter very little would change if we assumed the control system to be implemented by some sort of hardwired logic functions. Figure 2.1 also shows the DFM model of the integrated system.

Although the nature and features of this system will be discussed in detail in Section 2.5 to illustrate how DFM modeling and analysis steps can be executed in a typical, though simplified, application, discussing the DFM modeling concept and building blocks within the context of this simple system should enhance the reader's comprehension of DFM. The reader is encouraged to jump ahead to Section 2.5 to gain a better understanding of this simple system.

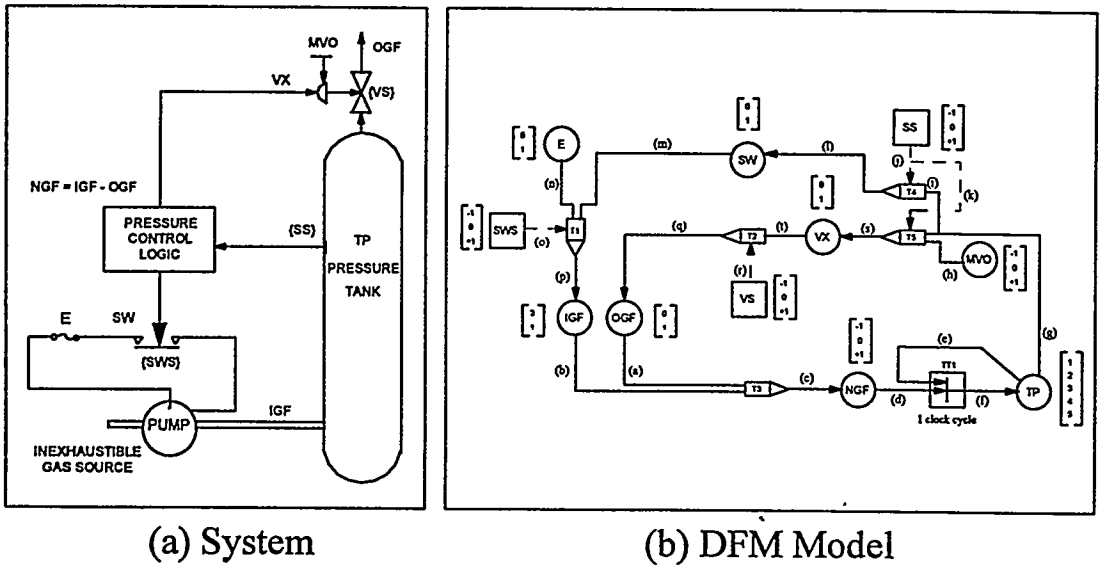


Figure 2.1 : A Simple Digital Control System and its DFM Model

The networks mentioned above are constructed from the DFM modeling elements. These modeling elements, as well as the manner in which they are assembled to form the three networks of a DFM model, are discussed below.

2.2.1 DFM Modeling Elements

A DFM model makes use of certain basic modeling elements to represent the temporal relations and the logical relations that exist in the system and the associated software. More specifically, a DFM model integrates a "time-transition network" which describes the sequence in which software subroutines are executed and control actions are carried out, a "causality network" that shows the functional relationships among key hardware and software parameters, and a "conditioning network" which models discrete software behavior due to conditional switching actions and discontinuous hardware performance due to component failures. The building blocks of these three types of networks are process variable nodes, condition nodes, causality edges, condition edges, and transfer/transition boxes and their associated decision tables. These basic modeling elements are shown in Figure 2.2.

2.2.1.1 Process Variable Nodes

Process variable nodes represent physical and software variables that are required to capture the essential functional behavior, continuous or discrete, of the digital control system. For example, the process variable node TP in Figure 2.1 represents the pressure in the gas tank.

A variable represented by a process variable node is discretized into a number of states. The reason for the discretization is to simplify the description of the relations between different variables. The choice of the states for a process variable node is often dictated by the logic of the system. For instance, it is natural to set a state boundary at a value that acts as a trigger point for a switching action or a value that indicates the system is progressing towards failure. The number of states for each variable must be chosen on the basis of the balance between the accuracy of the model and the complexity introduced by higher numbers of variable states.

For example, using the process variable node TP in Figure 2.1 for illustration, this parameter represents the tank pressure and it can vary from very low to very high. TP is discretized into 5 states, and the discretization scheme of this process variable node is shown in Table 2.I. This scheme reflects the knowledge that state 1 signifies very low pressure and the tank is almost empty. State 2, state 3 and state 4 represent low pressure, normal pressure and high pressure respectively, while state 5 corresponds to dangerously high pressure, which can cause the tank to burst. In addition, the state boundary between 2 and 3 is set to correspond to the trigger point where gas inflow is activated to replenish the tank. Similarly, the boundary between states 3 and 4 corresponds to the set-point for opening the relief valve to decrease the pressure in the tank.

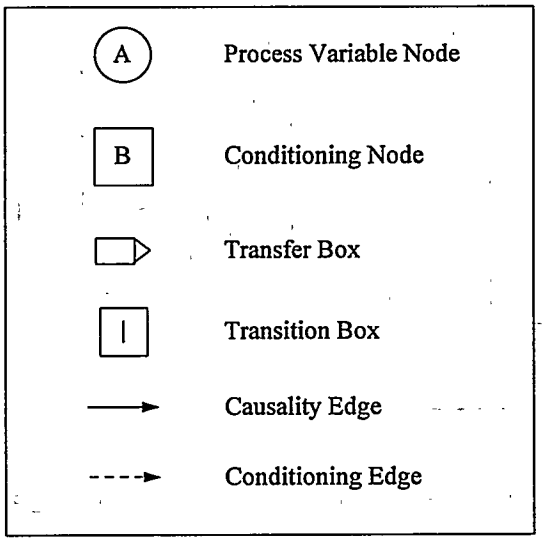


Figure 2.2 : The Basic DFM Modeling Elements

Table 2.I : Discretization Scheme for the Process Variable Node TP

State	Description
1	Tank pressure is very low
2	Tank pressure is low
3	Tank pressure is normal
4	Tank pressure is high
5	Tank pressure is very high

2.2.1.2 Causality Edges

Causality edges are used to connect process variable nodes to indicate the existence of a cause-and-effect relationship between the variables described by the nodes. For example, the causality edges (a), (b) and (c) in Figure 2.1 show that the value of the process variable NGF (net gas flow into the tank) is directly related to the values of the process variables IGF (gas inflow into the tank) and OGF (gas outflow through the valve at the top of the tank). The precise nature of the functional relationship (or the transfer function) is described by a transfer box that is always directly associated with each causality edge (please see discussion in Section 2.2.1.3 below).

2.2.1.3 Transfer Boxes and Associated Decision Tables

A transfer box represents a transfer function between process variable nodes. The quantification of the transfer function, i.e., the manner in which the states of the input process variable nodes are correlated with those of the output process variable nodes, is described by decision tables associated with each transfer box.

A decision table is associated with each transfer box and is used to quantify the relationships between its input and output process variable nodes. This table is a mapping between the possible combinations of the states of the input

process variable nodes and the possible states of the output process variable nodes. Decision tables are extension of truth tables in that they allow each variable to be represented by any number of states. These tables have been used in earlier developments to model components of engineering systems (Salem, et al., 1977; Salem, et al., 1979; Henley and Kumamoto, 1992).

Because each transfer box input or output variable is a vector of states, and each combination of input states maps to a state of each of the output variables, each decision table is actually a multi-dimensional matrix whose dimension is equal to one plus the number of its inputs. For simplicity and convenience of representation, all decision tables can be reduced to a two-dimensional form. In this simplified form, there will be a column for each input variable and a column for each output variable of interest. For example, in Figure 2.1, transfer box T3 links the input nodes IGF and OGF to the output node NGF. IGF is discretized into 2 states (0,1), as is the other input node OGF (0,1), while the output node NGF is discretized into 3 states (-1,0,+1). Hence in the decision table, there are 3 columns (1 for each of the two inputs and 1 for the output). The decision table in Table 2.II shows the output states produced from different combinations of the states of the inputs.

Table 2.II : Decision Table for the Transfer Box T3 in Figure 2.1

IGF	OGF	NGF
0	0	0
0	1	-1
1	0	+1
1	1	0

Decision tables can be constructed from empirical knowledge of the system, from physical equations that govern the system behavior, or from available software code and/or pseudo code. Building decision tables with empirical knowledge and/or the pseudo code provides a means of modeling the intended behavior of a system, and thus allow analysis to be performed on the specifications or the design concept, even before the system exists. On the other hand, using physical equations and running module testing to fill the decision table rows with detailed input/output state mappings creates a model reflecting the actual behavior of the system, thus enabling the actual system to be verified. The accuracy of the decision tables is crucial for the analysis because it directly correlates to the fidelity of the model (its ability to predict system behavior). Hence, to keep decision tables from growing too big, a judicious selection of the number of states into which each node is discretized should be made, without at the same time losing too much of the more detailed system-behavior information.

2.2.1.4 Condition Edges

Unlike causality edges, condition edges are mostly used to represent true discrete behavior in the system. They link parameter nodes to transfer boxes, indicating the possibility of using a different transfer function to map input variable into output variable states. For example, as shown in Figure 2.1, depending on the value of the parameter VS (unfaulted/faulted state of the valve), the output OGF (gas outflow through the valve) can be proportional to the input VX (valve position), or OGF can be stuck at minimum or maximum values regardless of VX.

2.2.1.5 Condition Nodes

Condition nodes, like process variable nodes, represent physical or software parameters. However, condition nodes are used in DFM to more explicitly identify component failure states, changes of process operation regimes and modes, and software switching actions. Condition nodes represent variables that can affect the logic superstructure of the digital control system by modifying the causal relations between the process variable nodes. Condition nodes that are linked to causality edges and upstream process variable nodes are at the same time process variable nodes as well as condition nodes, but condition nodes whose states are not determined by other upstream process variable nodes are treated in DFM as "random variables", i.e., as variables that can be assumed to be in any of their possible states. In the latter case, a distribution of "relative frequency" of the associated states could also be assumed, for purposes of probabilistic quantification. For example, node VS in Figure 2.1 is a condition node that is not affected by any upstream process, as the failure of the valve is assumed to be a random event and is not explicitly modeled. It

should be noted that the effect of a condition node on an output variable is modeled through a decision table, as is the case for a process variable node. The reason for having the added modeling elements of condition nodes and condition edges is to offer a clear distinction between continuous and discontinuous behavior in a system.

2.2.1.6 Transition Boxes and Associated Decision Tables

Transition boxes are similar to transfer boxes in that they connect process variable nodes to indicate cause-and-effect relationships. Condition nodes can be associated with transition boxes to represent discontinuous behavior between the input and output process variable nodes. Decision tables are again used to describe the relationships between the input and output process variable nodes. However, transition boxes differ from transfer boxes in the essential aspect that a time lag or time transition is assumed to occur between the time when the input variable states become true and the time when the output variable state(s) associated with the inputs is(are) reached. This time delay is a characteristic of the transition which is being modeled and is treated as an attribute of the transition box. For example, in Figure 2.1, the transition box TT1 indicates that a new value of TP (an updated value of the tank pressure) depends on the value of NGF (the net gas flow into the tank) and the old value of TP (the tank pressure at the previous clock cycle). Transition boxes are routinely used to model the execution of software routines and the handling of interrupts, which often play an important role in the execution flow of digital control systems software. They can of course also be used to model hardware time transitions..

2.2.2 Model Construction and Integration

To construct a DFM model for a digital control system, the first step is to select the physical components and the software functions that are to be included in the model. Following that, the physical parameters and software variables that capture the essential behavior of these components and software functions are identified and represented as process variable nodes. These process variable nodes are then linked together by causality edges through transfer boxes or transition boxes to form an integrated "causality" and "time-transition" network. Discrete behaviors such as component failures and logic switching actions are then identified and represented as condition nodes, which are tied to transfer boxes and transition boxes expressly to show how a "conditioning network" of discrete actions and events actually interacts with and affects the integrated "causality" and "time-transition" network. The parameters represented by the process variable nodes and condition nodes are discretized into meaningful states, and decision tables are constructed to relate these states. The decision tables can be constructed by empirical knowledge of the system, from physical equations that govern the system behavior, or from available software code and/or pseudo code. The completed DFM model then reflects the essential causal, temporal, and logical behavior of the digital control system. The example discussed in Section 2.5 will illustrate how these steps are carried out.

2.3 Framework for Model Analysis (Step 2)

2.3.1 Introduction to Fault Trees and Cut Sets

The analysis of a DFM system model constructed according to the rules described above (Step 1) is, in the applications of interest to this study, conducted by tracing sequences of events backward from effects to causes (i.e., "deductively") through the model structure, to identify the paths and the order by which combinations of hardware and/or software conditions can propagate through the system to produce system events of interest. This kind of DFM analysis thus shares many of the conceptual features of fault tree analysis. A fault tree is a graphical model that represents the combinations of individual component failures which can lead to the occurrence of an overall system failure (referred to as the top event). In conventional binary fault tree analysis, once a fault tree has been developed, Boolean algebra can be used to reduce the tree to a logically equivalent mathematical form in terms of the tree **minimal cut sets**. A **cut set** is defined as a set of events that, if they all occur, will lead to the top event. A *minimal* cut set is a cut set that does not contain any other cut set as a subset. The removal of any event from a minimal cut set would cause it to no longer be a cut set.

To illustrate the above in formal notation, let X_{top} be an indicator variable for the top event. An indicator variable can take the value of either 0 or 1 (0 if the top event is false, and 1 if it is true). Similarly, let $X_i^{(j)}$ be an indicator variable for the i-th primary event in the j-th minimal cut set. Then the indicator variable for the j-th minimal cut set, MCS_j, is a monomial that can be expressed as the conjunction of the indicator variables of its primary events:

$$MCS_j = \prod_1^n X_i^{(j)} \qquad \text{(Eq. 2.1)}$$

where n is the number of primary events in the j-th minimal cut set. The indicator variable for the top event can then be expressed in disjunctive form as:

$$X_r = 1 - \prod_{j=1}^{m} (1 - MCS_j) \qquad \text{(Eq. 2.2)}$$

A useful property of binary fault trees is that, if the binary variables that appear in them are appropriately defined, the formula that expresses the top event as a function of the basic events, equation (2.2), shows that when a basic event variable changes from the value 0 to the value 1 (i.e., in the customary conventions, from the unfaulted to the faulted state) the top event variable can remain at the value 0 or change from 0 to 1 (if it was at 0 before the basic event change), or remain at the value 1 (if it was already at 1 before the basic event change), but never go from 1 back to 0. A binary logic tree or function that displays this type of behavior is called a "coherent" binary tree or function.

2.3.2 Multi-Valued Logic Trees and Prime Implicants

A fundamental limitation to conventional fault tree analysis is that the above method can only be applied to systems in which the primary events, $X_i^{(j)}$, are binary. Because DFM models represent physical variables (e.g., pressure, temperature, voltage, etc.), binary logic (in which only two states may be used to characterize each variable space) is, in general, not sufficient for an adequate representation of the behavior of the system. DFM models thus employ multi-valued logic (MVL), wherein each variable space may be discretized into an arbitrary number of states. A DFM fault tree, therefore, would contain non-binary primary events (or certain equivalent binary expressions containing groups of mutually exclusive binary primary events, which may be defined ad-hoc to signify whether the assertion that a given multi-valued variable is in any one of its states is true or false). Although a definition of coherent MVL tree can be given, most MVL trees of practical interest (and their equivalent binary expressions), including DFM-derived fault trees, are non-coherent. An intuitive, rather than formal way, of understanding this is by noting that DFM variable states are not ordered in such a way that higher states always indicate "increasingly-faulted" conditions and lower states always indicate "increasingly-nominal" conditions. Thus, as a basic variable changes from a lower to a higher state, the system-state indicator variable of choice for the particular analysis of interest may be going in the opposite direction, i.e., from a higher to a lower state.

The top event of a MVL fault tree can still be expressed in disjunctive form (the form of a disjunction of conjunctions of primary events), but the MVL analogue of the minimal cut sets encountered in binary fault trees are known as **prime implicants** (Henley and Kumamoto, 1992, Ogunbiyi, 1980, Ogunbiyi and Henley, 1981, Garriba, et al., 1985 and Shields, et al., 1994). A prime implicant is any monomial (conjunction of primary events) that is sufficient to cause the top event, but does not contain any shorter conjunction of the same events that is sufficient to cause the top event. The prime implicants of a function are unique and finite (Quine, 1955); however, finding them is a more challenging task than finding binary logic minimal cut sets.

DFM uses decision tables to map the combinatorial states of transfer box inputs to their outputs. Decision tables allow each variable to be represented by any number of states, and they have been applied in fault tree analysis in the past to model component behavior. Given the state of a transfer box output node, the decision table gives the

complete sets of inputs that could have caused it. Since a decision table is, itself, essentially a disjunction of conjunctions of states, it is possible to generate prime implicants from the table (Henley and Kumamoto, 1992). Methods have been developed for obtaining system prime implicants from component decision tables (Henley and Kumamoto, 1992 and Ogunbiyi, 1980). The fundamental approach is to combine the individual component decision tables into a single **critical transition table** (Henley and Kumamoto, 1992 and Kumamoto and Henley, 1979), and performing **Quine's consensus operation** (a series of absorption and merging operations (Quine, 1955, Quine, 1952 and Mott, 1960)) on the rows of the table to reduce it to the complete set of prime implicants.

When referring to prime implicants in the context of a DFM analysis, another important observation is that the presence of the time element in the DFM modeling framework introduces the possibility of prime implicants that would not be possible in ordinary time-invariant logic. In the latter, in fact, a prime implicant of the form:

<variable A = 2 AND variable A = 3>

would not be possible, and, if found in the course of a time-invariant analysis, would have to be eliminated by application of explicit "physical consistency rules". In the application of DFM to time-dependent systems however, if a time-transition has been encountered and the prime implicant is thus "time-stamped" to indicate:

<variable A = 2 @ time t = T1 AND variable A = 3 @ time t = T2>,

then the logical inconsistency no longer exists, and the prime implicant can be considered possible (unless of course it violates a "dynamic consistency rule", which still applies in time-dependent logic; please refer to Section 2.3.3.3). All prime implicants identified in a DFM analysis are conjunctions of primary events with associated time stamps, and they are simply referred to as "timed prime implicants" (TPI's).

DFM, therefore, represents a significant advancement beyond conventional fault tree analysis. In particular, a conventional fault-tree produces cut-sets for one, and only one, binary top event, with no associated time dependent information. The DFM representation is one or two orders of magnitude more powerful, because it produces multi-valued logic and time-dependent prime implicants for a very large number of possible top-events. A DFM top-event can in fact be chosen to be any state among all the possible states of any of the variables, or even any combination of states of separate variables. This is in addition to the fact that, once a DFM system model has been constructed, it can be used repeatedly to investigate many different top events.

The algorithms for the identification of TPI's can produce different types of information, depending on the level of detail included in the original DFM model. More specifically, if the system is only modeled to the module level, so that each software subroutine or module is represented in DFM as a relatively high-level "transfer box" between "global" system-level principal input and output variables, then by definition the top-event prime implicants will only be expressed in terms of the states of such system-level variables (i.e., not in terms of local software variables that are "internal" to each software module). Another option in the type of information sought is whether the DFM backtracking is conducted module by module and component by component, so that, when the process is completed, information equivalent to an actual "timed fault tree" (TFT) is produced as output of the analysis, along with its TPI's. It should be noted that, as discussed further in Section 2.3.3.4, the backtracking process is conducted step by step within the DFM algorithmic procedure and therefore decision-table-format information equivalent in substance to a TFT is produced as an intermediate result on the way to identifying the top-event TPI's. The TFT, when read from the basic events to the top, provides the "explanation" and illustration of how, starting from the basic events contained in the prime implicants at the bottom of the tree, the system evolves through a time-sequence of states which finally lead to the top-event identified at the top of the tree. Please note that the actual progression of cause and effect in the process is exactly in reverse order with respect to the order in which the DFM model analysis unravels the event-sequence, backward in causality and time, from the ultimate system-level effect down to the basic events that are at its origin.

2.3.3 Model Analysis Procedure

2.3.3.1 Timed Fault Tree (TFT) Construction

To obtain a timed fault tree from the system model constructed in Step 1, we first have to identify a particular system condition of interest (desirable or undesirable). This system condition is usually expressed in terms of the state(s) of one or more process variable nodes, which are thus taken to be the fault tree "top event(s)". The DFM model is then analyzed by backtracking, via a computer-implementable analytical procedure, through the network of nodes, edges and transfer boxes and through the time transition network which keeps track of timing effects. This "automated back-tracking procedure" is continued for a few steps back in time, producing along the way the definition of the TFT associated with the particular top-event of interest, to find the possible "cause(s)" of that top event, that is, all the combinations of states of basic system variables which may produce the top event. The order in which the transfer boxes are visited in reverse is dictated by the logical sequence of these boxes in the DFM model, as well as by the sequence of transitions (corresponding to the execution order of software modules or physical events associated with a time delay) in the time-transition network. The information discovered at each step of the backtracking process is represented in the timed fault tree.

To illustrate the timed fault tree construction process, as it may be implemented in a manual execution, consider the analysis of the tank pressure control system shown in Figure 2.1, in which a top event has been defined as a situation in which the pressure in the tank reaches a dangerously high level. This top event is first translated into the state of the process variable node { TP = 5 @ t = 0 } and is shown in Figure 2.3(a). This event is to be expanded by backtracking through the model. From the DFM model in Figure 2.1, TP at t = 0 is calculated from TP at t = -1 and NGF at t = -1 through the transfer function associated with the transition box TT1. The decision table for transition box TT1 is then consulted to identify combinations of TP and NGF at a previous time step that can cause TP = 5 at the current time step. In this case, the two events {TP = 5 @ t = -1} OR {(TP = 4 @ t = -1) AND (NGF = +1 @ t = -1)} are found to be the causes and they are entered into the fault tree as in Figure 2.3(b). Note that a dotted line separates the top event and the events at the second level to indicate the presence of a time transition between the events at the two different levels. Next we backtrack through transfer box T3, in the DFM model in Figure 2.1, to find the combinations of IGF and OGF which can cause NGF = +1. One combination is identified and is shown in Figure 2.3(c) as an AND gate joining the particular states of IGF and OGF. Backtracking through the transfer boxes T1 and T2 will give us the causes for IGF = 1 and OGF = 0 respectively. The backtracking steps are repeated to produce the branch shown in Figure 2.4.

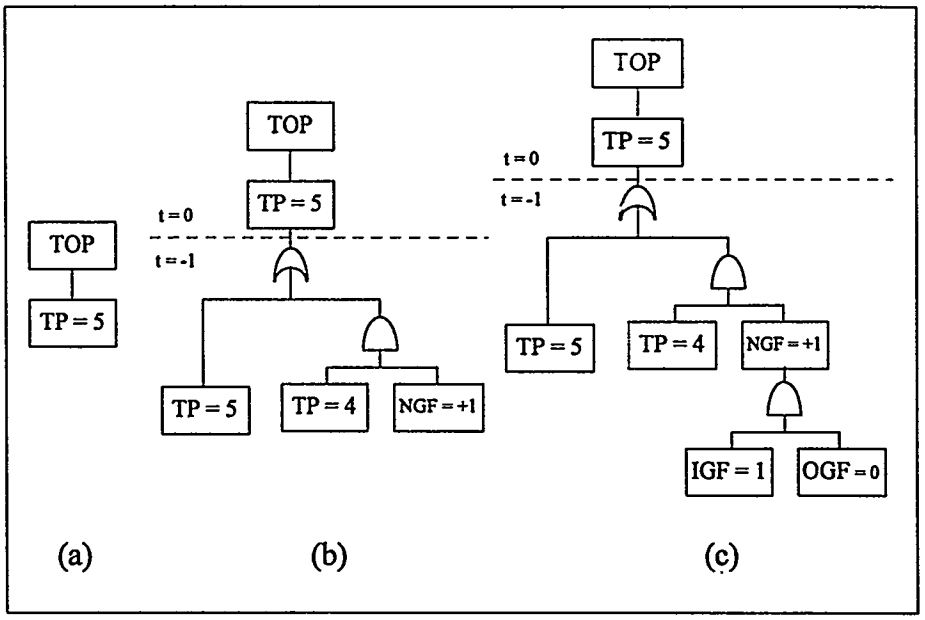


Figure 2.3 : Example of Timed Fault Tree Construction

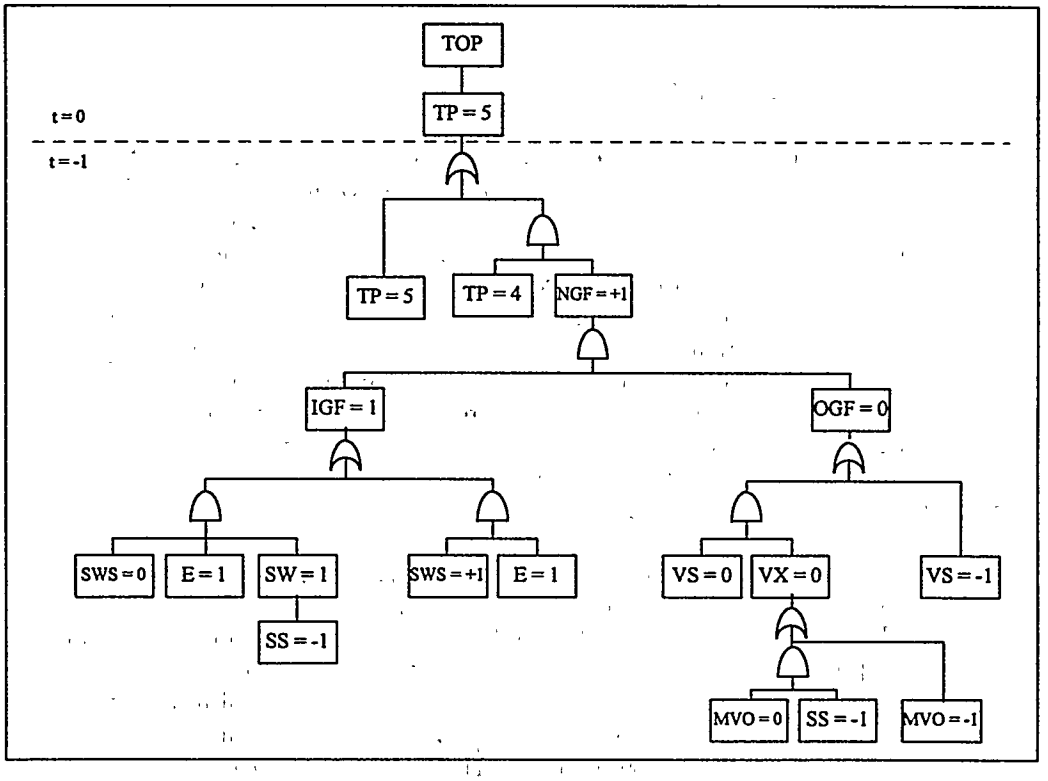


Figure 2.4 : Timed Fault Tree for Very High Tank Pressure

In many digital control systems, there are feedback or feedforward characteristics. This can cause a node to be traced back to itself in the fault tree construction. Consistency rules must be applied when these situations are encountered. Inconsistent branches are then pruned from the timed fault tree. Two major classes of consistency rules have been identified, they are "physical" consistency rules and "dynamic" consistency rules.

2.3.3.2 Physical Consistency Rules

Physical consistency rules are applied to eliminate physically impossible conditions from the timed fault trees. An example of this would be a system parameter taking on two different values at the same time step in the timed fault tree. This class of consistency rule is similar to the consistency rules applied in conventional static fault tree analysis. If the same variable appears twice, but in different states, in the same time step and under the same AND gate, then everything beneath the first AND gate above the second occurrence of the event must be pruned from the tree due to physical inconsistency. This is illustrated in Figure 2.5(a). If pruning this AND gate causes events above to become impossible, then these events must be pruned as well. Such is the situation illustrated in Figure 2.5(b).

2.3.3.3 Dynamic Consistency Rules

Dynamic consistency rules, likewise, are applied to the timed fault trees to eliminate branches which cannot occur due to constraints on the dynamic behavior of the system under consideration. These rules are developed from the analyst's knowledge and assumptions about the system's dynamic behavior. Dynamic consistency rules are expressed in terms of allowable variations of parameter values across different time steps. Table 2.III shows the form of some possible rules of this type.

Rule type 1 can be a result of the analyst's knowledge about the dynamic constraints of the system. For instance, in modeling a drain tank system, the level in the tank cannot increase with time as inventory is constantly being used up and is not being replenished. Rule type 1 can also come from modeling assumptions. For example, if the analyst

assumes the equipment in the tank system can only fail permanently, then a failed valve cannot return to the normal state in a later time step.

Table 2.III : Example Dynamic Consistency Rules

Rule	Description
1	A parameter cannot change in a certain direction between two time steps
2	A parameter cannot change by more than a certain amount between time steps
3	Several parameters must vary in a specific way between two time steps

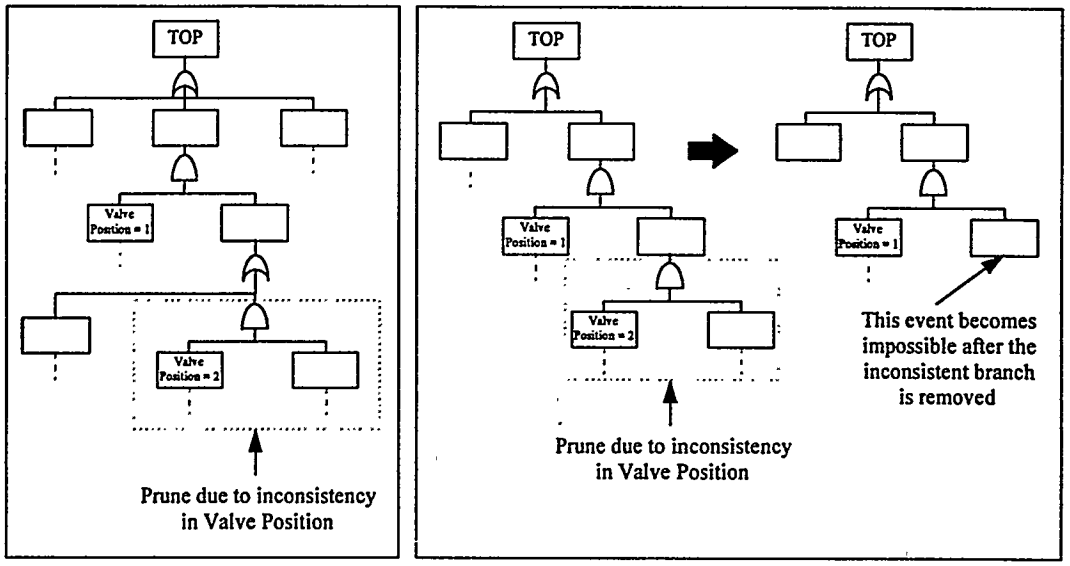


Figure 2.5 : Illustration of Physical Inconsistency

Rule types 2 and 3 come from knowledge of the system. For instance, a type 2 rule can state that the position of the valve cannot vary by more than two states in one time step, as it takes a finite amount of time for the valve to open or close. An example of a type 3 rule can be the constraint that the valve position and flowrate must vary in a proportional manner as required by physical law.

Dynamically inconsistent branches are pruned in a way similar to physically inconsistent branches. If a dynamically inconsistent event occurs in a timed fault tree, the dynamically inconsistent event, including all of the sub-branches connected to it via the first parent AND gate, must be pruned. This is illustrated in Figure 2.6. As with physical consistency rules, further pruning may be necessary if eliminated branches cause other events to become impossible.

2.3.3.4 Timed Prime Implicant (TPI) Identification

As discussed above, TPI's may be identified directly from a system DFM model. In the analytical algorithm actually implemented by the model-analyzer module of the DFM Software Toolset (Section 3.2.2), decision tables encountered during the backtracking process are expanded and joined, one by one, to form a single **critical transition table**, which contains directly all of the system parameter states that are produced along the sequence leading to the top event. As mentioned earlier in Section 2.3.2, the process of expanding and joining the decision tables in the backtracking process is logically equivalent to generating a timed fault tree, except that the events are not presented graphically as a tree structure, but in tabular form as intermediate transition tables. The critical transition table, on the other hand, is logically equivalent to the basic events produced in a timed fault tree. The reader should note that for a multi-state representation, the basic events identified in a timed fault tree (or the rows in

a critical transition table) are the sufficient conditions for the top event. The complete set of unique timed prime implicants (i.e., the necessary and sufficient conditions) are produced by performing**Quine's consensus operation** on the rows of the critical transition table. Quine's consensus operation is a series of absorption and merging procedures which are performed on the table to reduce it to an irredundant form. For example, consider the decision table in Table 2.IV, which is the equivalent of a sum-of-products expression for some function, called TOP. The variables are assumed to be multi-state and their states are:

$A \in [-1, 0, +1]$,
$B \in [N, R, F]$,
$C \in [-2, -1, 0, +1]$,
$D \in [H, N, L]$.

(These variables and the corresponding decision table do not necessarily reflect any particular logic, but are merely intended to illustrate Quine's consensus operation.)

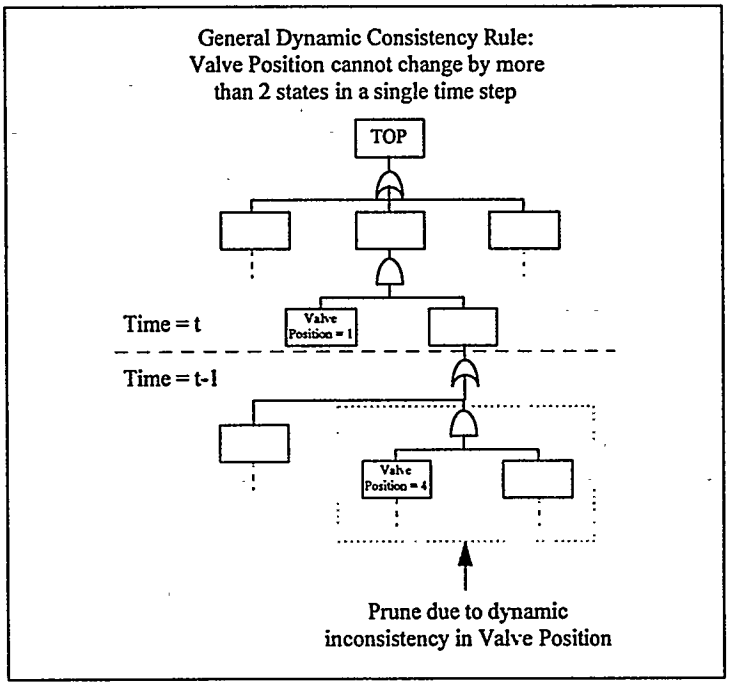


Figure 2.6 : Illustration of Dynamic Inconsistency

In the application of the consensus operation procedure for Table 2.IV, rows 7 and 9 merge with row 5, yielding a "don't care" (which is represented by a "-") in column 1 of row 5 and a new decision table (Table 2.V).

Table 2.IV : Decision Table for Function TOP

ROW	A	B	C	D	TOP
1	-	R	-1	N	1
2	0	-	+1	H	1
3	-	R	0	-	1
4	-	-	-1	L	1
5	0	R	-1	H	1
6	-	N	-2	-	1
7	-1	R	-1	H	1
8	0	R	-2	H	1
9	1	R	-1	H	1
10	0	F	-	H	1

Table 2.V : Decision Table for TOP After Merging Operation

ROW	A	B	C	D	TOP
1	-	R	-1	N	1
2	0	-	+1	H	1
3	-	R	0	-	1
4	-	-	-1	L	1
5	-	R	-1	H	1
6	-	N	-2	-	1
7	0	R	-2	H	1
8	0	F	-	H	1

Rows 6-8 of Table 2.V can then undergo a reduction operation, yielding a "don't care" in column 2 of row 7. Rows 1, 4 and 5 of the table also undergo a reduction-merging operation, yielding Table 2.VI. Table 2.VI contains only irredundant terms since none of the simplifying operations can be applied to any term in the table.

Table 2.VI : Irredundant Form of Decision Table for Function TOP

ROW	A	B	C	D	TOP
1	-	R	-1	-	1
2	0	-	+1	H	1
3	-	R	0	-	1
4	-	-	-1	L	1
5	-	N	-2	-	1
6	0	-	-2	H	1
7	0	F	-	H	1

Rows 1-3 and 6 of Table 2.VI yield a consensus term which is given in row 8 of Table 2.VII. Table 2.VII contains all of the prime implicants of the function since no new consensus terms can be generated from it and none of its terms can be simplified any further.

Table 2.VII : Decision Table for Function TOP After Consensus

ROW	A	B	C	D	TOP
1	-	R	-1	-	1
2	0	-	+1	H	1
3	-	R	0	-	1
4	-	-	-1	L	1
5	-	N	-2	-	1
6	0	-	-2	H	1
7	0	F	-	H	1
8	0	R	-	H	1

Of course, physical and dynamic consistency rules must still be applied during the construction of the critical transition table. The only difference is that, instead of applying them to individual events in the timed fault tree, they are applied to entire rows in the critical transition table.

2.4 Framework for DFM Analysis-Driven Testing

2.4.1 Overview of Testing

Testing is traditionally one of the most important activities carried out to assure that a given design is, in its actual implementation, complying with certain assigned constraints and specifications, be they in the realm of "peak

performance", safety, or reliability. Testing assures the quality of the final product, validates that the product will perform as it is designed to do, and provides reasonable assurance that the product will not threaten life or endanger the user. For systems such as spaceships, aircraft, and nuclear reactors, where failures may threaten life, testing costs may account for as much as 80% of the total manufacturing cost (Beizer, 1990). This is also true for software systems, where the dominating cost is often not the cost of design and programming, but the costs associated with logic and implementation errors: the cost of detecting them, the cost of correcting them, the cost of designing tests that discover them, and the cost of running those tests (Beizer, 1990).

In traditional "black box" testing, combinations of inputs for the control system software are chosen and the software is executed to produce outputs. These outputs are verified for conformance to specified behavior. However, this kind of testing is not the "silver bullet" in identifying errors in control system software, since it is limited by several factors, e.g.:

1) it is practically impossible (except for the simpler situations) to identify input sampling patterns which provide coverage and assurance for all of the execution paths of the control system software;

2) "hidden" errors in the software which only manifest themselves in conjunction with some other system conditions are very hard to identify.

A major difficulty in functional testing is the selection of inputs. A large set of inputs reflecting normal and exceptional circumstances are subjected to testing. Exceptional inputs cannot be overlooked as they often are the inputs that trigger undetected faults in software programs, resulting in system failures. However, there is no guideline as to how these exceptional inputs can be sampled. Selecting them is largely based on judgment. The coverage of all possible inputs is both impossible and impractical. It is very likely that some exceptional inputs are overlooked in testing. When these inputs arise in conjunction with some other unpredictable system conditions, serious consequences can result.

Other than the problem associated with the coverage of inputs in testing, there are also difficulties in assuring the execution paths of the software. Since the implementation details are not considered in functional testing, it is possible that expected outputs can be produced via unexpected paths in the software. Even though the behavior of the software seems correct to the testing team, there exists fundamental errors in the software which cause the selection of the incorrect path. In addition, the inability to verify the execution paths makes it difficult to trace the source of errors found. Functional testing indicates the existence of bugs in the software when unexpected outputs are produced, but it does not tell us how to find them. Locating the source of errors is not an easy task. Different bugs can have the same manifestations, and one particular bug can have many symptoms. A great number of small tests must be run in order to locate the bug.

In addition to being unable to verify the execution path of the embedded system software, functional testing is also ineffective in identifying "hidden" bugs which only manifest themselves together with some other system conditions (i.e., conditions in the hardware part of the system that interfaces with the software). It is impossible and impractical to test the software under all conditions. But scenarios can arise where a combination of low probability system conditions causes the "hidden" bugs to produce failures.

2.4.2 DFM Analysis Based Testing

Performing a preliminary DFM analysis on the system before testing could drive functional testing to focus on a limited domain of inputs. The objective was to eliminate the need to blindly select inputs to test the software. In a nutshell, each system state (desirable or undesirable) being investigated via DFM (usually referred to as a top event), is resolved into the combinations of primary events which can cause it to occur. Each event is described by a range encompassing the value of a particular variable, either in the physical system or in the software, and the time interval in which the variable assumes that value. In addition, DFM identifies the paths and the time sequences by which the top event is produced by these software and physical system conditions. As discussed above in Section 2.3, the principal result of a DFM analysis, as defined earlier in Section 2.3.3, is the identification of "timed prime

implicants" (TPI's). A timed prime implicant is a minimum combination of events which is both necessary and sufficient to cause the top event.

Several DFM analysis-based software testing strategies have been identified. The two basic modes in which DFM can be used to support the development of testing strategies are:

1. Top-Event Decomposition Mode (TED-Mode)
2. Timed Fault Tree Derivation Mode (TFTD-Mode)

2.4.2.1 Top Event Decomposition Mode (TED-Mode)

TED-Mode represents a high level test of the control system. In this mode, individual software modules are treated as black boxes, and a detailed representation of the input-output relationships that the modules implement is not sought. This mode may be used, for example, when analyzing a system design at a stage when functional requirements of the software modules have been identified, but have not yet been translated into code. In such a case, DFM would provide a re-definition, or "decomposition" of a system top event (i.e., the system failure mode to be avoided) in terms of "intermediate-level implicants" (ILI's) defined as combinations of states of the output variables of those software modules. Testing can then be broken down to the level of making sure that these combinations of output variable states cannot be produced by any allowed combination of input variable states. Note that, in general, the ILI's may also contain the states of certain hardware parameters or components. These hardware states would then become boundary conditions for the functional testing of the software.

2.4.2.2 Time Fault Tree Derivation Model (TFTD-Mode)

In the Timed Fault Tree Derivation Mode, a detailed DFM analysis produces not only timed prime implicants, but also the timed fault tree which describes the evolution of the system from these conditions towards the top event. Testing in this case can be executed to verify that the temporal behavior (evolution) of the system corresponds to what is expected by the analyst per the DFM model. Because DFM can be employed to produce TPI's for success as well as failure top events, a TFTD-Mode DFM analysis, followed by testing, can be used to verify that the system executes according to the expected and desired execution paths. Beyond the testing purposes, the utility of the TFTD-Mode is that it provides the user with an "explanation" of how the TPI's produce/give rise to the top event.

In addition, the form of the timed prime implicants, whether these consist of software "single event prime implicants" (SEPIs) or software/hardware "multiple event prime implicants" (MEPIs), can dictate how testing can best be carried out. Software conditions which are identified as SEPIs point to certain individual software states that are active without the need for outside "triggers". These prime implicants can contain normal conditions external to the software, such as those which are understood and are included within the "design envelope" represented by the software requirements and specifications. For example, the prime implicant identified in the analysis of the Demonstration Test Case (which is presented in Section 5.3.2) is a conjunction of software input conditions (the steam generator level, the steam flow and a variable in the memory) and an external condition (the steam generator pressure). Since the steam generator pressure is within the range encountered under normal operating conditions, the prime implicant is hence classified as a SEPI. MEPIs (i.e., multi-event prime implicants consisting of both software conditions and one or more physical system conditions) indicate system states which can only be caused by the combination of software conditions and unexpected or unwanted system conditions outside of the software, i.e., "external-world" conditions not included in the original "design envelope" of the software. Note that these conditions may exist concurrently, or they may take place at different times. This is a direct result of the fact that DFM models explicitly depict the progression, due to physical cause-and-effect, of the system state as it evolves in time towards failure. Thus, a multi-event prime implicant is essentially a unique combination of events which gives rise to a system evolution sequence leading to the top event. An example of a MEPI is the prime implicant identified in the analysis of the Interim Test Case (which is presented in Section 4.4.2). In addition to containing a software input condition (the tank level) and other normal external conditions (the sensors, the stop valves and the control valves being normal), this prime implicant also requires the failure of the check valve, and hence is classified as a MEPI.

A basic process of critical software testing based on a TFTD-Mode DFM analysis can be defined as follows:

a) If the DFM analysis identifies any single software implicants for a system state, this would therefore indicate the existence of fundamental bugs or logic errors in the software. The only remedy would thus be to correct the software so that these software conditions are made inactive and unreachable. The corrected version of the software would thus have to be reanalyzed to ensure that the corrections do not bring about new errors.

b) If the DFM analysis identifies a multi-event prime implicant, i.e., one or more of the external-world conditions associated with the system state history is off-nominal, then one of two subcases may arise:

b1) if the combination of external-world conditions is highly probable, the error-causing software condition must be removed and the corrected version must be reanalyzed as before.

b2) if the conjunction of external-world and software conditions is not believed to be easily achievable or likely, functional testing of the software can be performed in the "neighborhood" of these "unlikely" conditions to determine the actual margin of safety in the system. This means that the inputs selected for functional testing are concentrated in the ranges specified by the events which form the prime implicant, and the boundary conditions are constrained to correspond to that implicants state history, resulting in a much smaller domain of test inputs from which to sample. Testing in the neighborhood of the prime implicant conditions is stressed because a DFM model and the resulting prime implicants that it produces can be expected to be only a finite approximation of the actual system. The purpose of "neighborhood testing" would be to confirm the existence of identified faults and to ensure that "neighboring states" of the TPI variable states and conditions will not result, themselves, in system failures.

2.5 Example of DFM Modeling and Analysis

In this section, a simple control system is used to illustrate how DFM can be applied to identify the failure modes of the system. The system selected is a slight modification of the pressure tank example used in Chapter VIII of NUREG-0492, "Fault Tree Handbook". The sub-sections that follow will show the reader how a system definition is translated into a DFM model and how top events can be analyzed to identify the basic failure modes. The reader should note that this section intends to illustrate the basic concepts of DFM in semi-tutorial fashion, and that DFM is not limited to analyzing simple systems such as the one used here for this purpose. Chapters 4 and 5 will show how DFM can be applied to analyze more complicated systems with feedback control loops and software modules defined in line-code detail.

2.5.1 System Description

The example system consists of an inexhaustible gas source, a pressure tank, a pressure sensor, a pressure controller, a pump, an AC power source for operating the pump, an electric switch and an outlet valve. The schematic of this system is shown in Figure 2.7, and the controller function is to maintain the pressure of the tank at a certain level. The electric switch and the valve are controlled by the pressure controller, but the command from the controller to the valve can be overridden by a human operator's command. The operator's command is modeled as the node MVO in Section 2.5.2, and the discretization of the node is shown in Table 2.XII. The controller monitors the pressure in the tank via signals from the pressure sensor. If the reading is too low, the controller will close the outlet valve and close the electric switch. Closing the electric switch will activate the pump to replenish the gas inventory in the tank. On the other hand, if the pressure reading is too high, the controller will open the electric switch, thus disabling the pump, and open the outlet valve to vent the gas in the tank. Table 2.VIII summarizes the control actions that may be undertaken.

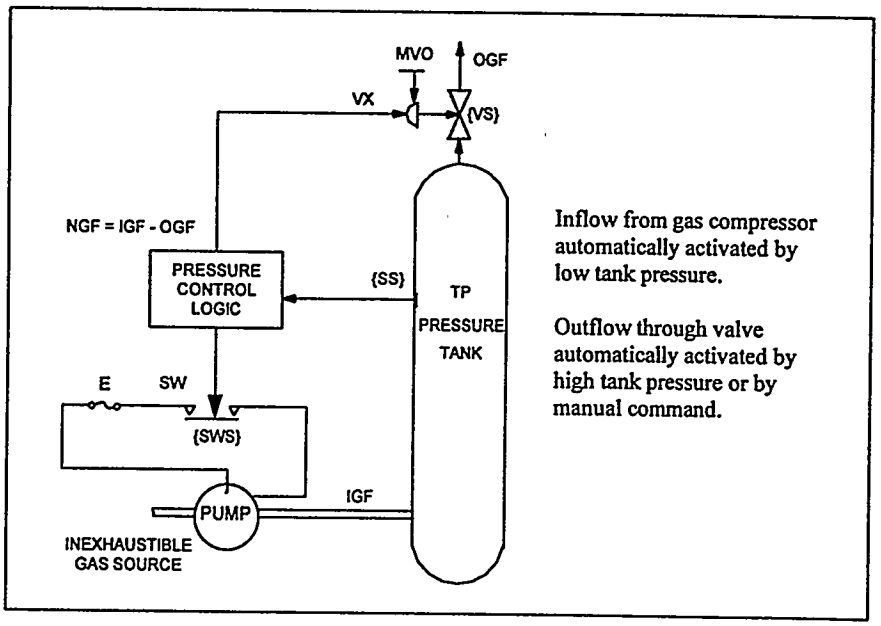


Figure 2.7 : Schematic of the Pressure Control System

Table 2.VIII : Summary of Control Commands

Pressure Reading	Outlet Valve Command	Electric Switch Command
Too Low	Close	Close
Normal	Close	Open
Too High	Open	Open

2.5.2 Example of DFM Model Construction

The first step in applying DFM is to construct a model to capture the behavior of the system. To accomplish this, the components to be modeled are first chosen. In this case, all the components will be included in the DFM model. Next, parameters that capture the attributes of these components are identified and they become the process variable nodes (Table 2.IX). These process variable nodes are linked together by causality edges through transfer boxes and transition boxes to model the cause-and-effect relationships among the parameters (Figure 2.8). For example, transfer box T1 represents the pump in which the AC power and the switch position will yield the gas inflow into the tank. On the other hand, transfer boxes T4 and T5 represent the pressure controller where the pressure reading triggers the outlet valve position and the electric switch position. The reader should note that the pressure tank is represented by a transition box (TT1) instead of a transfer box because pressure variation is dynamic. The current pressure depends on the net gas flow into the tank as well as the pressure a split second before.

Table 2.IX : Process Variable Nodes

Parameter	Meaning
E	AC Power
IGF	Gas flow into the tank through the pump
MVO	Manual override command to the outlet valve
NGF	Net gas flow into the tank
OGF	Gas flow out of the tank through the outlet valve
SW	Electric switch position
TP	Tank pressure
VX	Outlet valve position

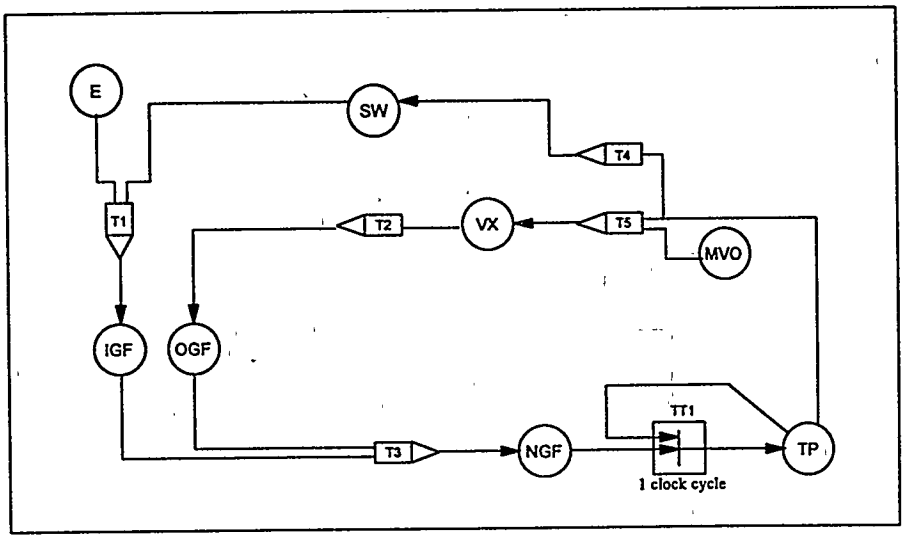
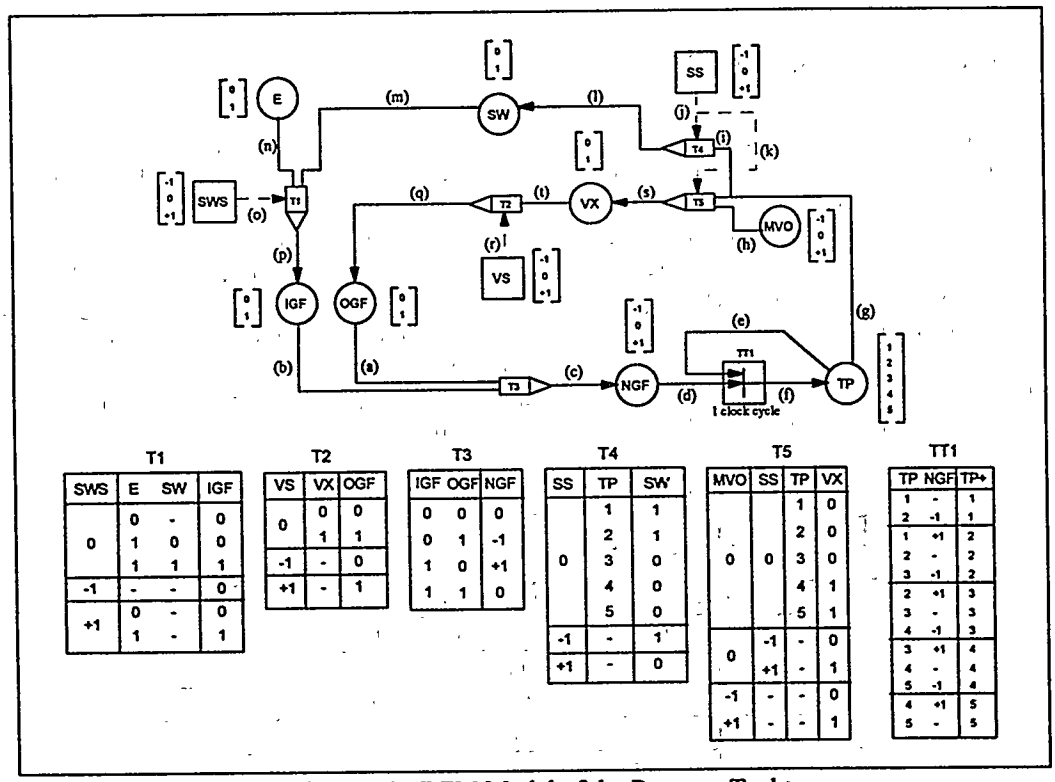


Figure 2.8 : Integrated Causality and Time Transition Network

After the construction of the integrated causality and time transition network, discontinuous behavior such as component failures are identified and represented in the model as condition nodes and condition edges (Figure 2.9). In this figure, SS represents the state of the pressure sensor and it has an impact on the controller action as the pressure control command is based on the pressure reading, not on the actual tank pressure. Similarly, SWS, the state of the electric switch can affect the gas flow into the tank, as a stuck open switch will prevent the pump from working even though power is available. The process variable nodes and the condition nodes are then discretized into finite number of states, and they are shown in Tables 2.X to 2.XIX. These discretization schemes are also shown in Figure 2.9 for easy reference.



T1

SWS	E	SW	IGF
0	0	-	0
	1	0	0
	1	1	1
-1	-	-	0
+1	0	-	0
	1	-	1

T2

VS	VX	OGF
0	0	0
	1	1
-1	-	0
+1	-	1

T3

IGF	OGF	NGF
0	0	0
0	1	-1
1	0	+1
1	1	0

T4

SS	TP	SW
0	1	1
	2	1
	3	0
	4	0
	5	0
-1	-	1
+1	-	0

T5

MVO	SS	TP	VX
0	0	1	0
		2	0
		3	0
		4	1
		5	1
	-1	-	0
	+1	-	1
-1	-	-	0
+1	-	-	1

TT1

TP	NGF	TP+
1	-	1
2	-1	1
1	+1	2
2	-	2
3	-1	2
2	+1	3
3	-	3
4	-1	3
3	+1	4
4	-	4
5	-1	4
4	+1	5
5	-	5

Figure 2.9 : DFM Model of the Pressure Tank

Table 2.X : Discretization of E

States	Meaning
0	AC power is unavailable
1	AC power is available

Table 2.XI : Discretization of IGF

States	Meaning
0	No gas flow into the tank
1	Gas flows into the tank

Table 2.XII : Discretization of MVO

States	Meaning
-1	Operator commands valve to close
0	Operator does not override controller command
+1	Operator commands valve to open

Table 2.XIII : Discretization of NGF

States	Meaning
-1	Net gas flow out of the tank
0	No net gas flow
+1	Net gas flow into the tank

.Table 2.XIV : Discretization of OGF

States	Meaning
0	No gas outflow through the valve
1	Gas flows out through the valve

Table 2.XV : Discretization of SS

States	Meaning
-1	Pressure sensor stuck low
0	Pressure sensor is normal
+1	Pressure sensor stuck high

Table 2.XVI : Discretization of SW

States	Meaning
0	Electric switch is opened
1	Electric switch is closed

Table 2.XVII : Discretization of SWS

States	Meaning
-1	Electric switch failed opened
0	Electric switch is good
+1	Electric switch failed closed

Decision tables are then constructed from the knowledge of the behavior of this system. These tables are shown in the model in Figure 2.9. In the construction of the decision tables, it was assumed that gas inflow is possible when the tank pressure is high and gas outflow is possible when the tank pressure is low. This assumption is reflected in the decision table for transition box TT1. At this point, the DFM model is completed and can be analyzed to identify failure modes.

Table 2.XVIII : Discretization of TP

States	Meaning
1	Tank pressure is very low
2	Tank pressure is low
3	Tank pressure is normal
4	Tank pressure is high
5	Tank pressure is very high

Table 2.XIX : Discretization of VX

States	Meaning
0	Outlet valve is closed
1	Outlet valve is opened

2.5.3 Example of DFM Model Analysis

The DFM model constructed for the pressure control system can be analyzed to generate timed fault trees and timed prime implicants. The timed prime implicants are the necessary and sufficient conditions for specific failure events, whereas the timed fault trees show how the necessary and sufficient conditions can cause the failure events. For example, to analyze how the pressure in the tank becomes dangerously high and causes the tank to rupture, we first define the top event in terms of the state of the process variable node TP (TP = 5 @ t = 0). The top event is then backtracked through the DFM model (via the steps discussed in Section 2.3.3) for one time step to generate the timed fault tree shown in Figure 2.10. The timed prime implicants for this corresponding timed fault tree are listed in Table 2.XX. Take for example prime implicant #1, the failed sensor gives a low pressure reading which causes the controller to command the electric switch to close and the outlet valve to close. The absence of manual override command and the fact that the valve is normal will lead to the closure of the outlet valve. At the same time, with AC power being available and the electric switch being operational, the closing of the switch will pump more gas into the tank. With the prior tank pressure being high, the net inflow of gas into the tank will cause the pressure to become dangerously high. The reader should note that even though a single failure is characterized by prime implicant #1, the fact that the other key components are normal has to be expressed explicitly in this prime implicants. This is a feature of multi-state representations of systems. If power had been unavailable, the tank pressure would not have reach the dangerously high level as the pump could not operate, and thus there would be no net gas flow into or out of the pressure tank. It is also important to point out that the assumptions made in constructing the DFM model had a direct impact on the prime implicants identified. In this particular case, if the assumption that gas inflow is possible at high tank pressure were to be removed, none of the prime implicants shown in Table 2.XX would be identified.

As the DFM model of the pressure control system implicitly contains most, if not all, conceivable behaviors believed to be exhibited by the system, one single model can be analyzed for as many top events as the analyst desires. The same model analyzed for the top event TP = 1 @ t = 0 (Tank pressure is low at time 0) produced the timed fault tree shown in Figure 2.11 and the timed prime implicants listed in Table 2.XXI.

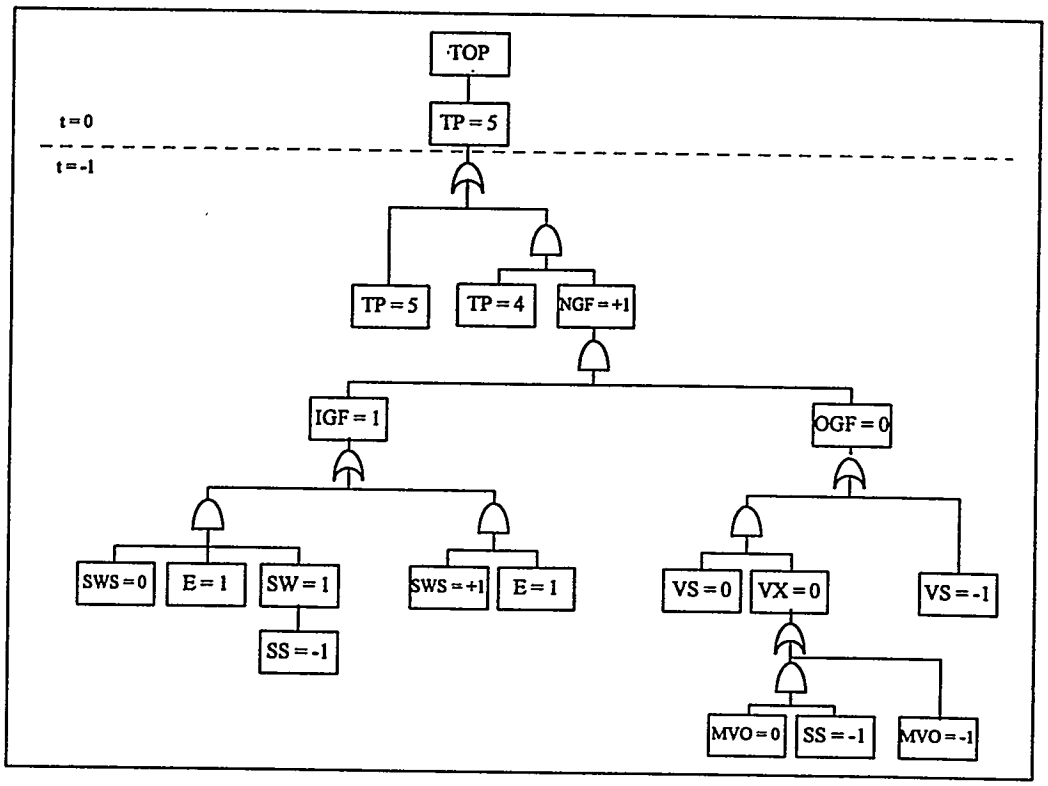


Figure 2.10 : Timed Fault Tree for the Top Event TP = 5 @ t = 0

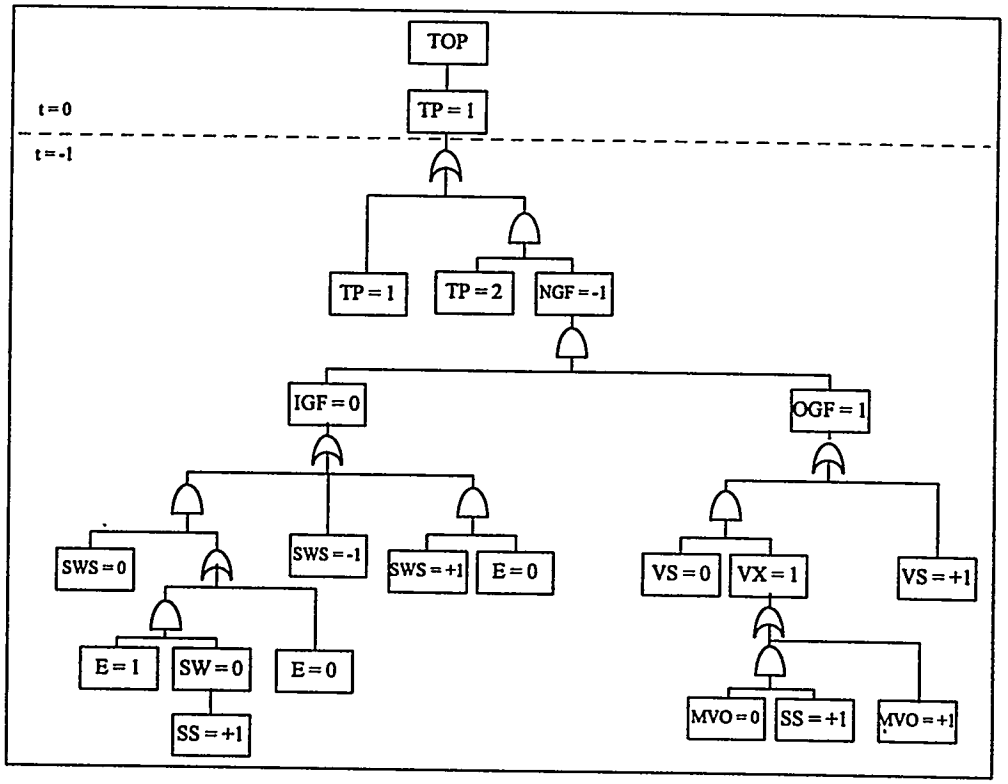


Figure 2.11 : Timed Fault Tree for the Top Event TP = 1 @ t = 0

Table 2.XX : Prime Implicants for the Top Event TP = 5 @ t = 0

Number	Prime Implicant		
1	Electric switch was normal	@ t = -1	AND
	*Power was available	@ t = -1	AND
	Outlet valve was normal	@ t = -1	AND
	*No manual valve command	@ t = -1	AND
	*Sensor failed low	@ t = -1	AND
	*Tank pressure was high	@ t = -1	
2	Electric switch was normal	@ t = -1	AND
	*Power was available	@ t = -1	AND
	Outlet valve was normal	@ t = -1	AND
	*Valve closed manually	@ t = -1	AND
	*Sensor failed low	@ t = -1	AND
	*Tank pressure was high	@ t = -1	
3	Electric switch was normal	@ t = -1	AND
	*Power was available	@ t = -1	AND
	*Outlet valve failed closed	@ t = -1	AND
	*Sensor failed low	@ t = -1	AND
	*Tank pressure was high	@ t = -1	
4	*Electric switch failed closed	@ t = -1	AND
	*Power was available	@ t = -1	AND
	Outlet valve was normal	@ t = -1	AND
	*No Manual valve command	@ t = -1	AND
	*Sensor failed low	@ t = -1	AND
	*Tank pressure was high	@ t = -1	
5	*Electric switch failed closed	@ t = -1	AND
	*Power was available	@ t = -1	AND
	Outlet valve was normal	@ t = -1	AND
	*Valve closed manually	@ t = -1	AND
	*Tank pressure was high	@ t = -1	
6	*Electric switch failed closed	@ t = -1	AND
	*Power was available	@ t = -1	AND
	*Outlet valve failed closed	@ t = -1	AND
	*Tank pressure was high	@ t = -1	
7	*Tank pressure was very high	@ t = -1	

It should be noted that the prime implicants shown in Table 2.XX and Table 2.XXI list both "faulted" and "normal" states of components and parameters. A "reduced" form of prime implicant definition can also be obtained which does not list states that can be always considered as unconditionally faulted. States that would continue to be listed in the reduced prime implicant form are marked with an asterisk in the two tables, whereas the states that are not marked would not be listed. More complete definition and discussion of this topic can be found in Chapter 6 (Section 6.3.2.3), and examples of "reduced prime implicants" are provided there for the test case analyses that are presented in Chapter 4 and Chapter 5.

Table 2.XXI : Prime Implicants for the Top Event TP = 1 @ t = 0

Number	Prime Implicant		
1	*Power was unavailable	@ t = -1	AND
	Outlet valve was normal	@ t = -1	AND
	*No manual valve command	@ t = -1	AND
	*Sensor failed high	@ t = -1	AND
	*Tank pressure was low	@ t = -1	
2	*Power was unavailable	@ t = -1	AND
	Outlet valve was normal	@ t = -1	AND
	*Valve opened manually	@ t = -1	AND
	*Tank pressure was low	@ t = -1	
3	*Power was unavailable	@ t = -1	AND
	*Outlet valve failed opened	@ t = -1	AND
	*Tank pressure was low	@ t = -1	
4	Electric switch was normal	@ t = -1	AND
	Outlet valve was normal	@ t = -1	AND
	*No Manual valve command	@ t = -1	AND
	*Sensor failed high	@ t = -1	AND
	*Tank pressure was low	@ t = -1	
5	Electric switch was normal	@ t = -1	AND
	Outlet valve was normal	@ t = -1	AND
	*Valve opened manually	@ t = -1	AND
	*Sensor failed high	@ t = -1	AND
	*Tank pressure was low	@ t = -1	
6	Electric switch was normal	@ t = -1	AND
	*Outlet valve failed opened	@ t = -1	AND
	*Sensor failed high	@ t = -1	AND
	*Tank pressure was low	@ t = -1	
7	*Electric switch failed opened	@ t = -1	AND
	Outlet valve was normal	@ t = -1	AND
	*No manual valve command	@ t = -1	AND
	*Sensor failed high	@ t = -1	AND
	*Tank pressure was low	@ t = -1	
8	*Electric switch failed opened	@ t = -1	AND
	Outlet valve was normal	@ t = -1	AND
	*Valve opened manually	@ t = -1	AND
	*Tank pressure was low	@ t = -1	
9	*Electric switch failed opened	@ t = -1	AND
	*Outlet valve failed opened	@ t = -1	AND
	*Tank pressure was low	@ t = -1	
10	*Tank pressure was low-low	@ t = -1	

3 DFM SOFTWARE TOOLSET

This chapter provides a discussion of the DFM Software Toolset, which is an integrated set of software tools developed in Phase II of this research project for implementing the model construction and analysis procedures of DFM. The topics covered include the development of the various modules (Section 3.1), description of the functionality and the user-interfaces of all the modules within this software toolset (Section 3.2), and the input and output relating to the analysis of the example discussed in Section 2.5.

The DFM Software Toolset is developed as a Microsoft Windows™ application which can run on Intel processor-based PCs. The goal of this software application is to assist the analysts to construct DFM models and to analyze the DFM models to generate prime implicants. This tackles the problem where manual construction of timed fault tree and generation of prime implicants, without the help of automated tools, is difficult for simple systems, and practically impossible for complex systems.

3.1 Development of the DFM Software Toolset

The DFM Software Toolset is an integration of two principal modules: the Model Editor and the Model Analyzer (Figure 3.1). The Model Editor aids analysts in the construction of DFM models for any given system of interest. It features a relational database, supporting a library of the pre-defined DFM modeling elements, which stores the information relating to the structure and the display attributes of the DFM model created with the Model Editor. The Model Analyzer analyzes DFM system models and obtains the prime implicants for any system state of interest defined by the user. The development and integration of these two principal modules are discussed below.

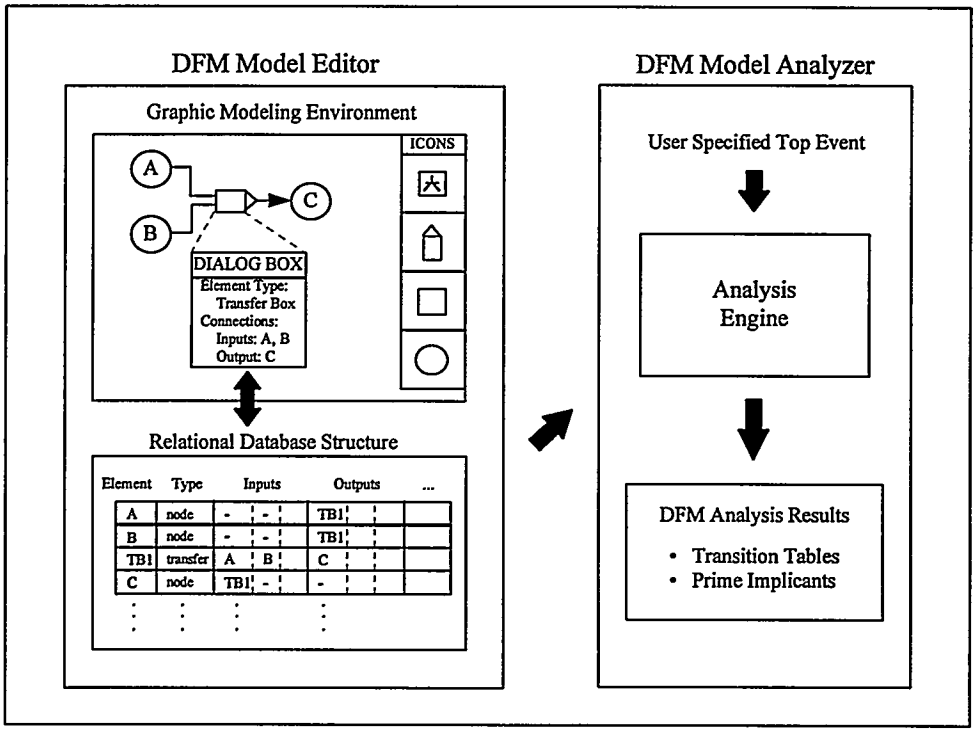


Figure 3.1 : Architecture of the DFM Software Toolset

3.1.1 Development of the Model Editor

The DFM Model Editor is a Windows-based graphical model building tool with which the user can create and edit DFM models. The user interface resources (windows, menus, dialog boxes, dialog controls, etc.) were created using

the GUILD™ GUI-development tool. GUILD is a resource editor that allows the user to visually design Windows resources.

The rest of the Model Editor was implemented in C. There are basically four types of C functions that provide the functionality of the Model Editor. The "laydown" functions deal with drawing, moving and otherwise visually manipulating the graphical objects on the Model Editor laydown page. The "callback" functions are the event handlers that process the mouse messages (button clicks and movements) on the laydown page. The "dialog" functions deal with the transmission of data from the data structures in memory to the various dialog box displays (list boxes, edit fields, etc.), and back again. The "load/save" functions handle all of the tasks related to disk input and output.

3.1.2 Development of the Model Analyzer

The DFM Model Analyzer is a Windows-based tool with which the user can analyze models created with the DFM Model Editor. The user interface resources, which include all the dialog boxes for defining top events, displaying prime implicants and intermediate transition tables, were created using the GUILD™ GUI-development tool. The analysis engine, which contains all the modules for carrying out different operations on the intermediate transition tables, was implemented in C. Figure 3.2 shows the algorithm which is implemented in the analysis engine. In addition, the modules for interfacing with the database and the dialog boxes were also written in C.

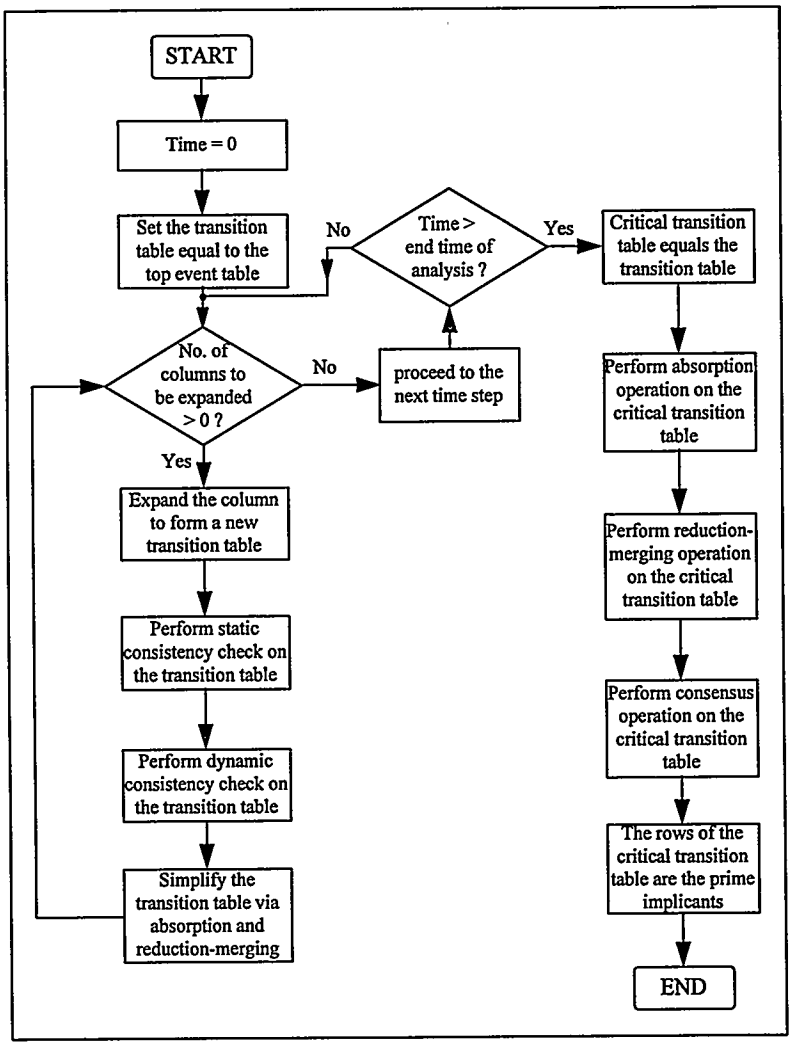


Figure 3.2 : Algorithm used in the Analysis Engine

3.2 Functionality of the DFM Software Toolset

The functionality and the user interfaces for the Model Editor and the Model Analyzer are discussed below.

3.2.1 Functionality of the Model Editor

The Model Editor facilitates the construction of DFM models by the analyst and converts the graphic representation of these models into a set of data that can be stored in a database, and later used by the Model Analyzer. The Model Editor consists of a graphic model building environment in which the user creates DFM models and a database structure which stores information about the model created.

3.2.1.1 Graphic Model Building Environment

The graphic model building environment assists the user to construct a DFM model. It provides a toolbox of graphic icons representing DFM modeling elements with which the user can build a DFM model. A screen capture of this graphic modeling environment is shown in Figure 3.3. This graphic model building environment is developed using a combination of the "C" programming language and the GUILD™ graphic user interface (GUI) development tool. The user defines the structure of a DFM model by picking the modeling elements from the icon menu and placing them on the screen. Connections are made by picking the source and the target as well as any intermediate points. Figure 3.4 shows the DFM model of the tank pressure control system (discussed in Section 2.5) created using the Model Editor.

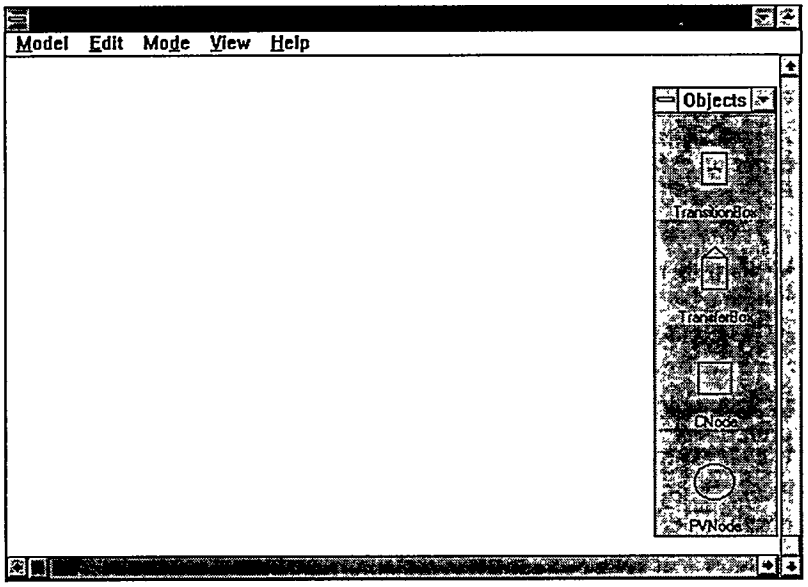


Figure 3.3 : Screen Capture of the Model Editor Graphic Model Building Environment

Associated with each modeling element is a dialog box in which the user can define the attributes of that element. The user accesses the dialog box by double clicking the mouse on top of the graphic icon. Figure 3.5 shows the dialog box accessed by double clicking on the node TP where the properties of the tank pressure process variable node can be defined. The properties that need to be defined in that dialog box are summarized and explained in Table 3.I. Similarly, Figures 3.6 and 3.7 show the dialog boxes for defining the properties of a transfer box and a transition box respectively, while Tables 3.II and 3.III provide explanations for the properties thus defined. These dialog boxes are accessed by double clicking on the transfer box T2 and the transition box TT1 respectively.

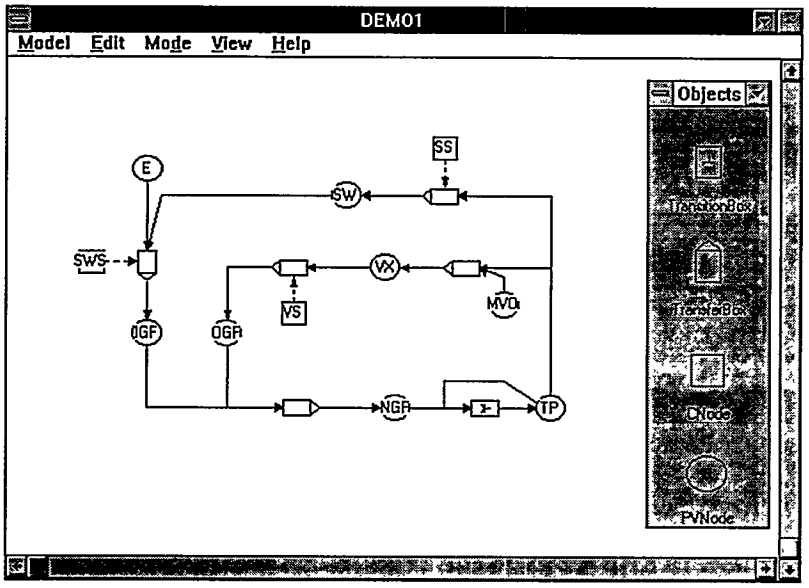


Figure 3.4 : DFM Model of the Pressure Control System Created with the Model Editor

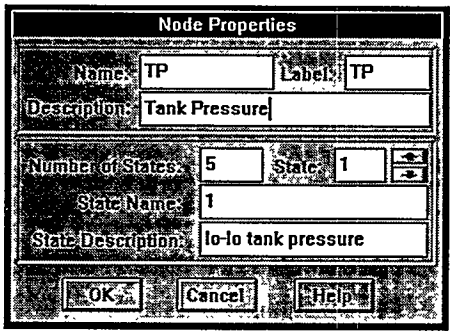


Figure 3.5 : Dialog Box for Defining Properties of a Node

Table 3.I : Properties of a Node

Properties	Meaning
Name	The name of the parameter that the node represents
Label	The label to be seen on top of the node in the graphic modeling environment
Description	A brief description of what the node represents
Number of States	The number of states into which the node is discretized
State	The arrows allow the user to select any possible state
State Name	The name of the states
State Description	A brief description of what the state means
Orientation	The radio buttons toggle the orientation of the transfer box on the screen

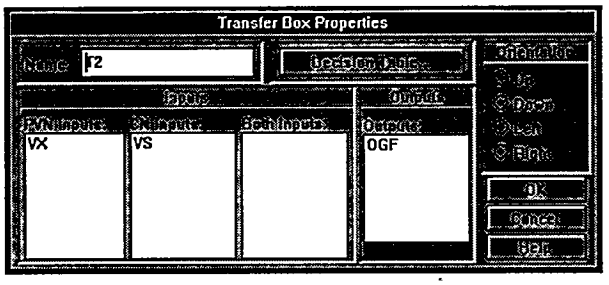


Figure 3.6 : Dialog Box for Defining Properties of a Transfer Box

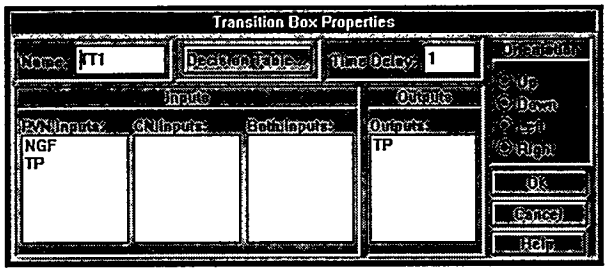


Figure 3.7 : Dialog Box for Defining Properties of a Transition Box

Table 3.II : Properties of a Transfer Box

Properties	Meaning
Name	The name for identifying the transfer box
Decision Table	A button for accessing another dialog box to define the decision table for this transfer box
PVN Inputs	List the Process Variable Nodes which are inputs to this transfer box[1] through causality edges
CN Inputs	List the Condition Nodes which are inputs to this transfer box[1] through condition edges[1]
Both Inputs	List the nodes which connect to this transfer box through both causality edges and condition edges[1]
Outputs	Lists the nodes which are outputs of this transfer box
Orientation	The radio buttons toggle the orientation of the transfer box on the screen

[1] These lists are created automatically once the connections are defined.

3.2.1.2 Database Structure

The database structure is created with a built-in feature of the GUILD™ GUI development tool. The database is in the form of a "B-trieve" database structure, and is directly accessible by any C code routine. The relational database structure consists of two major classes of data. One class characterizes the graphic attributes of the model, the other class characterizes the structure attributes of the model. The graphic attributes define the positions, the sizes and the orientations of all the nodes and boxes shown on the screen. In addition, these attributes determine how the connections are to be drawn, i.e., the color, the starting point, the end point and any intermediate points. The graphic attributes allow the Model Editor to "remember" how to regenerate the picture of the model. On the other hand, the structure attributes are essential if the model is to be analyzed. They define the structure of the DFM model so that the Model Analyzer can backtrack the model correctly through all the boxes and time transitions.

Table 3.III : Properties of a Transition Box

Properties	Meaning
Name	The name for identifying the transition box
Decision Table	A button for accessing another dialog box to define the decision table for this transition box
Time Delay	The time delay associated with this transition box
PVN Inputs	List the Process Variable Nodes which are inputs to this transition box[1] through causality edges
CN Inputs	List the Condition Nodes which are inputs to this transition box[1] through condition edges[1]
Both Inputs	List the nodes which connect to this transition box through both causality edges and condition edges[1]
Outputs	Lists the nodes which are outputs of this transition box
Orientation	The radio buttons toggle the orientation of the transition box on the screen

[1] These lists are created automatically once the connections are defined.

3.2.2 Functionality of the Model Analyzer

The function of the Model Analyzer is to backtrack the model to produce time fault trees and timed prime implicants for top events defined by the user. The Model Analyzer consists of the user interface resources and the analysis engine.

3.2.2.1 User Interface Resources

The user interfaces provide the environment for defining the goal of and displaying the results of the analysis. There are altogether 5 user interfaces for the Model Analyzer; the Top Event Interface, the Analyze Interface, the Display Result Interface, the Prime Implicants Interface and the Tables Interface.

The Top Event Interface (Figure 3.8) provides the interface for the user to define the top event for an analysis. The user defines a top event by specifying the states of the nodes and the associated time stamps. The list box on the left hand side of the dialog box (under the label "Nodes:") displays all the nodes in the model created using the Model Editor. The user can pick the node from this list to appear in the top event. The states defined for the node that has been selected will be shown in the list box in the middle of the dialog box (under the label "States:") and can be chosen to appear in the top event. The list box on the right hand side of the dialog box (under the label "Time:") allows the user to define a time stamp associated with the top event. After the node, the state and the time are all defined, the "Select" button can be pressed to add this node state to the top event. This will be summarized in the box at the bottom of the dialog box. Defining a top event with more than one state of a single node is just a matter of repeating the above procedure. The top event defined in Figure 3.8 is the one used in the example in Section 2.5.3 (TP = 5 @ t = 0).

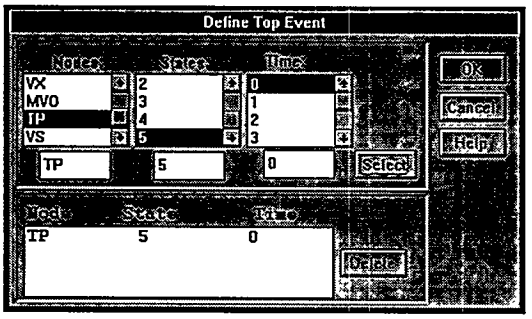


Figure 3.8 : User Interface for Defining the Top Event

The Analyze Interface (Figure 3.9) provides the interface for the user to specify how the analysis is to be carried out. The user can specify the number of time steps to be backtracked, as well as define the dynamic consistency rules to be used in the course of the analysis. The interface also notifies the reader that the decision table will be imported from a file of which the name is shown. Figure 3.9 defines the analysis that was performed for the example discussed in Section 2.5.3. The analysis procedure can be initiated by pressing the "Start" button.

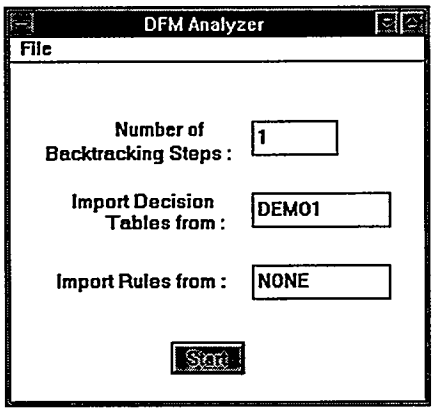


Figure 3.9 : User Interface for Defining the Scope of the Analysis

The Display Results Interface (Figure 3.10) shows the number of prime implicants found in the analysis. The two button "PI's" and "Tables" allow the user to access the details regarding the analysis and the prime implicants. Pressing the "PI's" button will take the user to the Prime Implicants Interface, whereas pressing the "Tables" button will take the use to the Tables Interface. Figure 3.10 shows the result that was obtained for the example in Section 2.5.3.

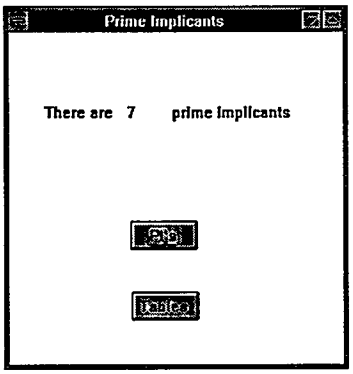


Figure 3.10 : User Interface for Displaying a Summary of the Analysis Results

The Prime Implicants Interface (Figure 3.11) displays the details of the prime implicants found in the analysis as a Notepad text file. As Notepad is a Windows application, the user gets all the convenience of printing a hardcopy of the file, cutting and pasting to incorporate the results into another word processing software. The prime implicants shown in Figure 3.11 are those obtained for the example discussed in Section 2.5.3.

The Tables Interface (Figure 3.12) shows the user how the Model Analyzer obtains the prime implicants as a Write text file. This file keeps track of all the intermediate transition tables. Like a Notepad file, the results shown can be printed out or incorporated into another word processing software. The intermediate tables shown in Figure 3.12 are those obtained for the analysis of the example discussed in Section 2.5.3.

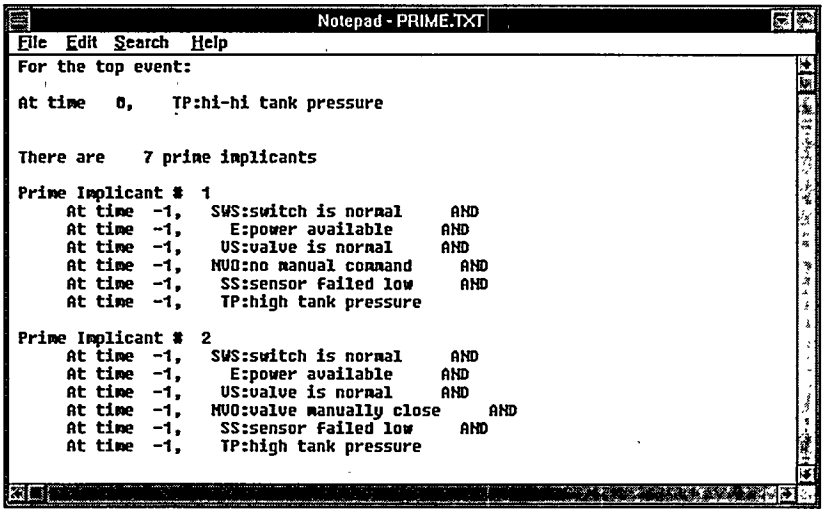


Figure 3.11 : User Interface for Displaying the Prime Implicants

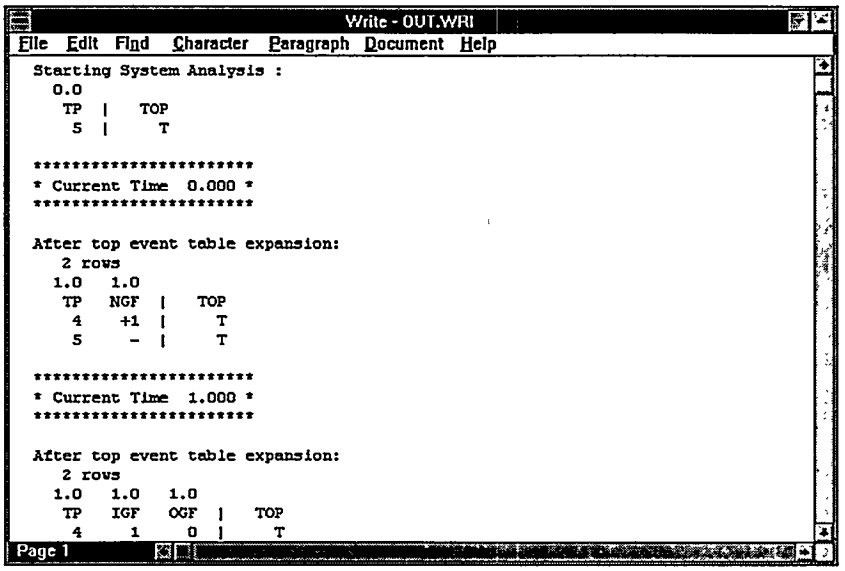


Figure 3.12 : User Interface for Displaying all the Intermediate Transition Tables

3.2.2.2 The Analysis Engine

The analysis engine is the part of the Model Analyzer which performs the backtracking steps. It carries out the steps of expanding the decision tables to form the intermediate transition tables, applying physical and dynamic consistency rules to remove inconsistent rows from the intermediate transition tables, simplifying the intermediate transition tables to obtain the critical transition table, and finally applying Quine's consensus theorem to generate the timed prime implicants.

4 INTERIM TEST CASE

The testing and demonstration of the DFM modeling and analytical approach has been executed by applying the technique in two realistic test cases, which are referred to within our project as the "Interim Test Case" (ITC) and the "Demonstration Test Case" (DTC). The latter, which is discussed in detail in Chapter 5, refers to the analysis of a PWR (Pressurized Water Reactor) steam generator level control system, the logic and algorithms of which are implemented via software. The DFM demonstration task called for a detailed analysis of this steam generator digital control system and this required the development of a detailed thermal hydraulic simulator of the steam generator portion of the system, which in turn was recognized from the beginning as being a relatively lengthy and complex task. Thus, the interim test case was conceived and constructed as a methodology test and demonstration tool that would not require itself as much effort to construct as the DTC.

The ITC was constructed to be a realistic system, that is a system that could conceivably exist and be used in an actual industrial application. The system was to be defined in such a way as to be easy to model and simulate in terms of its physical behavior, so that its simulated representation could be readied quickly for the purpose of enabling testing of the DFM approach and techniques. At the same time, to provide a true test for the DFM application and generate feedback on how DFM may need modifications and/or improvements, it was decided that the system would include a digital control system with logic and functional characteristics of a relatively high degree of complexity. The resulting ITC, which is described in detail, has been used to interactively test and develop the basic features and procedures of DFM, that is, to test how well the existing DFM features and procedures worked, and what extensions or additions might be needed to make DFM more readily usable and useful in more complex applications.

This chapter is organized into four sections. Section 4.1 (ITC System Description) describes the overall structure of and the functions carried out by the ITC system, as well as the system components and its control logic. Section 4.2 (ITC System Simulation) discusses how the system has been abstracted, i.e., the modeling assumptions and the physical laws used for representing the various portions and components of the system. This section also presents the procedures for simulating the behavior of the system and explains in detail the simulation algorithms employed in the simulation code which was used to understand the detailed behavior of the system. Section 4.3 (DFM Model of the ITC System) discusses the DFM model for this system, including the assumptions and the details regarding the definition of DFM nodes, transfer boxes and transition boxes. Finally, Section 4.4 (ITC DFM Model Analysis) summarizes some key ITC analyses that were executed, e.g., the top events that were analyzed and the resulting prime implicants and system sequences.

4.1 ITC System Description

The tank level and flow control system is shown in Figure 4.1. The key features of this system are summarized below:

- A water tank, fed by water pump on the inflow pipe and regulated by control and stop valves on the inflow and outflow pipes.
- A 3-element (level sensor, inflow sensor, outflow sensor) tank flow and level control system, with control logic implemented in a software-driven controller.
- A tank bypass is allowed for emergency mode of operation (e.g., tank overflow). In this mode, the inflow and outflow pipes are directly connected and the tank is isolated via the actuation of the three stop valves located on the inlet and outlet sides of the tank piping.
- Stop-valve actuation and control logic selection implemented within the digital controller software.

The major components of this system are the pump, the pipe, the control valves, the stopped valves, the water tank and the digital controller. Details of these components are discussed below.

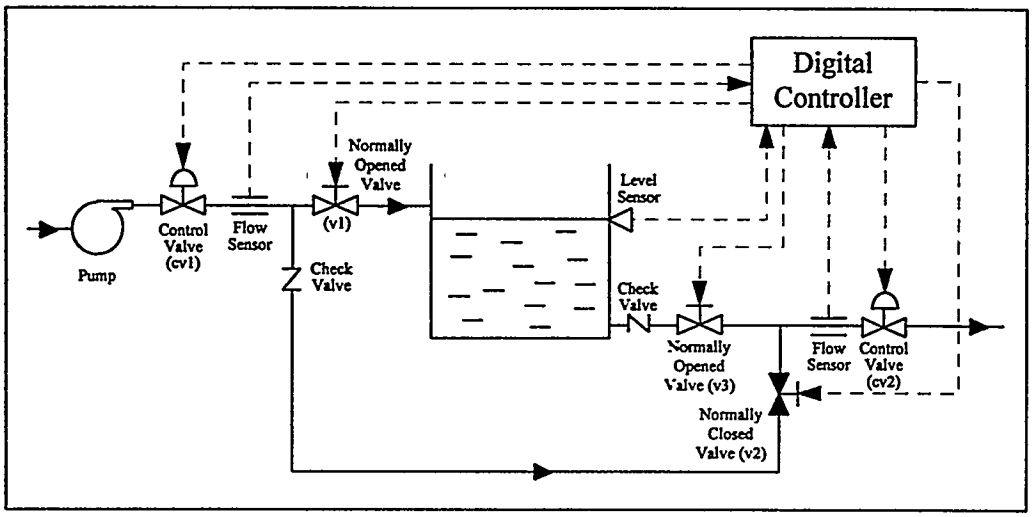


Figure 4.1 : ITC Digital Tank Level and Flow Control System

4.1.1 Pump

The pump is a centrifugal pump. It is used for pumping water to the water tank at an elevation of 50 ft. The pump operates at a constant speed and the pump characteristics curve is shown in Figure 4.2.

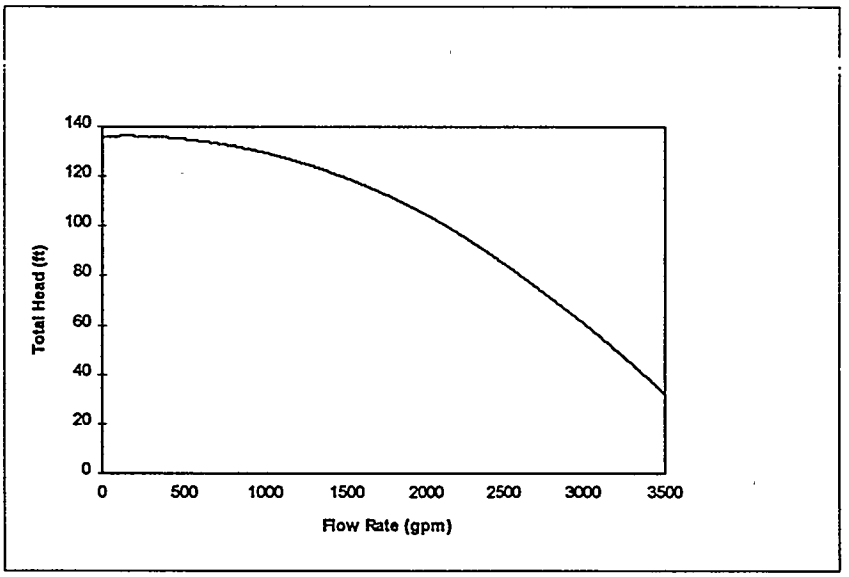


Figure 4.2 : Pump Characteristics Curve

4.1.2 Pipes

The pipes have a very smooth internal surfaces. The diameter of the pipes is 6 in. The upstream pipe is 600 ft long, while the downstream pipe and the bypass pipe are both 100 ft long.

4.1.3 Control Valves

The control valves are globe valves which are good for throttling operations. These valves can be throttled from 5% opened all the way to fully opened.

4.1.4 Stop Valves

The stop valves can either be fully opened or fully closed.

4.1.5 Water Tank

The water tank is 30 ft high and has a diameter of 15 ft.

4.1.6 Digital Controller

The digital controller operates in cycles of 100 ms. Its function is to maintain the water level at 15 ft and the downstream flowrate at a certain value by throttling the upstream and downstream control valves cv1 and cv2. The water level is controlled via cv1, while downstream flowrate is controlled via cv2. The controller receives inputs from the two flowrate sensors and the water level sensor, implements the control logic and then gives commands to the two control valves and the three stop valves.

The control flow is shown in Figure 4.3. In the control logic, the measured water level is compared with the level set-points. If the level is within a safe boundary, the controller will try to maintain the water level and the downstream flowrate at the corresponding set-points. The control logic uses Proportional Integral and Derivative (PID) control law to maintain the water level and Proportional Integral (PI) control law to control the downstream flowrate. Stop valves v1 and v3 will remain opened, while stop valve v2 will remain closed.

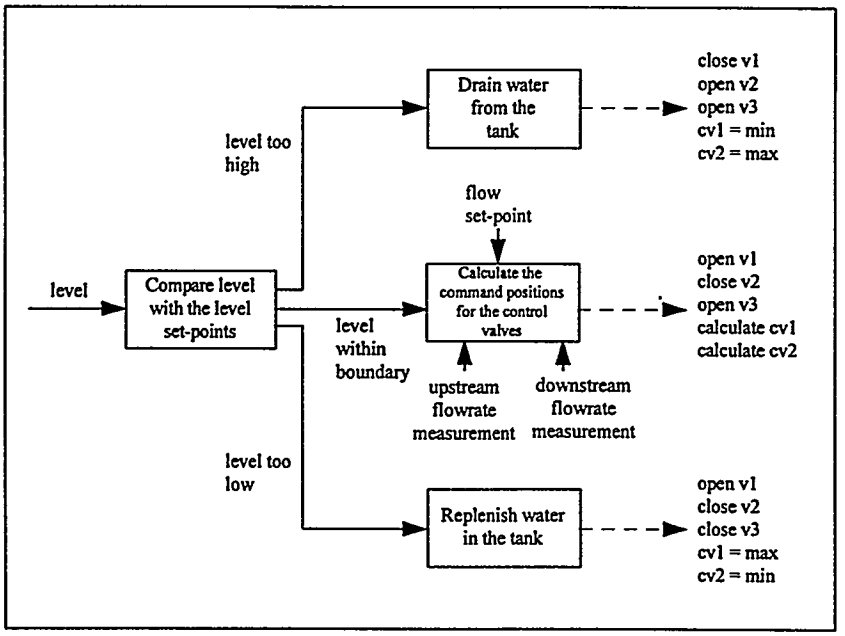


Figure 4.3 : Control Flow

However, if the water level is below a dangerously low level, the controller will bypass the normal control logic. Stop valve v1 will remain opened, but stop valves v2 and v3 will be closed in an attempt to replenish the water supply in the tank. On the other hand, if the water level is too high, the stop valve v1 will be closed and stop valves

v2 and v3 will be opened. This will drain the water from the tank. In both circumstances, the normal operation will be resumed once the water level returns to within the safe boundaries.

The control logic implemented is summarized in Table 4.I and the level and flowrate set-points are shown in Tables 4.II and 4.III.

Table 4.I : Control Logic

If (level < low-low set-pt.)	• Open stop valve v1 • Close stop valve v2 • Close stop valve v3 • Open control valve cv1 to maximum • Close control valve cv2 to minimum
If (low-low set-pt. ≤ level < low set-pt.)	• Open stop valve v1 • Close stop valve v2 • Open stop valve v3 • Calculate positions for control valves cv1 and cv2 using the normal level set-point and the low flowrate set-point
If (low set-pt. ≤ level ≤ high set-pt.)	• Open stop valve v1 • Close stop valve v2 • Open stop valve v3 • Calculate positions for control valves cv1 and cv2 using the normal level set-point and the normal flowrate set-point
If (high set-pt. < level ≤ high-high set-pt.)	• Open stop valve v1 • Close stop valve v2 • Open stop valve v3 • Calculate positions for control valves cv1 and cv2 using the normal level-set-point and the high flowrate set-point
If (high-high set-pt. < level)	• Close stop valve v1 • Open stop valve v2 • Open stop valve v3 • Close control valve cv1 to minimum • Open control valve cv2 to maximum

Table 4.II : Level Set-Points

Level Set-Point	Level (ft)
low-low	5
low	10
normal	15
high	20
high-high	25

Table 4.III : Flowrate Set-Points

Flowrate Set-Point	Flowrate (gpm)
low	500
normal	700
high	900

4.2 ITC System Simulation

As pointed out in the general discussion provided in Chapter 2, if the DFM analysis of the system is to be carried out with a high degree of completeness and fidelity, then the key features of the expected cause and effect and dynamic behavior of the system being modeled need to be known in detail by the analyst. This may be assumed to be true in

the assurance or safety analysis of a system that is either already operational, or that has at least been defined and designed to its detailed component level. A more qualitative knowledge, on the other hand, may be entirely sufficient for a DFM analysis that is conducted at the system specification level, when only a first-tier, preliminary definition of the system design is available. Because the ITC exercise was intended to test the more complete set of DFM capabilities, the first type of analysis was used in this exercise, and a behavior simulation model of the tank and piping system described in Section 4.1 was developed to understand the quantitative and dynamic aspects of the system behavior which may be useful in the construction of the DFM model of the system.

The simulation model that was developed finds the operating condition of the system and is also capable of predicting the dynamic evolution of the system parameters during operational transients. It calculates, as a function of time during a transient set in motion by the change of certain system boundary conditions, the pressure head developed by the pump, the pressure losses across the components, the upstream flowrate and the downstream flowrate and keeps track of the water level in the tank.

The pressure head developed by the pump and the upstream flowrate is calculated by finding the intersection between the pump characteristics curve and the upstream friction loss curve (Figure 4.4). The downstream flowrate is calculated by equating the pressure head at the water tank exit and the sum of the pressure losses in the downstream control valve (cv2) and the downstream pipe. If the stop valve v2 (the bypass valve) is opened, this flowrate is added to the upstream flowrate to get the total downstream flowrate. The variation in the water tank level is calculated by integrating the difference between the flowrate into the tank and the flowrate out of the tank.

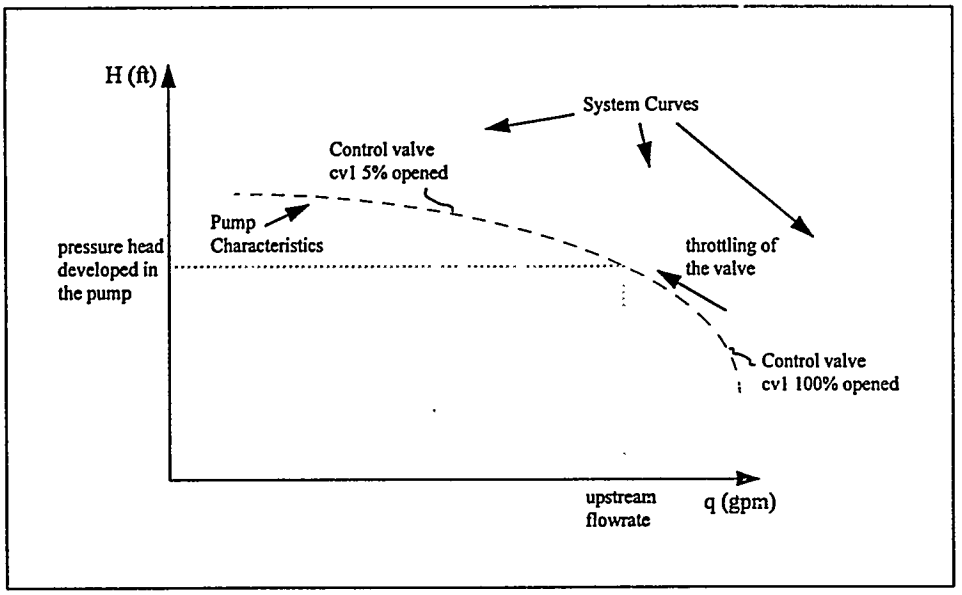


Figure 4.4 : Finding the Operating Condition of the System

The following describes how the different components are modeled by physical laws in the simulation program:

4.2.1 Pump

The pump behavior is described by the pump characteristics curve shown in Figure 4.2. A second order polynomial is used to approximate this curve. The equation is

$$H = -9.28 \times 10^{-6} q^2 + 2.8 \times 10^{-3} q + 136 \qquad \text{(Eq. 4.1)}$$

where H is the total head in ft
q is the flowrate in gpm

4.2.2 Pipes

The friction head loss in the pipes in modeled by the empirical Hazen-Williams equation:

$$H_L = \frac{4.73L}{C^{1.852}D^{4.87}}q^{1.852} \qquad \text{(Eq. 4.2)}$$

where H_L is the friction head loss in ft
D is the pipe diameter in ft
L is the pipe length in ft
q is the volumetric flowrate in ft^3/s
C is the Hazen-Williams coefficient
For a highly smooth pipe, C = 140

4.2.3 Control Valves

The head loss in the control valves is also estimated by using the Hazen-Williams equation. An equivalent (L/D) ratio of 400 is used. The opening and closing of the valve is assumed to affect the pressure loss similar to an orifice.

$$H_L = \frac{4.73(L/D)}{C^{1.852}D^{3.87}}q^{1.852}\left(\frac{100}{x}\right)^2 \qquad \text{(Eq. 4.3)}$$

where x is the valve position in %

4.2.4 Stop Valves

No pressure loss is assumed for a fully opened stop valve.

4.2.5 Digital Controller

Under normal conditions, the controller controls the water tank level by varying the upstream valve (cv1) position command. The upstream valve position command is calculated using a PID control logic, as shown in Eq. 4.4:

$$\Delta cvxl = -k_{11}\left(\Delta l + R_{11}\int \Delta l dt + R_{12}\left(q_{in} - q_{out}\right)\right. \qquad \text{(Eq. 4.4)}$$

where Δcvx1 is the change in control valve position command
Δl is the actual level - level set-point
q_{in} is the flowrate into the tank
q_{out} is the flowrate out of the tank
k_{11}, R_{11}, R_{12} are control parameters
$k_{11} = 10$, $R_{11} = 0.1$, $R_{12} = 0.005$

The controller controls the downstream flowrate by varying the downstream control valve position. The downstream valve position is calculated using a P-I control logic, as shown in Eq. 4.5:

$$\Delta cvx2 = -k_{21}\left(\Delta q_{down} + R_{21}\int \Delta q_{down} dt\right. \qquad \text{(Eq. 4.5)}$$

where Δcvx2 is the change in the control valve position command
Δq_{down} is actual downstream flowrate - flowrate set-point
k_{21} and R_{21} are control parameters
$k_{21} = 0.1$, $R_{21} = 5$

4.2.6 Simulation Code Algorithms

As discussed previously, the simulation code finds the operating condition of the system and keeps track of the water level in the tank. The algorithm used in the simulation code is shown in Table 4.IV. It should be noted that when both stop valves v1 and v2 are opened, most of the water will flow through v1 into the tank because of the lower pressure drop. Hence, it was assumed in the algorithm that the bypass flowrate is zero in both cases. A simulation time step of 10 ms is used. A sample output of this simulation program is shown in Figures 4.5 - 4.7. For this simulation run, the initial water level in the tank is 18 ft, both control valves start at their minimum positions. In addition, the stop valves v1 and v3 are opened, while the stop valve v2 is closed.

Table 4.IV : Simulation Algorithm (1/2)

1	Initialize the simulation time Initialize the positions for the stop valves v1, v2 and v3 Initialize the positions for the control valves cv1 and cv2 Initialize the water level in the tank Initialize the upstream flowrate, the bypass flowrate, and the downstream flowrate

2	Based on current positions for valves v1, v2, v3, cv1, cv2 and the water level, calculate the upstream flowrate, the bypass flowrate, the downstream flowrate, q_{in}(flowrate into the tank) and q_{out}(flowrate out of the tank).
	
	If (v1 is opened AND v2 is opened AND v3 is opened)
	Solve
	Pump Head = Static Loss + Upstream Pipe Loss + Control Valve Loss .....(1)
	for Upstream flowrate
	Bypass flowrate = 0
	Solve
	Water Level = Downstream Pipe Loss + Downstream Control Valve Loss .....(2)
	for Downstream flowrate
	If (v1 is opened AND v2 is opened AND v3 is closed)
	Solve (1) for Upstream flowrate
	Bypass flowrate = 0
	Downstream flowrate = 0
	If (v1 is opened AND v2 is closed AND v3 is opened)
	Solve (1) for Upstream flowrate
	Bypass flowrate = 0
	Solve (2) for Downstream flowrate
	If (v1 is opened AND v2 is closed AND v3 is closed)
	Solve (1) for Upstream flowrate
	Bypass flowrate = 0
	Downstream flowrate = 0

Table 4.IV : Simulation Algorithm (2/2)

2... cont	If (v1 is closed AND v2 is opened AND v3 is opened)
	Solve
	Pump Head = Static Loss + Upstream Control Valve Loss + Upstream Pipe Loss + Bypass Pipe Loss + Water Tank Level .....(4)
	for Upstream flowrate
	Solve (2) for Downstream flowrate
	If (Downstream flowrate < Upstream flowrate)
	Solve
	Pump Head = Static Loss + Upstream Control Valve Loss + Upstream Pipe Loss + Bypass Pipe Loss + Down Pipe Loss + Downstream Valve Loss .....(3)
	for Upstream flowrate
	Downstream flowrate = Upstream flowrate
	Bypass flowrate = Upstream flowrate
	If (v1 is closed AND v2 is opened AND v3 is closed)
	Solve (3) for Upstream flowrate
	Bypass flowrate = Upstream flowrate
	Downstream flowrate = 0
	If (v1 is closed AND v2 is closed AND v3 is opened)
	Upstream flowrate = 0
	Bypass flowrate = 0
	Solve (2) for Downstream flowrate
	If (v1 is closed AND v2 is closed AND v3 is closed)
	Upstream flowrate = 0
	Bypass flowrate = 0
	Downstream flowrate = 0
	q_{in} = Upstream flowrate - Bypass flowrate
	q_{out} = Downstream flowrate - Bypass flowrate

3	For every 100 ms (10 simulation steps), calculate the sensor inputs. From sensor inputs, calculate command positions for valves for v1, v2, v3, cv1 and cv2
4	Based on q_{in}, q_{out} and the current water level, calculate the new water level
5	Increment simulation time
6	Go to step 2

Controller cycle = 100 ms
Simulation cycle = 10 ms

4.3 DFM Model of the ITC System

This section describes the DFM model of the ITC system. In building the DFM model for this system, certain standard assumptions were made regarding the possible failure modes of the individual system hardware components. Both the flowrate sensors and the level sensor were assumed to be allowed to fail high, fail low, or fail as-is. Similarly, both control valves can fail closed, fail open, or fail as-is, while the stop valves can either fail closed or fail open, and the check valves can fail open.

The DFM model of the tank level and flow control system was constructed by following the steps outlined in Section 2.2, and is shown in Figure 4.8. The digital controller model is shown as a black box in this figure, but is expanded in full detail in Figure 4.9. The description of the variables that appear in the model as DFM "nodes" can be found in Table 4.V. In the DFM model in Figure 4.8, transfer boxes 1 and 2 represent the control valve actuators, and transfer boxes 3, 4 and 5 model the stop valve actuators. Similarly, transfer boxes 6 and 7 represent the flowrate sensors, while transfer box 8 models the level sensor. In addition, transfer box 9 represents the pump, transfer box 10 shows the bypass pipe and transfer boxes 11 and 12 model the inlet pipe and the outlet pipe respectively. The transfer box 13 and the transition box 14 model the behavior of the water level in the tank. In Figure 4.9, the transfer boxes 15,

16 and 17 represent the A/D converters for the upstream flowrate sensor signal, the downstream flowrate sensor signal and the level sensor signal respectively. Transfer boxes 18-21 model the PID control logic for maintaining the water level, while transfer boxes 22 and 23 model the PI control logic for regulating the downstream flowrate. In addition, transition box 24 represents the module in the software that switches the positions of the stop valves v1, v2 and v3, while transition boxes 25 and 26 model the D/A converters for the upstream (i.e., level-control) valve position command and the downstream flow-control valve position command, respectively.

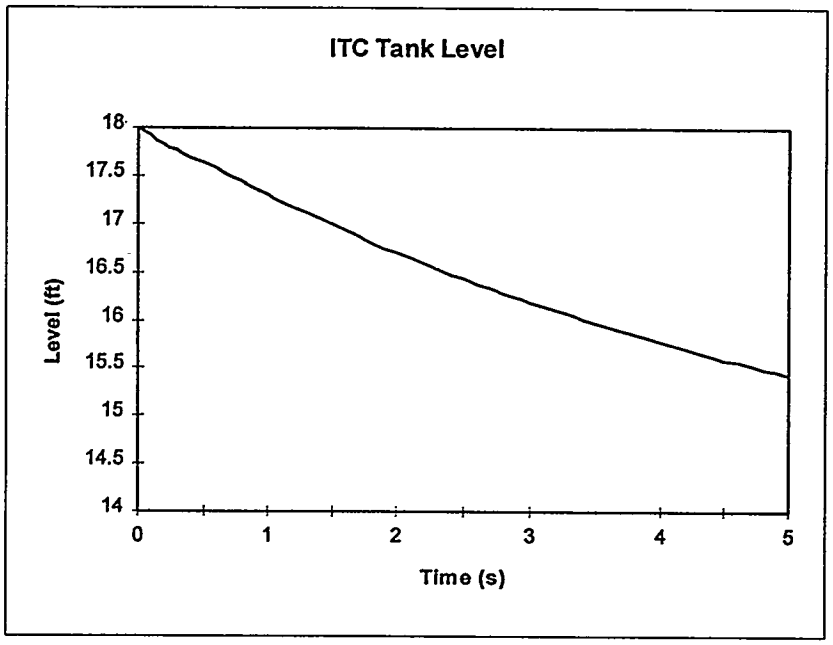


Figure 4.5 : Variation of the Tank Level with Time

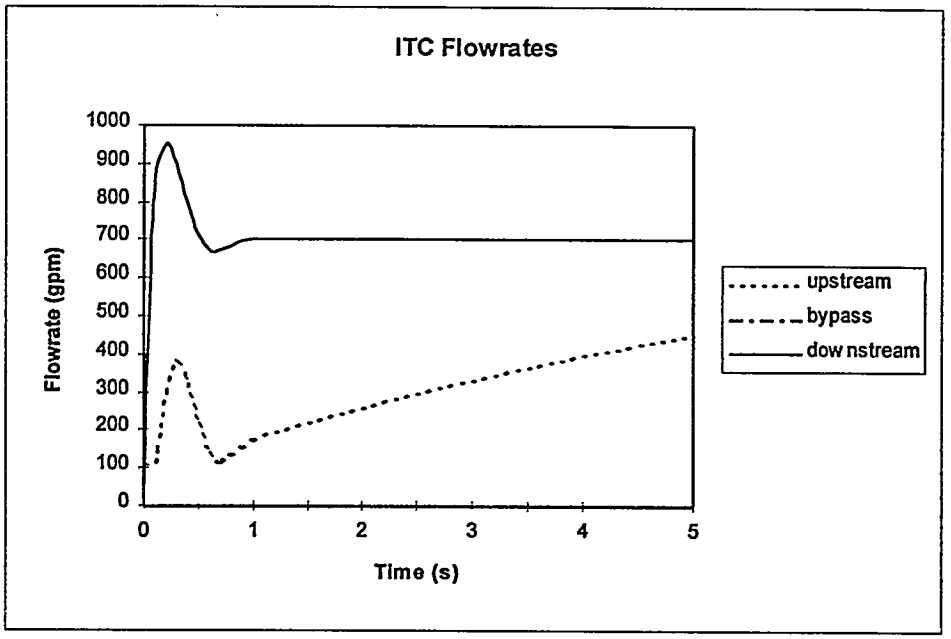


Figure 4.6 : Variations of the Upstream Flowrate and the Downstream Flowrate with Time

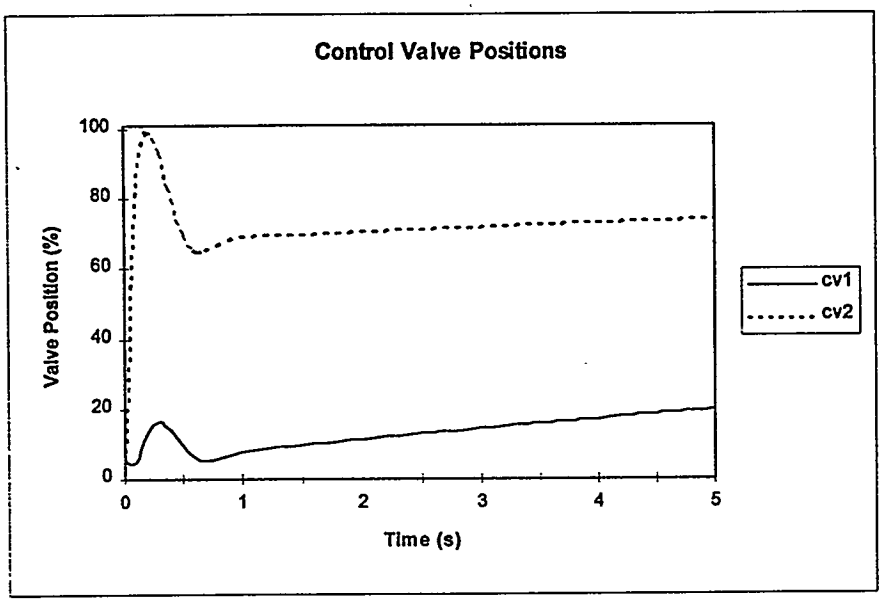


Figure 4.7 : Variations of the Control Valve Positions with Time

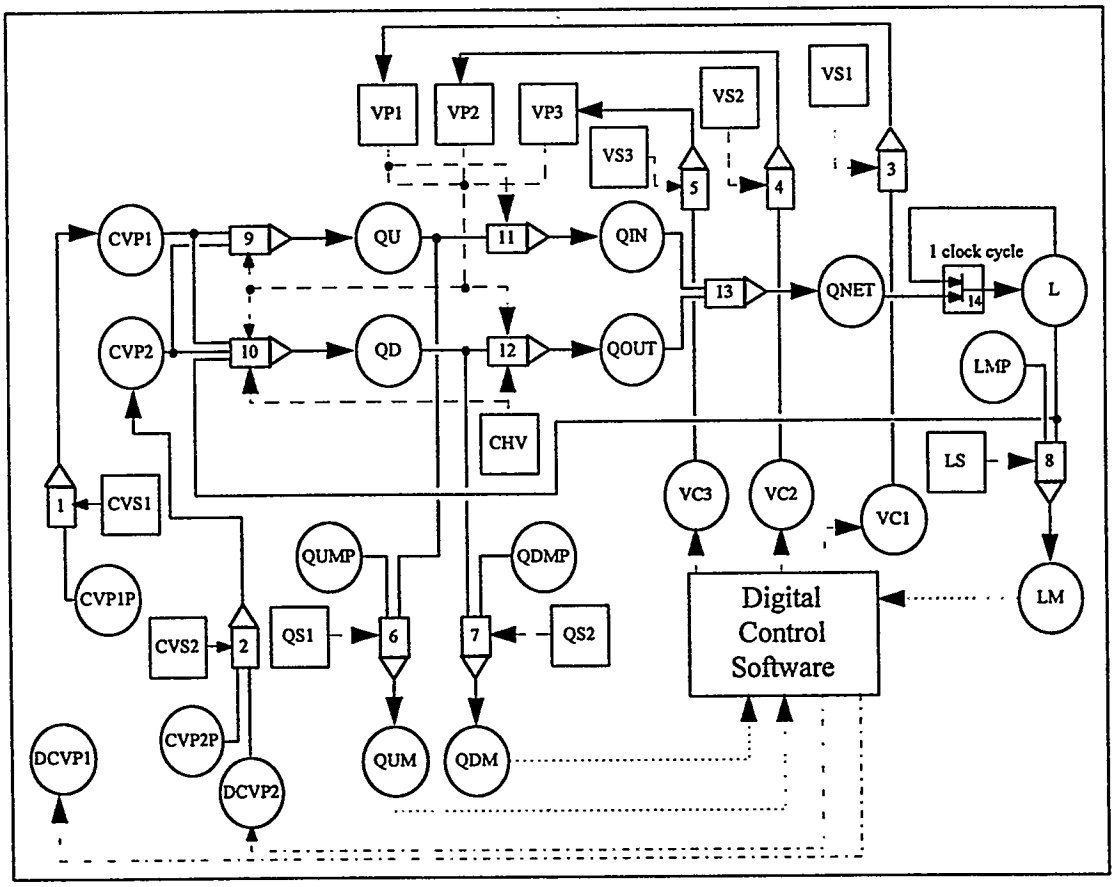


Figure 4.8 : DFM Model of the Tank Level and Flow Control System

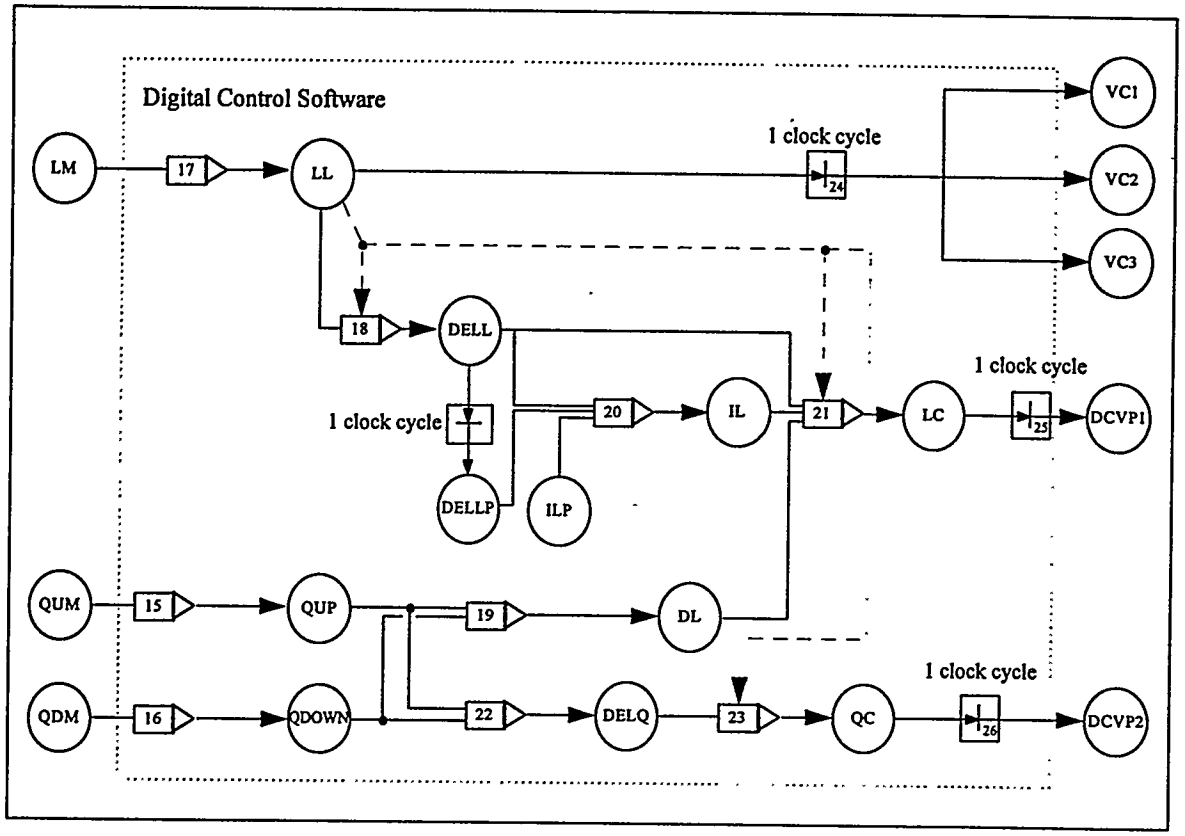


Figure 4.9 : DFM Model of the Digital Controller

In developing the DFM model for the tank level and flow control system, the issue of how to model different types of control logic (such as PI, PID, etc.) was encountered. The ensuing investigation resulted in modeling templates for representing these different classes of control logic. The related findings are discussed in detail in Chapter 6. The branch linking nodes DELL, DELLP, ILP, IL, DL and LC in Figure 4.9 is an example of the template which can be used to represent a PI control logic.

The nodes in the DFM model are discretized into finite number of states. The discretization schemes are shown in Tables 4.VI - 4.XX. These schemes reflect the knowledge about the system and assumptions regarding the failure modes of the components. For example, the discretization scheme for L (tank water level) in Table 4.XIV is such that the state boundaries correspond to the set-points used in the control logic. On the other hand, the scheme for discretizing CVS1 (state of the upstream control valve) in Table 4.VIII is defined in accordance with the assumption regarding the failure modes of this component.

Table 4.V : Description of the Variables in the DFM Model

Variable	Description
CHV	State of the check valve
CVP1	Position of the upstream control valve cv1
CVP1P	Position of the upstream control valve cv1 in the previous cycle
CVP2	Position of the downstream control valve cv2
CVP2P	Position of the downstream control valve cv2 in the previous cycle
CVS1	State of the upstream control valve cv1
CVS2	State of the downstream control valve cv2
DCVP1	Change in position of the upstream control valve cv1
DCVP2	Change in position of the downstream control valve cv2
DELL	Level error term in the software
DELLP	Level error term in the software in the previous cycle
DELQ	Downstream flowrate error term in the software
DL	Mismatch between upstream flowrate and downstream flowrate
IL	Integral control term for level in the software
ILP	Integral control term for level in the previous cycle
L	Water Level in the Tank
LC	Upstream valve position command
LL	Software representation of the water level in the tank
LM	Measurement of the water level in the tank
LMP	Measurement of the water level in the tank in the previous cycle
LS	State of the water level sensor
QC	Downstream valve position command
QD	Downstream flowrate
QDM	Measurement of the downstream flowrate
QDMP	Measurement of the downstream flowrate in the previous cycle
QDOWN	Software representation of the downstream flowrate
QIN	Flowrate into the tank through the inlet
QNET	Net flowrate into the tank
QOUT	Flowrate out of the tank through the outlet
QS1	State of the upstream flowrate sensor
QS2	State of the downstream flowrate sensor
QU	Upstream flowrate
QUM	Measurement of the upstream flowrate
QUMP	Measurement of the upstream flowrate in the previous cycle
QUP	Software representation of the upstream flowrate
VC1	Command to stop valve v1
VC2	Command to stop valve v2
VC3	Command to stop valve v3
VP1	Position of stop valve v1
VP2	Position of stop valve v2
VP3	Position of stop valve v3
VS1	State of stop valve v1
VS2	State of stop valve v2
VS3	State of stop valve v3

Table 4.VI : Discretization of CHV

State	Description
0	Normal
1	Failed opened

Table 4.VII : Discretization of CVP1, CVP2, CVP1P and CVP2P

State	Description
0	0%-10%
1	10%-30%
2	30%-50%
3	50%-70%
4	70%-90%
5	90%-100%

Table 4.VIII : Discretization of CVS1 and CVS2

State	Description
-1	Failed stuck closed
0	Normal
1	Failed stuck opened
2	Failed as is

Table 4.IX : Discretization of DCVP1, DCVP2, LC and QC

State	Description
-3	-100% to -60%
-2	-60% to -20%
-1	-20% to -5%
0	-5% to +5%
+1	+5% to +20%
+2	+20% to +60%
+3	+60% to +100%

Table 4.X : Discretization of DELL and DELLP

State	Description
-3	-15 ft to -5 ft
-2	-5 ft to -1 ft
-1	-1 ft to -0.2 ft
0	-0.2 ft to +0.2 ft
+1	+0.2 ft to +1 ft
+2	+1 ft to +5 ft
+3	+5 ft to +15 ft

Table 4.XI : Discretization of DELQ

State	Description
-3	-1000 gpm to -600 gpm
-2	-600 gpm to -200 gpm
-1	-200 gpm to -50 gpm
0	-50 gpm to +50 gpm
+1	+50 gpm to +200 gpm
+2	+200 gpm to +600 gpm
+3	+600 to +1000 gpm

Table 4.XII : Discretization of DL

State	Description
-2	-1500 gpm to -800 gpm
-1	-800 gpm to -150 gpm
0	-150 gpm to +150 gpm
+1	+150 gpm to +800 gpm
+2	+800 gpm to +1500 gpm

Table 4.XIII : Discretization of IL and ILP

State	Description
-2	-7000 to -1500
-1	-1500 to -200
0	-200 to +200
+1	+200 to +1500
+2	+1500 to +7000

Table 4.XIV : Discretization of L, LL, LM and LMP

State	Description
0	0-5 ft, Very low
1	5-10 ft, Low
2	10-15 ft, Slightly low to normal
3	15-20 ft, Normal to slightly high
4	20-25 ft, High
5	25-30 ft, Very high

Table 4.XV : Discretization of LS, QS1 and QS2

State	Description
-1	Failed Low
0	Normal
1	Failed High
2	Failed As Is

Table 4.XVI : Discretization of QD, QDM, QDMP, QDOWN and QOUT

State	Description
0	0 to 250 gpm
1	250 to 500 gpm
2	500 to 700 gpm
3	700 to 900 gpm
4	900 to 1300 gpm

Table 4.XVII : Discretization of QNET

State	Description
-2	-1300 to -1000 gpm
-1	-1000 to -500 gpm
0	-500 to +500 gpm
+1	+500 to +1000 gpm
+2	+1000 to +1300 gpm

Table 4.XVIII : Discretization of QIN, QUM, QUMP and QUP

State	Description
0	0 to 250 gpm
1	250 to 500 gpm
2	500 to 750 gpm
3	750 to 1000 gpm
4	1000 to 1300 gpm

Table 4.XIX : Discretization of VC1, VC2, VC3, VP1, VP2, and VP3

State	Description
0	Close
1	Open

Table 4.XX : Discretization of VS1, VS2 and VS3

State	Description
-1	Failed Closed
0	Normal
1	Failed Opened

Decision tables were constructed to complete the definition of this DFM model. The decision tables for the physical components were built by running the corresponding subroutines in the simulation code. For instance, Table 4.XXI is the decision table for transition box 14 and is constructed by running the software module in the simulation code that updates the water level in the tank. It is important to note that this would most likely be true even in the case in which one were modeling a materially existing system, since it would be completely impractical to exercise all kinds of arbitrary transients on the actual system, just to determine what its behavior is.

An equally important observation needs to be made with regard to construction of the decision tables for the system software. In fact, since an exact copy of the control software can be in most cases be obtained for an existing system, the decision tables for the digital controller can be constructed by executing, off-line, module by module, the control software that is actually implemented. This activity is essentially the exact equivalent, within the overall implementation of the DFM analytical technique, of performing "module testing" (as normally referred to in the software test practice) on the software, and was discussed in this context in Sections 2.1 and 2.2. Of course, in the analysis of the ITC system, the distinction between "actual copy" and "simulation copy" of the control software cannot be quite made, as we are dealing with a fictional system which exists only in its "simulated" version. The observations just made above remain valid, however, for their significance in the analysis of actual systems.

Table 4.XXII shows the decision table for transfer box 20, as built by actually running the subroutine that performs the PID control logic.

4.4 ITC DFM Model Analysis

The section discusses the analysis performed on the DFM model of the ITC system. To test the capability of DFM in a system and software assurance mode of application, a fault was intentionally injected in the control software. We present here the results of this "faulted-case analysis" to illustrate the capability of DFM for identifying and isolating software errors in such a mode of application. A number of analyses were also performed on the original unfaulted system, mostly for the purpose of refining and debugging the DFM analytical procedures and software tools. Since these analyses are judged to be of little interest for the reader, they are left out of the discussions in this section.

Table 4.XXI : Decision Table for Transition Box 14

QNET	L	L+
-	0	0
-2	1	0
-1	1	0
0	1	0
-	1	1
-2	2	1
-1	2	1
0	2	1
0	0	1
1	0	1
2	0	1
-	2	2
-2	3	2
-1	3	2
0	3	2
0	1	2
1	1	2
2	1	2
-	3	3
-2	4	3
-1	4	3
0	4	3
0	2	3
1	2	3
2	2	3
-	4	4
-2	5	4
-1	5	4
0	5	4
0	3	4
1	3	4
2	3	4
-	5	5
0	4	5
1	4	5
2	4	5

4.4.1 Description of the Fault Injected

A fault was injected into the digital control software. The fault was placed in the module of the software code that sets the position of the control valves and the stop valves when the measured water level is above the high-high set-point. Under that condition, the digital controller should close the stop valve v1, open the stop valves v2 and v3, close the upstream control valve to the minimum position (5%) and open the downstream control valve to the maximum position (100%), as was defined in Table 4.I. A comparison of the unfaulted software and the faulted software is shown in Figure 4.10. Instead of setting the variable cvx2_command to 100, the programming error causes the software to set the variable to -100. This fault has the effect of closing the downstream control valve to 5%. The reader should note that this branch of the code will not be executed unless the level is above the high-high set-point, and that this programming error requires an additional hardware failure to cause a system failure. Hence, blind testing may not be able to catch this software error unless this branch is specifically tested under the special condition.

Table 4.XXII : Decision Table for Transfer Box 20

DELL	DELLP	ILP	IL
-3	-3	-1	-2
-	-	-2	-2
-3	-	0	-1
-	-	-1	-1
-	-3	0	-1
-2	-2	0	-1
-3	-3	+1	0
-	-	0	0
+3	+3	-1	0
-	-	+1	+1
-	+3	0	+1
+2	+2	0	+1
+3	-	0	+1
-	-	+2	+2
+3	+3	+1	+2

Unfaulted Software

```
/************************************/
/* Subroutine Controller()          */
/* Simulates the digital controller */
/************************************/
void Controller()
{
        .
        .
        .
        else if ( l_measured >
Setpoint.level[4] )
        {
                vx1_command  = Closed;
                vx2_command = Opened;
                vx3_command = Opened;
                cvx1_command =  -100;
                cvx2_command = 100 ;
        }
        .
        .
        .
}
```

Faulted Software

```
/************************************/
/* Subroutine Controller()          */
/* Simulates the digital controller */
/************************************/
void Controller()
{
        .
        .
        .
        else if ( l_measured >
Setpoint.level[4] )
        {
                vx1_command  = Closed;
                vx2_command = Opened;
                vx3_command = Opened;
                cvx1_command =  -100;
                cvx2_command = -100 ;
        }
        .
        .
        .
}
```

Figure 4.10 : Comparison of the Unfaulted Software and the Faulted Software

The DFM model was constructed without using any prior knowledge of the software error. This is possible because, as mentioned earlier, the decision tables are built directly by "testing" the individual modules of the digital control software. Figure 4.11 shows the difference between the decision tables for the correct software and the faulted software. Note that the decision table on the left hand side (corresponding to the correct version) is produced by testing the module shown on the left hand side of Figure 4.10, while the decision table on the right hand side is generated by testing the module on the right hand side of Figure 4.10.

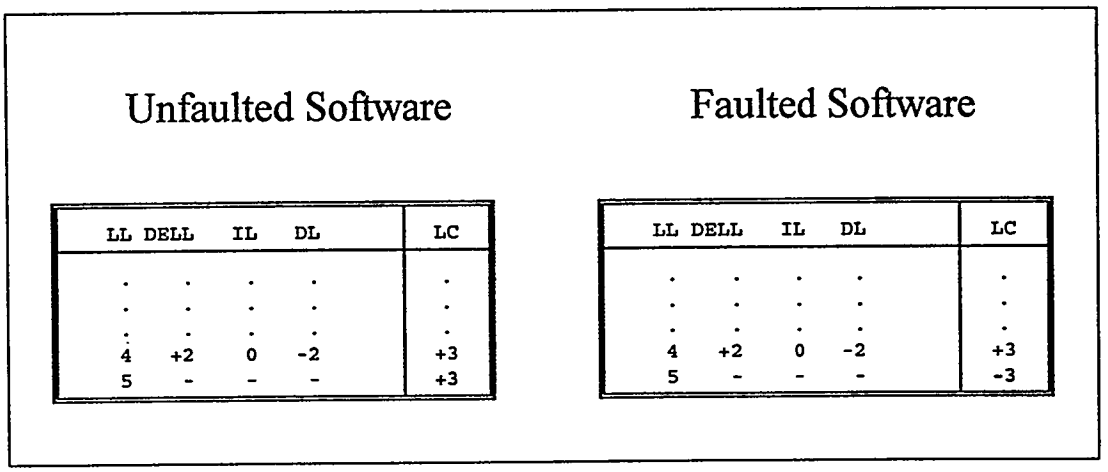


Figure 4.11 : Comparison of the Decision Tables for the Unfaulted Software and the Faulted Software

4.4.2 Analysis of the System with the Faulted Control Software

4.4.2.1 Definition of the Top Event

The system failure was defined as the tank "overflowing". This translates into a definition of the states of the pertinent DFM nodes as:

$$\{ (L = 5 \text{ @ } t = 0) \text{ AND } (L = 5 \text{ @ } t = -1) \text{ AND } (QNET = +1 \text{ @ } t = 0) \}.$$

The meaning of the above definition is that the tank level is very high in both the current and the previous time step and that there is a net inflow of water into the tank. In the course of the various ITC analyses that were carried out, it was discovered that defining the top event as specifically as possible, such as using a combination of several conditions across different time steps to describe the tank overflowing, would enable the analysis to be performed more efficiently. Defining a top event in very precise terms ensures that the DFM Model Analyzer software needs less computer memory to store the intermediate transition tables developed during the analysis and spends less computing time tracing events that are irrelevant. As a comparison, when the top event was defined more simplistically as { L = 5 @ t = 0 } (the level is high at the current time step), the Model Analyzer ran out of memory before the analysis was completed. The care that has to be exercised in a specific and precise definition of the possible top events of interest is one of the key findings of our test cases and will be revisited in Section 6.3.

4.4.2.2 Constraints Imposed on the Analysis

Dynamic consistency rules are defined to prune out the branches that encompass events that the analyst assumes to be impossible due to the dynamic constraints of the system. The dynamic consistency rules so defined are listed in Table 4.XXIII. These rules reflect the assumption that if any sensor or valve has failed, then it remains in the original failure state.

In addition to the use of dynamic consistency rules, we also determined that it is very beneficial, in terms of use of computational resources during an analysis, to permit the specification of "check rules", which can be used to limit the DFM search to the identification of specific classes of errors. In this particular case, the rules were defined in such a way as to force the Model Analyzer to analyze, store and display only those failure sequences that are related to software errors. For this purpose, the definition of software error must be referred to a formal catalogue of system behavior specifications, and includes any kind of software-produced action that violates the given set of top-level software behavior specifications. The specifically relevant portion of these specifications requires the controller not to command further opening of the upstream valve and/or further closing of the downstream valve when the water tank level is in the "high-high" range. This rule was defined as a boundary condition that the Model Analyzer used

to distinguish between branches that needed to be expanded further and those for which further expansion was not required. The benefit and effectiveness of the definition of this sort of check-rules is another key finding of our study and will be discussed further in Section 6.3.

Table 4.XXIII : The Dynamic Consistency Rules Defined for the Analysis

	Rule	Meaning
1	For CVS1, states -1, 1 and 2 are sink states	The upstream control valve, once failed, cannot be repaired.
2	For CVS2, states -1, 1 and 2 are sink states	The downstream control valve, once failed, cannot be repaired.
3	For LS, states -1, 1 and 2 are sink states	The level sensor, once failed, cannot be repaired.
4	For QS1, states -1, 1 and 2 are sink states	The upstream flow sensor, once failed, cannot be repaired.
5	For QS1, states -1, 1 and 2 are sink states	The downstream flow sensor, once failed, cannot be repaired.
6	For VS1, states -1 and 1 are sink states	The stop valve v1, once failed, cannot be repaired.
7	For VS2, states -1 and 1 are sink states	The stop valve v2, once failed, cannot be repaired.
8	For VS3, states -1 and 1 are sink states	The stop valve v3, once failed, cannot be repaired.

4.4.2.3 Result of the Analysis

The analysis was carried out for one step backward in the reference time frame and the prime implicant that was correspondingly identified is shown in Table 4.XXIV. The software error that causes the tank to overflow is found to be the downstream control valve commanded to close to its minimum position (software condition) AND the failure of the check valve (external condition). The other conditions present in the prime implicant are those that specify that all the sensors, the control valves and the stop valves are normal, and that the level was high in the previous time step. The presence of these other non-failure conditions is a characteristic of the multi-state, non-coherent representation of the system, as is in the example given in Section 2.5.3. To facilitate the efficient presentation of the prime implicants, it will be helpful to give the user the option to exclude certain normal component states from the display. This point will be revisited in Section 6.3.2.3, Findings and Insights.

Table 4.XXIV : Prime Implicant for the Event in which the Tank Overflows

	Prime Implicant		
1	Level sensor is normal	@ t = 0	AND
	Upstream flow sensor is normal	@ t = 0	AND
	Downstream flow sensor is normal	@ t = 0	AND
	Upstream control valve is normal	@ t = 0	AND
	Stop valve v1 is normal	@ t = 0	AND
	Stop valve v2 is normal	@ t = 0	AND
	Stop valve v3 is normal	@ t = 0	AND
	Downstream control valve is normal	@ t = 0	AND
	check valve failed open	@ t = 0	AND
	Upstream control valve commanded to close to its minimum	@ t = -1	AND
	Downstream control valve commanded to close to its minimum	@ t = -1	AND
	Tank level was very high	@ t = -1	

The reader should note that the prime implicant in Table 4.XXIV is not the only cause for the tank to overflow. In fact, many other prime implicants can lead to the same top event; one of which is, for example, the failure of the

level sensor in the "stuck low" mode. The prime implicant in Table 4.XXIV, however, is the only one containing a software error as a contributor to producing the top event. The fact that the non-software-related prime implicants were not produced by the DFM analysis that uncovered this particular time implicant is due to the application of the check rule of which we have made mention above. The effect of the application of this rule is to narrow the analysis into searching for a particular class of errors. Appropriate use of the check-rules allows the analyst to focus on particular failure paths, if he/she so desires, and to make more efficient use of the computational resources available for the analysis.

5 DEMONSTRATION TEST CASE

The testing and demonstration of the DFM modeling and analytical approach has been executed by applying the technique in two realistic test cases, which are referred to within our project as the "Interim Test Case" (ITC) and the "Demonstration Test Case" (DTC). The former has been discussed in detail in Chapter 4. The DTC refers to the analysis of a Pressurized Water Reactor (PWR) steam generator level control system, the logic and algorithms of which are implemented via software. This DFM demonstration case study called for a detailed analysis of this steam generator digital control system, and thus the development of a detailed thermal hydraulic simulator of the steam generator portion of the system was included as part of the task.

This chapter is organized in three sections. Section 5.1 (Steam Generator Simulation Package) provides a detailed discussion of how the thermal hydraulic and digital control portions of the test case are set up, Section 5.2 (DFM Model of the DTC System) gives a summary of how this test case is modeled using DFM, and Section 5.3 (DTC DFM Model Analysis) presents the definitions for and results of some key analyses.

5.1 Steam Generator Simulation Package

A dynamic simulation model of a vertical U-tube steam generator (SG) typical of a two loop Combustion Engineering Pressurized Water Reactor (PWR) was developed. The simulation model consists of the steam generator, Main Feedwater and Auxiliary Feedwater systems, Steam Header, SG Pressure Control System, and the SG Level Control System. This model is converted to a simulation code written in FORTRAN.

5.1.1 Steam Generator Model

The function of a steam generator is to remove heat from the primary coolant during the operation of a PWR. Reactor coolant enters the SG hot leg plenum, flows through the SG tubes to the cold leg plenum, and enters the primary system cold leg. While flowing through the tubes, heat is transferred from the primary coolant to the SG shell side and boils the secondary coolant.

The secondary side of the steam generator consists of an evaporative section and a steam drum. The evaporative section contains the U-tubes, and is located in the lower shell, while the steam drum houses the steam separator and dryer equipment. The steam drum section has a larger overall diameter than the evaporative region. There is a flow restrictor at the top of the steam drum where the steam line connects to the SG.

The shell side of the steam generator is modeled as two non-equilibrium regions separated by a moving boundary which is the SG level (see Figure 5.1). The simulation model recognizes the different flow areas of the evaporating and steam drum sections. As the level moves between the two sections, the model accounts for the flow area change when computing SG level. The governing equations for the shell side of the SG are: Conservation of mass in each region, conservation of energy in each region, equation of state, and constant volume constraint.

The SG shell side inventory is normally in a saturated state. There are however transients that may lead to non-equilibrium conditions. The two regions of the non-equilibrium SG model may be in the following thermodynamic states: 1) The lower region (F region) is either subcooled liquid or saturated liquid with bubbles forming and rising to the surface; and 2) The upper region (G region) is either superheated steam or saturated steam with liquid droplets forming and flowing to the liquid region.

5.1.1.1 Governing Equations

The two regions of the steam generator may have four different combinations of thermodynamic states and there is a different set of governing equations for each combination:

- Upper region (G) superheated steam, lower region (F) subcooled liquid,
- Region G superheated steam, region F saturated liquid with bubbles forming,

- Region G saturated steam with droplets forming, region F subcooled liquid, and
- Region G saturated steam with droplets forming, region F saturated liquid with bubbles forming.

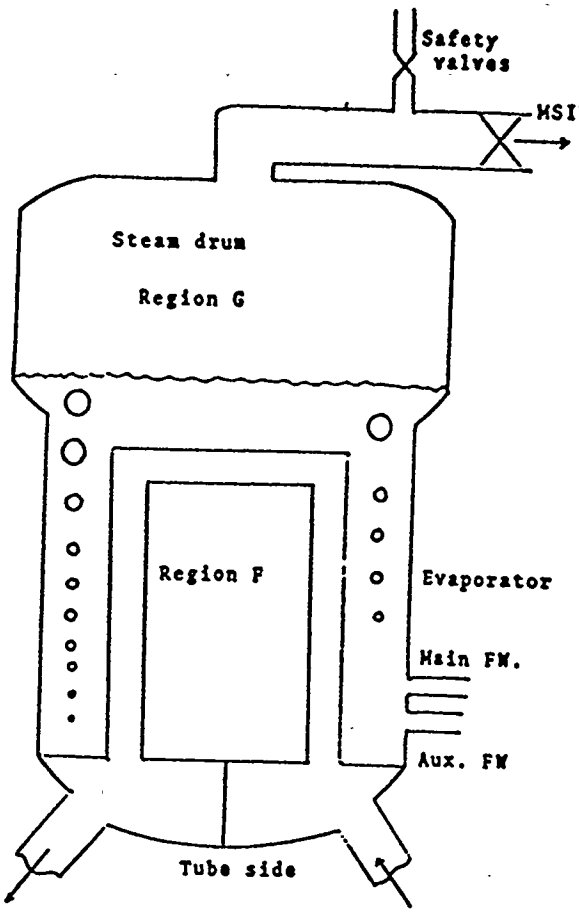


Figure 5.1 : Schematic of the U-tube Steam Generator

The steam generator model accounts for heat and mass transfer between the two regions. Mass transfer is modeled in the bubble rise and condensate drop models. The governing differential equations for each thermodynamic state are derived by first applying the mass and energy equations as well as the equations of state to each region of the steam generator. The resulting equations are analytically reduced until explicit state equations are obtained for all dependent variables (Motamed, 1983). The governing equations for the four possible combinations of states in the steam generator are as follows. All of the parameters in this section are defined in Tables 5.I and 5.II, unless otherwise noted.

a) **State 1: Upper region superheated, lower region subcooled.**

The final form of the governing equations are:

$$\frac{dM_G}{dt} = \sum_i \dot{m}_{Gi} \qquad \text{(Eq. 5.1)}$$

$$\frac{dM_F}{dt} = \sum_i \dot{m}_{Fi} \qquad \text{(Eq. 5.2)}$$

Table 5.I : Notations used in the Equations

Symbol	Meaning
A	Flow area
g	32.2 ft/s^2
h	Enthalpy
J	778 ft-lbf/Btu
M	Mass
$\dot{m}$	Mass flow rate
P	Pressure
t	Time
U	Internal energy
u	velocity
V	Volume
v	Specific volume
α	Void fraction
ρ	Density
τ	Time constant
Q,q	Heat flow rate

Table 5.II : Subscript Notations used in the Equations

Subscript	Meaning
b	Bubble
c	Condensate
cs	Condensate on spray
F	Fluid in the lower region
Ff	Fluid portion in the lower region
Fg	Vapor portion in the lower region
f	Saturated liquid
fg	Saturated liquid to vapor
G	Fluid in the upper region
Gf	Liquid portion in the upper region (condensate)
Gg	Vapor portion in the upper region
g	Saturated vapor
i	Summation convention indicating boundary flows
sp	Spray
HTR	Heater
FG	Interfacial transport
Loss	Indicating heat loss to the environment
in	Indicating flow into
o	Flow out of

$$\frac{dh_G}{dt} = \frac{1}{M_G}\left[\sum_i (\dot{m}h)_{Gi} - h_G \sum_i \dot{m}_{Gi} + \frac{144}{J} V_G \frac{dP}{dt}\right] \qquad \text{(Eq. 5.3)}$$

$$\frac{dh_F}{dt} = \frac{1}{M_F}\left[\sum_i (\dot{m}h)_{Fi} + Q_{in} - h_F \sum_i \dot{m}_{Fi} + \frac{144}{J} V_F \frac{dP}{dt}\right] \qquad \text{(Eq. 5.4)}$$

Where "Q_{in}" is the heat transfer from the tube to the shell side.

$$\frac{dP}{dt} = -\left\{ \nu_F \sum_i \dot{m}_{Fi} + \nu_G \sum_i \dot{m}_{Gi} + \frac{\partial \nu_F}{\partial h_F}\left[\sum_i (\dot{m}h)_{Fi} + Q_{in} - h_F \sum_i \dot{m}_{Fi}\right] + \frac{\partial \nu_G}{\partial h_G}\left[\sum_i (\dot{m}h)_{Gi} - h_G \sum_i \dot{m}_{Gi}\right]\right\} \Bigg/ \left\{ M_F \frac{\partial \nu_F}{\partial P} + \frac{144}{J} V_F \frac{\partial \nu_F}{\partial h_F} + M_G \frac{\partial \nu_G}{\partial P} + \frac{144}{J} V_G \frac{\partial \nu_G}{\partial h_G} \right\} \quad \text{(Eq. 5.5)}$$

$$M_F \nu_F = V_F \quad \text{(Eq. 5.6)}$$

$$M_G \nu_G = V_G \quad \text{(Eq. 5.7)}$$

$$V = V_F + V_G \quad \text{(Eq. 5.7a)}$$

Equations (5.1) to (5.5) are integrated to determine the mass and enthalpy of each SG region. Equation (5.5) is solved for the SG pressure. These equations are simultaneously integrated to calculate the steam generator state. Equations (5.6), (5.7), (5.7a) and the equation of state are used to determine the volume of each region.

b) State 2: Upper region superheated, lower region saturated.

The final form of governing equations for this case are:

$$\frac{dM_G}{dt} = \sum_i \dot{m}_{Gi} \quad \text{(Eq. 5.8)}$$

$$\frac{dM_F}{dt} = \sum_i \dot{m}_{Fi} \quad \text{(Eq. 5.9)}$$

$$\frac{dh_G}{dt} = \frac{1}{M_G}\left[\sum_i (\dot{m}h)_{Gi} - h_G \sum_i \dot{m}_{Gi} + \frac{144}{J} V_G \frac{dP}{dt}\right] \quad \text{(Eq. 5.10)}$$

$$\frac{dM_{Fg}}{dt} = \sum_i \dot{m}_{Fi} - \frac{dM_{Ff}}{dt} \quad \text{(Eq. 5.11)}$$

$$\frac{dM_{Ff}}{dt} = -\frac{1}{h_{fg}}\left[\sum_i (\dot{m}h)_{Fi} + Q_{in} - h_g \sum_i \dot{m}_{Fi} - M_{FH} \frac{dP}{dt}\right] \quad \text{(Eq. 5.12)}$$

Where,

$$M_{FH} = M_{Ff} \frac{dh_F}{dP} + M_{Fg} \frac{dh_g}{dP} - \frac{144}{J} V_F \quad \text{(Eq. 5.13)}$$

$$\frac{dP}{dt} = -\left\{ \nu_{fg} \left[\sum_i (\dot{m}h)_{Fi} + Q_{in} - h_g \sum_i \dot{m}_{Fi} \right] + h_{fg} \left\{ \nu_g \sum_i \dot{m}_{Fi} + \nu_G \sum_i \dot{m}_{Gi} + \left[\sum_i (\dot{m}h)_{Gi} - h_G \sum_i \dot{m}_{Gi} \right] \frac{\partial \nu_G}{\partial h_G} \right\} \right\} / \left\{ h_{fg} \left[M_{Fv} + M_G \frac{\partial \nu_G}{\partial P} + \frac{144}{J} V_G \frac{\partial \nu_G}{\partial h_G} \right] - \nu_{fg} M_{FH} \right\} \quad \text{(Eq. 5.14)}$$

Where,

$$M_{Fv} = M_{Ff} \frac{d\nu_f}{dP} + M_{fg} \frac{d\nu_g}{dP} \quad \text{(Eq. 5.15)}$$

$$M_G \nu_G = V_G \quad \text{(Eq. 5.16)}$$

$$M_{Fg} \nu_g + M_{Ff} \nu_f = V_F \quad \text{(Eq. 5.17)}$$

$$V = V_F + V_G \quad \text{(Eq. 5.18)}$$

Equations (5.8) through (5.15) are numerically solved to obtain the following:

1) Mass of fluid in the upper and lower regions,
2) enthalpy of fluid in the upper region,
3) vapor and liquid masses in the lower region, and
4) SG pressure.

Having calculated the state variables, equations (5.16) through (5.18) and equation of state are used to compute the volume of each region.

c) State 3: Upper region saturated, Lower region subcooled.

The final form of the governing equations are:

$$\frac{dM_G}{dt} = \sum_i \dot{m}_{Gi} \quad \text{(Eq. 5.19)}$$

$$\frac{dM_F}{dt} = \sum_i \dot{m}_{Fi} \quad \text{(Eq. 5.20)}$$

$$\frac{dh_F}{dt} = \frac{1}{M_F} \left[\sum_i (\dot{m}h)_{Fi} + Q_{in} - h_F \sum_i \dot{m}_{Fi} + \frac{144}{J} V_F \frac{dP}{dt} \right] \quad \text{(Eq. 5.21)}$$

$$\frac{dM_{Gg}}{dt} = \sum_i \dot{m}_{Gi} - \frac{dM_{Gf}}{dt} \quad \text{(Eq. 5.22)}$$

$$\frac{dM_{Gf}}{dt} = -\frac{1}{h_{fg}}\left[\sum_i (\dot{m}h)_{Gi} - h_g \sum_i \dot{m}_{Gi} - M_{GH}\frac{dP}{dt}\right] \quad \text{(Eq. 5.23)}$$

Where,

$$M_{GH} = M_{Gf}\frac{dh_f}{dP} + M_{Gg}\frac{dh_g}{dP} - \frac{144}{J}V_G \quad \text{(Eq. 5.24)}$$

$$\frac{dP}{dt} = -\left\{\nu_{fg}\left[\sum_i (\dot{m}h)_{Gi}\right] - h_g \sum_i \dot{m}_{Gi} + h_{fg}\left\{\nu_g \sum_i \dot{m}_{Gi} + \nu_F \sum_i \dot{m}_{Fi} + \left[\sum_i (\dot{m}h)_{Fi} + Q_{in} - h_F \sum_i \dot{m}_{Fi}\right]\frac{\partial \nu_F}{\partial h_F}\right\}\right\} / \left\{h_{fg}\left[M_{G\nu} + M_F\frac{\partial \nu_F}{\partial P} + \frac{144}{J}V_F\frac{\partial V_F}{\partial h_F}\right] - \nu_{fg}M_{GH}\right\} \quad \text{(Eq. 5.25)}$$

Where,

$$M_{G\nu} = M_{Gf}\frac{\partial \nu_f}{\partial P} + M_{Gg}\frac{\partial \nu_g}{\partial P} \quad \text{(Eq. 5.26)}$$

$$M_{Gg}\nu_g + M_{Gf}\nu_f = V_G \quad \text{(Eq. 5.27)}$$

$$M_F \nu_F = V_F \quad \text{(Eq. 5.28)}$$

$$V = V_F + V_G \quad \text{(Eq. 5.28a)}$$

Equations (5.19) through (5.26) are numerically integrated to obtain the following system parameters:

1) Mass within the lower region,
2) enthalpy in the lower region,
3) masses of vapor and liquid in the upper region, and
4) steam generator pressure.

The equation of state and equations (5.27) through (5.28a) are then used to calculate fluid properties and volume of each region.

d) State 4: Upper region saturated, lower region saturated.

The final form of the governing equations are:

$$\frac{dM_G}{dt} = \sum_i \dot{m}_{Gi} \quad \text{(Eq. 5.29)}$$

$$\frac{dM_F}{dt} = \sum_i \dot{m}_{Fi} \qquad \text{(Eq. 5.30)}$$

$$\frac{dM_{Gf}}{dt} + \frac{dM_{Gg}}{dt} = \sum_i \dot{m}_{Gi} \qquad \text{(Eq. 5.31)}$$

$$\frac{dM_{Ff}}{dt} + \frac{dM_{Fg}}{dt} = \sum_i \dot{m}_{Fi} \qquad \text{(Eq. 5.32)}$$

$$\frac{dM_{Gf}}{dt} = -\frac{1}{h_{fg}}\left[\sum_i (\dot{m}h)_{Gi} - h_g \sum_i \dot{m}_{Gi} - M_{GH}\frac{dP}{dt}\right] \qquad \text{(Eq. 5.33)}$$

$$\frac{dM_{Ff}}{dt} = -\frac{1}{h_{fg}}\left[\sum_i (\dot{m}h)_{Fi} - h_f \sum_i \dot{m}_{Fi} + Q_{in} - M_{FH}\frac{dP}{dt}\right] \qquad \text{(Eq. 5 34)}$$

Where,

$$M_{FH} = M_{Ff}\frac{dh_f}{dP} + M_{Fg}\frac{dh_g}{dP} - \frac{144}{J}V_F \qquad \text{(Eq. 5.35)}$$

$$\frac{dP}{dt} = -\left\{\nu_{fg}\left[\sum_i (\dot{m}h)_{Gi} + \sum_i (\dot{m}h)_{Fi} - \left(\sum_i \dot{m}_{Gi} + \sum_i \dot{m}_{Fi}\right)h_g + Q_{in}\right]\right.$$
$$\left. + h_{fg}\left[\nu_g\left(\sum_i \dot{m}_{Gi} + \sum_i \dot{m}_{Fi}\right)\right]\right\}$$
$$/\left\{h_{fg}\left(M_{G\nu} + M_{F\nu}\right) - \nu_{fg}\left(M_{GH} + M_{FH}\right)\right\} \qquad \text{(Eq. 5.36)}$$

Where,

$$M_{GH} = M_{Gf}\frac{dh_f}{dP} + M_{Gg}\frac{dh_g}{dP} - \frac{144}{J}V_G \qquad \text{(Eq. 5.37)}$$

$$M_{G\nu} = M_{Gf}\frac{d\nu_f}{dP} + M_{Gg}\frac{d\nu_g}{dP} \qquad \text{(Eq. 5.38)}$$

$$M_{F\nu} = M_{Ff}\frac{d\nu_f}{dP} + M_{Fg}\frac{d\nu_g}{dP} \qquad \text{(Eq. 5.39)}$$

$$M_{Gg}\nu_g + M_{Gf}\nu_f = V_G \qquad \text{(Eq. 5.40)}$$

$$M_{Fg}\nu_g + M_{Ff}\nu_f = V_F \qquad \text{(Eq. 5.41)}$$

$$V = V_F + V_G \qquad \text{(Eq. 5.41a)}$$

Equations (5.29) through (5.39) constitute the governing equations and are numerically integrated over time to obtain:

1) Mass of vapor and liquid in the upper region,
2) mass of vapor and liquid in the lower region, and
3) steam generator pressure.

As in the previous cases, the equation of state and equations (5.40) through (5.41a) are used to calculate the volumes of each region. . In this case since both regions are saturated, the enthalpy in each region is uniquely defined once mass and pressure are calculated.

Heat transfer from the primary to the secondary side of the steam generator is modeled by forcing functions based on plant data for similar transients. Heat transfer at the interface between the upper and the lower region is not modeled in the simulation code as it is negligible; because the two phases are at or close to thermal equilibrium.

5.1.1.2 Bubble Rise and Condensate Droplet Models

The upper and lower regions of the steam generator are separated by the mixture level. The mixture level representation and the bubble density distribution in this model are the same as Wilson's model (Nahavandi, 1980).

When the SG upper region is in a saturated thermodynamic state, it may contain liquid droplets. The differential equation for the mass flow rate of condensate droplets removed by the steam separator and entering the lower region is approximated by a first order lag. The lag is modeled as a function of the average distance that the droplet has to travel.

5.1.2 Main Steam System

The main steam system in this model consists of the system of pipes and valves between the steam generator and the turbine. The system piping includes the main steam header and main steam line. Valves include the Main Steam Isolation Valve (MSIV), nine Safety Valves (SV), the Turbine Stop Valve, and Turbine Governor Valve. A flow restrictor located at the junction between the steam line and steam generator is also included in the model. The operations of the steam dump and steam bypass systems are not included in this model.

The thermodynamic state of the main steam system is governed by conservation of mass and energy. The equation of motion is applied to determine the flow rate between the steam generator and the steam header (Motamed, 1983). The flow is limited to choked flow conditions. The flow at the flow restrictor, MSIV, the safety valves, and the turbine valve is also governed by the equation of motion and limited to choked flow conditions.

During transients which exceed the capacity of the pressure control system, the steam generator pressure is controlled by a set of nine spring-loaded safety valves. The valve operations are expressed by a set of bistable actions. The model accounts for different lift settings between safety valves to simulate lift and reset sequence.

5.1.3 Main Feedwater and Auxiliary Feedwater Systems

The Main Feedwater System (MFWS) is designed to deliver water to the steam generators during power operations and after reactor trip. For the purpose of this study, feedwater flow delivered to the feedwater regulating and bypass valves are modeled. The feedwater regulating and bypass valves are controlled by the SG level control system, which is discussed later.

The Auxiliary Feedwater System (AFWS) is designed to deliver water to the steam generator upon actuation of the emergency feed signal on low steam generator level. The AFWS flow is controlled by a bistable controller. Its

actuation on low steam generator level is independent of the MFWS. The SG level instruments associated with the AFWS operation are redundant and safety related.

5.1.4 Steam Generator Level Control System

5.1.4.1 Overview

The function of the steam generator level control system is to maintain the water level at a pre-defined set-point (68% narrow range level under normal operating conditions). The system consists of sensors that measure steam generator level (narrow range), steam flow and feed flow, D/A and A/D converters, a digital control software that executes at a clock cycle of 0.1s, and actuators that regulate the position of the main feed valve. The system is implemented as a three-element control system, where measurements of the steam generator level, the steam flow and the feed flow are taken every tenth of a second as sensor inputs to the control software. The software then uses these inputs to generate a target position for the main feed valve. This command is the output to the valve actuators. A schematic of the steam generator control system is shown in Figure 5.2.

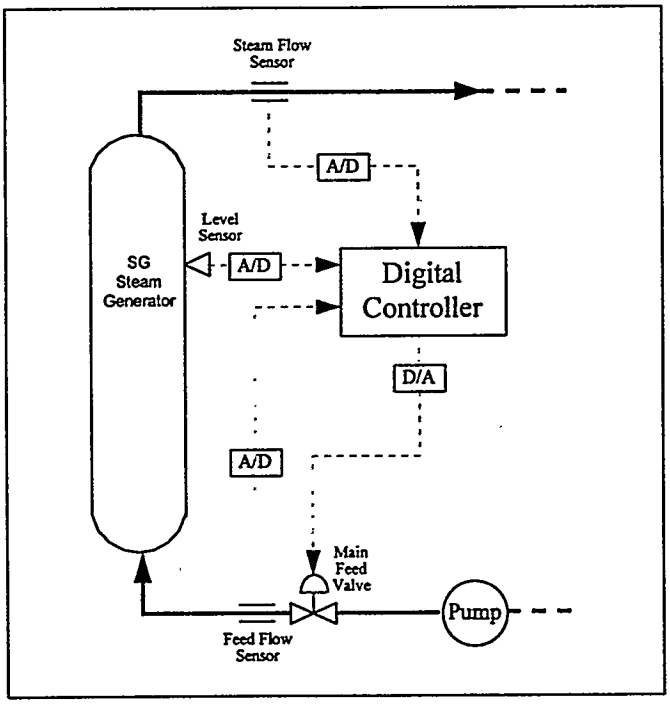


Figure 5.2 : Schematic of the Steam Generator Level Control System

5.1.4.2 Control Logic

Three sets of control logic are implemented by the steam generator control system; they are Proportional Integral and Derivative (PID) logic, High Level Override (HLO) and Reactor Trip Override (RTO). Reactor Trip Override logic is used when the digital control software receives a reactor trip signal, in which case the target main feed valve position is then set to 5%. The reader should note that the valve position would not change to 5% instantaneously, instead it could take as long as 16 seconds (the stroke time). High Level Override logic is employed when the steam generator reading is greater than 89%, in which case the target main feed valve position is set to fully closed. The HLO control action is irreversible; this means that once a HLO signal is triggered, the system will not return to the normal PID control action unless the system is reinitialized. Proportional Integral and Derivative logic is implemented in all cases not covered by the other two sets of control logic. Figure 5.3 shows a block diagram of how the PID logic is implemented.

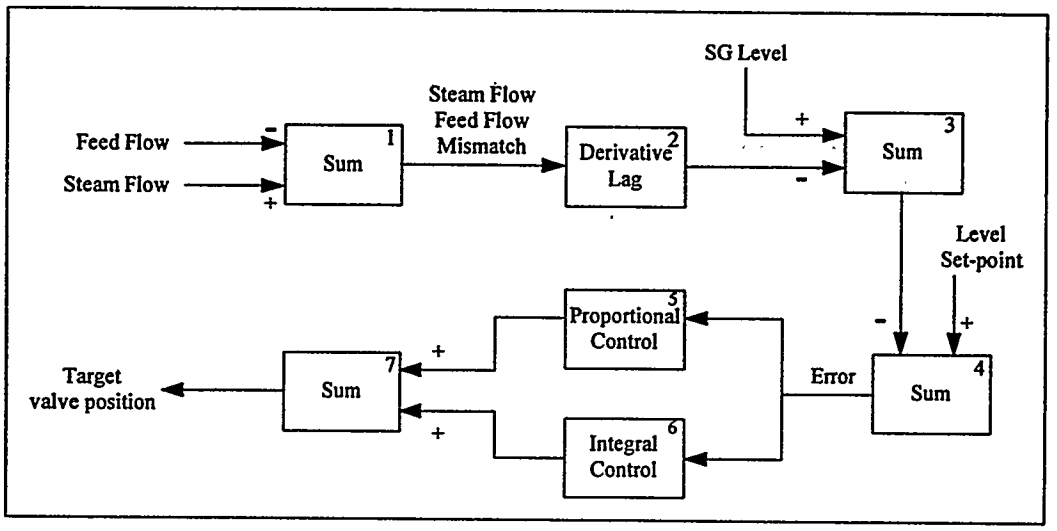


Figure 5.3 : Block Diagram of the PID Control Logic

5.1.5 Testing the Steam Generator Simulation Package

The steam generator simulation code was tested under five different scenarios. The purpose of these tests is to ensure that the simulator accurately reproduces the behavior of an actual steam generator, and thus can be used as a realistic case study for DFM. The five test scenarios are listed below, and they are events that can be encountered by a real steam generator control system.

1. Steady state
2. Turbine trip
3. Level sensor failure
4. Step power reduction
5. Ramp power reduction

For each test run, the steam generator level, the steam flow, the feed flow, the steam generator pressure and the auxiliary feed flow were monitored over a period of several minutes. The plots of these parameters are shown in the sections that follow. The results of these tests were examined by a utility plant simulator expert and compared with actual records, and were found to be consistent with actual plant transients.

5.1.5.1 Steady State

The simulation code was tested under steady state conditions. The initial steam generator level was set at 68% narrow range, the initial steam flow was set at 100%, the initial feed flow was set at 100% and the initial steam generator pressure was set at 1000 psi. The parameters were monitored for 400 seconds and the plots are shown in Figures 5.4 - 5.7.

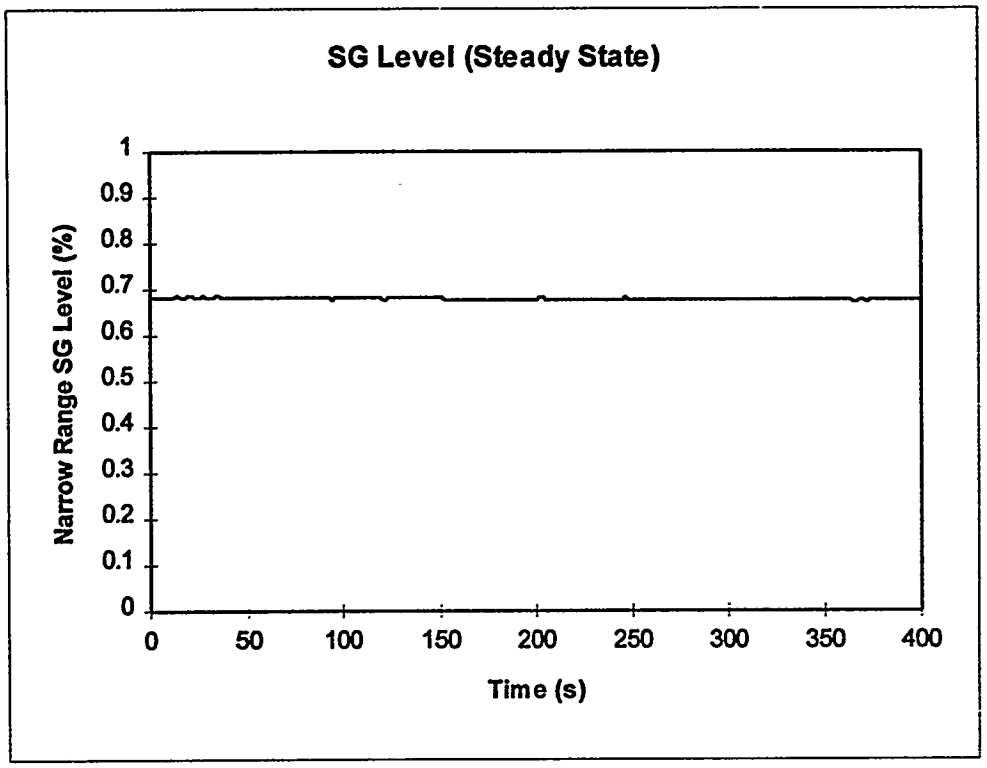


Figure 5.4 : Variation of the Narrow Range SG Level in Steady State

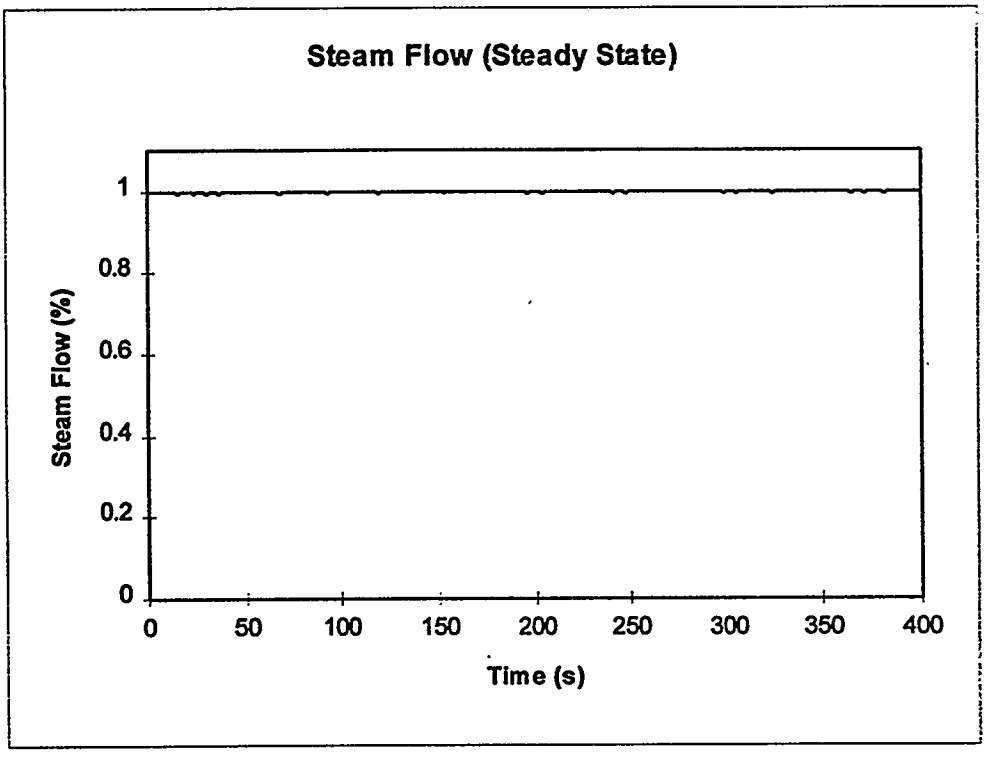


Figure 5.5 : Variation of the Steam Flow in Steady State

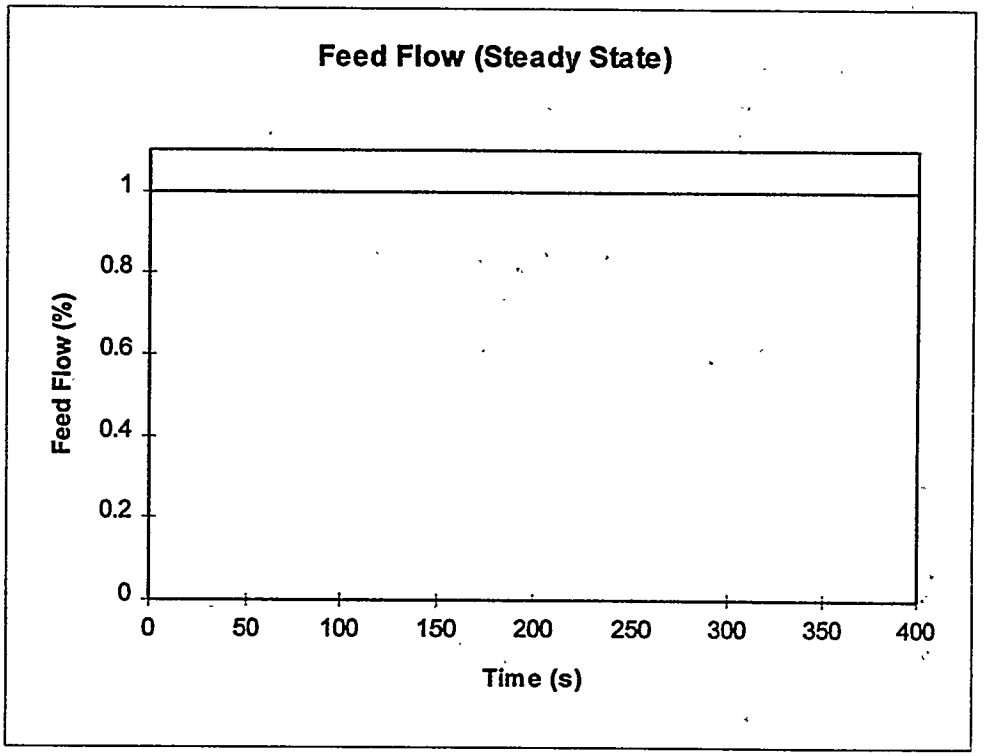


Figure 5.6 : Variation of the Feed Flow in Steady State

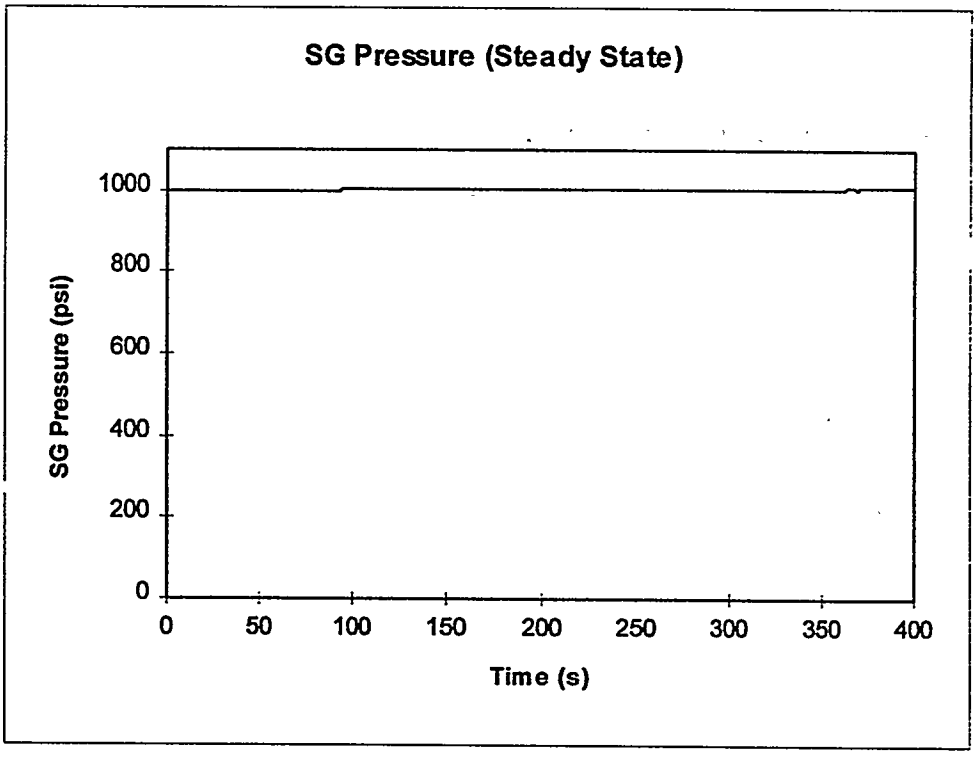


Figure 5.7 : Variation of the SG Pressure in Steady State

5.1.5.2 Turbine Trip

The simulation code was tested for its turbine trip response. The initial conditions for level, steam flow, feed flow and pressure were the same as in the steady state case. A turbine trip signal was generated at 0.2 sec after the start of the simulation. The parameters were monitored for 400 seconds and the plots are shown in Figures 5.8 - 5.12.

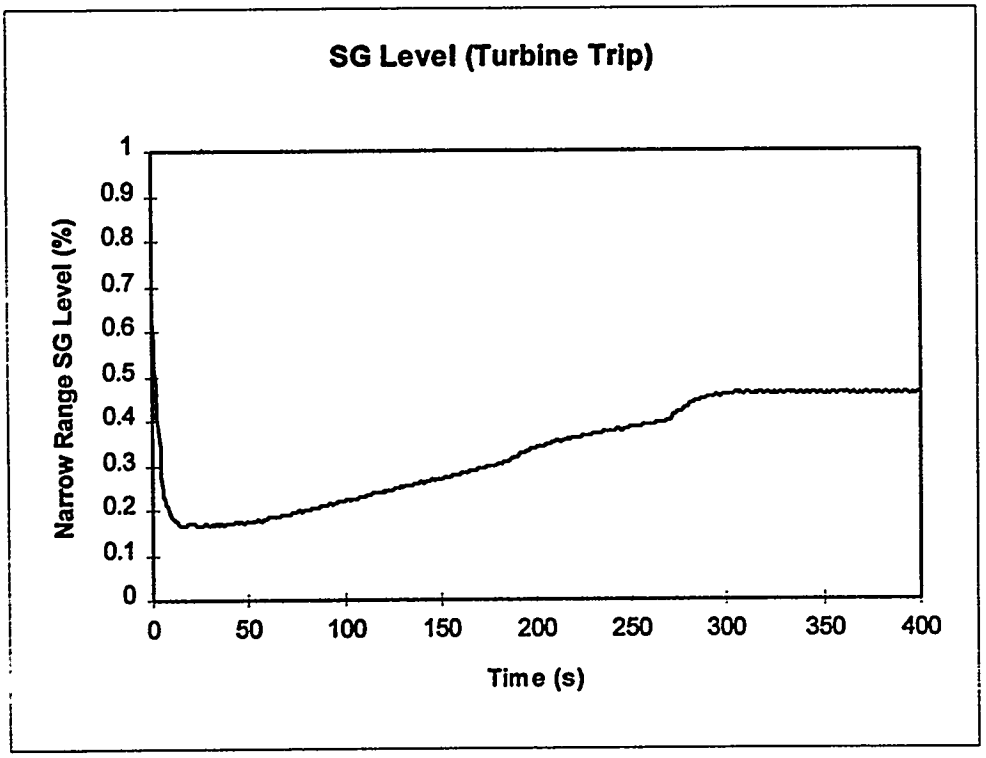


Figure 5.8 : Variation of the Narrow Range SG Level After the Turbine Has Tripped

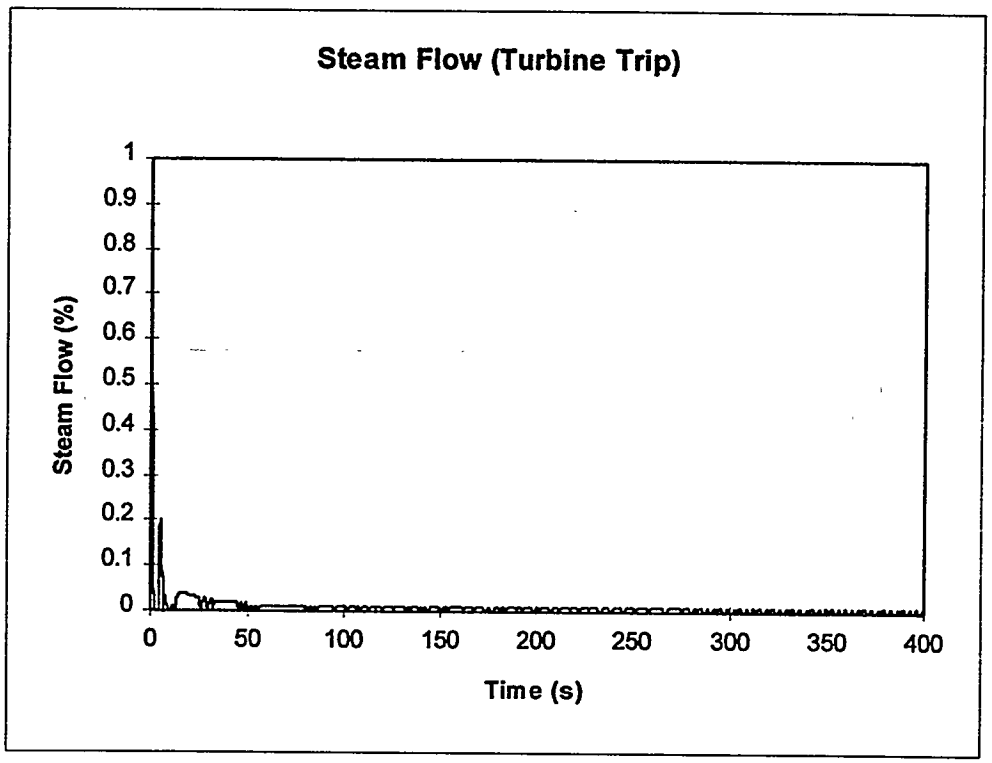


Figure 5.9 : Variation of the Steam Flow After the Turbine Has Tripped

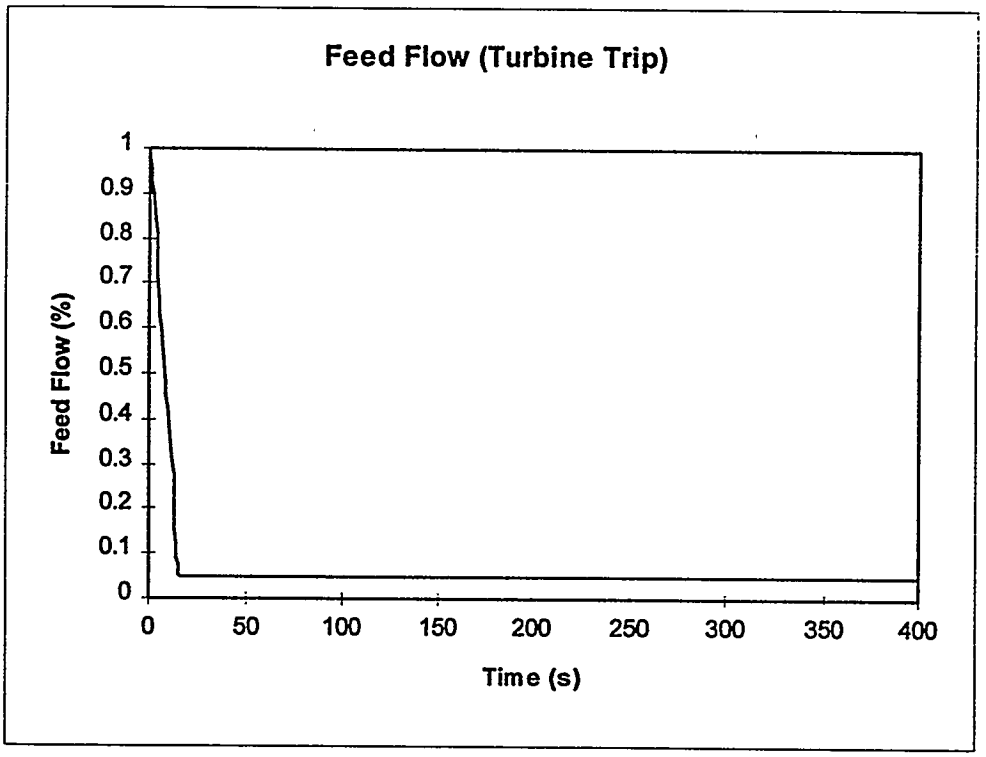


Figure 5.10 : Variation of the Feed Flow After the Turbine Has Tripped

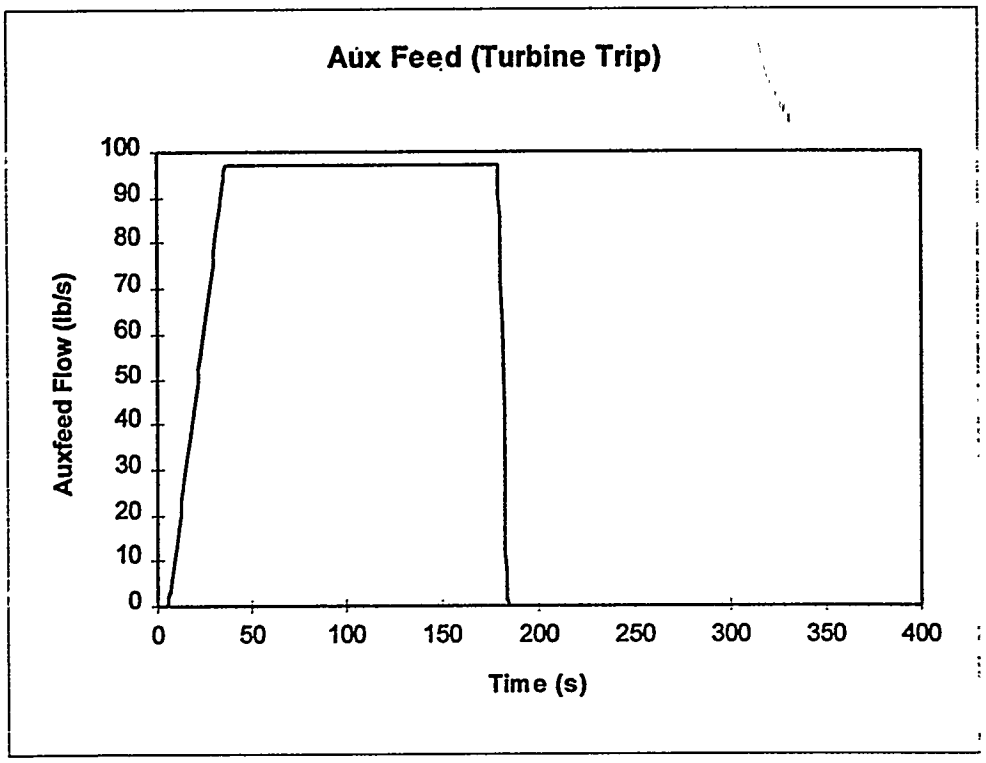


Figure 5.11 : Variation of the Auxiliary Feed Flow After the Turbine Has Tripped

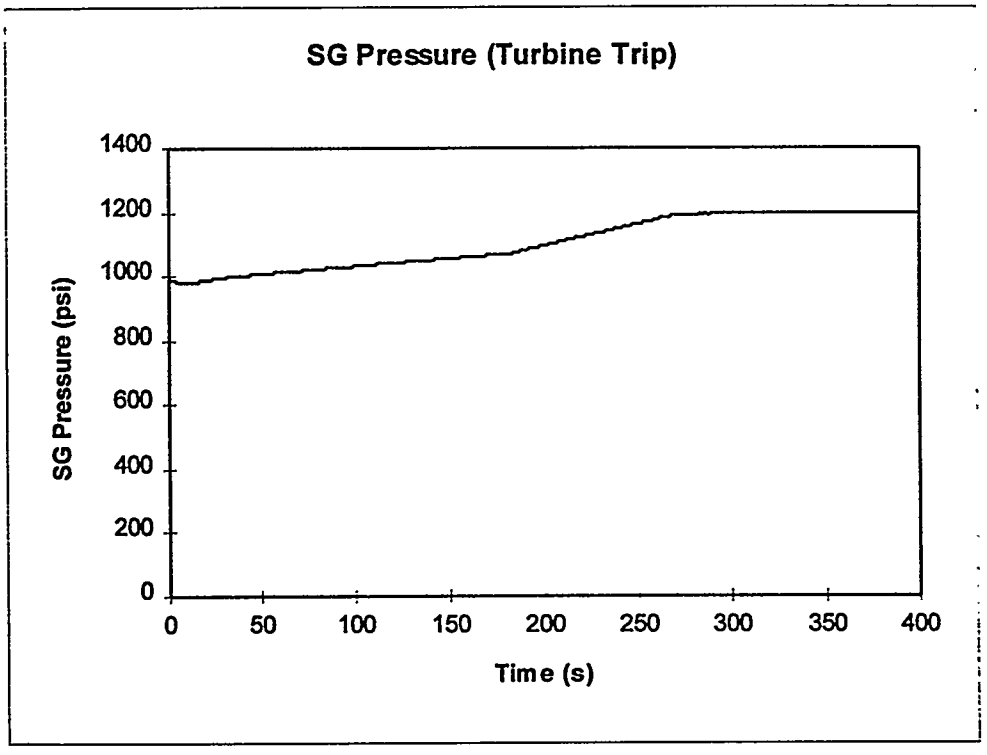


Figure 5.12 : Variation of the SG Pressure After the Turbine Has Tripped

5.1.5.3 Level Sensor Failure

The simulation code was executed under the condition of a level sensor failure. The initial conditions were the same as in the steady state case. The sensor failed stuck high at 0.5 second after the start of the simulation and the reading remained at 95%. The parameters were monitored for 75 seconds and the plots are shown in Figures 5.13 - 5.17.

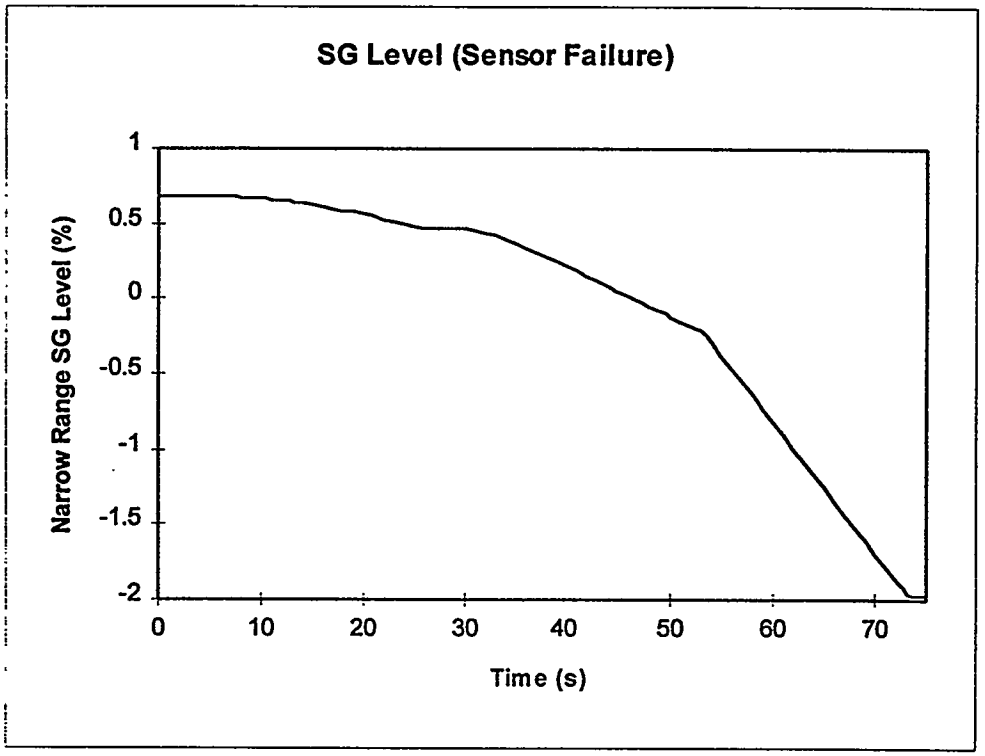


Figure 5.13 : Variation of the Narrow Range SG Level After the Level Sensor Has Failed

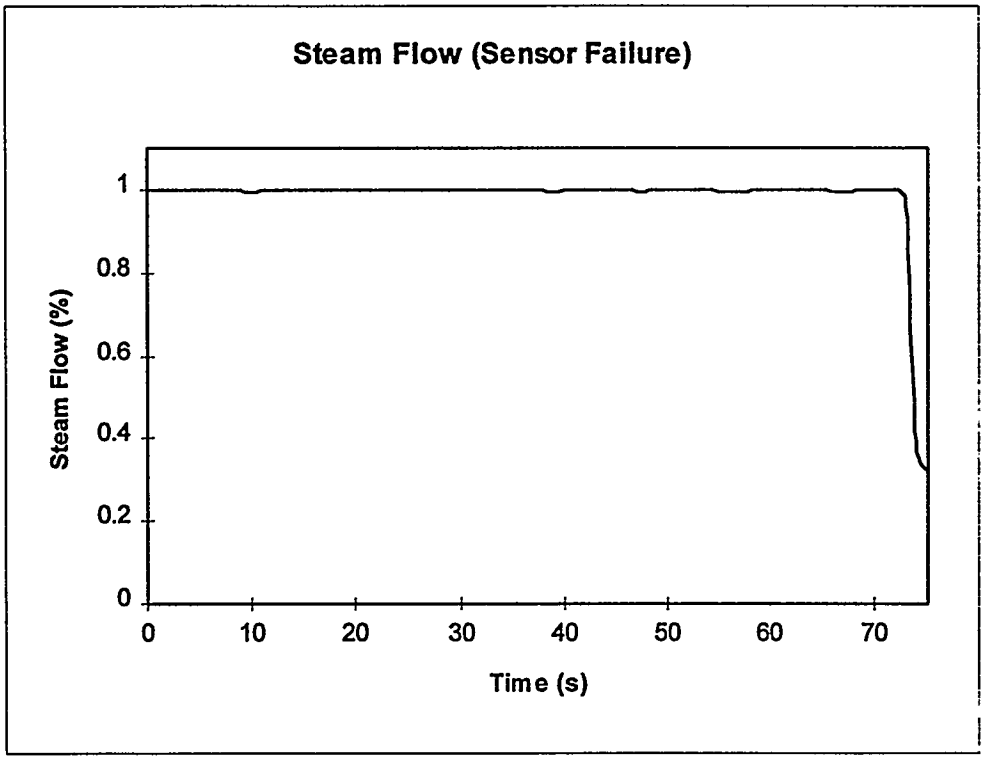


Figure 5.14 : Variation of the Steam Flow After the Level Sensor Has Failed

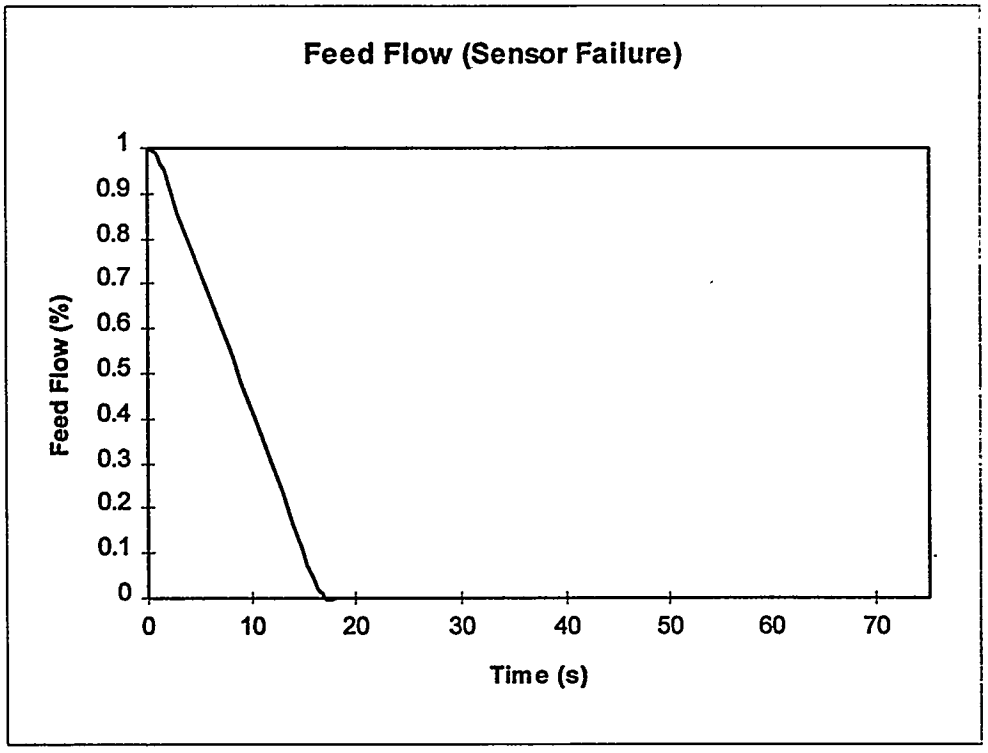


Figure 5.15 : Variation of the Feed Flow After the Level Sensor Has Failed

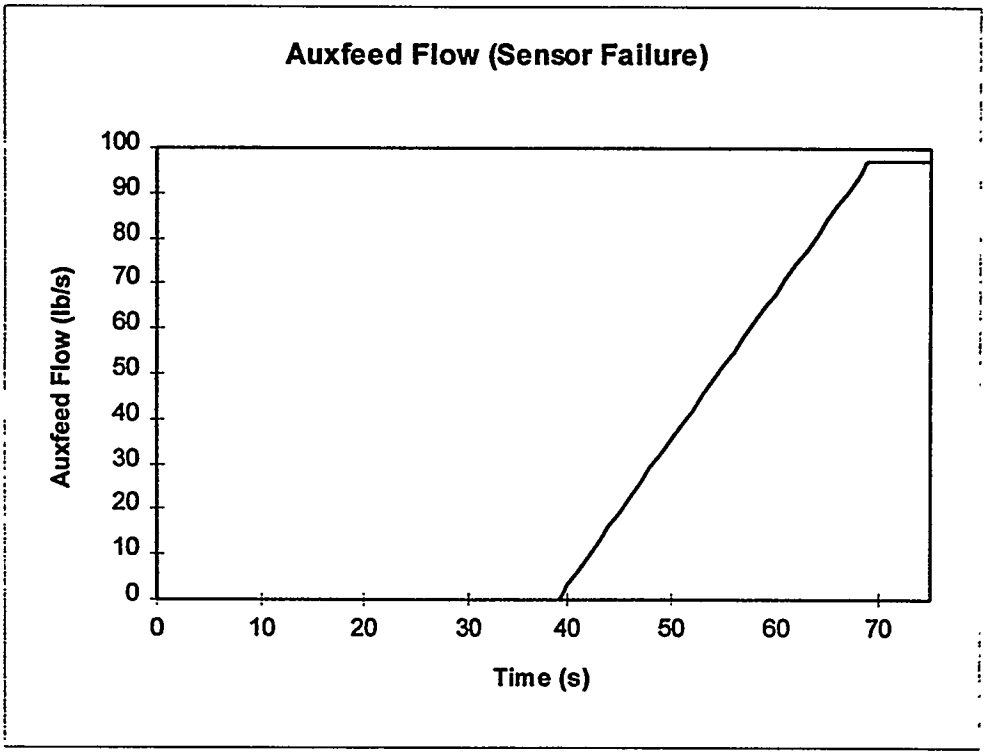


Figure 5.16 : Variation of the Auxiliary Feed Flow After the Level Sensor Has Failed

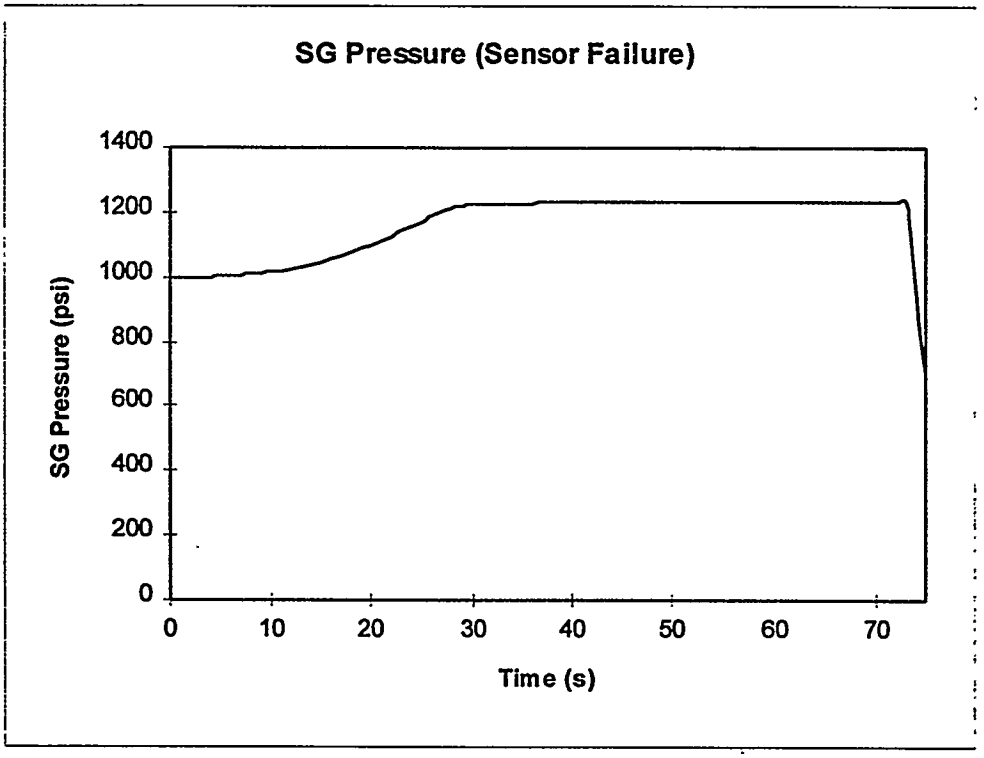


Figure 5.17 : Variation of the SG Pressure After the Level Sensor Has Failed

5.1.5.4 Step Power Reduction

The simulation code was tested under the condition of a step load change. The initial conditions were the same as in the steady state case. Power was reduced from 100% at 2 s to 80% at 7 s. The parameters were monitored for 1400s and the plots are shown in Figures 5.18 - 5.21.

5.1.5.5 Ramp Power Reduction

The simulation code was executed under the condition of a ramp load change. The initial conditions were the same as in the steady state case. Power was reduced from 100% at 2 s to 50% at 352 s. The parameters were monitored for 1400 seconds and the plots are shown in Figures 5.22 - 5.25

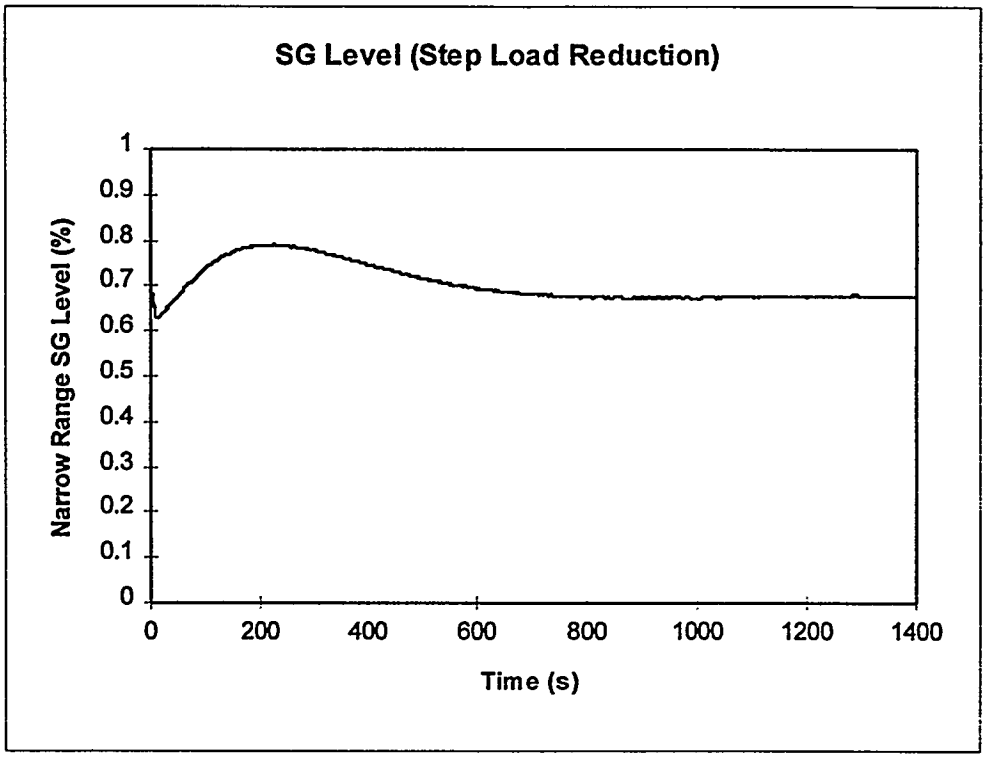


Figure 5.18 : Variation of the Narrow Range SG Level During Step Power Reduction

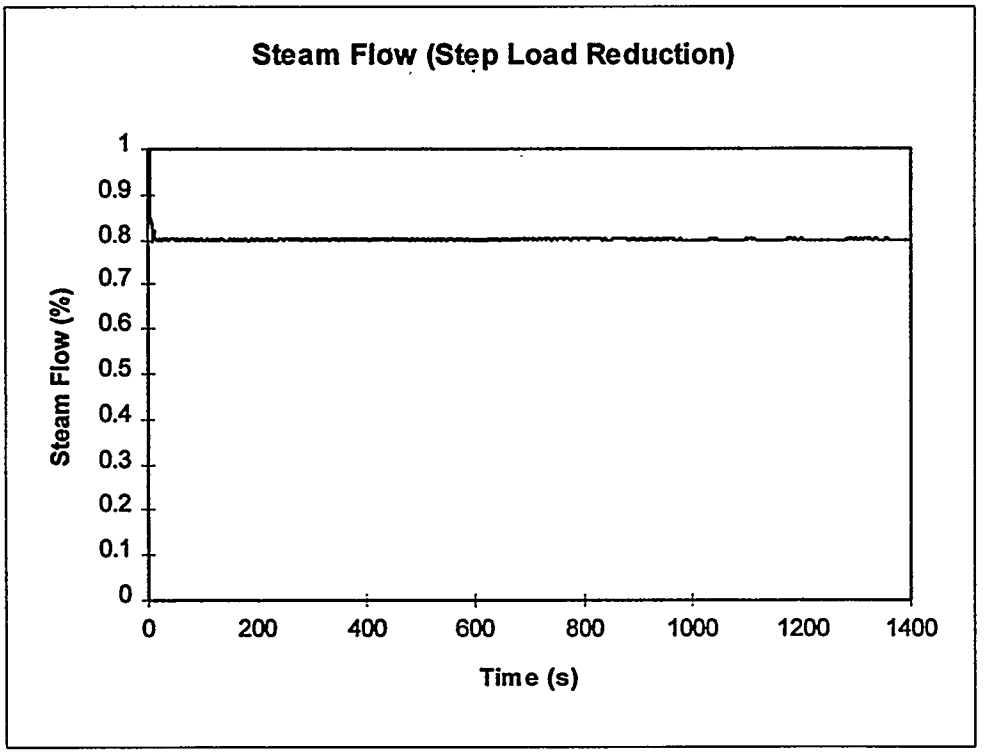


Figure 5.19 : Variation of the Steam Flow During Step Power Reduction

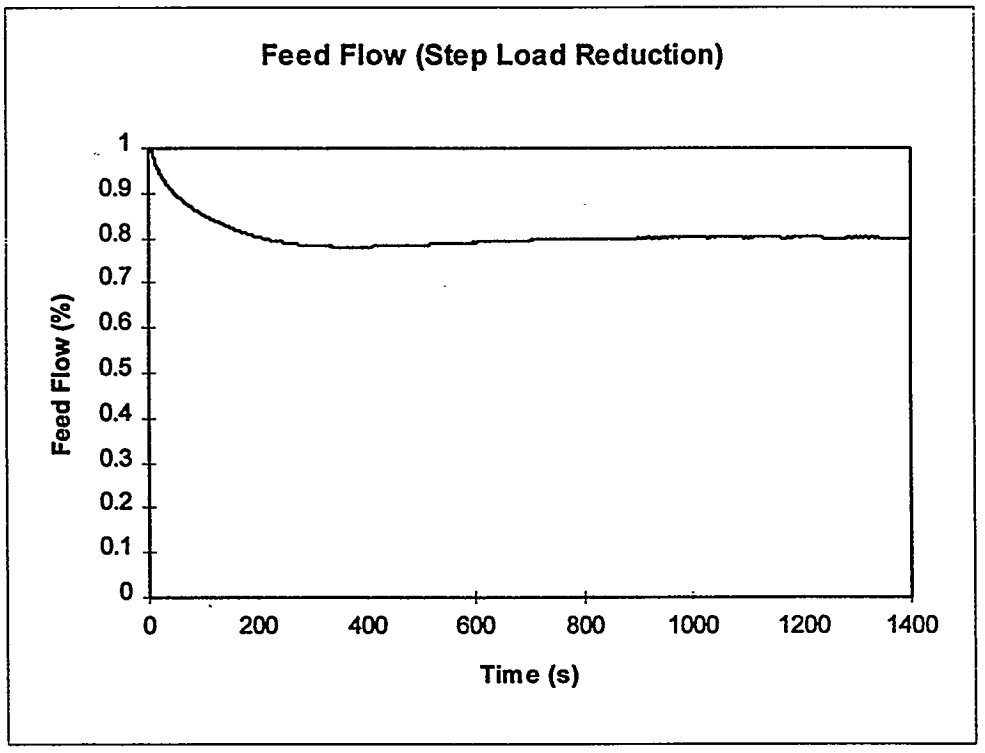


Figure 5.20 : Variation of the Feed Flow During Step Power Reduction

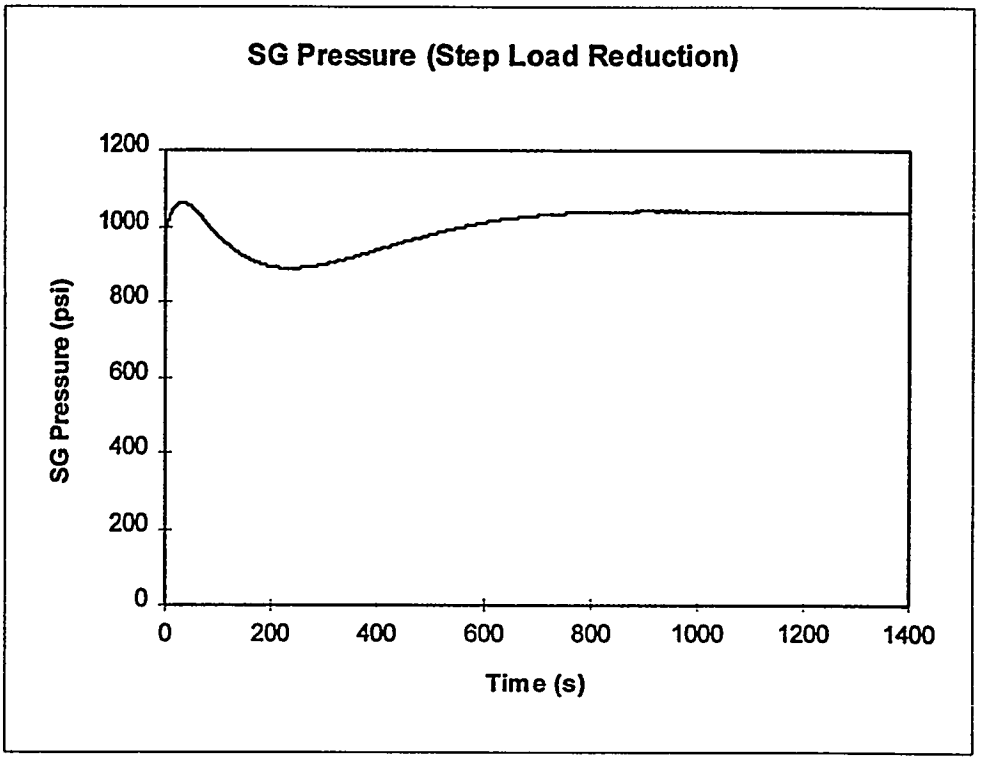


Figure 5.21 : Variation of the SG Pressure During Step Power Reduction

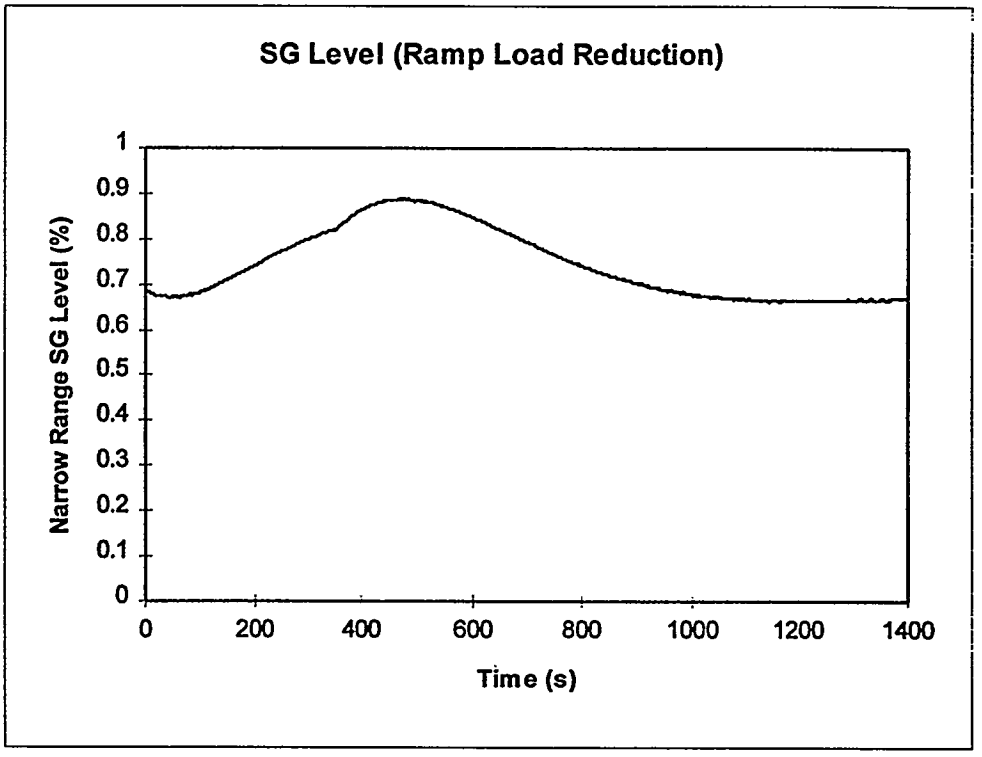


Figure 5.22 : Variation of the Narrow Range SG Level During Ramp Power Reduction

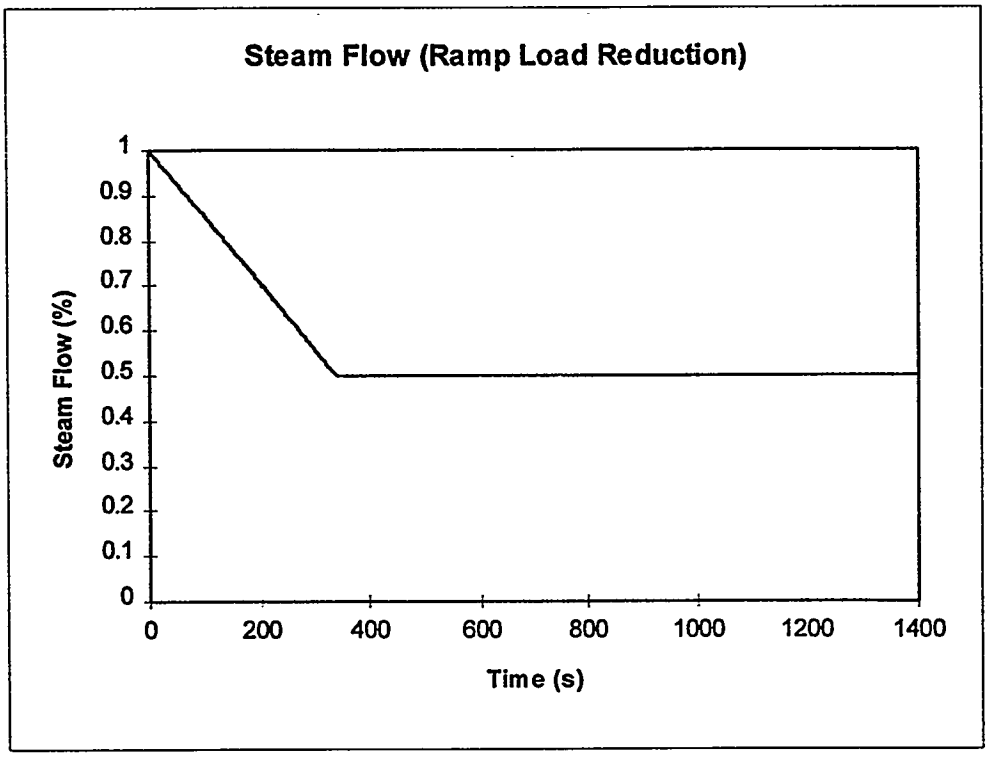


Figure 5.23: Variation of the Steam Flow During Ramp Power Reduction

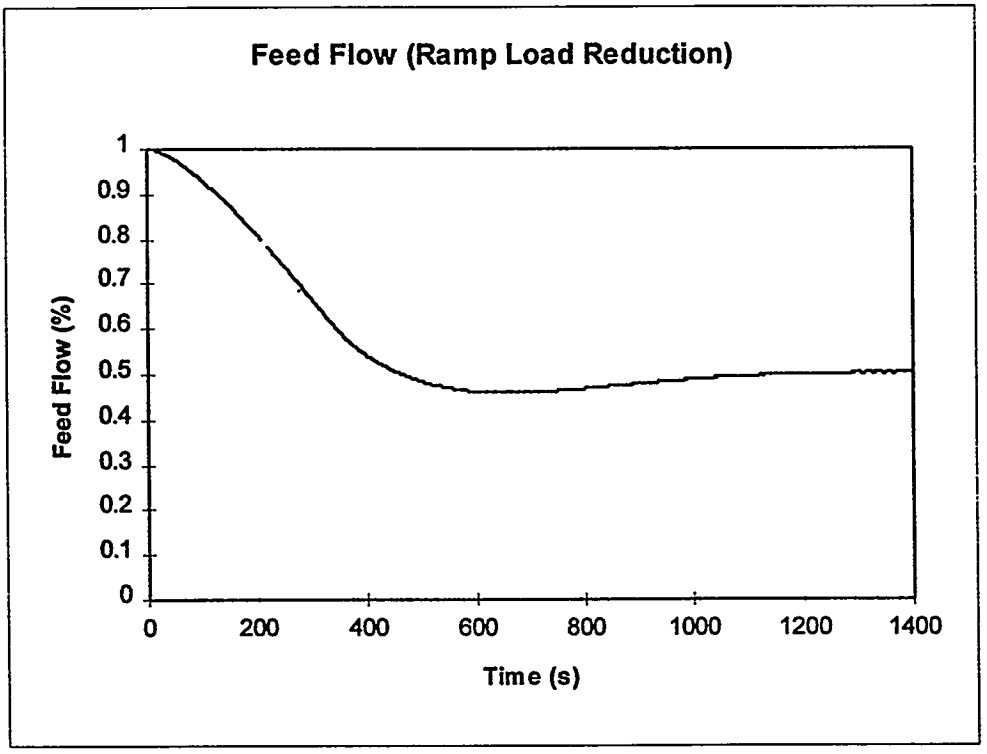


Figure 5.24: Variation of the Feed Flow During Ramp Power Reduction

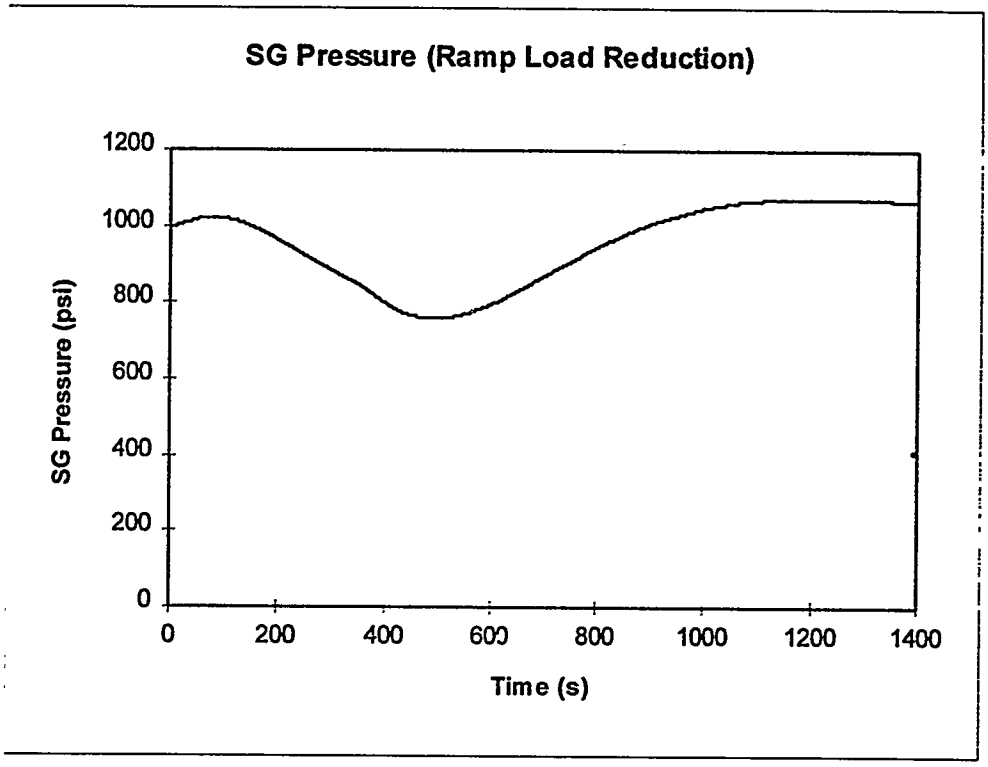


Figure 5.25: Variation of the SG Pressure During Ramp Power Reduction

5.2 DFM Model of the DTC System

This section describes the DFM model that was constructed to represent the DTC system. In building the DFM model for this system, certain standard assumptions were made regarding the possible failure modes of the individual system hardware components. They are:

- The steam flow sensor, the feed flow sensor and the SG level sensor can fail high or fail low.
- The main feed valve can fail closed or fail open.

The DFM model of the DTC system was constructed by following the steps outlined in Section 2.2, and is shown in Figure 5.26. The control software is shown as a black box in this figure, but is expanded in full detail in Figure 5.27. The description of the variables that appear in the model as DFM "nodes" can be found in Table 5.III.

Some of the features represented in the DFM model of Figure 5.26 are listed below:

- Transfer boxes 26, 27 and 18 represent the level sensor, the feed flow sensor and the steam flow sensor respectively.
- Transfer box 28 shows the actuation of the auxiliary feed flow.
- Transfer box 16 models the variation of the water level in the steam generator.
- Transfer box 20 represents the variation of the pressure in the steam generator.
- Transfer box 25 models the actuator of the main feed valve.

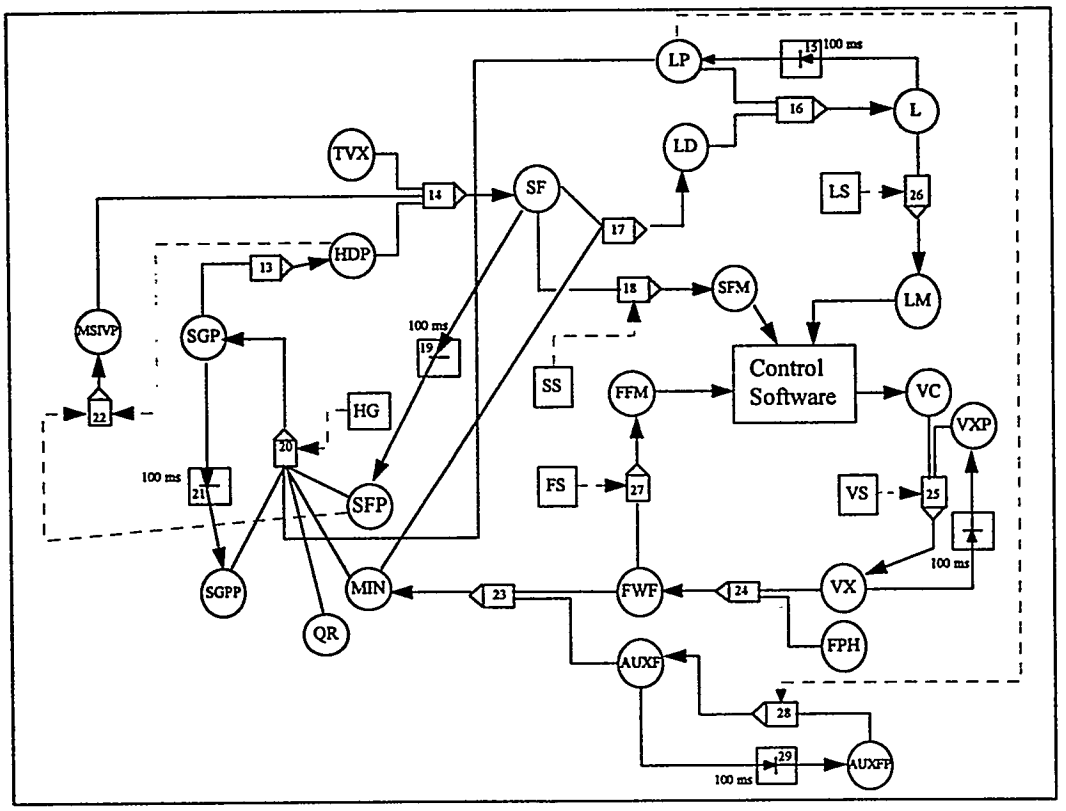


Figure 5.26 : DFM Model of the DTC System

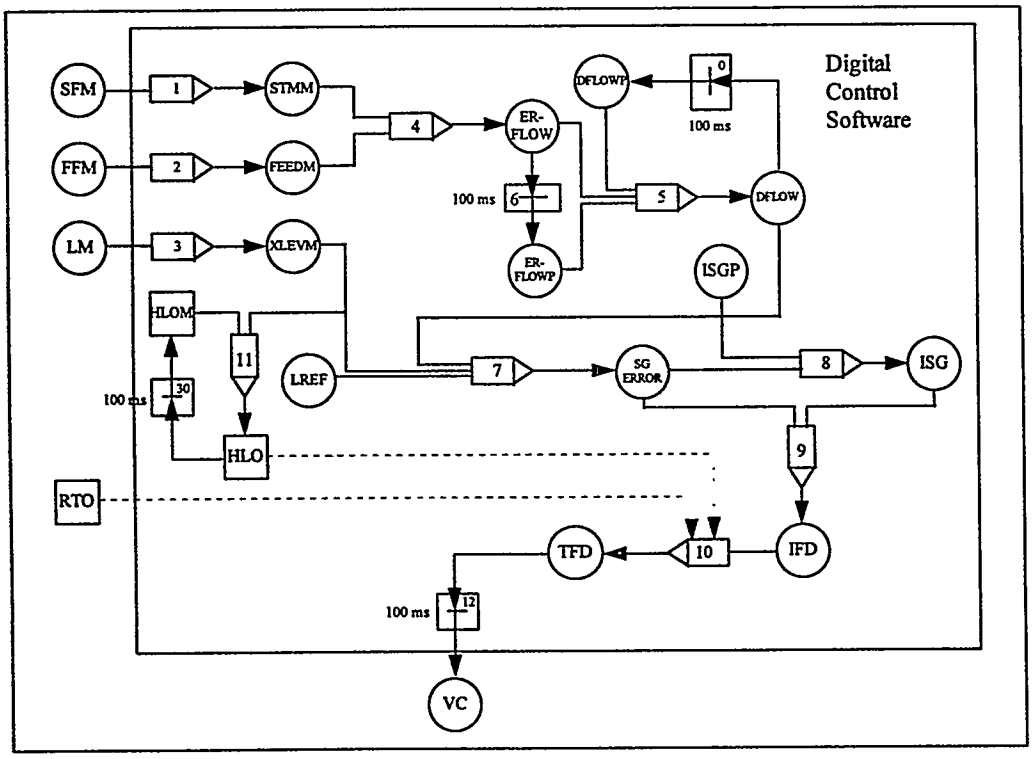


Figure 5.27 : DFM Model of the DTC Control Software

While the features included in the DFM model of Figure 5.27 are as follows:

- Transfer boxes 1, 2 and 3 model the A/D converters for the steam flow signal, the feed flow signal and the level signal respectively.
- Transfer boxes 4 and 5 represent the software module that calculates the derivative-lag term for the steam flow/feed flow mismatch.
- Transition boxes 0 and 6 show the updating of the steam flow/feed flow mismatch and the derivative-lag term.
- Transfer box 7 models the software module that generates the error term for controlling the steam generator level.
- Transfer boxes 8 and 9 represent the software module that generates the intermediate flow demand signal.
- Transfer box 11 and transition box 30 model the activation of the High Level Override signal.
- Transfer box 10 shows how the final flow demand signal can be overridden by the Reactor Trip Override and High Level Override signals.
- Transition box 12 represents the D/A converter that converts the final flow demand signal to a command for the main feed valve position.

The nodes in the DFM model are discretized into finite number of states. The discretization schemes are shown in Tables 5.IV - 5.XXVI. These schemes reflect the knowledge about the system and assumptions regarding the failure modes of the components. For example, the reasoning behind the discretization scheme for L (the steam generator water level), shown in Table 5.XIV, is listed as follows:

- If the level goes below 25%, the auxiliary feed flow will be turned on.
- When the level rises back above 30%, the auxiliary feed flow will be turned off.
- The low level alarm will be triggered if the level drops below 40%.
- 68% is the control reference point.
- The high level alarm will be activated if the level rises above 87%.
- If the level goes beyond 89%, the High Level Override signal is activated.
- 90% level is the high level trip point.
- The level between 40% and 87% is further discretized into a number of states to represent slight and moderate deviations from the control set-point.

On the other hand, the discretization scheme for FS (the state of the feed flow sensor) shown in 5.VIII follows from the assumption regarding the failure mode of this component.

Table 5.III : Description of the Variables in the DFM Model

Variable	Description
AUXF	Auxiliary feedwater flowrate
AUXFP	Auxiliary feedwater flowrate in the previous controller clock cycle
DFLOW	Software representation of the derivative-lag control term
DFLOWP	Value of DFLOW in the previous controller clock cycle
ERFLOW	Software representation of the steam flow/feed flow mismatch
ERFLOWP	Value of ERFLOW in the previous controller clock cycle
FEEDM	Software representation of the feedwater flowrate
FFM	Measurement of the feedwater flowrate
FS	State of the feedwater flow sensor
FWF	Feedwater flowrate
HDP	Steam header pressure
HG	State of vapor at the top of the SG
HLO	High Level Override signal
HLOM	High Level Override signal in the digital controller memory
IFD	Intermediate flow demand signal
ISG	Integral control term for level in the software
ISGP	Integral control term for level in the previous controller clock cycle
L	Steam generator level (narrow range)
LD	Change in the SG level
LM	Measurement of the SG level (narrow range)
LP	SG level in the previous controller clock cycle
LREF	SG level set-point used in the software
LS	State of the SG level sensor
MIN	Total liquid mass flowrate into the SG
MSIVP	Main Steam Isolation Valve position
QR	Heat transfer from the primary side
RTO	Reactor Trip Override signal
SF	Steam flowrate
SFM	Measurement of the steam flowrate
SFP	Steam flowrate in the previous controller clock cycle
SGERROR	Software representation of the SG level mismatch
SGP	Steam generator pressure
SGPP	SG pressure in the previous controller clock cycle
SS	State of the steam flow sensor
STMM	Software representation of the steam flowrate
TFD	Final flow demand signal
TVX	Turbine governing valve position
VC	Valve command
VS	State of the main feed valve
VX	Main feed valve position
VXP	Main feed valve position in the previous controller clock cycle
XLEVM	Software representation of the SG level (narrow range)

Table 5.IV : Discretization of AUXF and AUXFP

State	Description
0	Off
1	0% - 50%
2	50% - 100%

Table 5.V : Discretization of DFLOW and DFLOWP

State	Description
-2	Largely negative
-1	Slightly negative
0	Close to zero
+1	Slightly positive
+2	Largely positive

Table 5.VI : Discretization of ERFLOW and ERFLOWP

State	Description
-2	Largely negative
-1	Slightly negative
0	Close to zero
+1	Slightly positive
+2	Largely positive

Table 5.VII : Discretization of FEEDM, FFM and FWF

State	Description
0	0% - 5%
1	5% - 30%
2	30% - 60%
3	60% - 80%
4	80% - 100%

Table 5.VIII : Discretization of FS, LS and SS

State	Description
-1	Failed Low
0	Normal
+1	Failed High

Table 5.IX : Discretization of HDP

State	Description
0	Very Low
1	Low
2	Normal
3	High

Table 5.X : Discretization of HG

State	Description
0	Superheated
1	Saturated

Table 5.XI : Discretization of HLO and HLOM

State	Description
0	Signal is active
1	Signal is inactive

Table 5.XII : Discretization of IFD

State	Description
1	0% - 25%
2	25% - 55%
3	55% - 75%
4	75% - 100%

Table 5.XIII : Discretization of ISG and ISGP

State	Description
-1	Negative
0	Close to zero
+1	Positive

Table 5.XIV : Discretization of L, LM, LP and XLEVM

State	Description
0	0% - 25%
1	25% - 30%
2	30% - 40%
3	40% - 55%
4	55% - 65%
5	65% - 71%
6	71% - 78%
7	78% - 87%
8	87% - 89%
9	89% - 90%
10	90% - 100%

Table 5.XV : Discretization of LD

State	Description
-1	-3% to -1%
0	-1% to 1%
+1	+1% to +3%

Table 5.XVI : Discretization of MIN

State	Description
0	0% - 5%
1	5% - 30%
2	30% - 60%
3	60% - 80%
4	80% - 100%

Table 5.XVII : Discretization of MSIVP

State	Description
0	Fully Closed
1	Closing or opening
2	Fully Open

Table 5.XVIII : Discretization of QR

State	Description
0	0% - 10% power
1	10% - 50% power
2	50% - 90% power
3	90% - 100% power

Table 5.XIX : Discretization of RTO

State	Description
0	Signal is active
1	Signal is inactive

Table 5.XX : Discretization of SF and SFP

State	Description
0	0% - 5%
1	5% - 30%
2	30% - 60%
3	60% - 80%
4	80% - 100%
5	100%+

Table 5.XXI : Discretization of SFM and STMM

State	Description
0	0% - 5%
1	5% - 30%
2	30% - 60%
3	60% - 80%
4	80% - 100%

Table 5.XXII : Discretization of SGERROR

State	Description
-3	Largely negative
-2	Negative
-1	Slightly negative
0	Close to zero
+1	Slightly to largely positive

Table 5.XXIII : Discretization of SGP and SGPP

State	Description
0	Very low
1	Low
2	Normal
3	High

Table 5.XXIV : Discretization of TFD, VC, VX and VXP

State	Description
0	0% - 5%
1	5% - 30%
2	30% - 60%
3	60% - 80%
4	80% - 100%

Table 5.XXV : Discretization of TVX

State	Description
0	Fully closed
1	Closing or opening
2	Fully open

Table 5.XXVI : Discretization of VS

State	Description
-1	Stuck closed
0	Normal
+1	Stuck open

Decision tables were constructed to complete the definition of this DFM model. The decision tables for the physical components were built by running the corresponding subroutines in the simulation code. For instance, Table 5.XXVII shows the decision table for transfer box 28 in Figure 5.26. It shows the actuation of the auxiliary feedwater system. In particular:

- If LP (previous SG level) is in state 0, the inactive auxiliary feedwater system will be switched on (AUXFP = 0 → AUXF = 1). On the other hand, if the auxiliary feedwater system has been turned on, the auxiliary feed flow will try to reach 100% (AUXFP = 1,2 → AUXF = 2).
- If LP is in state 1, the inactive auxiliary feedwater system will remain inactive (AUXFP = 0→ AUXF = 0). On the other hand, if the auxiliary feedwater has previously been triggered, the auxiliary feed flow again will try to reach 100% (AUXFP = 1,2 → AUXF = 2).
- If LP is in state 2, the inactive auxiliary feedwater system will remain inactive (AUXFP = 0→ AUXF = 0). Whereas, if the auxiliary feedwater has previously been triggered, it will be gradually shut down (AUXFP = 1 → AUXF = 0, AUXFP = 2 → AUXF = 1).
- If LP is in any state at or above state 3, the auxiliary feedwater system will remain inactive.

Table 5.XXVII : Decision Table for Transfer Box 28

LP	AUXFP	AUXF
0	0	1
0	1	2
0	2	2
1	0	0
1	1	2
1	2	2
2	0	0
2	1	0
2	2	1
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0
10	0	0

As in the study of the Interim Test Case, the decision tables for the system software were constructed by executing off-line, module by module, the control software that is actually implemented. The observation regarding the relationship between this activity and "module testing", which was made in Sections 2.1, 2.2 and 4.3, is still valid in this case.

Table 5.XXVIII shows the decision table for transfer box 10 in the DFM model of Figure 5.27. It includes the information that:

- If neither the High Level Override signal nor the Reactor Trip Override signal is active (HLO = 1 and RTO = 1), there is a one-to-one correspondence between the states of IFD (intermediate flow demand signal) and those of TFD (final flow demand signal).
- If either the High Level Override signal or the Reactor Trip Override signal is active (HLO = 0 or RTO = 0), no matter what the intermediate flow demand signal is, the final flow demand signal will always be the minimum state (IFD = - → TFD = 0). The "-" is a "don't care" symbol and it indicates that IFD can be at any state.

Table 5.XXVIII : Decision Table for Transfer box 10

HLO	RTO	IFD	TFD
1	1	1	1
1	1	2	2
1	1	3	3
1	1	4	4
0	1	-	0
1	0	-	0
0	0	-	0

5.3 DTC DFM Model Analysis

This section discusses the analyses performed on the DFM model of the DTC system. To test the capability of DFM in a system and software assurance mode of application, two different faults were intentionally injected into the control software. We present here the results of these "faulted-case analyses" to illustrate the capability of DFM for identifying and isolating software errors in such a mode of application. The first fault was injected into the software specification and the corresponding analysis is presented in Section 5.3.1, while the second fault was injected as a programming error in the software code and the corresponding analysis is discussed in Section 5.3.2. A number of analyses were also performed on the original unfaulted system, the results of which did not point to any unexpected errors. Since these analyses are judged to be of little interest to the reader, they are left out of the discussions in this section.

5.3.1 The First Faulted-Case Analysis

5.3.1.1 Description of the Fault Injected

For this first faulted-case analysis, it was assumed that an error had been initially introduced into the design specification of the control software. The assumption was that, instead of subtracting the derivative-lag signal of the steam flow-feed flow mismatch from the steam generator level, as shown in Box 3 in Figure 5.3, the faulted specification called for the addition of these two terms. The software developed from the faulted specification had thus an inherent fault. This fault can be triggered into an execution error if there exists a significant steam flow-feed flow mismatch, comparable in magnitude to the level mismatch. A comparison of the original specification and the faulted specification is shown in Figure 5.28, and a comparison of the corresponding software modules developed from these specifications is shown in Figure 5.29.

The DFM model was constructed without assuming any prior knowledge of the software specification error. This is possible because, as in the ITC case discussed in Section 4.4.1, the decision tables were built directly by executing the individual modules of the digital control software. Figure 5.30 shows the difference between the decision tables for the correct software and the faulted software. Note that the decision table on the left hand side (corresponding to the correct version) is produced by testing the module shown at the top half of Figure 5.29, while the decision table on the right hand side is generated by executing the module at the bottom half of Figure 5.29.

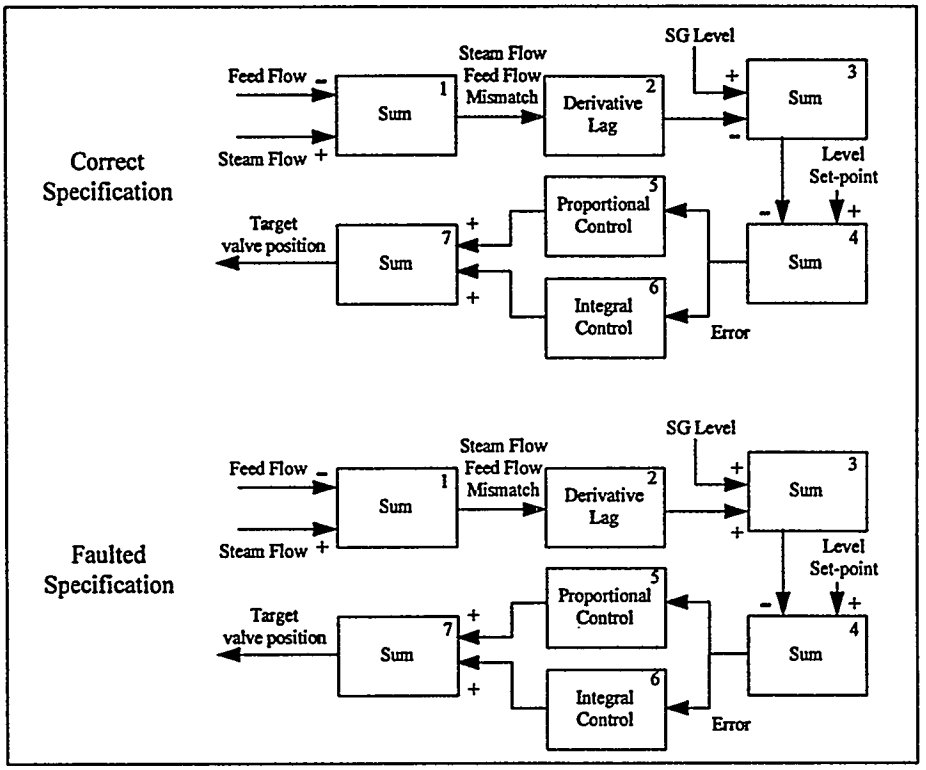


Figure 5.28 : Comparison of the Original Specification and the Faulted Specification

Correct Software Module

```
*****************************************************************
* SUBROUTINE TO DERIVE FLOW DEMAND                              *
* This Subroutine will use data generated in the main program to *
* compute and return desired flow demand.                       *
*****************************************************************
      SUBROUTINE CONTROL(FEEDM,XLEVM,STMM,DT,FEEDACT,RTO,HLO,TIME)
      .
      .
      .
***** level mismatch ***************************************************
      LERROR = 10*XLEVM - DFLOW
      SGERROR = 10*SGREF - LERROR
      .
      .
      .
```

Faulted Software Module

```
*****************************************************************
* SUBROUTINE TO DERIVE FLOW DEMAND                              *
* This Subroutine will use data generated in the main program to *
* compute and return desired flow demand.                       *
*****************************************************************
      SUBROUTINE CONTROL(FEEDM,XLEVM,STMM,DT,FEEDACT,RTO,HLO,TIME)
      .
      .
      .
***** level mismatch ***************************************************
      LERROR = 10*XLEVM + DFLOW
      SGERROR = 10*SGREF - LERROR
      .
      .
      .
```

Figure 5.29 : Comparison of the Correct Software Module and the Faulted Software Module

Unfaulted Software

XLEVM	DFLOW	SGERROR
2	-2	-3
3	-2	-3
4	-2	-3
5	-2	-3
6	-2	-3
7	-2	-3
8	-2	-3
9	-2	-3
10	-2	-3
-	-2	-2
6	-1	-2
7	-1	-2
8	-1	-2
8	0	-2
9	-1	-2
9	0	-2
10	-1	-2
.	.	.
.	.	.
.	.	.

Faulted Software

XLEVM	DFLOW	SGERROR
2	+2	-3
3	+2	-3
4	+2	-3
5	+2	-3
6	+2	-3
7	+2	-3
8	+2	-3
9	+2	-3
10	+2	-3
-	+2	-2
6	+1	-2
7	+1	-2
8	0	-2
8	+1	-2
9	0	-2
9	+1	-2
10	-1	-2
.	.	.
.	.	.
.	.	.

Figure 5.30 : Comparison of the Decision Tables for the Unfaulted Software and the Faulted Software

5.3.1.2 Analysis of the DTC System with the Software Specification Error

Definition of the Top Event

The system failure was defined as the steam generator "overflowing". This translates into a definition of the states of the pertinent DFM nodes as:

$$\{ (L = 10 \text{ @ } t = 0) \text{ AND } (L < 10 \text{ @ } t = -1) \}$$

The above top event definition assumes that system failure occurs when the steam generator level increases from a non-maximum level in the previous time step to the maximum level in the current time step. Thus, the analysis focused on identifying modes in which the steam generator level could be raised to the maximum level, and ignored the cases in which the steam generator level was maintained at the maximum level.

Constraints Imposed on the Analysis

Dynamic consistency rules were defined to prune out the branches that encompass events that the analyst assumed to be impossible due to the dynamic constraints of the system. The dynamic consistency rules so defined are listed in Table 5.XXIX. These rules reflect the assumption that if any sensor or valve has failed, then it remains in the original failure state.

As in the ITC analysis presented in Section 4.4, it is very beneficial to specify check rules to limit the DFM search to the identification of classes of errors. In this particular case, the rules were defined in such a way as to force the Model Analyzer to analyze, store and display only those failure sequences that are related to software errors. For this purpose, the definition of software error that was used includes any kind of software-triggered action that violates the given set of top-level software behavior specifications. The specifically relevant portion of these specifications requires the control software to maintain the SG level at the 68% level, and thus not to command further opening of the main feed valve when the level is above this set-point. This rule was defined as a boundary condition that the Model Analyzer used to distinguish between intermediate transition table rows that need to be expanded further and those for which further expansion was not required. In addition, we assumed that the control software had not accumulated a large error that would bias the control action to correcting this previously accumulated error instead of

responding to the current condition. This assumption was enforced in the analysis by defining another set of boundary conditions.

Table 5.XXIX : The Dynamic Consistency Rules Defined for the Analysis

	Rule	Meaning
1	For FPH, states -1 and +1 are sink states	The main feed pump, once failed, cannot be repaired.
2	For LS, states -1 and +1 are sink states	The level sensor, once failed, cannot be repaired.
3	For FS, states -1 and +1 are sink states	The feed flow sensor, once failed, cannot be repaired.
4	For SS, states -1 and +1 are sink states	The steam flow sensor, once failed, cannot be repaired.
5	For VS, states -1 and +1 are sink states	The main feed valve actuator, once failed, cannot be repaired.

Furthermore, boundary conditions were defined to allow the analysis to be carried out for cases in which the steam generator is working within its normal operating regime. The boundary conditions so defined are:

- The steam generator pressure is in state 2 (between 960 psi and 1185 psi).
- The reactor is operating close to full power.
- The turbine governing valve is fully opened.

Result of the Analysis

The analysis was carried out for one step backward in the reference time frame and the 10 prime implicants that were correspondingly identified are shown in Table 5.XXX.

Table 5.XXX : Prime Implicants for the Top Event in which the Steam Generator Overflows (1/3)

	Prime Implicant		
1	Main feed valve is normal	@ t = 0	AND
	Main feed pump is normal	@ t = 0	AND
	High Level Override signal was inactive	@ t = -1	AND
	Reactor Trip Override signal was inactive	@ t = -1	AND
	Main feed valve was opened between 60% and 80%	@ t = -1	AND
	Feed flow was between 60% and 80%	@ t = -1	AND
	Steam flow was between 30% and 60%	@ t = -1	AND
	SG level was at level 8	@ t = -1	AND
	Feed flow sensor was normal	@ t = -1	AND
	Steam flow sensor was normal	@ t = -1	AND
	Level sensor was normal	@ t = -1	
2	Main feed valve is normal	@ t = 0	AND
	Main feed pump is normal	@ t = 0	AND
	High Level Override signal was inactive	@ t = -1	AND
	Reactor Trip Override signal was inactive	@ t = -1	AND
	Main feed valve was opened between 60% and 80%	@ t = -1	AND
	Steam flow was between 30% and 60%	@ t = -1	AND
	SG level was at level 8	@ t = -1	AND
	Feed flow sensor stuck high	@ t = -1	AND
	Steam flow sensor was normal	@ t = -1	AND
	Level sensor was normal	@ t = -1	

Table 5.XXX : Prime Implicants for the Top Event in which the Steam Generator Overflows (2/3)

	Prime Implicant		
3	Main feed valve is normal	@ t = 0	AND
	Main feed pump is normal	@ t = 0	AND
	High Level Override signal was inactive	@ t = -1	AND
	Reactor Trip Override signal was inactive	@ t = -1	AND
	Main feed valve was opened between 60% and 80%	@ t = -1	AND
	Steam flow was between 30% and 60%	@ t = -1	AND
	SG level was at level 8	@ t = -1	AND
	Steam flow sensor was normal	@ t = -1	AND
	Level sensor stuck low	@ t = -1	
4	Main feed valve stuck fully opened	@ t = 0	AND
	Main feed pump is normal	@ t = 0	AND
	Steam flow was between 30% and 60%	@ t = -1	AND
	SG level was at level 8	@ t = -1	AND
	Steam flow sensor was normal	@ t = -1	
5	Main feed valve is normal	@ t = 0	AND
	Main feed pump is normal	@ t = 0	AND
	High Level Override signal was inactive	@ t = -1	AND
	Reactor Trip Override signal was inactive	@ t = -1	AND
	Main feed valve was opened between 60% and 80%	@ t = -1	AND
	Steam flow was between 80% and 100%	@ t = -1	AND
	SG level was at level 8	@ t = -1	AND
	Level sensor stuck low	@ t = -1	
6	Main feed valve stuck fully opened	@ t = 0	AND
	Main feed pump is normal	@ t = 0	AND
	Steam flow was between 80% and 100%	@ t = -1	AND
	SG level was at level 8	@ t = -1	
7	Main feed valve is normal	@ t = 0	AND
	Main feed pump is normal	@ t = 0	AND
	High Level Override signal was inactive	@ t = -1	AND
	Reactor Trip Override signal was inactive	@ t = -1	AND
	Main feed valve was opened between 60% and 80%	@ t = -1	AND
	Steam flow was between 30% and 60%	@ t = -1	AND
	SG level was at level 9	@ t = -1	AND
	Steam flow sensor was normal	@ t = -1	AND
	Level sensor stuck low	@ t = -1	
8	Main feed valve stuck fully opened	@ t = 0	AND
	Main feed pump is normal	@ t = 0	AND
	Steam flow was between 30% and 60%	@ t = -1	AND
	SG level was at level 9	@ t = -1	AND
	Steam flow sensor was normal	@ t = -1	
9	Main feed valve is normal	@ t = 0	AND
	Main feed pump is normal	@ t = 0	AND
	High Level Override signal was inactive	@ t = -1	AND
	Reactor Trip Override signal was inactive	@ t = -1	AND
	Main feed valve was opened between 60% and 80%	@ t = -1	AND
	Steam flow was between 80% and 100%	@ t = -1	AND
	SG level was at level 9	@ t = -1	AND
	Level sensor stuck low	@ t = -1	

Table 5.XXX : Prime Implicants for the Top Event in which the Steam Generator Overflows (3/3)

	Prime Implicant		
10	Main feed valve stuck fully opened	@ t = 0	AND
	Main feed pump is normal	@ t = 0	AND
	Main feed valve was opened between 60% and 80%	@ t = -1	AND
	Steam flow was between 80% and 100%	@ t = -1	AND
	SG level was at level 9	@ t = -1	

Prime implicants 3, 5, 7 and 9 reveal that the steam generator level sensor stuck at the low reading combined with the level being very high (at states 8 or 9) will cause the steam generator to overflow. The low reading provided by the level sensor will cause the control software to act as if there is not enough water in the steam generator and command the main feed valve to open up to add water into the steam generator. Combined with the fact that the SG level is already very high, the surplus feed flow into the SG over the steam flow out of the SG will cause the level to rise and the SG to overflow. The presence of the other non-failure conditions in the prime implicants is a characteristic of the multi-state, non-coherent representation of the system, as it was earlier seen in the example given in Section 2.5.3. For instance, the main feed pump being normal is part of the necessary condition in the prime implicant since a failed pump cannot sustain feed flow into the SG to cause the overflow condition. From the point of view of the best form of presentation of the results of a deductive analysis, however, some of the non-failure condition can be omitted, as mentioned earlier in Section 2.5.3 and a "reduced prime implicant" can be shown. This is discussed further in Chapter 6 (Section 6.3.2.3).

Similarly, prime implicants 4, 6, 8 and 10 imply that the main feed valve stuck fully opened combined with the level being very high will cause the steam generator to overflow. The main feed valve stuck fully opened will force the maximum rate of feed flow into the SG. Together with the SG level already very high, the surplus feed flow over the steam flow will again cause the level to rise and the SG to overflow. Just like the prime implicants discussed above, these prime implicants also contain necessary non-failure conditions of some hardware components.

If a prime implicant does not contain basic component failure modes that can cause the top event directly, this usually means that a hidden software error is identified. The event sequence leading from the prime implicant to the top event needs to be analyzed in detail to locate the software error. In this particular analysis, prime implicant 1, unlike prime implicants 3-10, does not contain any basic component failure modes, but consists of non-failure hardware component conditions and input conditions of the control software (specification of the level, the feed flow and the steam flow). This prime implicant points to the possibility of a software fault, but it is not directly obvious where the fault is and how the overflow condition is brought about. The intermediate transition tables provided by the Model Analyzer can be used to reconstruct the sequence of events from the prime implicant to the top event. The sequence of events reconstructed in such a manner is show in Figure 5.31. As shown in that figure, SF (steam flow) at state 2 and SS = 0 (steam flow sensor normal) give the correct steam flow reading to the control software (STMM = 2). Similarly, FWF (feed flow) at state 3 and FS = 0 (feed flow sensor normal), and L (SG level) at state 8 and LS = 0 (level sensor normal) provide the control software with the correct feed flow reading (FEEDM = 3) and the SG level reading (XLEVM = 8) respectively. Inside the control software, STMM = 2 and FEEDM = 3 produce the corresponding steam flow-feed flow mismatch (ERFLOW = -1). This mismatch, combined with the lack of accumulated error through the previous controller clock cycles, gives the appropriate derivative lag signal (DFLOW = -1, -2). The fact that DFLOW can be in either state -1 or state -2 is due to the fact that each state of a node presents a range of values, thus a mismatch between STMM = 2 (30%-60%) and FEEDM = 3 (60%-80%) can result in a range of DFLOW values. Continuing to follow the sequence of events, DFLOW = -1, -2 and XLEVM = 8 give the error term SGERROR = 0, +1. However, this does not correspond to a desirable system behavior, as a very high SG level and a negative derivative steam flow-feed flow mismatch indicate a net mass influx into the SG while the level is very high, and thus should give a negative error (the level set-point is less than the anticipated level). This points to the presence of an error in the software module that calculates SGERROR from DFLOW and XLEVM. By checking this module in the software specification diagram, the inappropriate addition of the derivative lag to the SG level could be identified. The identification of software errors by means of reconstructed sequences of events is another major findings of this research, and it will be revisited in Section 6.3.2.3. At the current state of development, DFM requires the analyst to possess sufficient knowledge and understanding of the overall system so

that he/she can identify undesirable behaviors from the sequence of events. The possibility of enhancing DFM to allow users with lower degrees of system knowledge to presented in Chapter 7 (Conclusions and Recommendations).

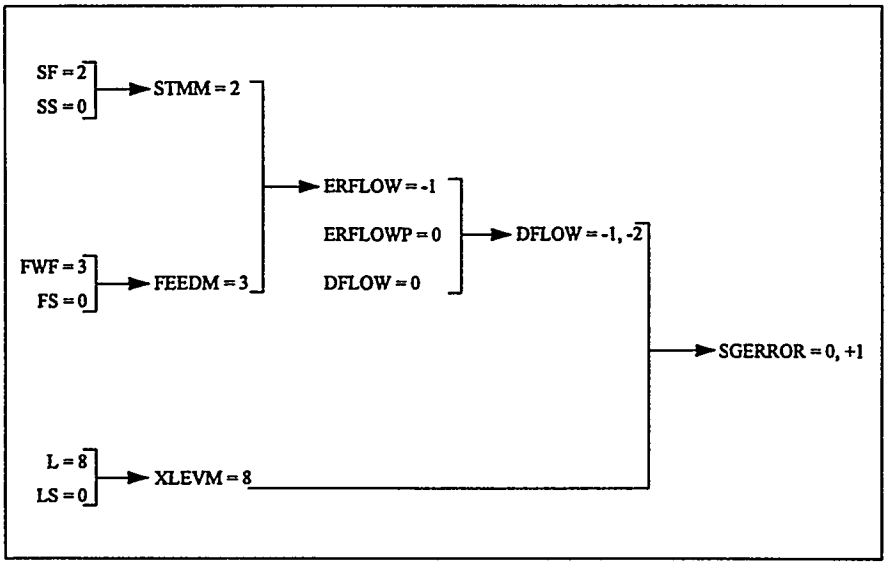


Figure 5.31 : Sequence of Events for Prime Implicant #1

Even though prime implicant 2 contains a basic component failure (the feed flow sensor failing high), this failure does not contribute directly towards the top event as in prime implicants 3-10. Instead, a sequence of events can be generated, similar to the one shown in Figure 5.31, to show that this prime implicant causes the top event because of the same software error identified by the sequence of events for prime implicant 1.

DFM Analysis Driven Testing

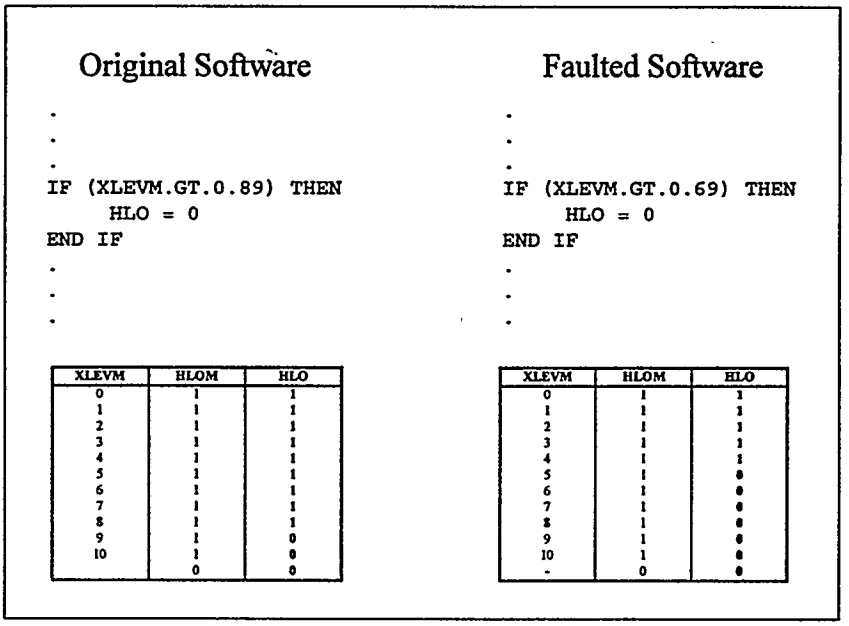
The result of the analysis identified prime implicants #1 and #2 as sets of input conditions that would trigger the software to reverse the control action. Namely, these input conditions can cause the software to command further opening of the main feed valve while the SG level is already very high. However, the prime implicants so identified contain conditions that represent ranges of values, such as feed flow between 60% and 80%, steam flow between 30% and 60%, and SG level at state 8 (87%-89%) for prime implicant #1. To ascertain the exact combinations of input parameters that can cause the top event, testing can be carried out by sampling the above input parameters within the identified ranges. The reader should note that testing with the knowledge of the prime implicants and sequences of events is an advancement over pure "black box" testing of the software, as the results provided by a DFM analysis point to the key parameters that need to be tested, and the ranges within which these parameters should best be sampled in the tests. The insights gained in testing will be discussed further in Section 6.3.3.

5.3.2 The Second Faulted-Case Analysis

5.3.2.1 Description of the Fault Injected

For the second faulted-case analysis, it was assumed that an error had been introduced into the control software code. The assumption was that, instead of triggering the High Level Override (HLO) signal at 89% level, this programming error causes the HLO signal to be activated at 69% level. As the level set point is at 68%, a sight increase in SG level from the set point will cause the software to command the closing of the main feed valve to 5%.

As in the other faulted case studies, the DFM model of the faulted system was built without prior assumption about the software error. The decision tables were constructed by directly executing the individual modules of the digital control software. Figure 5.32 compares the original unfaulted software module and the software module with the HLO error, and the corresponding decision tables built by executing these software modules respectively.



XLEVM	HLOM	HLO
0	1	1
1	1	1
2	1	1
3	1	1
4	1	1
5	1	1
6	1	1
7	1	1
8	1	1
9	1	0
10	1	0
	0	0

XLEVM	HLOM	HLO
0	1	1
1	1	1
2	1	1
3	1	1
4	1	1
5	1	0
6	1	0
7	1	0
8	1	0
9	1	0
10	1	0
-	0	0

Figure 5.32 : Comparison of the Original and Faulted Software and the Corresponding Decision Tables

5.3.2.2 Analysis of the DTC System with the Programming Error

Definition of the Top Event

The system failure was defined as the "steam generator level dropped to 0% narrow range". This was translated into a definition of the states of the pertinent DFM nodes as:

$$\{ L = 0 \,@\, t = 0 \}$$

Constraints Imposed on the Analysis

Check rules were defined to limit the DFM search to the identification of software errors. Thus, the Model Analyzer was directly to analyze, store and display only those failure sequences that are related to software errors. In this particular case, the check rules constrained the analysis to event sequences in which the SG level is low and the software commands further closing of the main feed valve, as these sequences violate the top level specification that the level be maintained at 68% narrow range level.

In addition to the above check rules, two sets of boundary conditions were enforced in the analysis to limit the DFM search to:

- event sequences which did not contain any hardware component failure.
- event sequences where the control software had not accumulated a large error that would bias the control action to correcting this previously accumulated error.

Result of the Analysis

The analysis was carried out for five steps backward in the reference time frame and the one prime implicant that was correspondingly identified is shown in Table 5.XXXI.

Table 5.XXXI : Prime Implicant for the Top Event "SG Level Dropped to 0% Narrow Range"

	Prime Implicant	
1	High Level Override signal was inactive in the memory	@ t = -5 AND
	Steam generator level was between 65% and 71%	@ t = -5 AND
	Steam generator pressure was between 960 psi and 1185 psi	@ t = -5 AND
	Steam flow was between 80% and 100%	@ t = -5

The prime implicant does not contain any basic events, but it encompasses input conditions that can trigger possible errors in the software. To locate the error in the software, the intermediate transition tables obtained by the DFM Model Analyzer in the backtracking process can be used to reconstruct the sequence of events that lead from the prime implicant to the top event. The sequence of events thus reconstructed is shown in Figure 5.33. The steam generator level between 65% and 71% (L = 5) and the assumption that the level sensor is normal give the correct reading to the control software (XLEVM = 5). This SG level reading triggers the activation of the HLO signal (HLOM = 1 → HLO = 0) and causes the flow demand to become 5% (TFD = 0). Thus, the activation of the HLO signal at the incorrect set-point is identified. The sequence of events that follows shows that this irreversible control action eventually causes the SG level to drop to 0% narrow range.

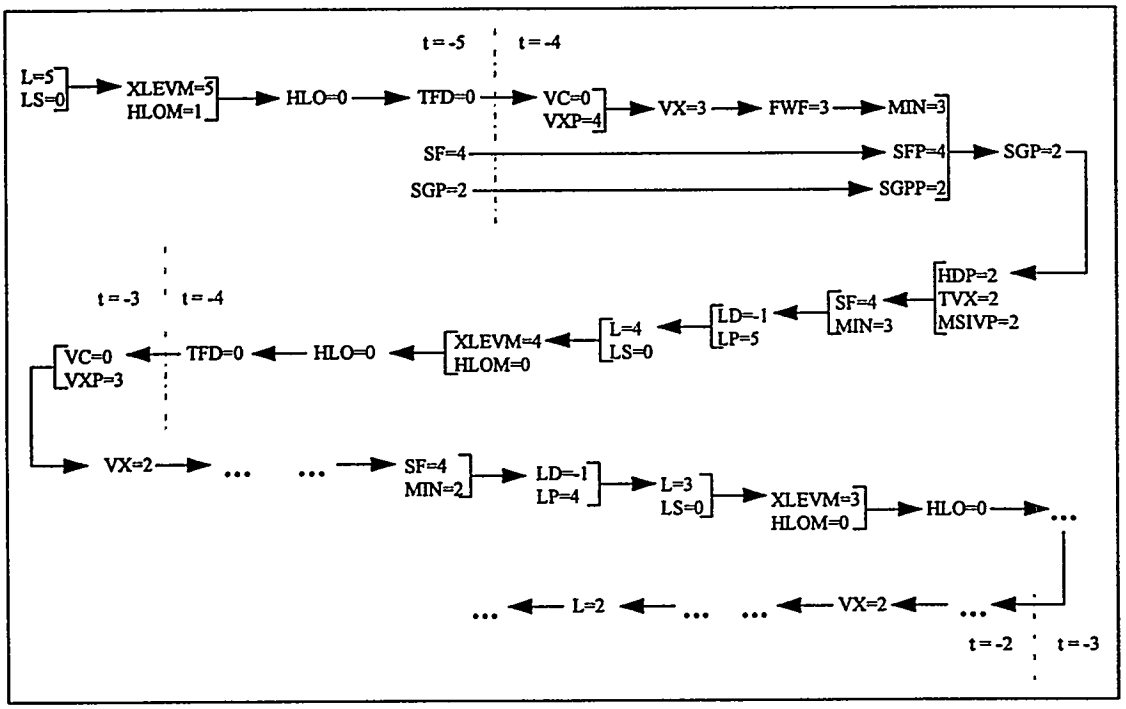


Figure 5.33 : Sequence of Events for Prime Implicant #1

It is important to point out that even though the prime implicant was identified by backtracking the DFM model for 5 time steps, this does not imply that the prime implicant leads to the top event in 5 computing cycles of the digital control software. This is because the decision tables in the DFM model represent state transitions of parameters in the system. Take for instance the event (LD = -1 AND LP = 5) → L = 4 at t = -4 shown in Figure 5.33. It shows that a drop in the SG level (LD = -1) from the 65% to 71% range (LP = 5) will yield a new SG level in the 55% to 65% range. In reality, the level transition from state 5 to state 4 will take a much longer period of time than one computing cycle, unless the level is already very close to the boundary separating the two states (i.e., slightly above 65%). Thus, a DFM analysis implicitly compresses the time to identify the prime implicants in backtracking the fewest possible number of time steps.

DFM Analysis Driven Testing

From the prime implicant and the sequence of events generated from the DFM analysis, it was determined that a SG level in the 65%-71% range could inadvertently trigger the High Level Override mode of control. As in the first faulted case analysis, testing can be carried out to ascertain the exact level at which HLO is triggered. Thus, the results of a DFM analysis can identify the key parameters to be tested, and narrow down the ranges from which these parameters are to be sampled in the testing. In this particular case, testing can be focused on sampling SG levels within the 65%-71% range, and not be diverted to trying other SG level ranges or testing with other parameters. This observation will be revisited in the Findings and Insights in Section 6.3.3.

6 FINDINGS AND INSIGHTS

In the execution of this research, lessons and insights were learned in several relevant areas of interest for the application of the analytical techniques that we have discussed in the preceding chapters. A possible broad categorization of the principal areas in which these insights were gathered can be made as indicated below:

- insights on possible expanded objectives and uses of a DFM analysis;
- insights on the applicability of DFM to the analysis of systems other than those containing software exercising closed-loop continuous control actions;
- insights on the optimization of DFM modeling, analysis and testing procedures.

The various types of insights and lessons learned in the three broad categories defined above are summarized and discussed in this chapter. More specifically, Section 6.1 presents the insights in the first of the three areas defined above, Section 6.2 addresses the second, and Section 6.3 addresses the third.

6.1 Objectives and Uses of a DFM Analysis

The application of deductive procedures to DFM models has been discussed in Chapter 2 and demonstrated in Chapters 4 and 5. In this mode of analysis, the objective is to identify the possible combinations of basic failure modes of the elemental components of a system that may result in a pre-defined type of system failure and the logic-sequential progression of events from the basic failure mode combinations to such a system failure. Timed fault trees (or the logically equivalent series of intermediate transition tables) and the associated timed prime implicants are the products that are generated by the DFM analysis to satisfy that objective.

As we have noted earlier, however, the descriptive formalism adopted in DFM does not necessarily limit the analysis to be performed only in the "backtracking" (i.e., deductive) mode of execution. An inductive procedure can, in fact, also be carried out. In the course of our study, we have briefly explored this possibility and arrived at the conclusion that such a mode of DFM analysis could be particularly useful in verifying that the system behavior corresponds to a given specification of desired behavior. The objective being pursued would, in this type of application, be the comparison and verification of the as-designed or as-built system behavior with respect to the system design goals and objectives. In particular, the capability of DFM to represent the essential functionality of a specified control software allows it to be used as a modeling framework for design verification. Since a DFM model captures the causal relationships between variables in a control software, both logical (discrete) and functional (continuous), a prediction of the behavior of the software in response to specified stimuli (i.e., input from the non-software portions of the modeled system) can be inductively inferred from it. Section 6.1.1 outlines the possible analytical procedures and shows an example of such an inductive analysis.

6.1.1 Design Verification of Control Software

A design verification objective can be fulfilled by using the DFM model of a digital controller and interfacing system in a forward-simulation mode of analysis, to derive a system state-transition graph or generate the equivalent set of system state-transition relations, which can then be used to verify the design of the controller via a model checking algorithm. Instead of traversing the DFM model backward in causality and obtaining a sequence of intermediate transition tables in reverse sequential order, as in a deductive analysis, the DFM model can be traversed forward in causality from a set of initial conditions to generate the sequence of events which follow from the initial conditions. The sequence of events can be summarized in tabular form as transition relations or presented in graphical form as a state-transition graph. Binary decision diagrams (BDD), a tool for modeling binary systems, have indeed been used to generate state-transition graphs to verify logic circuit designs (Browne, et al., 1986). A natural extension for non-binary systems is thus to obtain state-transition graphs from a multi-valued representation such as DFM. Indeed, there is no inherent property of state-transition graphs that precludes the use of multi-valued logic.

The pressure tank example in Section 2.5 is used here again to illustrate how to derive, from its DFM model, a state-transition graph and apply to this a design verification procedure. The focus here is in the verification of design and

not with the identification of physical faults. Thus, the variables E, SWS, SS, VS and MVO, as well as their "faulting effects", are ignored in this exercise.

For the purposes of this example, suppose that the pressure control system has been incorrectly designed , so that the set-point for closing the electric switch is specified at the boundary between TP = 3 and TP = 4, instead of between TP = 2 and TP = 3. The DFM model of this design is presented in Figure 6.1. Starting from an arbitrary initial tank pressure, the DFM model can be traversed forward to produce the sequences shown in Table 6.I. This sequence can then be simplified and translated into the state-transition graph shown in Figure 6.2.

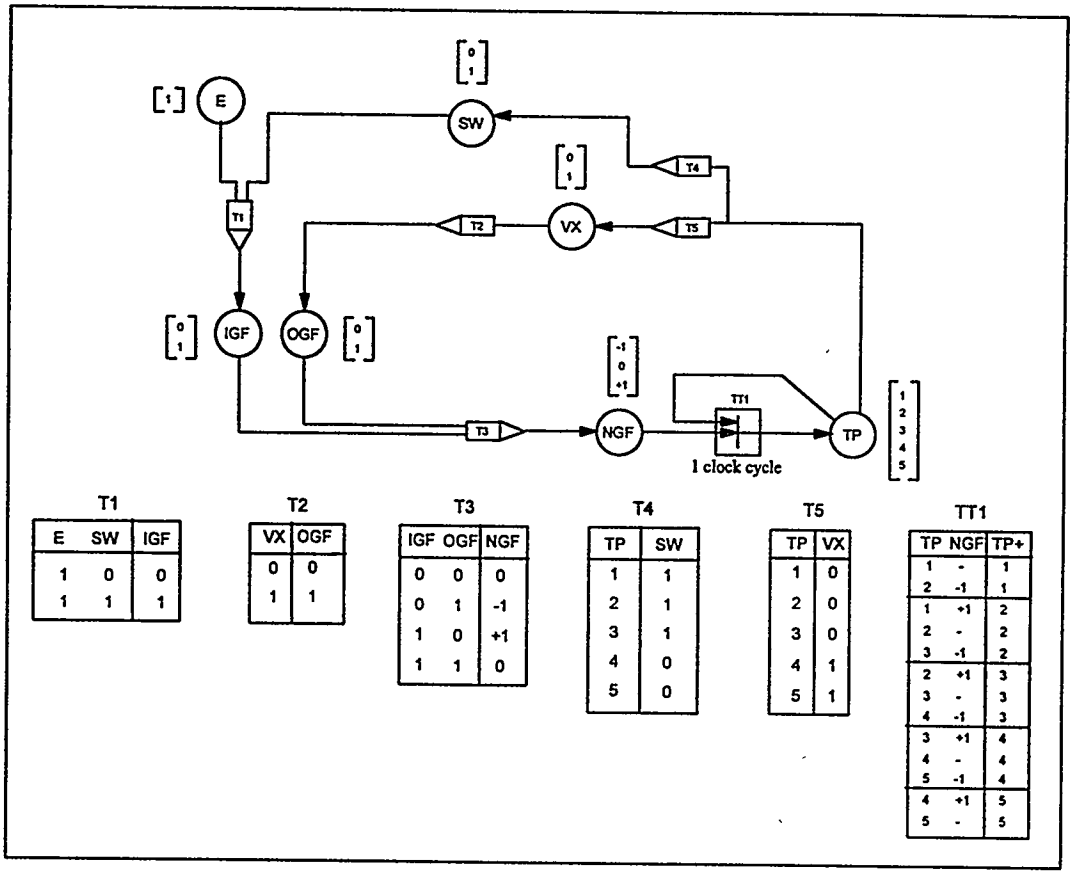


Figure 6.1 : DFM Model for Verification of the Pressure Control Software

Table 6.I : List of Pressure Transition Sequences

TP = 1 @ t = 0 →	SW = 1 → VX = 0	IGF = 1 → OGF = 0	NGF = +1 →	TP = 2 @ t = 1
TP = 2 @ t = 0 →	SW = 1 → VX = 0	IGF = 1 → OGF = 0	NGF = +1 →	TP = 3 @ t = 1
TP = 3 @ t = 0 →	SW = 1 → VX = 0	IGF = 1 → OGF = 0	NGF = +1 →	TP = 4 @ t = 1
TP = 4 @ t = 0 →	SW = 0 → VX = 1	IGF = 0 → OGF = 1	NGF = -1 →	TP = 3 @ t = 1
TP = 5 @ t = 0 →	SW = 0 → VX = 1	IGF = 0 → OGF = 1	NGF = -1 →	TP = 4 @ t = 1

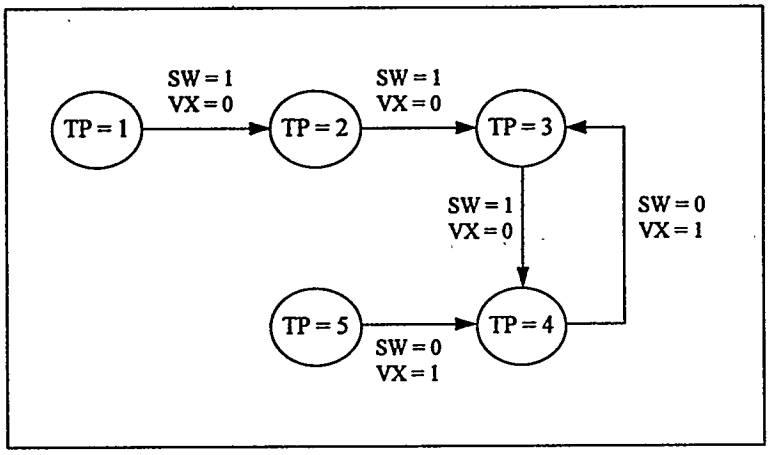


Figure 6.2 : State Transition Graph

The control software can be verified against the requirement that the pressure is to be stabilized at level 3. Given the state-transition graph, an analyst can follow the history of the tank pressure for any number of time steps. If the above condition is not satisfied for all transition histories, the controller is not verified. It can be seen from Figure 6.2 that the pressure will eventually reach level 3 from any other level, but will not be stabilized at level 3 (it will oscillate between levels 3 and 4). Thus, the requirement is not satisfied. The reader should note that the design error will not result in the tank bursting, as the tank pressure cannot reach level 5 from other levels, but the error can create a serious problem as the pressure oscillates about the normal level and cannot reach steady state.

6.2 Applicability of DFM to Other Types of Systems

The basic elements of DFM can be used to model any causality driven behavior. Thus, as argued earlier in Section 2.1, DFM can be applied to analyze a broad variety of systems, and not just those characterized by the presence of digital control software and closed-loop digital control actions, such as those that have been the principal subject of our investigation and that were discussed earlier in this report. Section 6.2.1 presents our views and insights with respect to the feasibility of applying DFM to open loop control systems. This type of system is commonly found in the safety systems of nuclear power plants.

6.2.1 Feasibility of Applying DFM to Open Loop Control Systems

Open loop control systems are control systems in which the control actions are solely determined by pre-defined system conditions which are used as discontinuous trigger-points for the actions. Unlike closed-loop control systems that constantly apply mathematical manipulation of monitored parameters to provide continuous control adjustments, open loop systems usually control one-shot, discrete actions. These systems are commonly found in the safety systems used in nuclear power plants. Typical examples are the safety injection system and the reactor trip control system.

The capabilities provided by DFM analyses are well suited for verifying systems with open loop control actions. By application of a DFM deductive analysis, i.e., one that produces timed fault trees and prime implicants, the analyst can find out whether the overall system integrity can be violated despite the actions of the open loop control system. For example, by including the safety relief valve control in the DFM model of a steam generator and defining the top event as steam generator overpressure, a DFM analysis can be carried out to investigate the possibility that the safety relief valve fails to prevent the pressure from building up to an undesirable level. If this event is possible, a DFM analysis can be used to determine the necessary and sufficient conditions for it to occur, and the sequence of events that would lead to overpressure. Or, for a more complex open-loop logic implemented in software, such as the reactor trip logic itself, the DFM analysis could be applied to derive timed fault trees for the trip control software, and make sure that no prime implicants resulting from software logic errors are possible.

On the other hand, an inductive analysis, i.e., one that generates transition relations, can help verify that an open loop system will do what it is supposed to do. For instance, the trip control system can be modeled to generate transition relations for all the possible execution paths and check that the execution sequences are followed exactly as desired and specified.

If the use of DFM to verify open loop control systems can in principle be established, the question of its practicality needs to be addressed. Open loop control systems are causality driven systems where combinations of events can trigger different control actions, and the control actions lead to different sequences of events. In addition, the events can take place with or without time intervals in between. The elements of DFM are well suited to modeling these kinds of control flow. Furthermore, certain types of open loop control actions may be logically complex, with many possible combinations of system conditions leading to different actions. The multi-state representation used in DFM allows the variable space to be discretized to reflect the different trigger points, and the formalism for multi-valued logic adopted in DFM would enable decision tables to be constructed to represent the exact logic relationships that the control system enforces. The DFM system-model would thus be an integrated representation of all these logic relationships and could therefore be analyzed to identify and verify all the possible control-system-triggered actions.

A further argument can be used in trying to make a prediction on the practical level of difficulty that may or may not be experienced in applying DFM to systems with open loop control actions. The examples in Chapters 4 and 5 have shown the feasibility of applying DFM to systems with closed-loop control actions, and open loop control actions can be considered as a special case of closed loop control, where the gain in the feedback or feedforward loop is zero. The test cases discussed in Chapters 4 and 5 also dealt with systems in which open loop logic was intermixed to closed loop feedback control (e.g., the mode-of-control switch and trip logic included with the control of the water tank system discussed in Chapter 4). With the experience of modeling these systems, the authors feel confident that an extension of the methodology to the verification and safety assurance of complex open loop control logic does not entail any insurmountable difficulty, and that a practical demonstration could be carried out. This will be discussed further in Chapter 7 (Conclusions and Recommendations).

6.3 Optimization of DFM Procedures

In the course of applying DFM to the Interim Test Case and the Demonstration Test Case, ways to carry out the modeling, analysis and testing procedures with greater efficiency were identified. Simplification measures that are useful in producing DFM models efficiently are presented in Section 6.3.1; optimization rules for DFM analysis are discussed in Section 6.3.2; and the formulation of a set of software testing related steps is given in Section 6.3.3.

6.3.1 Modeling Procedures

In modeling the Interim Test Case and Demonstration Test Case digital control systems, the authors observed that common modeling structures are used to represent the control logic within these systems. These modeling structures can be grouped together to form a library of modeling templates from which DFM models of similar systems can be assembled. As more common logic structures are identified, modeling efforts can be reduced by making use of the available templates instead of constructing the model from the most basic modeling elements. This object-oriented approach to constructing DFM models is not limited to digital control systems. As experience is gained in modeling other classes of systems, templates for representing common modules can be identified. The discussions in Sections 6.3.1.1 and 6.3.1.2 will present templates that have been identified for the modeling of digital control systems.

6.3.1.1 Modeling Different Types of Control Logic

Proportional (P) control, proportional-integral (PI) control, proportional-derivative (PD) control and proportional-integral-derivative (PID) control are commonly used in both analog and digital control systems. Standardized procedures are available to design control systems with these types of control logic to meet specific requirements, such as rise time, overshoot and steady state error. As a tool for modeling control systems, it is advantageous for DFM to provide the user with a library of pre-defined structures for representing these different types of control logic. This section discusses the templates that were identified for such a purpose.

In the ensuing discussion, the parameter P represents the error in the variable being controlled (i.e. the difference between the value of the variable and the set-point), DP indicates the derivative of P, IP means the integral of P, and C is the control command generated by application of the control logic. In addition, a superscript "-" is used to indicate the value of the variable in the previous computation cycle.

Modeling Proportional Control Logic

Figure 6.3 shows a template for representing P-control logic. As seen from this figure, the command is directly related to the parameter through transfer box 1. The associated decision table for transfer box 1 models the correct gain of the relationship.

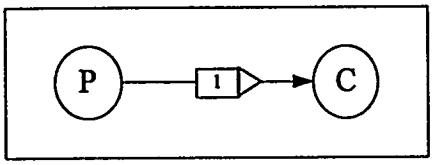


Figure 6.3 : DFM Template for Modeling P Control Logic

Modeling Proportional-Integral Control Logic

Figure 6.4 is a template for modeling P-I control logic. The integral control term is generated by P, the previous value of P and the previous value of the integral control term through transfer box 2, where the average of P and P^- multiplied by the time step yields the new area to be added to IP^-. P and IP are then used to calculate the command through transfer box 1. Transition box 3 updates the value of P so that the present value of P in the current cycle becomes the past value of P in the next cycle.

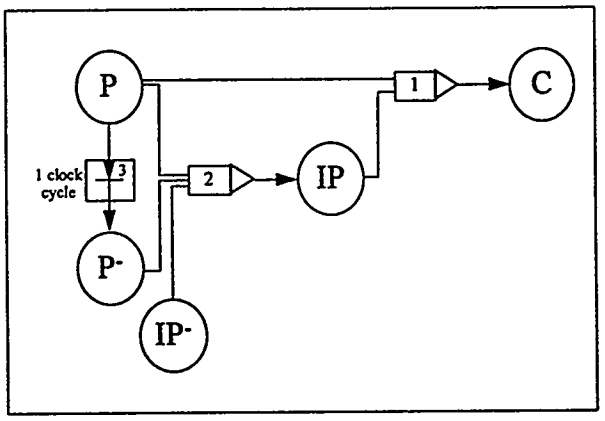


Figure 6.4 : DFM Template for Modeling P-I Control Logic

Modeling Proportional-Derivative Control Logic

Figure 6.5 shows a template for modeling P-D control logic. The derivative control term (DP) is generated by the present value of P and the previous value of P through transfer box 2, DP and P are then used to calculate the command through transfer box 1. Transition box 3 updates the value of P so that the present value of P in the current cycle becomes the past value of P in the next cycle.

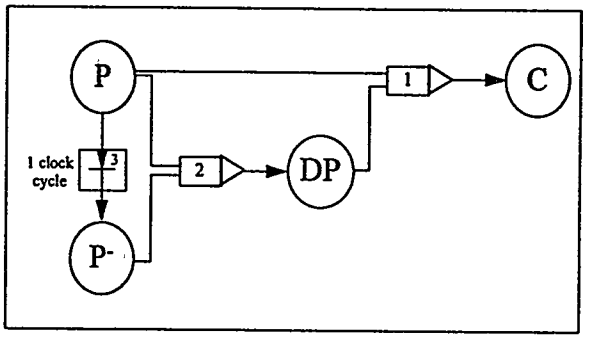


Figure 6.5 : DFM Template for Modeling P-D Control Logic

Modeling Proportional-Integral-Derivative Control Logic

Figure 6.6 is a template for modeling P-I-D control logic. The integral control term is generated and the parameter is updated as is in Figure 6.4, while the derivative control term is calculated as is in Figure 6.5. The control command is then calculated using the trio P, DP and IP through transfer box 1.

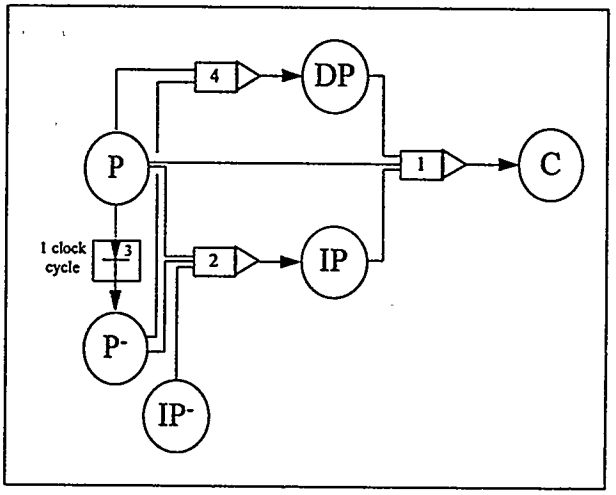


Figure 6.6 : DFM Template for Modeling P-I-D Control Logic

6.3.1.2 Modeling Irreversible Control Actions

Irreversible control actions are control actions which once triggered, cannot be switched off unless the entire system is shutdown or reset. These control actions are commonly found in safety systems. An example is the HLO mode of operation for the SG level control system presented in Chapter 5. It would seem obvious that the signal to activate the irreversible control action depends only on the parameters that trigger the signal, but the fact is that the signal also depends on the memory of itself in the system. If the memory was inactive in the previous cycle, the parameters entering the triggering domain will activate the irreversible control signal. However, if the memory was already active in the previous cycle, the signal will remain inactive, even though the parameters can be outside the triggering domain. The template for modeling this type of irreversible control action is presented in Figure 6.7, and the corresponding decision table is shown in Table 6.II. In the figure, S represents the signal for triggering the irreversible control action, SM represents the memory of this signal in the system, and Ps represents the set of parameters that can trigger the irreversible control action.

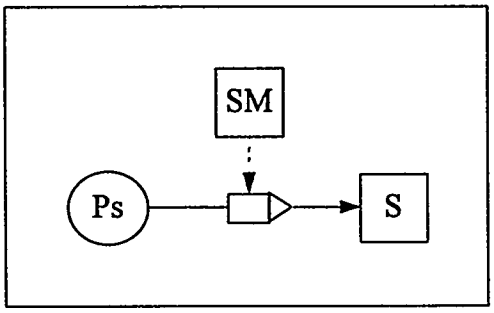


Figure 6.7 : DFM Template for Modeling Irreversible Control Action

Table 6.II : Decision Table for the Template shown in Figure 6.7

Ps	SM	S
Outside Triggering Domain	Inactive	Inactive
Within Triggering Domain	Inactive	Active
Don't Care	Active	Active

6.3.2 Analysis Procedures

In using the DFM Software Toolset to analyze the Interim Test Case and the Demonstration Test Case, the authors found that with the application of certain guidelines, the search algorithm of the Model Analyzer can be optimized to produce the results in a more efficient manner and present the results which yield the most useful information. These guidelines can assist the analyst to prioritize the analysis so that the most hazardous failure modes can be identified first. The guideline pertinent to the formulation/definition of the top event in terms of DFM model states is presented in Section 6.3.2.1, the guideline for narrowing down search paths is discussed in Section 6.3.2.2, and the guideline for presenting the analysis results is given in Section 6.3.2.3.

6.3.2.1 Guideline for Formulation of the Top Event

The way in which the top event is expressed in terms of a combination of the states of the nodes has an impact on the efficiency of the analysis. A more generic translation of a top event would yield more prime implicants, but not all the prime implicants accordingly identified would necessarily represent an equally critical threat to the integrity and safety of the system. Hence, the most efficient way of analysis is to first translate the top event into the most specific expression involving the states of those nodes which correspond to portions of the system of the greatest interest and concern to the analyst. The ensuing analysis will then produce prime implicants of the highest priority, i.e., those associated with failure modes which represent types of system hazards for which the analyst has the higher degree of specific concern. Thus, for example, if the focus of the analysis is to identify software related problems, it would be pointless to search for system failure modes originated by basic failures involving exclusively physical equipment and hardware. After the first high priority analysis, successive analyses can be performed with the top event expressed in more generic terms to identify additional failure modes of possible interest.

To further illustrate the concept expressed above, take for example the Interim Test Case discussed in Chapter 4, where the top event is first generically identified as the "tank overflowing". This top event can be first translated into the combination of the tank level being very high in the previous step and a net positive inflow of water into the tank. With this top event interpretation, the search is narrowed down to identifying the most hazardous conditions, whereby the control system itself brings about the overflow condition. A second analysis can be carried out where the top event is interpreted as the combination of the level being very high in the previous step and the level remaining very high in the current step. The search is now geared towards finding ways in which the control system fails to correct the undesirable event of the level being very high, but does not directly cause the overflow condition. Further analysis can be performed for the top event of the level being very high in the current time step to identify paths by which the system dynamics fills the tank up to the undesirable level. Successive levels of analysis carried

out in this manner, can prioritize the search to find the most critical conditions first, and find the less critical conditions in successive analyses.

6.3.2.2 Classification of Failure Modes

In addition to translating the top event in successive degree of generality, intelligent analysis can also be performed by grouping the prime implicants into classes of failure modes. The classification of failures can narrow down the search for prime implicants and enable the analyst to make intelligent interpretations out of the results. This is implemented by defining rules to reflect a certain class of behavior, and using the rules to distinguish between events in a timed fault tree (or, equivalently, rows in an intermediate transition table) that need to be further explored and those that may be stored and not analyzed further.

One particular approach may be used to look for single failure causes, whereby only event-sequences which contain one failure event at most are developed, i.e., the search is redirected to another branch as soon as a second failure event is determined to be necessary to make the sequence possible. Such an approach can similarly be extended to look for double failures, triple failures and so on. For example, the analysis of the Demonstration Test Case presented in Section 5.3.2 was carried out to search for event sequences in which there was no hardware failure.

Another possible approach is to classify the source of the failure modes, i.e., as to whether they originate from the software, from the hardware, or from both. If we assume that the analyst intends to look for software induced failures first, the rules to be defined will seek to specify the behavior of a correct software. In the search, any event that matches the rules, and hence indicating the software is behaving correctly, will not be explored further. In fact, this approach has been applied in the analysis of the Interim Test Case in Section 4.4.2 and the Demonstration Test Case in Section 5.3. The rule being defined limit the search to finding failure modes in which the software does not behave as it is designed to. The results presented in Sections 4.4.2 and 5.3 show how this approach can help an efficient identification of the software error present within the controller. Further analysis can be performed with similar definition of rules to identify failures emanating from the hardware or failures requiring both a software error and external triggering conditions. As discussed in Section 4.4.2, the definition of check rules to classify that errors originate from the software require the analyst to refer to a formal "catalogue" of system behavior specifications.

6.3.2.3 Presentation of the Analysis Results

Reconstruction of the Sequences of Events

As the reader can see in the results of the DTC analysis given in Section 5.3, the prime implicants are expressed as combinations of software input conditions, system boundary conditions and component failure conditions. Software errors are not identified as basic events, but as input conditions at the hardware/software interface. This is due to the fact that the faulted software module does not necessarily accept parameters at the hardware/software interface, the parameters it receives as inputs can be parameters calculated inside the software. In a DFM analysis, when these internal parameters are encountered, they will be backtracked to their predecessors, which might also be internal software parameters. Backtracking is terminated until the hardware/software interface is reached. To pinpoint the module in which the software error is located, it is necessary to reconstruct the sequence of events from the input conditions through the software modules. As the events are traced forward in causality through the software modules, undesirable responses emanating from any software module can be identified by comparing with the specification behavior, as in the cases discussed in Section 5.3.1 and 5.3.2. This forward causality tracing activity is similar to the generation of the state-transition sequences discussed in Section 6.1.1, except not all possible sequences resulting from the input condition are generated. The intermediate transition tables generated by the DFM Model Analyzer will limit the sequences to those which will ultimately cause the top event. The discussion in Section 5.3 pointed out that the analyst must possess sufficient knowledge and understanding of the system to recognize whether the generated sequence of events correspond to the desirable system behavior or not. The possibility of enhancing the capability of DFM to make it usable for users with lower degrees of system knowledge and understandings is discussed in Chapter 7, Conclusions and Recommendations.

Reduction of Prime Implicants

As seen in the ITC and DTC analyses, prime implicants may contain non-failure conditions. For example, the prime implicant shown in Table 4.XXIV contains the normal states of the level sensor, the flow sensor, the control valves and the stop valves. A "reduced form" of the prime implicant which was correspondingly identified is shown in Table 6.III. The software error that causes the tank to overflow is identified via its immediate effect, that is the command issued to the downstream control valve to its minimum position (software condition), AND the failure of the check valve (external condition). The reduced form of the prime implicant was obtained by the Model Analyzer from the full form initially identified, by deleting from the list of conditions in the prime implicant all those conditions which identify the states of sensors, control valves and stop valves related to the event sequence of interest as being normal, i.e. none of these components are failed.

Table 6.III : Reduced Form of the Prime Implicant for the ITC Analysis

	Prime Implicant		
1	check valve failed open	@ t = 0	AND
	Upstream control valve commanded to close to its minimum	@ t = -1	AND
	Downstream control valve commanded to close to its minimum	@ t = -1	AND
	Tank level was very high	@ t = -1	

In general, in a multi-state, non-coherent system representation such as that used in DFM, a parameter state can be always classified as "faulted" or "normal" only for the model parameters expressly set up to represent hardware failure / non-failure states. A reduced form of prime implicant can thus be obtained by not including in it the listing of normal states of this type of parameters. The states of process variables, on the other hand, are not definable a priori to be always "good" or "bad", and consequently are always listed even in the reduced prime implicant. This is because a process parameter state which is "good" in a certain type of situation may become "bad" when the situation changes. For example, in the prime implicant in Table 6.III, the state of the upstream control valve is "good" (the valve is trying to reduce the tank inflow to a minimum in the presence of a potential overflow situation), whereas the state of the downstream valve is "bad" (since this valve is trying to reduce outflow). This classification of good and bad, however, would be completely reversed if we were in an opposite situation in which the tank water level had fallen below the minimum allowable. In essence, the state of being "commanded to close to its minimum" cannot be determined for either valve to be good or bad until the context within which this happens has been identified. The prime implicants for the first faulted case analysis of the DTC (Table 5.XXX) can similarly be reduced to the those shown in Table 6.IV.

6.3.3 Testing Procedures

From the study of the Interim Test Case and the Demonstration Test Case, the authors found that intelligent testing strategies could be devised using the prime implicants and the sequence of events obtained from a DFM analysis. Better testing could be performed at the module level and the system level, and the discussions on these aspects are presented in Section 6.3.3.1 and 6.3.3.2 respectively.

6.3.3.1 Module Testing

Prime implicants for DFM analyses, such as those obtained from the ITC and the DTC studies, are combinations of input conditions to the software, boundary conditions in the rest of the system (e.g., the components which must be operational), and component failure conditions. In addition, the sequence of events reconstructed by retracing the intermediate transition tables could locate the software errors within specific software modules, such as the cases presented in Sections 5.3.1 and 5.3.2. These two pieces of information can be used to optimize the testing procedures. As the location of the software errors can be pinpointed in the sequence of events, more resources can be directed to test the specific modules where critical errors are identified. In addition, the input conditions, which are identified as part of the prime implicants, can guide the selection of the parameters to be tested, and the definition of the ranges within which the parameters can best be sampled in the tests.

Table 6.IV : Reduced Form of the Prime Implicants for the First DTC Analysis

	Prime Implicant		
1	High Level Override signal was inactive	@ t = -1	AND
	Reactor Trip Override signal was inactive	@ t = -1	AND
	Main feed valve was opened between 60% and 80%	@ t = -1	AND
	Feed flow was between 60% and 80%	@ t = -1	AND
	Steam flow was between 30% and 60%	@ t = -1	AND
	SG level was at level 8	@ t = -1	
2	High Level Override signal was inactive	@ t = -1	AND
	Reactor Trip Override signal was inactive	@ t = -1	AND
	Main feed valve was opened between 60% and 80%	@ t = -1	AND
	Steam flow was between 30% and 60%	@ t = -1	AND
	SG level was at level 8	@ t = -1	AND
	Feed flow sensor stuck high	@ t = -1	
3	High Level Override signal was inactive	@ t = -1	AND
	Reactor Trip Override signal was inactive	@ t = -1	AND
	Main feed valve was opened between 60% and 80%	@ t = -1	AND
	Steam flow was between 30% and 60%	@ t = -1	AND
	SG level was at level 8	@ t = -1	AND
	Level sensor stuck low	@ t = -1	
4	Main feed valve stuck fully opened	@ t = 0	AND
	Steam flow was between 30% and 60%	@ t = -1	AND
	SG level was at level 8	@ t = -1	
5	High Level Override signal was inactive	@ t = -1	AND
	Reactor Trip Override signal was inactive	@ t = -1	AND
	Main feed valve was opened between 60% and 80%	@ t = -1	AND
	Steam flow was between 80% and 100%	@ t = -1	AND
	SG level was at level 8	@ t = -1	AND
	Level sensor stuck low	@ t = -1	
6	Main feed valve stuck fully opened	@ t = 0	AND
	Steam flow was between 80% and 100%	@ t = -1	AND
	SG level was at level 8	@ t = -1	
7	High Level Override signal was inactive	@ t = -1	AND
	Reactor Trip Override signal was inactive	@ t = -1	AND
	Main feed valve was opened between 60% and 80%	@ t = -1	AND
	Steam flow was between 30% and 60%	@ t = -1	AND
	SG level was at level 9	@ t = -1	AND
	Level sensor stuck low	@ t = -1	
8	Main feed valve stuck fully opened	@ t = 0	AND
	Steam flow was between 30% and 60%	@ t = -1	AND
	SG level was at level 9	@ t = -1	
9	High Level Override signal was inactive	@ t = -1	AND
	Reactor Trip Override signal was inactive	@ t = -1	AND
	Main feed valve was opened between 60% and 80%	@ t = -1	AND
	Steam flow was between 80% and 100%	@ t = -1	AND
	SG level was at level 9	@ t = -1	AND
	Level sensor stuck low	@ t = -1	
10	Main feed valve stuck fully opened	@ t = 0	AND
	Main feed valve was opened between 60% and 80%	@ t = -1	AND
	Steam flow was between 80% and 100%	@ t = -1	AND
	SG level was at level 9	@ t = -1	

6.3.3.2 System Testing

Besides identifying the modules and the parameters where testing is most needed, the results provided by DFM analyses can also help guide system level testing activity to identify the most efficient way of improving the system. As the prime implicants are expressed in terms of combinations of input conditions, boundary conditions and component failure conditions, system level testing can be directed to determine the likelihood of encountering these conditions. The prime implicants can be ordered according to their likelihood of occurrence. The prime implicants higher on the list should be addressed first. For example, in the first faulted-case analysis of the DTC presented in Section 5.3.1, if system level testing could determine that the input conditions and the failure events in prime implicants 3-10 were unlikely to occur, the development activity could be redirected to focus entirely on fixing the software specification error, rewriting the code, and re-analyzing the system with the specification error removed. In the extreme case in which system level testing can determine that all the prime implicants are not likely to be encountered, a trade-off can be made where the user accept the system with the inherent error. This can be beneficial if the cost associated with fixing the error and re-qualifying the system is much higher than the cost of accepting the risk.

7 CONCLUSIONS AND RECOMMENDATIONS

The objective of this research was the development of a modeling environment and analytical framework to enable the execution of a practical process of verification and validation for software that is devoted to critical process control and safety functions, and more specifically for control software of the type that may be employed in the next generation of nuclear power plants, as well as in digital upgrades that are presently being proposed and implemented in the control systems of existing plants. This objective has been successfully pursued and achieved with the completion of the development and demonstration activities documented in this report.

The principal accomplishment of this research can be summarized as follows:

- The features of the Dynamic Flowgraph Methodology (DFM) analytical approach have been developed and defined in all detail necessary to establish a practical baseline for future applications. The approach is articulated in two steps, which involve, respectively, system modeling and system assurance analysis. These steps can also be integrated with, and facilitate, the traditional step of system assurance-testing.

- An integrated analytical software package, which implements the DFM procedures and algorithms, has been developed. This software runs on PC workstations under the Windows™ environment and relies on graphic models and user interfaces for data input and output, which directly reflect the "directed graph" representation at the base of DFM.

- Two extensive validation and demonstration activities were carried out to refine and test the capabilities of the DFM methodology. An "interim test case" (ITC) was developed and used to aid the development and finalization of DFM. A "demonstration test case" (DTC) was developed and used to prove the viability of the DFM methods and tools, as finalized in a baseline version at the end of the development phase concluded with the ITC exercise.

The DFM methodology trial and demonstration activities carried out in the project have established the suitability of the DFM analytical approach. Besides validating the detailed DFM modeling and analysis procedures, the two extensive test cases carried out within this research have demonstrated the validity of the "systems approach" in the assurance analysis of software-based control systems. This approach requires both the software and hardware portions of the system to be represented and analyzed by use of one integrated model, which includes all the major system functions and interfaces, instead of partitioned and separate representations that are analyzed and verified separately. The advantage of using and analyzing an integrated system model is that the specification of behavior, as well as the actual behavior, of individual elements can be, as the analysis is carried out, compared with the specification of behavior and the actual behavior of the whole system. This makes possible the uncovering of inconsistencies that may exist between the two levels of specifications and behaviors, which would be very hard to identify by the traditional approaches of "partitioning" the assurance analyses.

In the course of this development, an understanding has been developed of how DFM modeling and analysis activities can be carried out in coordination with established software testing activities and how they can facilitate a systematic and close-ended approach to carry out the latter. The DFM system modeling process requires in fact a systematic execution of software module testing (if of course the system software exists in executable form at the time the DFM analysis is carried out). After a DFM system model has been assembled by putting together software and hardware module representations, the process of DFM analysis can then take out the guesswork from system integration testing and identify, from the integrated hardware / software analysis, the test inputs and sequences that need to be applied as input "stimuli" to the software portion of the system in order to test for the existence or not of software faults. It must be acknowledged, however, that the amount of direct practical experience that has been gathered in the area of coordination of DFM analysis and software testing activities is still relatively limited. Thus, because of the importance that software testing has in current assurance and verification practices and the large amount of resources that are routinely invested in testing activities, a recommendation can be made to focus a portion of future DFM development and application efforts onto the objective of systematically expanding the current level

of understanding and experience base regarding how DFM analysis and software testing integration can be fully achieved and optimized.

Particularly important toward the objective of enabling the use of the DFM technique in practical industrial applications is the fact that an application toolset, which we have referred to in this report as the DFM Software Toolset, has been created and is now available for such a use. The DFM Software Toolset is an integrated set of software tools for implementing the model construction and analysis procedures of DFM. This software toolset, which is made up of two principal modules, i.e., the Model Editor and the Model Analyzer, runs as a Microsoft Windows™ application on any PC workstation (486 or Pentium class workstations with 16 Mbytes or more of memory are recommended). The availability of this tool makes the MVL (multi-valued logic) algorithms which are at the core of the DFM analytical capability transparent to practically-oriented users, while also providing a graphic model-editor environment for the construction of DFM representations of systems to be analyzed and for the input of user-provided data concerning such systems and the desired assurance analyses thereof.

With the use of the DFM application software, deductive analyses of systems presenting a level of complexity comparable to those of the ITC or DTC systems (discussed, respectively, in Chapter 4 and Chapter 5) can be carried out in relatively straightforward fashion. However, DFM analysis of control system software cannot be expected to be executable as a routine task. On the contrary, users should be aware that the correct application of the technique requires an in-depth understanding of the technical issues that have been presented and discussed in this report. Computational resources may also pose limitations to the mode of application of the analysis. If care is not applied to follow the recommendations for the definition of top-events and boundary conditions (or "check-rules") that have been discussed in Chapters 4, 5 and 6, greater amounts of computer resources (i.e., memory and/or processor time) become necessary to execute an analysis, possibly to the point of running against final limits of feasibility for systems of greater complexity.

Among all the insights that have been discussed in Chapter 6, two appear to be especially important for the future development and execution of DFM analysis. The first is linked to the observation that many software faults are identifiable by DFM not directly as basic conditions that are part of a "prime implicant" logic definition, but only via the observation of the actual sequence and logic path to the top event associated with the prime implicant itself. This requires the analyst to examine carefully the sequences originated by a prime-implicant condition, and compare them with a reference model of "good system behavior" in order to pinpoint the fault. The direct implication of this is, therefore, that the analyst must refer to a formal "catalogue" of system behavior specifications, from the highest to the lowest level available, and he/she must possess an overall knowledge and understanding of the system sufficient to recognize whether, and how, a DFM-generated sequence of system-events does not correspond to the desirable system behavior. A conclusion that may immediately follow from this observation is that, to evolve towards more "routine" applications of DFM by personnel with lower degrees of system knowledge and understanding, it maybe desirable to develop a systematic procedure for the identification and definition of desired system behavior, similar perhaps in nature to the one that is envisioned by the proponents of "formal specification methods" (Rushby 1995 and Thomas 1994). This observation also indicates the existence of an opportunity for an investigation of the relation of an engineering modeling and analysis method such as DFM with the formal methods of logic specification and verification, including the investigation of possible avenues by which the more intuitive and practical representation of the former may be utilized to facilitate the use and application of the latter in practical situations.

The second important indication for further development of DFM is to extend the analytical procedures to permit inductive logic analysis, in addition to the present capability for deductive analysis. By tracking system state evolution forward in time and cause-effect order of sequence, this added analytical capability would be useful for software specification and design verification purposes, as it would allow the analyst to test specifically for whether the control software and the associated system follow a certain type of desired behavior which has been specified in advance (e.g., whether the system reaches a controlled and stable state starting from certain specified initial conditions). A simple trial application of this mode of analysis which was conducted manually within our study (and reported in Section 6.1.1) shows that this extension is not only useful, but that it should be also achievable without excessive application of additional development effort.

With respect to the assurance analysis of open-loop software systems, such as those which implement nuclear plant safety and protection logic, the conclusion of this research is that DFM analysis should be well suited for this purpose. The importance of such software-based safety systems and their assurance in the nuclear regulatory process is readily recognized. Although the principal focus of this study has been the assurance analysis of closed-loop control systems, mainly because of the technical challenges posed to the analyst by their dynamic characteristics, consideration of the nature of the DFM analysis technique and direct evidence from the execution of the test-cases conducted in this research both support the above conclusion (please refer to Section 6.2.1 for details). Thus, an optimization of DFM analysis techniques for open-loop safety systems analysis, although not within the direct scope of this study, is judged to be an achievable objective. Given the current dearth of actual analytical tools to support safety-critical software assurance efforts, a recommendation can be made for the development of DFM application capabilities and specialized implementation tools in this direction.

At the conclusion of the development cycle and studies documented in this report, the DFM methodology has been brought to the working level of being applicable to software-driven control systems of considerable complexity. This results from both the successful demonstration of its basic features and capabilities in two realistic and large-scale test cases and from the development of engineering-workstation software that implements and partially automates the execution of a desired analysis. Further refinement of the DFM tools will be conditional upon user feedback from field applications which will be pursued, if possible, in the immediate future (e.g., as part of the SBIR Phase III, if commercial sponsors and users can be identified).

REFERENCES

Beizer, B.: *Software Testing Techniques*, Van Nostrand Reinhold, 2nd Edition, 1990.

Boenhert, P.: Memorandum to H. Lewis, Chairman, Computers in Nuclear Power Plant Operations Subcommittee, Advisory Committee on Reactor Safeguards, U.S. Nuclear Regulatory Commission, September 25, 1990.

Browne, M.C., E.M. Clarke, D.L. Dill and B. Mishra: "Automatic Verification of Sequential Circuits Using Temporal Logic," *IEEE Transactions on Computers* **C-35**, pp.1035-1043, 1986.

Cha, S.S., N.G. Leveson and T.J. Shimeall: "Safety Verification in Murphy Using Fault Tree Analysis," *Proceedings of the International Conference on Software Engineering*, Singapore, pp.377-386, 1988.

Garriba, S., E. Guagnini and P. Mussio: "Multiple-Valued Logic Trees: Meaning and Prime Implicants," *IEEE Transactions on Reliability* **R-34**, pp.463-472, 1985.

Garrett, S., S. Guarro and G. Apostolakis: "The Dynamic Flowgraph Methodology for Assessing the Dependability of Software Systems," *IEEE Transactions on Systems, Man and Cybernetics* **25**, pp.824-840, 1995.

Garrett, S., M. Yau, S. Guarro and G. Apostolakis: "Assessing the Dependability of Embedded Software Systems Using the Dynamic Flowgraph Methodology," in *Dependable Computing and Fault-Tolerant Systems Vol. 9*, F. Cristian, G. Le Lann, T. Lunt (eds.), Springer-Verlag Wien, New York, 1995.

Goel, A.L.: "Software Reliability Models: Assumptions, Limitations, and Applicability," *IEEE Transactions on Software Engineering* **SE-11**, 1985.

Guarro, S.B. and D. Okrent: "The Logic Flowgraph: A New Approach to Process Failure Modeling and Diagnosis for Disturbance Analysis Applications," *Nuclear Technology* **67**, pp.348-359, 1984.

Guarro, S.B.: "PROLGRAF-B: A Knowledge Based System for the Automated Construction of Nuclear Plant Diagnostic Models," *Proceedings of the International Topical Meeting on Artificial Intelligence and Other Innovative Computer Applications in the Nuclear Industry*, Snowbird, UT, August 1987.

Guarro, S.B.: "A Logic Flowgraph Based Concept for Decision Support and Management of Nuclear Plant Operation," *Reliability Engineering and System Safety* **22**, 1988.

Guarro, S.B., J.S. Wu, G.E. Apostolakis and M. Yau: *Findings of a Workshop on Embedded System Software Reliability and Safety*, Technical Report UCLA-ENG 90-25, University of California, Los Angeles, CA, June 1990.

Henley, E.J. and H. Kumamoto: *Probability Risk Assessment: Reliability Engineering, Design, and Analysis*, IEEE Press, 1992.

Jaffe, M.S., N.G. Leveson, et al.: "Software Requirements Analysis for Real-Time Process-Control Systems," *Proceedings of the 11th International Conference on Software Engineering*, Pittsburgh, PA, May 1989.

Kumamoto, H. and E.J. Henley: "Safety and Reliability Synthesis of Systems with Control Loops," *AlCHe Journal* **25**, pp. 108-113, 1979.

Leveson, N.G. and P.R. Harvey: "Analyzing Software Safety," *IEEE Transactions on Software Engineering* **SE-9**, pp.569-579, 1983.

Leveson, N.G. and J.L. Stolzy: "Safety Analysis Using Petri Nets," *IEEE Transactions on Software Engineering* **SE-13**, pp.358-363, 1987.

Littlewood, B. and D. Miller: "Preface: Special Issue of Reliability Engineering and System Safety on Software Reliability and Safety," *Reliability Engineering and System Safety*, 1990.

Morgan, E.T. and R.R. Razouk: "Interactive State-Space Analysis of Concurrent Systems," *IEEE Transactions on Software Engineering* **SE-13**, pp.1081-1091, 1987.

Motamed, M.E.: "Pressurized Water Reactor Dynamic Simulation Model," *Ph.D. Thesis*, University of California, 1983.

Mott, T.H.: "Determination of the Irredundant Normal Forms of a Truth Function by Iterated Consensus of the Prime Implicants," *IEEE Transactions on Electronic Computers* **9**, pp. 245-252, 1960.

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

Nahavandi, A.N.: "An Improved Pressurizer Model With Bubble Rise and Condensate Drop Dynamics," *Nuclear Engineering Design* **12**, pp.135-147, 1980.

Narayana, K.T. and A.A. Aaby: "Specification of Real-Time Systems in Real-Time Temporal Interval Logic," *Proceedings of the 1988 Conference on Real-Time Systems*, IEEE Press, 1988.

Ogunbiyi, I.E.: "Application of Decision Tables to Risk Analysis Studies," *Ph.D. Dissertation*, University of Houston, June 1980.

Ogunbiyi, E.I. and E.J. Henley: "Irredundant Forms and Prime Implicants of a Function with Multistate Variables," *IEEE Transactions on Reliability* **R-30**, pp. 39-42, 1981.

Neumann, P.G.: "Some Computer-related Disasters and Other Egregious Horrors," *ACM Software Engineering Notes*, January 1985.

Parnas, D.L., G.J.K. Amis and J. Madey: "Assessment of Safety-Critical Software in Nuclear Power Plants," *Nuclear Safety* **32**, No. 2, pp. 189-198, April-June 1991.

Petrella, S., P. Michael, W.C. Bowman and S.T. Lim: "Random Testing of Reactor Shutdown System Software," *Proceedings of the International Conference on Probabilistic Safety Assessment and Management*, Beverly Hills, CA, February 1991.

Quine, W.V.: "The Problem of Simplifying Truth Functions," *American Mathematical Monthly* **59**, pp. 521-531, 1952.

Quine, W.V.: "A Way to Simplify Truth Functions," *American Mathematical Monthly* **62**, pp. 627-631, 1955.

Razouk, R.R. and M.M. Gorlick: "A Real-Time Interval Logic for Reasoning about Executions of Real-Time Programs," *Proceedings of the ACM SIGSOF'89*, ACM Press Software Engineering Notes **14**, pp.10-19, 1989.

Rippon, S.: "Three Computerized Control Rooms," *Nuclear News* **33**, pp. 60-63, October 1990.

Roberts, N.H., W.V. Vesely, D.F. Haasl and F.F. Goldberg: "Fault Tree Handbook," U.S. Nuclear Regulatory Commission report NUREG-0492, 1981.

Rushby, J.: *Formal Methods and their Role in the Certification of Critical Systems*, Technical Report CSL-95-1, Computer Science Laboratory, SRI International, Menlo Park, CA, March 1995.

Salem, S.L., G.E. Apostolakis and D. Okrent: "A New Methodology for the Computer-Aided Construction of Fault Trees," *Annals of Nuclear Energy* **4**, pp. 417-433, 1977.

Salem, S.L., J.S. Wu and G.E. Apostolakis: "Decision Table Development and Application to the Construction of Fault Trees," *Nuclear Technology* **42**, pp. 51-64, 1979.

Shields, E.J., G.E. Apostolakis and S.B. Guarro: "Determining the Prime Implicants for Multi-state Embedded Systems," *Proceedings of PSAM-II*, San Diego, CA, March 1994.

Taylor, J.J. and B.K-H Sun: "Applications of Computers to Nuclear Power Plant Operations," *Nuclear News* **33**, pp.38-40, October 1990.

Thomas, M.: "The Role of Formal Methods in Achieving Dependable Software," Reliability Engineering and System Safety **43**, pp.129-134, 1994.

Ting, Y.T.D.: "Space Nuclear Reactor System Diagnosis: A Knowledge Based Approach," *Ph.D. Dissertation*, UCLA, 1990.

Vijuk, R. and H. Bruschi: "AP600 Offers a Simpler Way to Greater Safety, Operability and Maintainability," *Nuclear Engineering International*, November 1988.

Yau, M., S. Guarro and G. Apostolakis: "Demonstration of the Dynamic Flowgraph Methodology using the Titan II Space Launch Vehicle Digital Flight Control Software," *Reliability Engineering and System Safety* **49**, pp.335-353, 1995.

NRC FORM 335 (2-89) NRCM 1102, 3201, 3202

U.S. NUCLEAR REGULATORY COMMISSION

BIBLIOGRAPHIC DATA SHEET

(See instructions on the reverse)

1. REPORT NUMBER (Assigned by NRC. Add Vol., Supp., Rev., and Addendum Numbers, if any.)

NUREG/CR-6465

2. TITLE AND SUBTITLE

Development of Tools for Safety Analysis of Control Software in Advanced Reactors

3. DATE REPORT PUBLISHED

MONTH	YEAR
April	1996

4. FIN OR GRANT NUMBER

W6157

5. AUTHOR(S)

S. Guarro, M. Yau, M. Motamed

6. TYPE OF REPORT

Technical

7. PERIOD COVERED *(Inclusive Dates)*

September 30, 1993 - March 31, 1996

8. PERFORMING ORGANIZATION – NAME AND ADDRESS *(If NRC, provide Division, Office or Region, U.S. Nuclear Regulatory Commission, and mailing address; if contractor, provide name and mailing address.)*

ASCA, Inc.
2250 East Imperial Highway, Suite 200
El Segundo, CA 90245-3547

9. SPONSORING ORGANIZATION – NAME AND ADDRESS *(If NRC, type "Same as above"; if contractor, provide NRC Division, Office or Region, U.S. Nuclear Regulatory Commission, and mailing address.)*

Division of Systems Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001

10. SUPPLEMENTARY NOTES

L. Beltracchi, NRC Project Manager

11. ABSTRACT *(200 words or less)*

Software based control systems have gained a pervasive presence in a wide variety of applications, including nuclear power plant control and protection systems which are within the oversight and licensing responsibility of the U.S. Nuclear Regulatory Commission. While the cost effectiveness and flexibility of software based plant process control is widely recognized, it is very difficult to achieve high levels of dependability and safety assurance for the functions performed by process control software, due to the very flexibility and potential complexity of the software itself. The development of tools to model, analyze and test software design and implementations in the context of the system that the software is designed to control can greatly assist the task of providing higher levels of assurance than those obtainable by software testing alone. This report presents and discusses the development of the Dynamic flowgraph methodology (DFM) and its application in the dependability and assurance analysis of software-based control systems. The features of the methodology and full-scale examples of application to both generic process and nuclear power plant control systems are presented and discussed in detail. The features of a workstation software tool developed to assist users in the application of DFM are also described.

12. KEY WORDS/DESCRIPTORS *(List words or phrases that will assist researchers in locating the report.)*

Software Safety
Digital Control Systems

13. AVAILABILITY STATEMENT

Unlimited

14. SECURITY CLASSIFICATION

(This Page) Unclassified

(This Report) Unclassified

15. NUMBER OF PAGES

16. PRICE