Integrating Safety Analysis Techniques, Supporting Identification of Common Cause Failures

Giuseppe Mauri

Thesis submitted for the degree of Doctor of Philosophy

The University of York

Department of Computer Science

September 2000

Dedicated to the memory of my dear father

Abstract

When we apply safety analysis techniques on a new design, our primary objective is to anticipate potential scenarios of failure in the system under examination. If we assume that the system has a complex hierarchical structure, this task can be interpreted as one of identifying how failures originate at the low-levels of the design and how combinations or sequences of such low-level failures propagate to higher levels and give rise to system malfunctions. The ultimate aim is to identify weak areas of the design and stimulate design iterations that improve the safety of the system under examination. Unfortunately, the current industrial practise shows that this aim is seriously hindered by the lack of appropriate techniques for the analysis of complex hierarchical designs.

Classical safety analysis techniques, such as Functional Failure Analysis, Hazard and Operability Studies, Failure Mode and Effects Analysis and Fault Tree Analysis, are performed at different stages of the design lifecycle on the basis of models that reflect different levels of abstraction in the design. The selective and fragmented application of different methods, however, has a number of negative implications for the quality of the results gained from the assessment. Firstly, the results of the various safety studies are often inconsistent. Secondly, as hardware safety analysis and software hazard analysis typically form two separate parts of the assessment, the relationship between hardware and software failure often remains vague and unresolved. Finally there is an inherent difficulty in relating the results from low-level safety studies back to the high-level functional failure analysis.

In the first part of this thesis we propose a new method for safety analysis that enables integrated safety assessment of complex hierarchical designs. It helps analysts to identify potential functional failures at the application level and then to systematically determine the causes of those failures in progressively lower levels of the design decomposition. The result of the assessment is a collection of safety analyses that provides a consistent and meaningful picture of how low-failures are stopped at intermediate levels of the design, or propagate and give rise to hazardous malfunctions.

In the second part of this thesis we show how features of the new method support also effective common cause failure analysis. That is both the qualitative identification of components vulnerable to common cause failures and the quantitative estimation of the contribution of these events to critical failures of the system.

Contents

List of Tables

List of Figures

Acknowledgements

I wish to thank my supervisor John McDermid whose help, guidance and financial support have been invaluable throughout all the work. He has always shown interest and enthusiasm in my research and created an environment that encourages the autonomy of spirit and the development of new ideas that were necessary for this kind of work, hence he has my deepest thanks.

A number of colleagues here at York and at the European Commission in Ispra (Italy) have provided constructive comments and acted as sounding boards for my ideas. I would like to thanks Sergio Contini, Tim Kelly, Marcelo Masera, Mark Nicholson, Divya Prasad, David Pumfrey, Stefan Scheer, Marc Wilikens and Weihua Wu. A special thanks goes to Yiannis Papadopoulos a fellow doctoral student and friend with whom I have shared a common interest in the topic of safety analysis. Our collaboration over the last three years has helped me to clarify some aspects of my work in the area. Thanks also to Steve Wilson that modified the software platform that supports the method. Without his software it would have been much harder to run the case studies. Finally, I would like to thank Ginny Wilson and Filomena Ottaway who were wonderful in sorting out administrative issues connected to the work and the university.

On a personal note I would like to thank all the people who helped to keep life in perspective during the last five years in York. Amongst them particular thanks goes to Marco Carcassoni, Michele Cassano, Stratos Chatzikyriakos, Doriana Delfino, Sofie Emmertsen, Tse-Min Lin, Karsten Loer, Olga Miranda, Eman Nasr, Yiannis Papadopoulos, Michela Pilotto, Gloriana Regginato, Stefania Ziccolella and my invaluable friend Gerry Faith.

There are no words to express my thanks to my father Raimondo Mauri, who despite encouraging me for four years did not have the chance to see the end of my doctorate, to my mother Wilma Cervini, who has always encouraged me to keep going, and finally, to my brother Edoardo who, yet having lost a partner of adventures, has always been close to me.

Author's Declaration

I declare that the research in this thesis is original work conducted by the author between October 1995 and September 2000, in the High Integrity System Engineering (HISE) group. Some parts of the thesis have already been published, specifically, Chapter 2 is partially based on [Mauri, 1996], Chapter 3 is based on [Mauri, 1997a-b], Chapter 4 is based on [Mauri et al, 1998] and [Papadopoulos, Mauri, McDermid, 2000]. Chapter 5 is partially based on [Mauri, 1997b]. Finally, Chapter 6 is partly based on [Mauri, 1997b], [Mauri et al, 1998] and [Papadopoulos, Mauri, McDermid, 2000].

Chapter One

Introduction

The success of many modern applications is highly dependent on the correct functioning of complex computer based systems. In some cases, failures in these systems may bring serious consequences in terms of loss of human life [Hecht and Hecht, 1986]. Systems in which failure could endanger human life are termed *safety-critical*. The application of these systems ranges from transport (aircraft, driverless and high speed trains, active safety in cars) through power production plant (nuclear power plants), medicine (life-support, patient monitoring, pacemaker) to industrial processes (chemical and petro-chemical industries). Significant effort is required to assess and certify these systems since software is extensively used. Software behaves different from hardware upon which safety critical systems of the past were based. Hence *computer based safety critical systems*, which are the topic of this thesis, have to be analysed with new analysis methods. At the moment a number of safety analysis methods (most of them extension of methods used for the analysis of pure hardware artefacts) are used throughout the *lifecycle* of computer based safety critical systems to ensure that they meet the necessary standards.

1.1 Life-cycle

To develop safety critical systems a number of stakeholders' requirements have to be considered, but safety is paramount. According to recent guidelines [SAE-ARP 4754-4761, 1996; IEC 61508, 1997] the safety analysis process should be conducted throughout the lifecycle of safety critical systems from the specification stage through implementation, integration, verification, operation, maintenance and decommissioning. This means also that safety engineers have to work together with system engineers to meet the safety requirements for the requested artefact. In this thesis we will concentrate on the safety analysis performed during the part of the lifecycle represented in Figure 1-1. These are the safety analyses which support the *"Decomposition and Design"* and the *"Integration and Verification"* processes. The purpose of these analyses is to check

the developing system design against safety requirements, anticipate potential scenarios of failure and, eventually, provide feedback to system engineers on whether the system they are constructing will behave safely. This should avoid employing resources in developing systems that will not later be acceptable to regulatory authorities.

The safety lifecycle is often represented with a "*V*" shape. The left branch represents the continuous assessment of the design as it progresses towards the development of more and more details (lower level components). During this process a number of recommendations and safety related requirements are produced. They add up to stakeholders safety requirements. All these constraints are to be met by the system. The verification of those constraints takes places in the right branch of the safety lifecycle, which represents the assessment of the integration process. The overall design of the system is accepted only if it is demonstrated that specifications, recommendations and safety related requirements issued during the decomposition and design stages are met i.e. the system is *"not worse than"* the one specified. The process of verification starts from the lowest decomposition levels (i.e. component level) and proceeds towards top functional levels (that is the opposite of the process that happens during the decomposition and design). If requirements and recommendations given for each peer decomposition level are not met, they can either be renegotiated with stakeholders or designs have to be changed, increasing the overall developing cost.

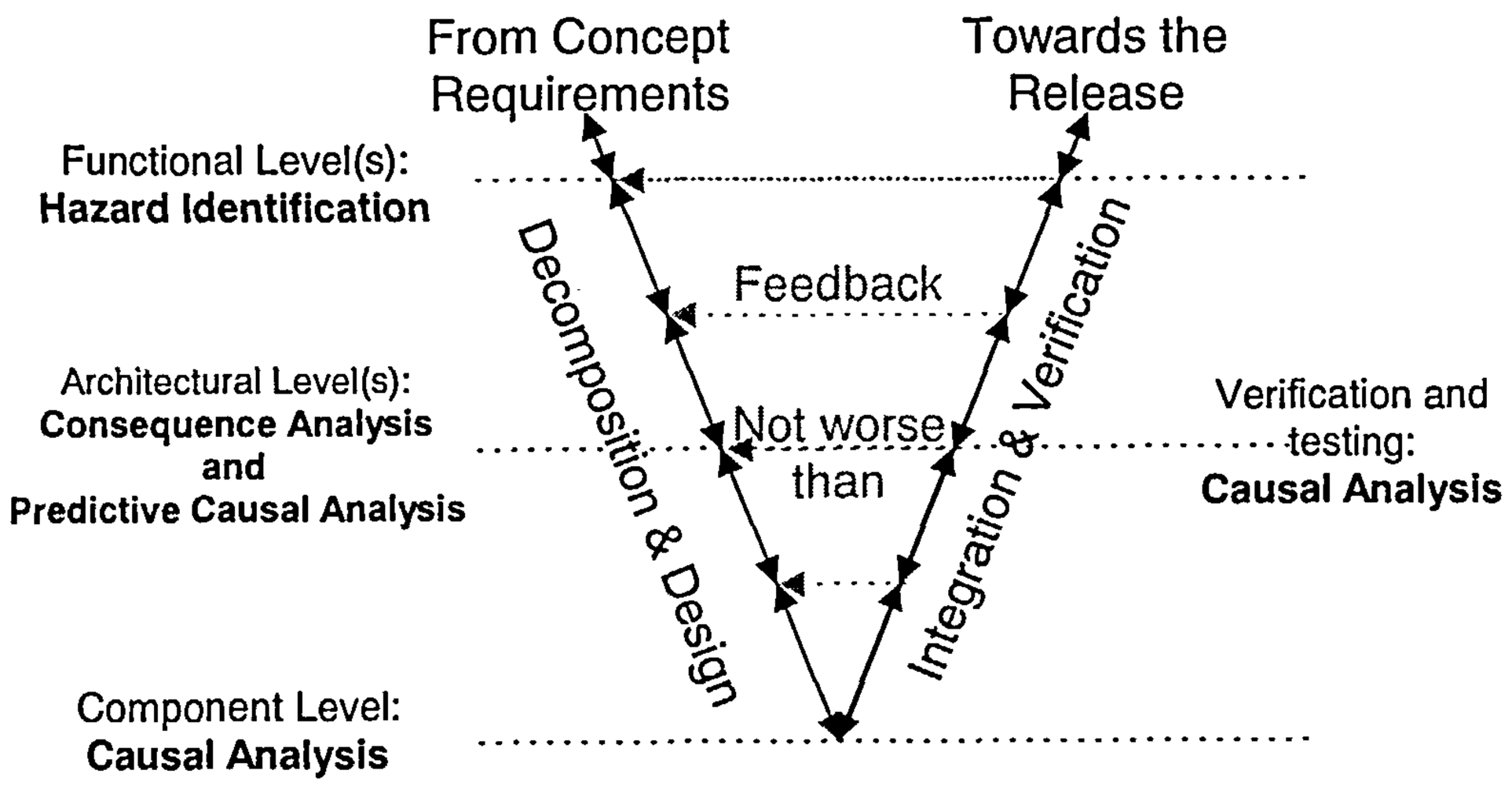


Figure 1-1: Safety life cycle

A safety critical system may also require a high level of reliability to be achieved. That is the case of systems that are requested to be fully (or partially) working to be safe, for example a flight control system in an aircraft or an emergency feed water system in a nuclear power plant. In these systems the high level of safety (as well as reliability) is traditionally achieved by using *fault tolerance*.

1.2 Fault tolerance

Fault tolerance is a particular technique that allows building systems that preserve the delivery of their expected (or a minimum) service despite the presence of errors caused by faults within the system itself [Avizienis, 1985]. To achieve this behaviour they employ *redundancy*. Redundancies can be classified into four types: 1) hardware redundancy; 2) software redundancy; 3) time redundancy; and 4) information redundancy. In the case of hardware redundancy the system is provided with more hardware components (e.g. *channels*) than it would need if the hardware were perfect. Upon failure of a hardware component (or channel) a spare one is switched in. In the case of software redundancy the system may be provided wi th different versions of tasks. Different and independent teams of programmers write tasks so that when one fails under certain inputs another version can be used and there is a chance that the alternate will function safely. In the case of time redundancies the scheduler has some slack so that some tasks can be rerun and still meet deadlines. In the case of information redundancies data are coded in such a way that a certain number of bit errors can be detected and/or recovered.

A fault tolerant system will only fail if multiple failure events happen. The smallest combination of failure events happening together (i.e. linked by an "*AND*" gate) which causes a system to fail is called *Minimal Cut Set* (MCS). A fault tolerant system usually has minimal cut sets that span various orders. The *order* of a minimal cut set is the number of failure events that occurring simultaneously will verify it. An order is defined also for a fault tolerant system. The order of a fault tolerant system is the order of the smallest minimal cut set that causes a critical failure. To be fault tolerant, a system cannot have minimal cut set of the first order.

The introduction of redundancies makes the work of safety engineers more difficult, since redundancies bring with them a new class of events named *common cause events*.

1.3 Common Cause Events

Common cause events affect safety analysis so that the measurable likelihood of a minimal cut set is bigger than the product of the likelihood of each single event in the minimal cut set considered alone. Common cause events make useless increasing the number of redundant channels beyond a certain limit as shown in [Mauri, 1995] and [Cojazzi, et al, 1995]. If engineers were able to build redundant systems with independent redundant channels, there would not be the need of Common Cause Failure (CCF) analysis. In addition, engineers would be able to reach the aimed level of safety (and reliability) by increasing the level of redundancy. Unfortunately, it is practically impossible to build independent redundant channels and the contribution of common cause events have to be evaluated to assure that safety and reliability requirements are met in fault tolerant systems.

The easiest way to consider common cause failures is to work on minimal cut sets. Events in a minimal cut set may represent the same failure mode in different components (i.e. common mode) or different failure modes. They can be generated by the same cause (i.e. common cause) or by different causes. However, the issue for the purpose of this thesis, is that, when all the events in a minimal cut set arise simultaneously by the same *root cause*, the fault tolerant system fails as if the events in the minimal cut set had arisen randomly. The likelihood of a minimal cut set occurring because of a common cause failure is usually extremely small, however, it is always greater than the likelihood of the minimal cut set to happen randomly. Purpose of common cause failure analysis is to evaluate this likelihood and to help improving the design. Without considering common cause events, the likelihood of critical minimal cut sets for fault tolerant systems would be underestimated.

A lot of confusion exists on an unequivocal definition of common cause events especially between the nuclear and the aerospace industry. This thesis will be mostly based on the well founded definition given in [Mosleh, et al., 1988] which was based on the results of the benchmark exercise on common cause failure [Amendola, 1986; Poucet et al., 1987], organised by the European Commission.

1.4 Motivation

When we apply safety analysis techniques on a new design, the immediate objective is to anticipate potential scenarios of failure in the system under examination. If we assume that the system has a complex hierarchical structure, this task can be interpreted as one of

identifying how failures originate at the low-levels of the design and how combinations or sequences of such low-level failures propagate to higher levels and give rise to system malfunctions. The ultimate aim of this analysis is to identify weak areas of the design and stimulate design iterations, which eventually improve the failure detection and control mechanisms of the system under examination. Unfortunately, the current industrial practise shows that this aim is seriously hindered by the lack of appropriate techniques for the analysis of complex hierarchical designs.

Classical safety analysis techniques (such as Functional Failure Analysis [SAE-ARP 4754, 1996], Hazard and Operability Studies [Kletz, 1992], Failure Mode and Effects Analysis [Palady, 1995] and Fault Tree Analysis [Vesely, 1981]) are performed at different stages of the design lifecycle on the basis of models that reflect different levels of abstraction in the design. The selective and fragmented application of different methods, however, has a number of negative implications for the quality of the results gained from the assessment. Firstly, the results of the various safety studies are often inconsistent. Secondly, as hardware safety analysis and software hazard analysis typically form two separate parts of the assessment, the relationship between hardware and software failures often remains vague and unresolved. Finally there is an inherent difficulty in relating the results from low-level safety studies back to the high-level functional failure analysis. Although fault trees are built precisely for this purpose, the traditional process of constructing these fault trees relies exclusively on expert knowledge, and lacks a systematic or structured algorithm which the analyst can apply on a system model in order to derive the tree. In the context of a complex system this process becomes tedious, time consuming and error prone, and the resultant fault trees are large but, more importantly, difficult to interpret and verify. In consequence, safety analyses are in practice not only voluminous but also fragmented and inconsistent. Such analyses are also difficult to interpret and do not always provide a useful resource in the design of the system.

Common cause failure analysis has always been matter of concern for system developers and regulatory authorities. This is mainly due to the difficulty and the uncertainty of the quantification of the likelihood of common cause events. Nuclear industries have been pushed since the sixties to address this problem. The reason was that regulatory authorities (in USA and Europe) were, already at that time, asking for nuclear power plants where the likelihood of any critical failure was well below 10^{-6} per

year[1]. Aerospace and automotive industries are not yet asked for such a low frequency for critical failures. At the moment it seems that they are pursuing frequencies for critical failure of 10^{-9} per hour [SAE-ARP 4761, 1996], which is 10 times bigger[2] than the minimum allowed for nuclear power plants. However the achieved failure rate for civil aircraft is around 10^{-6} critical accidents per hour[3]. This higher "accepted" frequency for critical failures (about 10^4 time bigger than for nuclear) is perhaps one of the reasons[4] for which the aircraft industry is still allowed to "escape" the quantification of the likelihood of common cause failure events. They perform only qualitative analysis on potential root causes of common cause events and their effect on the system [SAE-ARP 4761, 1996]. They achieve this by conducting careful design and verifying that components and sub-systems are sufficiently "strong" to resist environmental hazards specified in a checklist (that is what they call *Zonal Hazard Analysis*). Then, they produce evidence that the system, as a whole (e.g. the aircraft), will resist particular risks specified on another checklist, for example the impact of a bird, fire, tyre burst (by performing what they call *Particular Risks Analysis*). Finally, they verify that events in minimal cut sets are sufficiently uncoupled against possible causes of common failure specified into another checklist, this is achieved by performing what they call *Common Mode Analysis* [SAE-ARP 4761, 1996]. Checklists are provided by regulatory authorities, as well as being maintained by developers, and the aim of these analyses is to

---

[1] This is partly achieved since in the nuclear history of about $2*10^4$ civil reactor per year (i.e. 500 reactors running per 40 years) we have had only one critical accident: Chernobyl. However Russian reactors were built with a critical failure rate of 10^{-3} per year. Hence Chernobyl should not be taken into account. Three Mile Island accident is not to be considered a critical accident, since the container worked properly and avoided the spreading of long life radioactivity into the environment.

[2] The frequency of *10^{-6} critical reactor failures per year* is equivalent to *$1.1*10^{-10}$ reactor failures per hour* (i.e. *10^{-6} critical reactor failures per year* divided *$8.76 *10^3$ hours per year*). This is almost 10 time smaller than the failure frequency of $1*10^{-9}$ aimed for critical failures in civil aircrafts.

[3] The actual failure rate perceived by common people for critical failures in civil aircraft can be quantified as follows. If we suppose that there are 10^4 aircraft flying every day around the world, each flying $5*10^3$ hours per year, losing 12 aircraft every year, this means that the actual critical failure rate is around $4*10^{-6}$ per hour (i.e. *10^4 aircraft around the world* times *$5*10^3$ hours flown per year* divided *12 aircraft lost in one year*). http://www.ntsb.gov/Aviation/Table1.htm reports "*0.012 critical accidents per 10^5 flight hours*" that is not far from our estimation. [Boeing, 1996] also reports similar values.

[4] Another reason is the difficulty of estimating failure rates for some software components.

produce evidence that minimum requirements are met. However, to the best of our knowledge, regulatory authorities do not ask for any quantitative evaluation of the impact of couplings that cannot be removed.

One of the reasons that the quantitative estimation of common cause failure is "*escaped when possible*", is that in the way it is performed by the nuclear industries it is expensive and largely based on the estimation of some parameters which may often have a large uncertainty. In many cases values for these parameters are given by field experts (expert judgement), in other cases a conservative value is taken *a priori*. While the first option can be impractical (lack of experts for specific fields) and expensive (in some cases there are very few experts all over the world), the second option penalises good systems.

Hence, if we could mechanise the process of common cause failure analysis by supporting and facilitating expert judgement, we would also improve the chance of quantitative common cause failure analysis being more frequently used.

1.5 Central Proposition and Objectives

The central proposition of this thesis is the following:

> *"It is possible to produce an integrated safety analysis framework which can be used to produce a complete and consistent safety analysis, including treatment of common cause failure and which can be used to drive "a design-for-safety" process."*

The main objectives of this research work are:

a) Study the current industrial practice for safety analysis of critical computer based systems and for common cause failure analysis;

b) Provide a method and a notation usable throughout the lifecycle, that supports the *design-for-safety* of computer based safety critical systems;

c) Provide a method that supports common cause failure analysis;

d) Give specifications for a software tool that supports the proposed method.

1.6 Scope of Study and Methodology

The foundation of this thesis is the techniques widely used by the automotive, aeronautical and nuclear industry for the analysis of critical computer based systems. Some of these techniques, i.e. FHA, HAZOP, FMEA, FTA have been used for almost 30 years.

This thesis addresses the part of the lifecycle that goes from the decomposition and design to integration and verification stages. It concentrates on linking existing techniques and in proposing a novel method for the qualitative and quantitative estimation of common cause failures. Case studies have been done on a *Fuel System* and a *Computer Assisted Braking system*.

1.7 Organisation of the Thesis

The thesis is divided into seven chapters: chapters one and seven providing an introduction and a conclusion to the thesis, respectively. The key contribution of the thesis is contained in chapters four and five. The literature survey and the work that brought to the formulation of the main method presented in the thesis are in chapters two and three, respectively.

Chapter Two - Techniques for Safety Analysis

In the second chapter we review the main safety analysis techniques used for the assessment of critical computer based systems by presenting principles that underlie individual techniques. Although those techniques are mostly used in the nuclear and aerospace industry, particular attention is reserved for what has been done for software in safety critical applications. Then we focus on techniques for the analysis of common cause failures. We explain the mechanisms of common cause failures and explore the various ways common cause failures are currently investigated. We close the chapter pointing out areas where further research is needed and setting out the questions that we aim to address in the thesis.

Chapter Three – Preliminary Work

The third chapter summarises the work that was done at the beginning of our research and that brought us (through many refinements) to the formulation of the method, known as Failure Logic Analysis for System Hierarchies (*FLASH*). It highlights the process underneath the development of the technique and explores some alternative approaches.

Chapter Four – Failure Logic Analysis for System Hierarchies

This chapter presents the basic *FLASH* method, as it would be used in an idealised top down process. *FLASH* aims to support the lifecycle making possible *Design-for-Safety*. *FLASH* creates a framework, linking several continuous phases of the lifecycle, pointing out inconsistencies among designs representing different phases of the lifecycle, linking low level analyses to the *FHA* and supporting dependent failure analysis. *FLASH* is applied in two different stages of the lifecycle. In the first stage it checks the evolving design against higher-level safety requirements and supports the establishment of derived safety requirements for each sub-system. In the second stage it verifies whether the product as implemented and integrated meets its concept level and derived safety requirements.

Chapter Five – Common Cause Failure

This chapter extends the *FLASH* formalism presented in chapter four to treat common cause failures. We show how the hierarchy of *FLASH* tables can be used to identify those minimal cut sets that need to be analysed for common cause failures. Additionally, we provide a novel method for quantitative estimation of the likelihood of minimal cut sets with coupled events that uses some of the information collected during *FLASH* analysis.

Chapter Six – Case Studies

This chapter outlines the application of the proposed method on two case studies. We show different stages of the application of the method and highlight the most important features. Each case study is separately evaluated and compared with what could be achieved by using other analysis techniques. The pragmatics of dealing with complex evolving designs is presented here.

Chapter Seven – Conclusion

This chapter provides a summary of our research work, draws the conclusions of the thesis and highlights potential areas for further development.

Chapter two

Techniques for Safety Analysis

2.1 Introduction

In this chapter we review the main safety analysis techniques (as well as recently proposed variations of those techniques) used in the assessment of critical computer based systems. In the first part of the chapter we present the principles that underlie individual techniques and we use four criteria to compare and highlight similarities and differences among those techniques. In the second part of the chapter we focus on techniques for the analysis of common cause failures. We identify the mechanisms of common cause failures and explore the various ways common cause failures are investigated in current practice. Finally we point out areas where further research is needed and set out the questions that we aim to address in this thesis.

The four criteria against which we will examine and categorise the main safety analysis techniques are as follows:

1) Aim;
2) How they explore the relationship between causes and effects;
3) Position in the lifecycle;
4) Presentation of results.

The first criterion explores the primary "*Aim*" of the technique under examination. As Table 2-1 indicates there are techniques that primarily aim to produce a *qualitative* analysis, for example by generating a list of potential failures that affect a system, and techniques that produce a *quantitative* analysis for example predicting the frequency of some critical accidents. Besides those two classes of techniques, there is a third class formed by techniques that enable *both* qualitative and quantitative analysis.

Aims	Example of possible outputs
Qualitative Analysis	Generating a list of potential failures that affect a system
Quantitative Assessment	Predicting the frequency of critical events
Both Qualitative Analysis & Quantitative Assessment	A graph resembling a tree with probabilities associated with each leaf, branch, ramification and root

Table 2-1: Aims of Safety Analysis Techniques

The second criterion in our categorisation considers the way safety analysis techniques proceed in their investigation i.e. "*how they explore the relationship between causes and effects*". There are at least four different ways to proceed. There are *deductive* techniques that start from *known* effects to seek *unknown* causes, *inductive* techniques that start from *known* causes to forecast *unknown* effects, *exploratory* techniques that link *unknown* causes to *unknown* effects and *descriptive* techniques [Fenelon et al., 1994] that link *known* causes to *known* effects. The above categorisation scheme is illustrated in Table 2-2.

		Effects	
		Known	***Unknown***
Causes	***Known***	Descriptive techniques	Inductive techniques
	Unknown	Deductive techniques	Exploratory techniques

Table 2-2: Four ways to investigate the causes-effects relationship

The third criterion in our categorisation is "*Position in the lifecycle*". Some techniques are used at different stages in the development process to provide feedback to the design and development process. The techniques that are used at the beginning of the design process focus on the analysis of the abstract concept of the system. They identify potential failure modes to give advice for the development of the architecture of the system. We refer to these techniques as being used *early* in the life cycle. The group of techniques that follows, concentrates on the analysis of the architecture of the system. At this stage, the allocation of functions to sub-systems and components is known and the purpose is to identify hazards that may arise due to (abnormal) deviations of flows between components of the architecture. We refer to these techniques as being used in the *intermediate* phases of the lifecycle. Finally, there are techniques which are used after the full design process is completed. They mainly perform confirmatory analyses to determine whether or not the full design meets specifications and requirements. We refer

to these techniques as being used in *later phases* in the lifecycle. Beside these three groups of techniques, a further group is formed by techniques used *across* the design lifecycle. These techniques can usually provide continuous feedback to designers. This categorisation scheme is illustrated in Table 2-3.

Position in the lifecycle	Description
Early	Analysis of the abstract concept of the system
Intermediate	Analysis of the architecture of the system
Late	Assessment that the full design meets specifications and requirements
Across	Provide continuous feedback to designers

Table 2-3: Position in the lifecycle

The fourth and last criterion in our categorisation is based on the *"presentation of results"*. There are some techniques for safety analysis that provide results in a *graphical* format and others that provide them in *tabular* forms. A graphical format provides a more intuitive and perhaps easier to understand representation of the results from the assessment. It is also generally easier to relate the failure and recovery logic depicted in a graph back to the system design. However, as the graph grows, fragmentation becomes inevitable and the intuitive capacity is jeopardised, since the graph becomes difficult to read. Conversely, the tabular format can provide a quantity of detailed information which is easy to be read but less intuitive. There are only a few safety analysis techniques that provide *both* results in a graphical and tabular output for the same information. Those are among the techniques surveyed in the next section. Details of this criterion are summarised in Table 2-4.

Presentation of results	Features
Graphical	Intuitive, understandable, relate to the system representation of the logic or sequences of failures and recovery measures
Tabular	A lot of detailed information easy to be read
Both tabular and graphical	Intuitive and easy to read

Table 2-4: Presentation of results

2.2 Safety Analysis

Having explained the four criteria that will help us examine, relate and contrast different safety analysis techniques, we can now proceed to the review. The presentation of each technique starts with a brief historical background and proceeds with a more detailed description of the technique which also identifies the position of the technique in the above classifications.

2.2.1 Preliminary Hazard Analysis

Preliminary Hazard Analysis (PHA) was introduced in the late sixties (1966) after the Department of Defense of the United States of America requested safety studies to be performed at all the stages of product development. They issued guidelines that were applied from 1969 onward [MIL-STD-882, 1969] [MIL-STD-882d, 1999].

The Preliminary Hazard Analysis technique is used in the later stages of requirement analysis and in the early stages of the design process (*early in the lifecycle*). The purpose of Preliminary Hazard Analysis is to identify safety critical areas, to provide an initial assessment of hazards, and to define requisite hazard controls and subsequent actions. The technique is not well formalised. It typically consists of brainstorming where the preliminary design is discussed on the basis of the experience of people involved in the brainstorming activity. Check lists are commonly used to help in identifying hazards. Results are presented in a tabular format. Table 2-5 displays a piece of a Preliminary Hazard Analysis table as an example. It has been made out for two of the hazards that may arise with a computerised braking system in a car. The first column of the table reports hazards that have to be investigated, for instance the loss of the braking capabilities of a car and uneven braking. The second column describes the effects of the hazard, in our case the possible death and injury of people or directional instability. The third column reports the severity level for the hazard (e.g. catastrophic, critical, marginal or negligible). The fourth column sets out the conditions in which the hazard produces the most serious effects. The fifth column reports the exposure to danger, that is a measure of the time spent within the area of danger. The sixth and last column gives information about the ability of the system or the driver to avoid danger. Hazards listed in the first column are usually taken from a Preliminary Hazard List [MIL-STD-882c, 1993] that is compiled before the actual Preliminary Hazard Analysis.

Often Preliminary Hazard Analysis tables have a few additional columns. They are domain specific, defined by the company or even by the customer. In our table Effects,

Criticality, Co-effectors, Exposure to danger and Avoidance to Danger are the output of the analysis, while the Hazard is the input.

Preliminary Hazard Analysis is a *qualitative* technique. It *explores* relationships among potential causes (i.e. the hazard) to give *unknown* effects the (accident) hence it is *inductive*. It is applied only during the *early* stages of the developing process and produces a *tabular* output.

Hazard	Effect (accident)	Severity	Co-effectors	Exposure to danger	Avoidance of danger
Loss of Braking	Death or serious injury to occupants of the vehicle, other vehicles or pedestrians	Critical	High speed travel and requirement to slow down or stop	Frequent = 1e-2 [1/h]	Unlikely to avoid danger
Uneven Braking	Directional instability. Death or serious injury to occupants of the vehicle, other vehicles or pedestrians	Critical	Heavy traffic, Hazardous road condition	Frequent = 1e-2 [1/h]	Likely to avoid danger

Table 2-5: Preliminary Hazard Analysis table

2.2.2 Functional Hazard Assessment

Functional Hazard Assessment (FHA) approaches the analysis of the top-level design from the functional viewpoint [SAE-ARP 4754/4761, 1996]. The aim of this technique is to identify which functions of the system contribute to hazards, and thus assigning them a criticality level. Functional Hazard Assessment was developed by the aerospace industry to bridge between hardware and software, since functions are generally identified without specific implementations. It requires domain specific knowledge to produce meaningful results from Functional Hazard Analysis. The output is a set of tables which give for each function, for each failure condition, and for each phase, a description of effects, mitigation procedures, and often the type of analysis that has to be performed to have the system accepted by regulatory authorities. Table 2-6 shows a standard Functional Hazard Assessment output table as reported by the Aerospace Recommended Practice [SAE-ARP 4761, 1996]. The first column lists functions that have to be assessed (i.e. Decelerate Aircraft on the Ground). For that function, the second column lists the

failure conditions (i.e. Loss of Deceleration Capability, Partial Loss of Deceleration Capability) that may apply to each function. In our case each of the two failure conditions have four sub-cases (i.e. *a-b-c-d*). Identical failure conditions e.g. sub-cases *a* and *c* (or *b* and *d*) have different effects on the aircraft if they happen in different operational states e.g. taxying or landing of the aircraft.

Function	Failure Condition (Hazard Description)	Phase	Effects of failure Condition on Aircraft/Crew	Classification	Reference to Supporting Model	Verification
Decelerate Aircraft on the Ground	*1. Loss of Deceleration Capability*	*Landing /Run to take off/ Taxi*	*See Below*			
	1.a. Unannuciated loss of deceleration capability	Landing/ Run to take off	Crew is unable to decelerate the aircraft, resulting in a high speed overrun	Catastrophic		Aircraft Fault Tree
	1.b. Annuciated loss of deceleration capability	Landing	Crew selects more suitable airport, notifies emergency ground support, and prepares occupants for landing overrun	Hazardous	Emergency landing procedures in case of loss of stopping capability	Aircraft Fault Tree
	1.c. Unannuciated loss of deceleration capability	Taxi	Crew is unable to stop the aircraft on the taxiway or gate resulting in low speed contact with terminal, aircraft, or vehicles	Major		
	1.d. Annunciated loss of deceleration capability	Taxi	Crew steers the aircraft clear form any obstacles and calls for a tug or portable stairs	No Safety Effects		
	1.e. Inadvertent Deceleration after the aircraft cannot be safely stopped in the ground	Takeoff	Crew is unable to take off due to the application of brakes at the same time as high thrust settings, resulting in a high speed overrun	Catastrophic		Aircraft Fault Tree
	2. Partial Loss of Decelerating Capability	*Landing /Run to take off*	*See Below*			
	2.a. Unannuciated loss of deceleration capability	Landing /Run to take off	Crew is unable to completely decelerating the aircraft before the end of the runway resulting in a potential overrun	Hazardous		Aircraft Fault Tree
	2.b. Annuciated loss of deceleration capability	Landing	Crew selects more suitable airport, notifies emergency ground support, and prepares occupants for landing overrun	Major		
	2.c. Unannuciated loss of deceleration capability	Taxi	Crew may not be able to adequately stop the aircraft before obstacle, resulting in low speed collision.	Minor		
	2.d. Annunciated loss of deceleration capability	Taxi	Crew steers the aircraft clear from any obstacles and calls for a tug or portable stairs	No Safety Effects		
	... ...	...	...			

Table 2-6: FHA table

The operational state is called *Phase* in our table and it is reported in the third column. During the landing phase the failure condition 1.a is classified as *catastrophic*. In case of taxiing, the same failure condition is classified as *major*. The *classification* of failure conditions is reported in the fifth column. Mitigation measures that can be taken to limit effects are reported in the sixth column. Analyses that have to be undertaken to verify that the system meets safety requirements go into the seventh and last column.

Several other techniques have been proposed to achieve a Functional Hazard Analysis. One of these, the Functional Failure Analysis (FFA), is recommended in [Papadopoulos and McDermid, 1999a]. This technique considers three misbehaviours for each function. They are *1) function not provided when requested*; *2) function provided when not required*; and *3) malfunction*. The Functional Failure Analysis table differs slightly from the table described above, but it pursues the same objective.
The aim of the Functional Hazard Analysis is to perform a *qualitative* analysis in the *early* stages of the design process to identify which functions of the system contribute to hazards, thus it is an *deductive* technique. The output is *tabular*.

2.2.3 HAZOP and HAZOP based techniques

HAZard and OPerability study (HAZOP) [CISHEC, 1977] [Kletz, 1992] [Adelard, 1994] was developed by Imperial Chemical Industries in the early 1970's [Lawley, 1974] [Lawley, 1976] and extended to software in the early 1990's [McDermid et al., 1995]. HAZOP is performed after an outline equipment design is proposed showing the main design components and the flows between them. The results of the HAZOP may be either to accept the proposed architecture, subject to some safety-related derived requirements, or to ask for the design to be modified.

HAZOP is a team process, aimed at achieving an "imaginative anticipation of hazards". At a mechanistic level it consists of completing a table according to some "guide words" (e.g. *None*, *More of*, *Less of*, *Part of*, *More than*, *Other*). A guideword describes a hypothetical deviation from the normally expected attributes of a flow. Driven by these guidewords, failure causes and their effects are listed. The acceptability of the effects of the deviations is considered and measures proposed to decrease the likelihood of the failure cause, or to mitigate the effects. Table 2-7 shows an example of a HAZOP table for a hydrocarbon flow feeding a chemical reactor. The first column reports two of the guidewords that drive the analysis, i.e. none and more. The team starts

from these guidewords to identify deviations to the expected behaviour of the flow that are placed in the second column (i.e. No Flow, More Flow, More Pressure, More Temperature, etc.). In the third column the team records potential causes of deviation in the flow (in our case the flow feeding the chemical reactor). For instance, there may be no hydrocarbon available in the storage tank or a failure of the pump feeding the reactor. Consequences of each deviation are recorded into the fourth column. In our case this give rise to the formation of polymers in the heat exchanger. The last column reports actions that the team recommends for reducing the hazard.

The aim of the HAZOP is to perform a *qualitative* analysis in the *intermediate* stages of the design process to anticipated hazards, thus it is an *exploratory* technique. The output is *tabular*.

Guide Word	Deviation	Possible Causes	Consequences	Action Required
NONE	*No flow*	No hydrocarbon available from storage Transfer pump fails (motor fault, loss of power, impeller corroded etc.)	Loss of feed to reactor. Polymer formed in heat exchanger As above	1) Ensure good communication with storage area 2) Install low level alarm on settling tank Covered by 2)
MORE	More flow	Level control valve fails to open, or Level Control Valve bypassed in error	Settling tank overfills	3) Install high level alarm 4) Check size of overflow 5) Establish locking-off procedure for Level Control Valve bypass when not in use
	More Pressure	Isolation valve or Level Control Valve closed when pump running	Line subjected to full pump pressure	6) Install kickback on pumps
	More Temperature	High intermediate storage temperature	Higher pressure in transfer line and settling tank	7) Install warning of high temperature at intermediate storage
...	...	...	...	

Table 2-7: HAZOP table

HAZOP has been traditionally used for hazard identification at plant level. More recently though we have seen categorisations of abstract failure classes for software components [Ezhilchelvan and Shrivastava, 1986], [Bondavalli and Simoncini, 1990], and a number of HAZOP-inspired techniques for hazard analysis of software architectures [Burns and Pitblado, 1993]. The early extension of HAZOP to computers was called CHAZOP, for Computer HAZOP. However CHAZOP was really an extended checklist, and did not really build on ideas of flows and guidewords. Work in York produced the Software Hazard Analysis and Resolution in Design (SHARD) [McDermid and Pumfrey, 1994] which is much more HAZOP-like, but applied new guidewords i.e. *Early*, *Late*, *Omission*, *Commission*, and *Value*, rather then the classical guidewords. Like HAZOP,

SHARD is used to analyse an outline design and can produce derived safety requirements.

The aim of SHARD is to perform a *qualitative* analysis in the *intermediate* stages of the design process to anticipate hazards, thus it is an *exploratory* technique. When hazards are known, SHARD may also be used in a *deductive* mode (i.e. for the analysis of embedded systems). The output of SHARD is *tabular*.

Another technique that originated from HAZOP is the Failure Propagation and Transformation Notation (FPTN) [Fenelon & McDermid, 1993] [Fenelon et al., 1994]. This is a hierarchical graphical notation that represents system failure behaviour. It is linked to a design notation and, like HAZOP and SHARD, is both an inductive and deductive analysis. FPTN makes consistency checks and is designed to be used at all stages of the life cycle. FPTN represents a system as a set of interconnected modules; these might represent anything from a complete system to a few lines of program code. The connections between these modules are failure modes, which propagate between them. Figure 2-1 displays a FPTN module. Each module has a set of input failures, to which it is susceptible (i.e. A:t, B:t, C:Vu, X:Vd at the left side of the module), and a set of output failures, which it propagates (i.e. D:o, E:c, F:o at the right side of the module). A module can also generate new failures (e.g. F:o) and handle existing ones (e.g. X:Vd). Equations inside the module show how the input and the internally generated failure modes contribute to the output failure modes (i.e. D:o = A:t & B:t; and E:c = B:t | C:v). Figure 2-1 displays also that an FPTN module may record the criticality of the module (in the right top corner), and whether the module is further decomposed into the other more simple modules (the shadow).

FPTN is a *qualitative* technique that can be performed at any stages of the design process, thus *across* the lifecycle. Its role is to summarise analyses, thus it is a *descriptive* technique. The output is *graphical*.

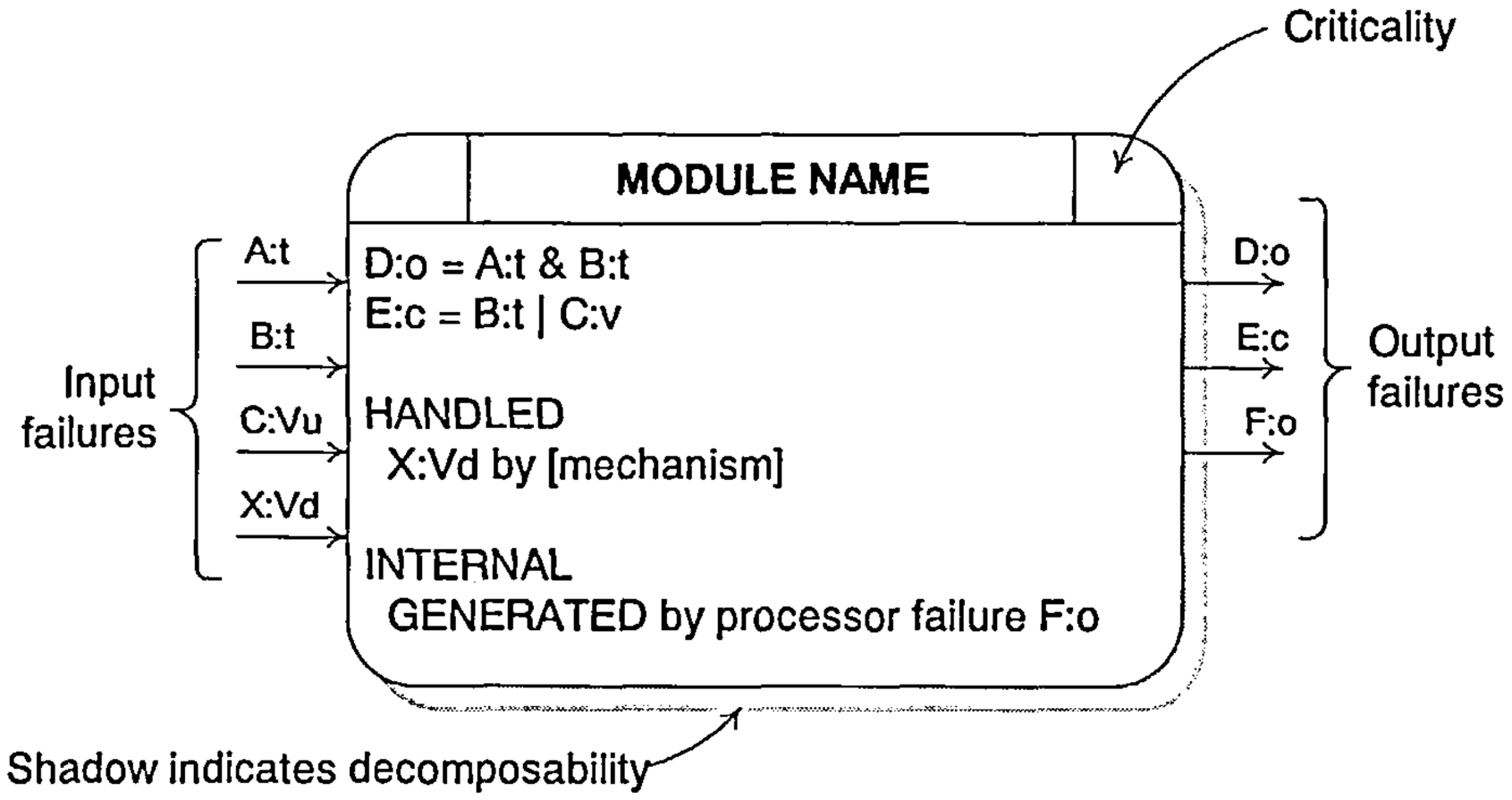


Figure 2-1: FPTN module

2.2.4 FMEA

Failure Modes and Effects Analysis (FMEA) has been developed since the sixties [Recht, 1966] for studying aircraft safety, then it was used for space applications [Bussolini, 1971], for chemical plants [King and Rudd, 1971] [Lees, 1980] and car manufacturing [Yamada, 1977]. FMEA was recommended for Nuclear installations after the accident at the Three Mile Island power station [NUREG 2300, 1983]. Many standards deal with FMEA. Guides were published by the US Department of the Navy [MIL-STD-1629a, 1980] and the Institute of Electric and Electronic Engineers [IEEE, 1975].

FMEA is an *inductive* analysis technique used to study the effects of component failure modes on a system. FMEA starts from knowledge of component failure modes and considers the effects of each failure on subsystems and the system. It involves the study of all the components in a system and is often applied also to higher level assemblies and systems. It checks whether proposed components, with their known failure modes, fulfil system-level safety requirements. The result of the FMEA may be to accept the proposed components or, perhaps, to issue recommendations for maintenance checks, or to ask for components to be substituted. In light of the FMEA, analysts are able to ensure that all the conceivable failure modes and their effects on the system operability are taken into account, although this is clearly a very costly process and, for a complex system might not be practical. It is also common to use FMEA to determine whether or not a design meets the general requirement that "no single point of failure" shall give rise to a hazard.

A classical FMEA output is shown in Table 2-8. The first column lists basic components of the system, the second column lists failure modes that apply to each component. The third and fourth columns respectively list effects on the subsystem and system. The fifth column classifies effects according to their severity, the sixth column gives the failure rate associated with the failure mode, and the last column is left for comments. Thus Table 2-8 tells us that the speed sensor (first column) in a car may fail in various modes, one of these is delivering *No Signal* (second column). This failure produces effects at subsystem level (third column). The subsystem believes that the vehicle is not moving. The system is indirectly affected by this failure since the speed indicator shows a null speed, the mileometer is not incremented and the electronic gearbox selects a wrong gear (fourth column). Obviously the hazard severity for the first and second failure modes is less severe than the third one which may cause loss of lives and the vehicle.

FMEA is a *qualitative* and *quantitative* technique that proceeds from *known* causes to *unknown* effect thus it is *inductive*. FMEA needs the knowledge of the full system design so it is performed *later* in the lifecycle. The output is *tabular*.

Component	*Failure Mode*	*Subsystem Effects*	*Vehicle Effects*	*Haz*	*Failure rate [1/h]*	*Comments*
Vehicle Speed Sensor	No signal	Vehicle speed will always be calculated as zero	1.No speed indication 2.Mileometer not incremented 3.Electronic gearbox control may select too low gear, possibly resulting in wheel lockup or transmission damage	Min Min Maj	5E-5	Effect 3) requires simultaneous failure of engine load calculation and mechanical interlocks on gearbox
Vehicle Speed Sensor	Noisy (too Many edges)	Calculated vehicle speed will be too high. If edges arrive at higher rate than specified, they will be lost	4.Indicated speed greater than actual 5.Mileometer over-reads 6.Electronic gearbox control may select too high gear, possible resulting in stall	Min Min Min	3E-5	Effect 6) is hard to detect via engine load calculation, unless noise is extreme
Vehicle Speed Sensor	Intermit-tent	Calculated vehicle speed will be too low	7.Speed indicated lower than actual 8.Mileometer under-reads 9.As 3)	Min Min Maj	4E-5	See above

Table 2-8: Failure Mode and Effect Analysis table

A natural extension of FMEA is *Failure Mode, Effects and Criticality Analysis* (FMECA). It was introduced almost immediately after FMEA. It is based on FMEA but in addition to this, it performs a criticality analysis verifying that failure modes with severe effects have sufficiently low occurrence probability. An FMECA table has at least two more columns that record the *probability-severity*[5] pair for each failure mode. If the likelihood is high or the consequences severe, the more critical is the failure mode and the need to take corrective measures.

2.2.5 Fault tree and Event tree analyses

Fault Tree

Fault Tree Analysis has developed since the early sixties (1961) when Bell Laboratories introduced this concept as a method to assess the safety of the launch control system of the Minuteman missile [Henley and Kumamoto, 1981]. A few years later fault tree analysis was adopted and improved by engineers working for Boeing [Haasl, 1965] [Fussell, 1973]. But it was not until the eighties that the fault tree construction process was formalised under pressure from the United States Nuclear Regulatory Commission and a handbook was written [Vesely, 1981]. Since then various procedures and tools to support fault tree analysis have been proposed in [Taylor, 1982] [Poucet et al., 1993 b]. In 1995 there were more than a hundred different tools [Sardella, 1995]. However, only recently has fault tree analysis extended to software [Leveson, 1983 and 1991].

The aim of fault tree analysis is to determine the possible combinations of causes that may give rise to some undesired events called *top events*. A fault tree consists of several levels of event connected in such a way that each event, at a given level, is a consequence of events at the level just below, through various logical operators (gates). Events may be equipment failures, human errors, software errors, etc. that are likely to cause an undesired outcome. Figure 2-2 represents a simple fault tree. The Top Event *D* occurs when both the *basic* event *A* and the *intermediate E* get rise. However *E* occurs only when any of the basic events *B* or *C* get rise.

---

[5] If we know the mission time for the system considered in Table 2-8, then we can calculate the likelihood of each component failure mode. Hence we can say that Table 2-8 contains sufficient information to be used also for an FMECA.

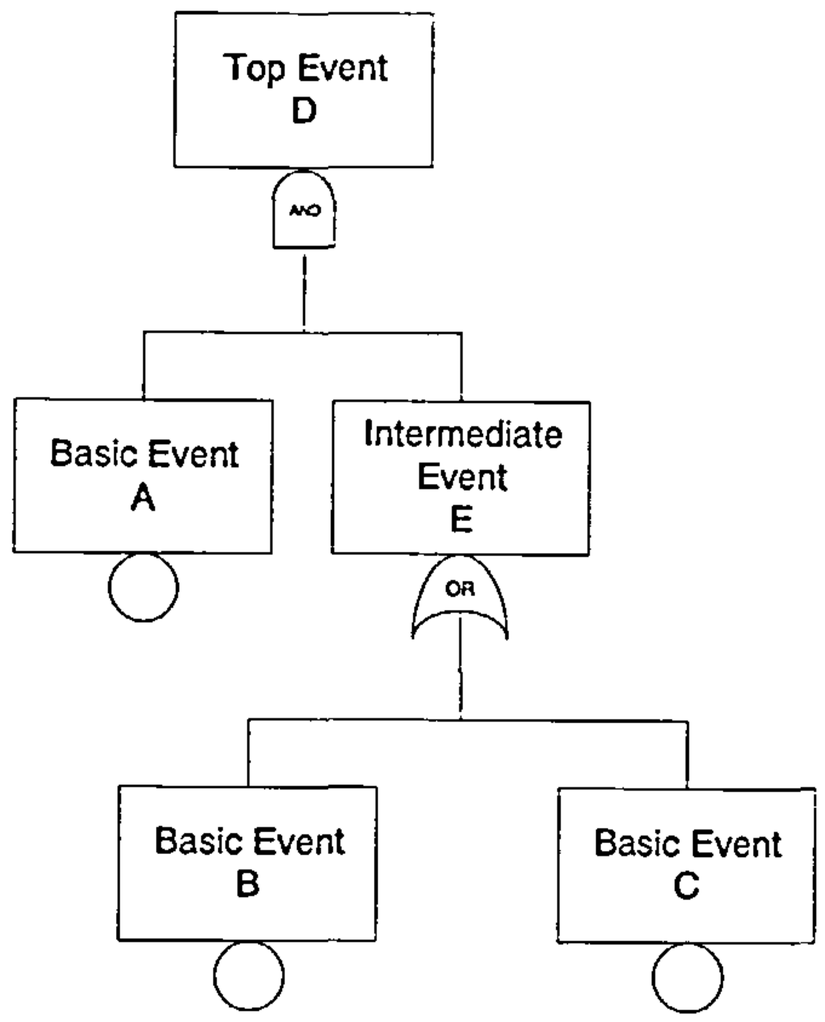
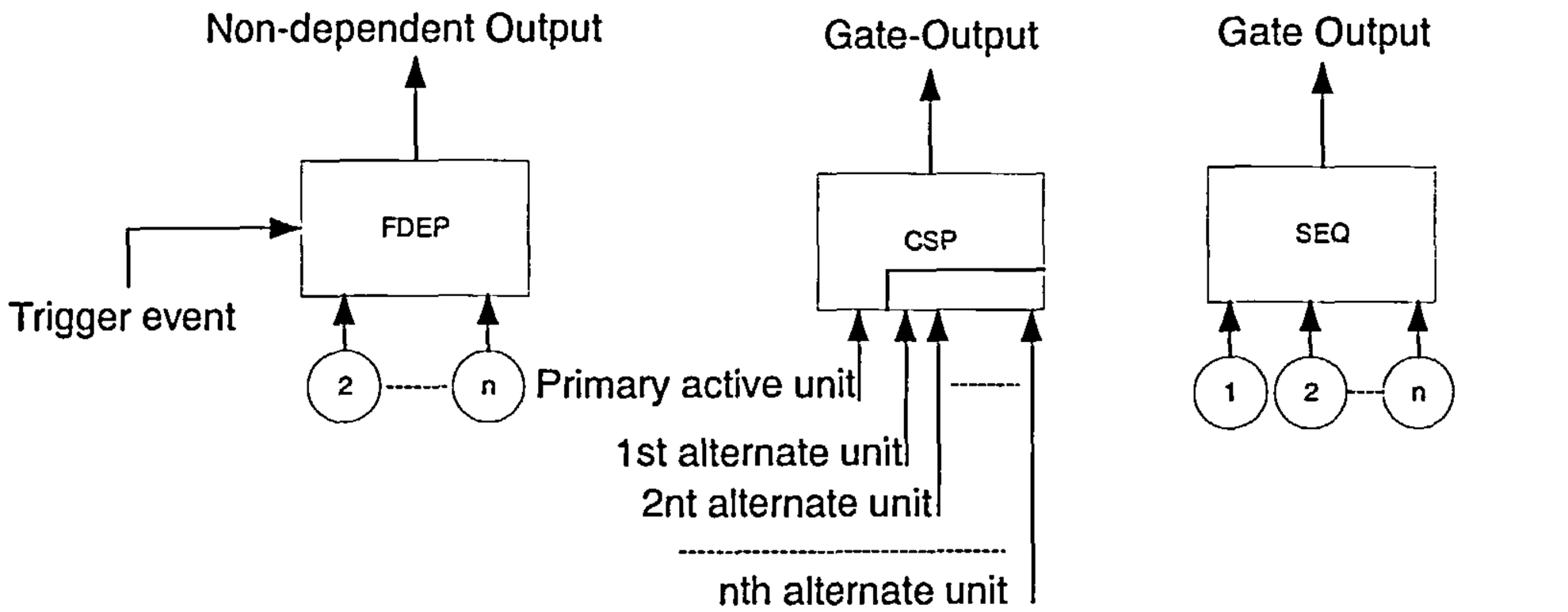


Figure 2-2: Fault Tree

In a fault tree, basic events must be independent of one another. Fault tree analysis is extensively used, as its simple graphical style is readily applied and well understood by practising engineers. In the many years since its introduction, the fault tree technique has gone through many extensions. One of these is the addition of new gates to represent the dynamic behaviour in advanced fault tolerant digital-systems. This extension also made fault trees fully compatible with Markov chains (explained later in this chapter). The fault tree handbook reports five gates AND, OR, XOR (exclusive OR), Priority-AND and INHIBIT [Vesely, 1981]. These gates capture the effects of failures that depend only upon the combination of causal events, but not those that depend on the sequence in which the events occur. There are three sequences of events for which dedicated gates were introduced [Dugan et al., 1993]. Figure 2-3 displays these three new gates. The *Functional dependency gate* (a) represents the functional dependency of the events below the gate from the trigger event depicted on the left side of the gate. When the trigger event happens all the functionally dependent events (below the gate) will happen. The occurrence of any of the functional dependent events has no effect on the trigger event. The *Cold spare gate* (b) models components that are not powered up until they are needed for backup purpose. When the primary event arises (event 1 in Figure 2-3b), then a cold spare is powered up and operates until it fails (event 2 in Figure 2-3b) causing another cold spare to be powered up. The gate is "true" when all the basic events have arisen, and hence all the spare components used up. The basic hypothesis behind this gate is that spare components are *as good as new* until they are powered up for the first

time. The *Sequence enforcing gate* (c) represents events happening in a particular order. This gate fires "true" if and only if all the events listed below the gate happen from left to right. For any other sequence of events the gate does not fire.



a) Functional dependency gate. b) Cold spare gate. c) Sequence enforcing gate.

Figure 2-3: Dynamic fault tree gates

Like FMEA and FMEA derived techniques, fault tree analysis can provide quantitative output, for any state of the system. In fact any fault tree can be reduced to sequences of events connected by only "AND", "OR" gates and negation "NOT" [Contini, 1999b], and eventually be fully represented by a list of minimal cut sets that are the minimum combination of events which, when they happen simultaneously, can cause the top event. The probability of the top event is then estimated by adding up the probability of all the minimal cut sets of the tree. It is not intended in this section to detail how the quantitative evaluation of fault trees proceeds. For that we refer to the fault tree handbook [Vesely, 1981].

Although fault tree analysis is extremely powerful in supporting both qualitative and quantitative analysis, the fault trees technique is very much dependent on the analyst: different teams draw different fault trees for the same system [Amendola, 1986]. To avoid this dependence, several tools which draw fault trees automatically, from Plant and Instrument (P&I) diagrams, have been developed e.g. in [Carpignano & Poucet, 1994]. At present, the weak points of those tools concern mainly the large size of generated fault tree diagrams when they are compared with hand-produced fault trees [Sardella, 1995].

The fault tree technique aims *both* at a *qualitative analysis* and a *quantitative assessment*. However the quantitative assessment is not always possible. It needs knowledge of probabilities associated with basic events (leaf events). In the case of software fault trees it is not possible to associate probabilities with some failure modes, hence fault tree analysis is used only qualitatively.

In addition, fault tree analysis proceeds from *known* effects to *unknown* causes thus it is a *deductive* technique.

The fault tree technique can be used at any stage of the design and development process. Fault tree leaf events may represent functional failures, system failure modes or component failure modes. Thus fault tree analysis can be used at any stage of the design process i.e. functional, architectural and component level, that is *across* the lifecycle.

Finally, the output of the fault tree is a *graph* (resembling a tree) and, when it is possible, it also provides the likelihood of the top event. However, since a fault tree can be represented by the list of its minimal cut sets, this list can also represent the output of the fault tree analysis, hence we can say that fault trees also have a *textual*[6] output.

Event Tree

The event tree technique is an inductive method that develops the possible consequences of a generic initiating event, e.g. a failure. The consequences of such an event can be mitigated, or made worse, by systems dealing with it immediately afterwards. Figure 2-4 shows the event tree that may originate from the initiating event *High Pressure* in the vessel of a chemical reactor. Emergency systems are designed to deal with this event, however they may fail in various ways and, in some circumstances, the vessel may explode with severe consequence. The event tree in the figure shows that when the safety sensor detects high pressure in the vessel, emergency systems are triggered. If the system called into action works, the upper path of each branch (i.e. *Y* = *Yes*) is true, otherwise the lower path (*N* = *No*) is taken. In this example, following always the upper branch, we can see that the initiating event is completely handled and safety is maintained (although the plant is now unavailable). Following this path we see that the input flow is cut off (to avoid any further increase of reagent in the reactor), the output flow is increased to maximum (to facilitate the depressurisation) and the warning lamp lit (to communicate the abnormal state to the operator). If a system does not work we follow

---

[6] They can be represented also in a tabular form.

the lower path. The worst outcome happens when neither the input flow is cut off nor the output flow is increased to maximum and the safety valve does not open (there are two paths like this that are highlighted in the picture). Between these two extremes there is a "grey area" that represents the cases in which some of the safety systems work and some others fail. Remaining paths represent these outcomes.

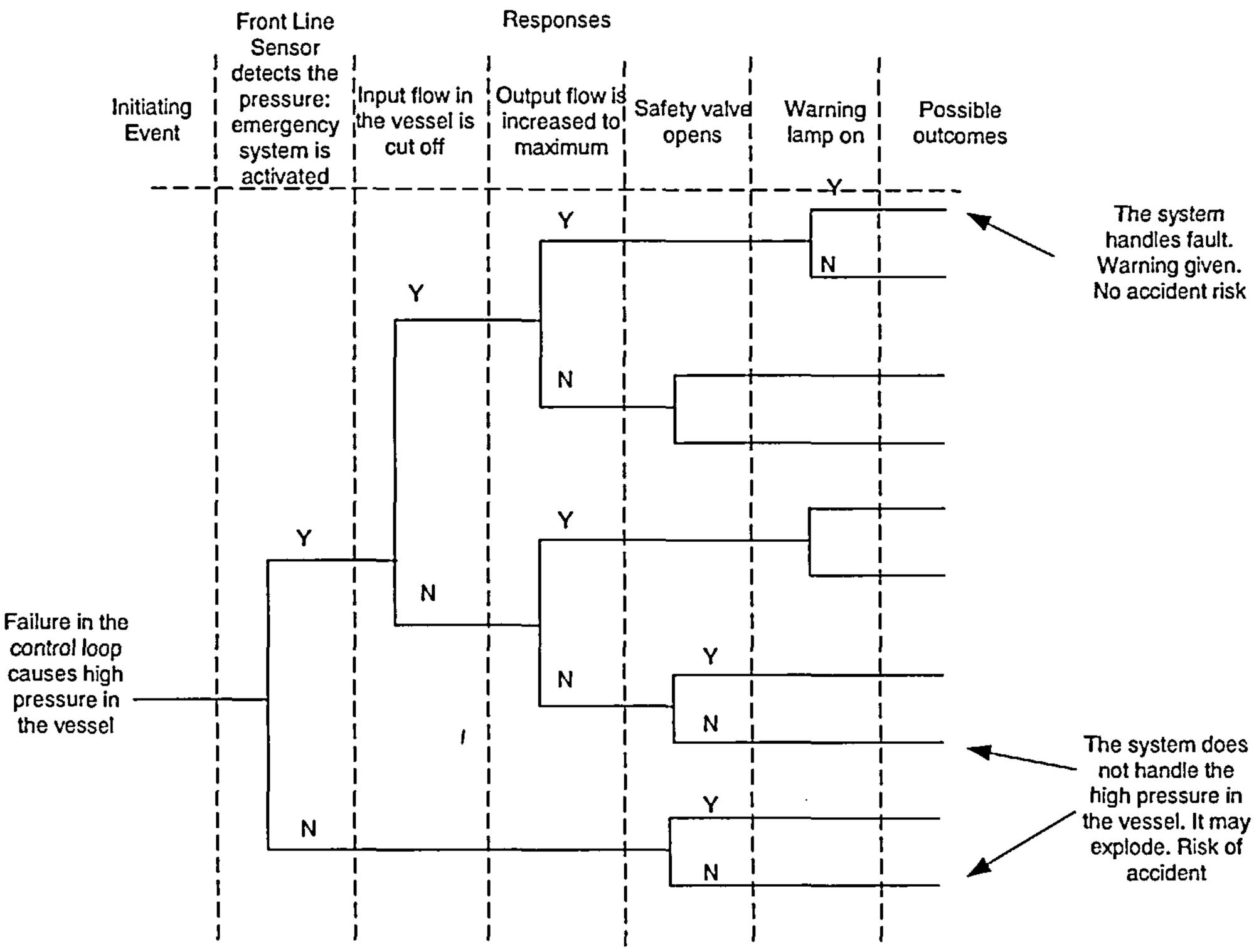


Figure 2-4: Event tree

When probabilities of mitigating events are known, it is possible to calculate the likelihood of each path. It is not intended in this review to detail the quantitative evaluation of event tree paths. For that we refer to the [NUREG 2300, 1983]. Further, event tree mitigating events may represent functional failures, system failure modes or component failure modes. Thus event tree analysis can be used at any stage of the design process i.e. functional, architectural and component level, that is *across* the lifecycle. The event tree is a *graph* and, when it is possible, also the likelihood of each path can be given. An event tree can be represented by the list of its paths, hence we can say that it has also a *textual* (or tabular) output.

Drawing some conclusions, the aim of event tree technique is to provide *both* a qualitative analysis and a quantitative assessment. Event trees proceed from known causes to investigate unknown effects hence they are *inductive*. They can be used at any stage of the design development i.e. *across* the lifecycle. The output is *both* graphical and textual, although use of the graphical is more common.

Large Fault Tree, Small Fault Tree

Two different approaches can be used for a Probabilistic Safety Assessment of complex systems, i.e. Nuclear Power Plants, airliners, etc. First, the Large Event Tree – Small Fault Tree (LET/SFT) approach called also event tree with boundary conditions or, event tree linking or small fault tree. Second, the Small Event Tree – Large Fault Tree (SET/LFT) approach called also fault tree linking or large fault tree. Both approaches use Event trees and Fault Trees to perform the Probabilistic Risk Analysis. The difference between those approaches lies in the fact that in the LET/SFT support systems (e.g. power supplies, water supplies etc.), are modelled in event trees, whereas in SET/LFT support systems are modelled in fault trees. Although LET and SET analysis conducted with the same level of detail give the same numerical result [Rasmussen, 1992], so far, the SET approach has always been preferred to the LET. This is because fault tree construction and analysis (being a *deductive* process) can be extensively automated while event tree construction and analysis (being an *inductive* process) cannot be automated except by using techniques like Monte Carlo simulation. Hence it is preferable to deal with big fault trees rather than with big event trees. Software that deals with large fault trees can be found in [Carpignano & Poucet, 1994; Sardella, 1995].

2.2.6 Markov chains

Markov methods are useful for evaluating components with multiple states i.e. several good, degraded, and critical states [Norris, 1998]. Let us consider the system in Figure 2-5 with three possible states 0, 1, and 2. In the Markovian model, each transition is characterised by a *transition rate* (i.e. failure rate = λ_{2-1}, λ_{1-0}, repair rate = μ_{1-2}, μ_{0-1}). If we define

$\Pr_i(t)$ = probability that the system is in state i at time t.

$\rho_{ij}(t)$ = the transition rate (either λ or μ) from state i to state j.

And if we assume that $\Pr_i(t)$ is differentiable it can be shown that:

$$\frac{d\Pr_i(t)}{dt} = -\left(\sum_j \rho_{ij}(t)\right) \bullet \Pr_i(t) + \left(\sum_j \rho_{ji}(t) \bullet \Pr_j(t)\right)$$

If a differential equation is written for each state and the resulting set of differential equation is solved we obtain the time dependent probability of the system being in each state [Modarres, 1993]. Markov chains are mainly a quantitative *technique* though the state and transition diagram also gives qualitative information about the behaviour of the system.

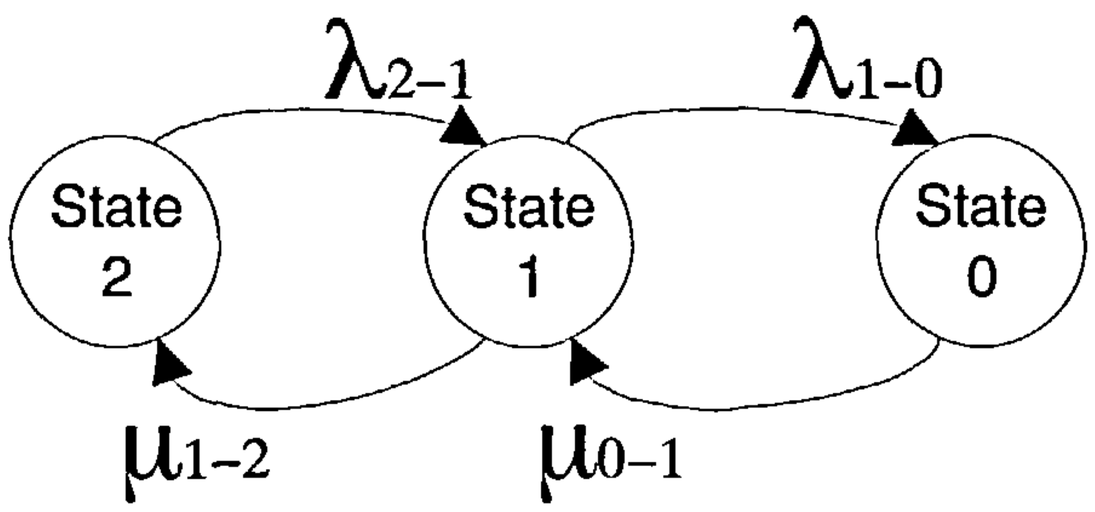


Figure 2-5: Markovian model for a system with three states

2.2.7 Master Plant Logic Diagram

The Master Plant Logic Diagram (MPLD) method was proposed in [Modarres, 1987] as an outgrowth of the Master Logic Diagram [NUREG 2300, 1983] to represent all the physical interrelationships among various plant systems and subsystems in a simple logic diagram. It is used for probabilistic safety assessment to model and integrate the relationship between all plant functions and equipment, therefore it is suitable for several safety applications [Modarres, 1992] such as:

- Understanding and propagating effects of equipment failures;
- Generating and quantifying accident sequences;

- Determining important elements of plant safety and ranking of major contributors to unsafe situations;
- Helping designers and analysts in the identification of risk-significant configurations;
- Evaluating safety implications of an actual event occurrence.

The aim of the MPLD method is to make the construction of a system safety model easy, and to make such a model easy to update. Although fault trees and event trees are well-established methods, as a matter of fact, they become inscrutable and resource-intensive when they extend to multiple pages. Their limitations are especially severe when they are updated following changes that have been made to the system i.e. operation, procedures, hardware, software, etc. Finally, fault trees and event trees are not easily traceable and their independent review and quality control is very time consuming. On the contrary, MPLD is a more intuitive representation of the system and it can be kept up to date more easily when there are changes in design or configuration of a plant. It can also be used to update risk estimates.

In success space, MPLD shows the manner in which various functions, sub-functions, and hardware components interact to achieve the overall system task. Conversely, a MPLD in failure space displays events, i.e. functional failures and relevant hardware failures causing system failures, therefore MPLDs can easily map the propagation of plant hardware failures to the system level [Modarres, 1992].

The hierarchy of an MPLD is shown by a dependency matrix (see Figure 2-6) in which the dependency is established and shown explicitly by a "•". The same picture shows that the failure of each of the functions F_1 and F_2 causes the system failure. Each of those functions is supported by two sub-functions, each of which is enough to provide F_1 and F_2.

The MPLD shows a clear Single Point of Failure (SPF) of the support system S_3 that directly causes the failure of sub-function $F_{2\text{-}1}$ and indirectly (causing the failure of support system S_2) causes the failure of sub-function $F_{2\text{-}2}$. Moreover, the MPLD shows that support system S_1 is provided by two functions ($S_{1\text{-}1}$ and $S_{1\text{-}2}$) that must fail simultaneously to cause S_1 to fail. Finally, the MPLD shows that the failure of the support system S_2 is not critical because that failure can cause neither F_1 nor F_2 to fail. Fault trees would not have allowed the same failure mechanisms to be shown in such an intuitive and compact way.

Like the fault tree and event tree techniques MPLD supports *both* qualitative analysis and quantitative assessment [Modarres, 1992]. It can be performed at any stage of the design process, thus *across* the lifecycle. The output is *both* graphical and textual or tabular. Table 2-9 summarises the graph in Figure 2-6. The likelihood of end state can be quantified.

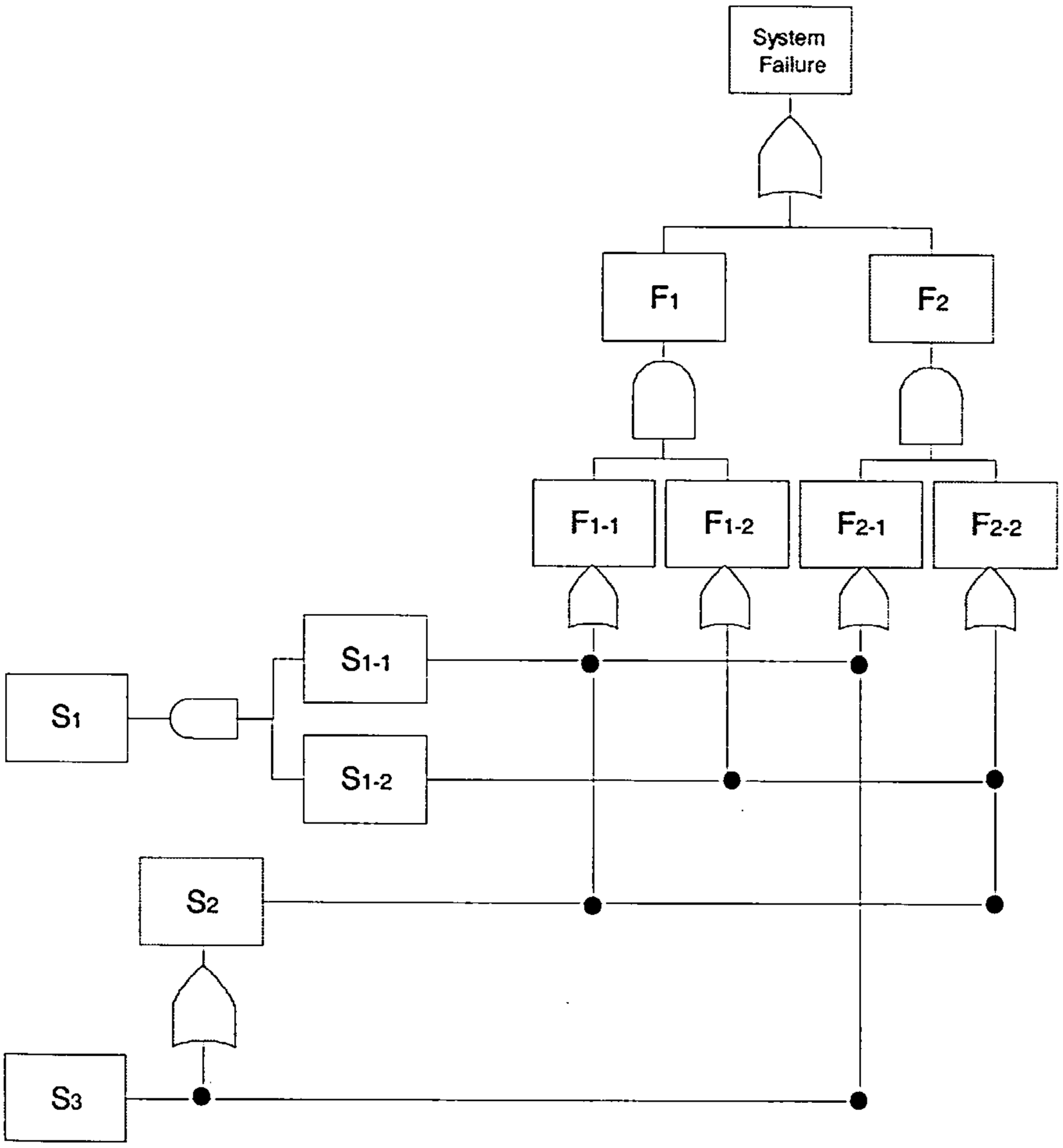


Figure 2-6: An example of MPLD in Failure space

Combinati-on number	Failed support function (or equipment)	Support Function (or Equip.) Failed Because of dependencies	Likelihood	End State
1	S_{1-1}	--		F_{1-1}, F_{2-1}
2	S_{1-2}	--		F_{1-2}, F_{2-2}
3	S_2	--		F_{1-1}, F_{2-2}
4	S_3	S_2		F_{1-1}, F_2
5	S_{1-1}, S_{1-2}	--		F_1, F_2
6	S_{1-1}, S_2	--		F_{1-1}, F_2
7	S_{1-1}, S_3	S_2		F_{1-1}, F_2
8	S_{1-2}, S_2	--		F_1, F_{2-2}
9	S_{1-2}, S_3	S_2		F_1, F_2
10	S_2, S_3	--		F_{1-1}, F_2
11	S_{1-1}, S_{1-2}, S_2	--		F_1, F_2
12	S_{1-1}, S_{1-2}, S_3	S_2		F_1, F_2
13	S_{1-1}, S_{1-2}, S_3	--		F_{1-1}, F_2
14	S_{1-1}, S_{1-2}, S_3	--		F_1, F_2
15	$S_{1-1}, S_{1-2}, S_2, S_3$	--		F_1, F_2
16	No failure	--		No failure

Table 2-9: Combination of support function failure and end states

2.2.8 Taxonomy of Techniques for safety analysis

Techniques for safety analysis discussed so far provide feedback to the design process so that their output is used either to let the design proceed without modification or to recommend improvements. However, it is evident that the presented techniques for safety analysis achieve the feedback to the design in various ways, which the four criteria presented in the introduction of this chapter highlight to some extent. Table 2-10 summarises the discussion that has been undertaken so far by ranking the presented techniques against the four criteria.

Criterion		Classes	Techniques
	Aim	*Qualitative analysis*	Preliminary hazard analysis, Functional Hazard Analysis, Functional failure analysis, HAZOP
		Quantitative assessment	Markov chains
		Both qualitative analysis & quantitative assessment	Fault tree, event trees and FMEA
	Relationship Causes-Effects	*Relationship among known causes and known effects*	*Descriptive techniques* FPTN, Master Plant Logic Diagram
		From known effect to unknown causes	*Inductive techniques* Event tree
		Relationship among unknown causes to unknown effects	*Exploratory techniques* FPTN, Preliminary Hazard Analysis, HAZOP, SHARD
		Relationship among known causes to known effects	*Deductive techniques* SHARD, HAZOP, Fault tree
	Position in the lifecycle	*Early*	Preliminary hazard analysis, Functional hazard analysis, Functional failure analysis
		Intermediate	HAZOP, SHARD, FPTN
		Late	FMEA
		Across	Fault tree, Event tree, Master plant logic diagram,
	Presentation of results	*Tabular*	Preliminary hazard analysis, Functional hazard analysis, Functional failure analysis, HAZOP, FMEA
		Graphical	FPTN
		Both tabular and graphical	Event tree, Fault tree, Master Plant Logic Diagram

Table 2-10: Techniques for safety analysis listed against the four criteria

2.3 Common Cause Failure Analysis

Common cause failure analysis has its own section in this review since it considers failure events that cannot be dealt with (explicitly) by techniques presented in the previous section. These failure events are not usually considered as independent events occurring within a system, but as *influences on the system* from some source that are common to redundant components, resulting in some abnormal output states.

The first problem in dealing with common cause failures is the definition of an unambiguous terminology. This was perceived in the many meetings that we were involved in during our research. Hence, we begin this section by presenting results of research into the terminology describing common cause failures by detailing the terms that we will be using in the development of the thesis.

The terminology on common cause failure has changed over the years. In the beginning only *common mode failures* were considered [Edwards and Watson, 1979]. Later, the definition of *common cause failure* was introduced referring to a slightly wider group of failures [Bourne et. al., 1981] superseding common mode failures. However, at that time the idea that common cause failure was synonymous with common mode failure was widespread. The issue regarding the difference between common *cause* and *mode* was clarified in 1985, when the term *dependent failures* was introduced to supersede and encompass common *cause*, common *mode* failures and "*cascade failures*". Table 2-11 gives the definitions of *dependent*, common *cause*, common *mode* and *cascade* failures as given by the safety and reliability directorate of the United Kingdom Atomic Energy Authority in an official document [Humphreyes and Johnston, 1987]. Cascade includes all dependent failures that are not common cause failures [EPRI, 1985; Johnston and Crackett, 1985]. Figure 2-7 summarises what we have said so far. Common mode failures are a subset of common cause failures, whilst dependent failures encompass both common cause and cascade failures. We agree with these definitions and we use them in the rest of the thesis.

Dependent failure (DF)	The likelihood of a set of events, the probability of which cannot be expressed as simple product of the unconditional failure probabilities of the individual events.
Common cause failure (CCF)	This is a specific type of dependent failure that arises in redundant components where simultaneous (or near simultaneous) multiple failures result in different channels from a single shared cause.
Common mode failure (CMF)	This term is reserved for common-cause failures in which multiple items fail in the same mode.
Cascade failure (CF)	These are all those dependent failures that are not Common Cause, i.e. they do not affect redundant components.
Further: The term "Dependent failure" as defined above is designed to cover all definitions of failures that are not independent. From this definition of dependent failure it is clear that an independent failure is one where the failure of a set of events is expressible as simple product of individual event unconditional failure probabilities.	

Table 2-11: Definitions

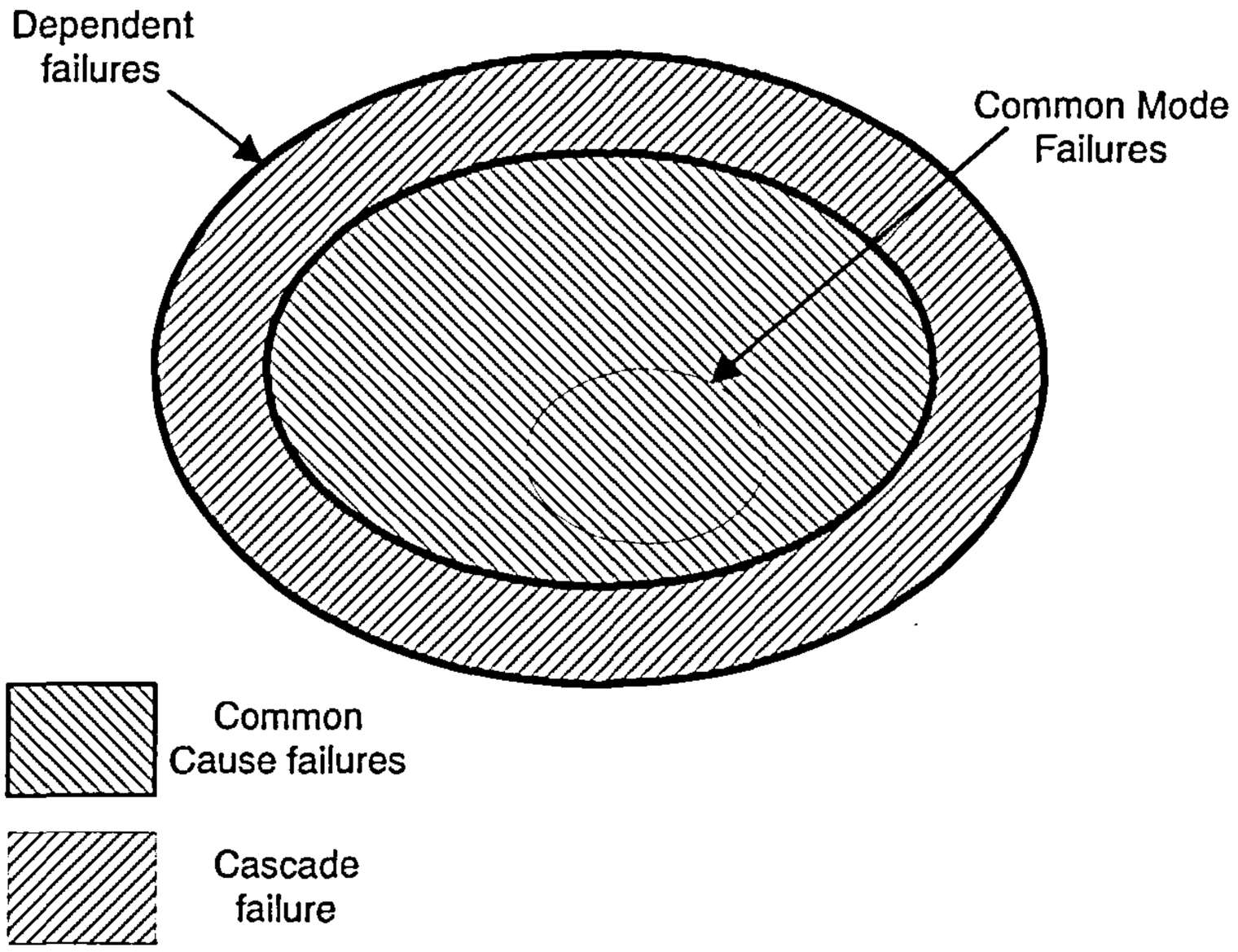


Figure 2-7: Dependent failures

2.3.1 Dependent failure events

The theoretical definition of dependent events can be found in statistics and probability books. In [McCord and Moroney, 1964; Peyton and Peebles, 1987] we can find that given two dependent events A and B, the probability that both events A and B happen, is not equal to the product of the two unconditional probabilities:

$$P(A \text{ and } B) = P(A) \bullet P(B|A) = P(B) \bullet P(A|B) \neq P(A) \bullet P(B) \qquad (2\text{-}1)$$

More specifically, in this thesis, we are concerned with the situation in which the likelihood of two (or more) events is greater than the product of the likelihood of each single event:

$$P(A \text{ and } B) > P(A) \bullet P(B)$$

2.3.2 Common cause failure events

The reference document for studying common cause failures is NUREG 4780 [Mosleh et al., 1993]. The author says that to understand the mechanisms leading to dependent events, and to model them, it is necessary to answer questions like:

- Why do components fail or why are they unavailable?
- What is it that can lead to multiple failures?
- Is there anything at a particular facility that could prevent such multiple failures occurring?

The *root cause*, the *coupling factor* and the existence or lack of *engineered or operational defences* against unanticipated equipment failures are the answers to such questions. The root cause explains the mechanism underlying the transition from available to failed or functionally unavailable. For example, if two components are located in the same room and they are susceptible to high humidity, a common cause failure could occur as a result of an event outside the room but causing high humidity in the room. In this case high humidity is the root cause of failure for the two components.

Given the existence of the root cause, the coupling factor explains why a particular cause affects several components. It creates linking conditions to cause multiple components to fail in a correlated fashion. For example, location in the same room is a coupling factor for those components susceptible to high humidity. Figure 2-8 shows the mechanism of failure of multiple components, that is whenever there is a coupling factor (e.g. same location) and a trigger event (e.g. failure of an air conditioning system) occurs, the root cause (e.g. high humidity) acts causing multiple components to fail.

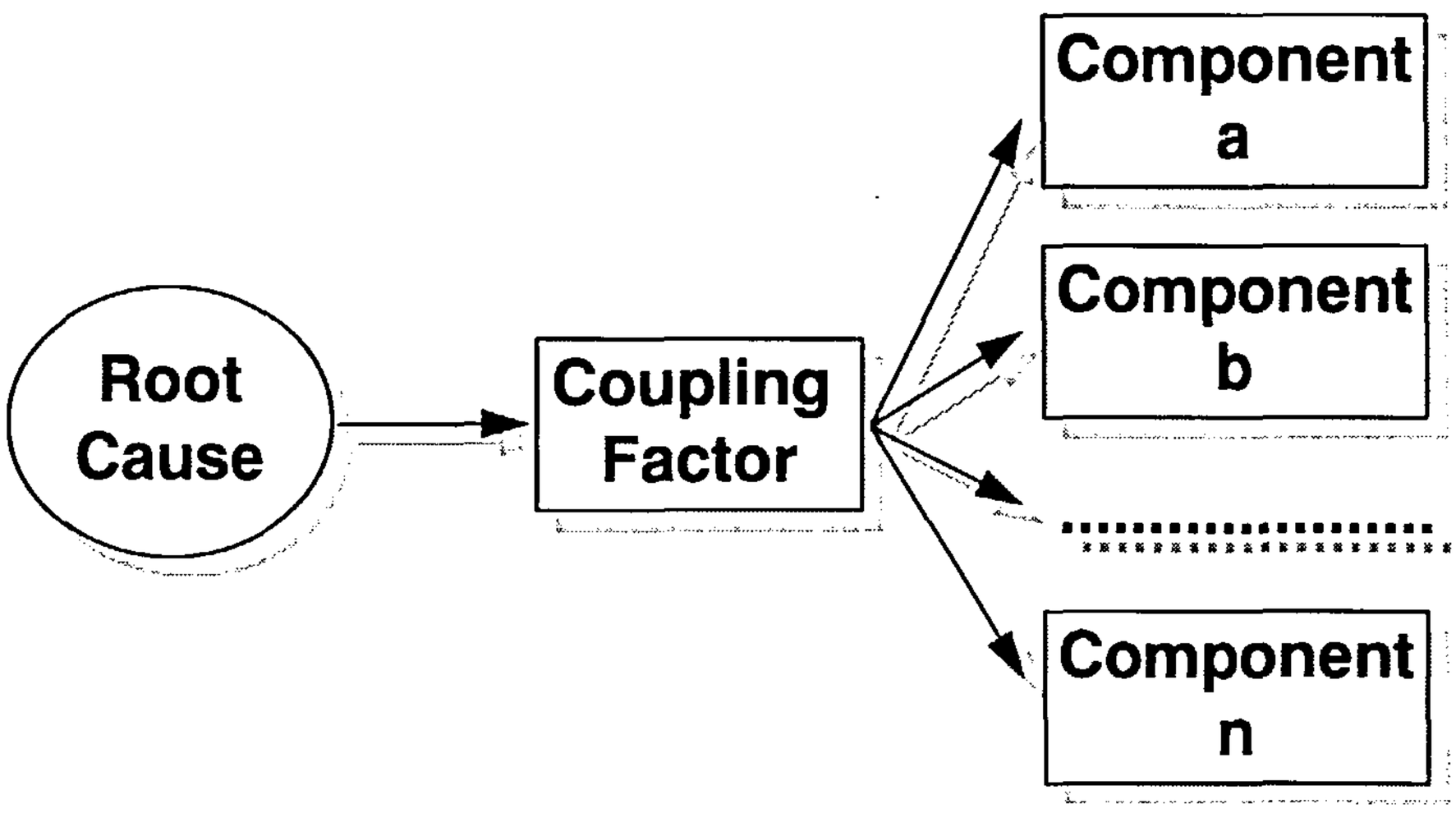


Figure 2-8: The root cause through the coupling factor affects several components

Engineered defences means all those mechanisms that could be adopted to prevent root causes and couplings from occurring. It is possible to act in two different ways: preventing root causes and/or reducing coupling factors.

In the first case the susceptibility of components to particular root causes (e.g. humidity) has to be reduced. In the second case we need to increase diversity. This is possible with techniques of design control and quality control that help in segregating equipment and in ensuring high quality construction.

2.3.3 Common mode failure events

Systems using redundancies, and fault tolerant systems in general, are able to continue operating despite the failure of a limited number of their hardware or software components. This is so when the failures are of individual components independently, but these systems are vulnerable to common mode failures. These failures may sometime endanger safety critical systems, hence they are of interest for safety analysts. It is generally recognised that there are four different types of common mode failures [Edwards and Watson, 1979; Humphreyes and Johnston, 1987]:

1) The coincidence of failures of two or more identical components in separate channels of a redundant system, due to a common cause (the failures will probably have common failure mode also).
2) The coincidence of failures of two or more different components in separate channels of a redundant system due to a common cause (the failures will probably have common failure mode also).
3) The failures of one or more components which result in the coincidence of failures of one or more other components not necessarily of the same type, as the consequence of some single initial cause (the primary and secondary failures might also be coincident, and any coincidental failures might have different failure modes but all will be in the same category).

N.B. In any of the above cases, the failure can occur at the same instant or at different times, but at some time the failed states will be coincident.

4) The failure of some single component or service which is common to all channels in an otherwise redundant system (e.g. common maintenance, test). This only includes component services which are an integral part of the system and on which system operation is dependent.

On the basis of these types of failure, Edwards and Watson gave their definition of common mode failure:

> *"A common-mode failure (CMF) is the result of an event(s) which because of dependencies, causes a coincidence of failure states of components in two or more separate channels of a redundancy system, leading to the defined system failing to perform its intended function".*

Causes of common mode failures

Causes of common mode failures can be depicted as in Figure 2-9 [Edwards and Watson, 1979]. To study common mode failure the boundary of the system has to be explicitly defined, i.e. what is included and excluded in the system. Hence what is included in the system has to be dealt with by safety analysis techniques presented in the previous section and what is excluded by the boundary of the system is the domain of common-mode failure analysis.

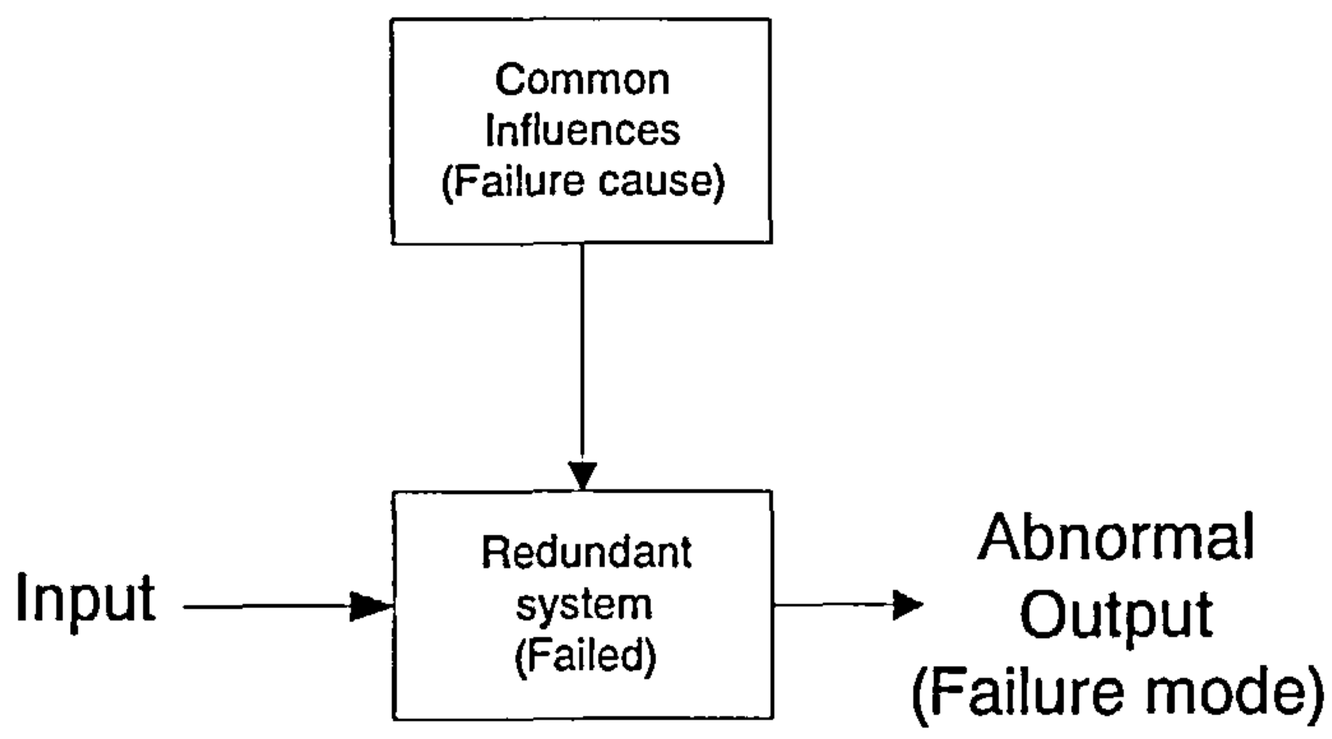


Figure 2-9: Causes of common-mode failure

The picture indicates the causes of common mode failure as *Common Influences* (i.e. Root Causes + Couplings). In a fault tolerant system they may occur either in the period prior to operation or during its operating life. In the first case the influences take place in activities such as specification, design, manufacture, installation and commissioning, in the second case they happen in maintenance or operation. These include deficiencies of the system that are due to common influences that happen in the period prior to operation and become apparent while the system is running. For example, the system might not be able to perform completely its task or in particular circumstances; or it may be vulnerable to common influences during operation due to inadequate design, quality control or commissioning.

However, to define which causes of failure are common influences we have to start defining what is a system and what is its boundary. Edwards and Watson say that a system is an *"interconnection of components that combine to form a specified functional relationship between inputs and outputs"*. Hence, we can understand that everything that is not needed by the system to provide the input-output relationship when it is functioning normally, is not part of the system and therefore it is a possible cause of common influences. These influences can be a failure cause like fire, explosion, missile impact, contamination interference etc. Remote sources that can have a significant common influence on the system are also the weather, earthquakes, floods etc.

Difficulties arise when there is the direct involvement of humans in the system as for operation, maintenance and test. If the human influence is required for the system to fulfil the functional relationship between input and output then the human influence has to be considered inside the boundary otherwise it has to be considered as a common influence. Thus, test and maintenance are to be considered as common influences, while operation may or may not according to the system or the application. If operator action is required for the system to perform its functionality then the operator action has to be considered part of the system. For instance, in a manually controlled system like an aircraft, pilots are an essential part of the functionality of the system since they control the aircraft from the information presented to them. If pilots were only interacting now and then with the control system of the aircraft like making initial or occasional adjustments and then the aircraft was completely operated by the auto-pilot then pilots would not be part of the system. Actually pilots contribute to most of the aircraft accidents especially with regard to navigation, therefore they have to be considered part of the system. While in automatic protective systems like in nuclear reactors the operator

is only responsible for certain adjustment and supervision. Hence operators have to be considered as common influence and possible cause of common-mode failure.

We said already that common cause failures supersede common mode failures. That is because common cause failures cause all the events in a minimal cut set to occur at the same time[7]. Whilst common mode failures are a specific type of common cause failure in which events in the minimal cut set are failure modes of the same type.

2.3.4 Defending against Root Cause

Defending against root causes seems to be quite straightforward, but it is not always possible to do, and sometimes is not economically viable. There are two main steps to provide defences against root causes:

- The identification of all possible root causes;
- The definition of affordable improvements for reaching the required system robustness.

Whereas the second point is purely a technological and economic matter, the first is quite a difficult issue, as the identification of all the possible root causes (that must be outside system boundaries to be considered by common cause failure analysis) may require expert judgement, and so depend on the expertise of the analyst.

A number of different schemes for classifying root causes of dependent events have been proposed both in the Nuclear and Aerospace domain. They have been developed to help analysts in identifying root causes. Each classification scheme is expected to be, ideally, exhaustive and its categories to be mutually exclusive.

2.3.5 Defending against couplings

Defending against couplings is subtler than identifying root causes. It implies the assessment of a number of types of couplings deriving not only from the positioning of each item inside the system, but also from the item design and construction phases. Therefore all the development and maintenance life cycle of the components must be analysed. Looking at different types of couplings, the nuclear sector has defined three main categories of dependencies: functional, physical and human [NUREG 2815, 1985].

---

[7] Or in a short time interval.

They use the term *functional dependencies* when an item depends on shared functions that can be achieved either by *shared hardware*, or on a *process coupling*. In the first case, multiple devices depend on the same equipment (e.g. a support system); in the second one, the function of one device depends on the function of another device (e.g. temporal dependence). They use the term *physical dependencies* when two or more devices are coupled through the same environment, so that an event affecting the environment affects also all the components inside that particular area[8]. They use the term *human-interaction dependencies* to address all those couplings caused by human actions. They analyse both the *cognitive behaviour* (e.g. failure of diagnosis) and the *procedural behaviour* (e.g. multiple maintenance errors).

A checklist helping in the identification of couplings is reported in the NUREG 5801 [Mosleh et al., 1993]. According to this publication the analyst should focus mainly on the identification of those components of the system which share one or more of the followings:

- Same design
- Same hardware
- Same function
- Same installation, maintenance, or operation procedures staff
- Same system/component interface
- Same location
- Same environment

Therefore it could be useful to develop checklists of key attributes such as design, location, operation etc., where the analyst can find most or all of the possible couplings. An example of such a checklist that helps in the identification of redundant components in a system and in the identification of the most commonly observed couplings for a Motor Operated Valve[9] is reported in Table 2-12.

---

[8] With the word area we do not mean just the same zone (e.g. a room), but also multiple volumes linked by a common ventilation duct or inside the same electromagnetic field are considered as a single area.

[9] A checklist to address software components, that we have developed during this work, is reported in Table: 5.1.

Component Type	• Component size • Material • Special features
Component Use	• System isolation • Flow modulation • Parameter sensing • Motive force
Component Manufacturer	• Brand
Component internal conditions	• Absolute or differential pressure range • Temperature range • Normal flow rate • Chemistry parameter range • Power requirements
Component boundaries and system interfaces	• Common discharge header • Interlocks
Component location name and code	• Room • Area
Component external environment conditions	• Temperature range • Humidity range • Barometric pressure range • Atmospheric particulate content and concentration
Component initial conditions and characteristics	• Normally closed, open • Energised • Normally running, standby
Component testing procedure and characteristics	• Test intervals • Test configuration • Effect of maintenance on system operation
Component maintenance procedures and characteristics	• Planned • Preventive maintenance frequency • Maintenance configuration • Effect of maintenance on system operation

Table 2-12: Checklist for a Motor Operated Valve

2.3.6 The aerospace industry

Aerospace industries approach dependency analysis in a slightly different way. They do not talk explicitly about root causes and couplings, and they use the term common cause analysis to address what the nuclear industries call dependent failure analysis. In common *cause* analysis they identify three different issues which they address with *zonal safety analysis*, *particular risks analysis* and *common mode analysis* [SAE-ARP 4754 and SAE-ARP 4761, 1996].

Zonal Safety Analysis addresses all those concerns regarding equipment installations, interference between systems, the robustness of the system against possible maintenance errors and the claimed independence of events in a fault tree. They look for all the installation aspects of each system and the mutual influence between systems

installed in close proximity on the aircraft. The whole aircraft is divided into several zones and for each of these zones a zonal safety analysis is performed. The objective of the zonal safety analysis is to ensure that the system design meets the safety objective with respect to:

- Basic installation;
- Effect of failures on aircraft;
- Implication of maintenance errors;
- Verification that the design meets the FTA independence claims.

Particular Risk Analysis addresses specific events listed by airworthiness regulations that potentially may cause a failure inside the system itself. For each risk the possible consequences for the whole aircraft should be evaluated; if one of the risks may affect safety, proper measures should be taken. Table 2-13 lists particular risks set out in [SAE-ARP 4761, 1996].

- Fire
- High energy devices (non-containment):
  - Engine
  - Auxiliary Power Unit
  - Fans
- High pressure bottles
- High pressure Air Duct Rupture
- High temperature Air Duct Leakage
- Leaking fluids:
  - Fuel
  - Hydraulic
  - Battery acid
  - Water
- Hail, Ice, Snow
- Birds strike
- Tyre burst, flailing tread
- Wheel rim release
- Lighting strike
- High Intensity Radiation Fields
- Flailing Shafts
- Bulkhead rupture

Table 2-13: Subjects of Particular Risks Analysis

Common Mode Analysis addresses redundancies. According to ARP 4761, common mode analysis should be performed in the lifecycle after Functional Hazard Analysis and Preliminary System Safety Analysis. Its aim is to verify that all the inputs to all AND gates (both explicit and implicit) in the failure logic analysis (Fault Tree Analysis, Dependence Diagram, Markov Analysis etc.) are independent. Basically, components with the same hardware and software could be susceptible to common mode failures due to couplings arising from particular risks, or other causes. Therefore the principal task of the analysis is to look for couplings and to evaluate to what extent 'root causes' could affect coupled components. Identifying coupling is the major task and is very much dependent on the expertise of the analyst; several check lists have been tailored to help in discovering couplings. Table 2-14 reports different common mode categories and Table 2-15 reports a checklist useful for the qualitative assessment (so far no quantitative assessment of common mode failure has been done). Both tables are taken from [SAE-ARP 4761, 1996].

- Software design errors
- Hardware design errors
- Hardware failures
- Production repair/flaw
- Stress related events
- Installation errors
- Requirements errors
- Environmental factors
- Cascading faults
- Common external source faults

Table 2-14: Common Mode Fault categories to be analysed

It is important to point out that whereas common cause failure analysis in the nuclear industry is *both* a qualitative and quantitative procedures, common cause failure analysis in the aerospace industry is purely a qualitative analysis.

COMMON MODE TYPE	COMMON MODE SUB-TYPE	EXAMPLES OF COMMON MODE SOURCES	EXAMPLES OF COMMON MODE FAILURES/ERRORS
Concept and Design	DESIGN ARCHITECTURE	Common discharge Header	Common discharge failure
		Common external sources (ventilation, electrical, power,..)	Failure of common sources
		Equipment Protections	Designer failure to predict an event, ...
		Operating characteristics (normally running, standby,..)	
		Others	General design error, ...
	TECHNOLOGICAL MATERIALS EQUIPMENT TYPE	New/Sensible technology	Hardware error, ...
		Component type (size, material,..)	
		Common Software	Software error...
		Component Use	...
		Internal Conditions (Temperature, ranges,..)	usage out of operating ranges (T, P)
		Initial conditions	...
		Others	...
	SPECIFICATIONS	Specification Origin	Origin error (human), lack of specific protection in equipment design, ...
		Same Specification	Defective specification, ...
		Others	...
Manufacturing	MANUFACTURER	Common Manufacturer	Common error due to manufacturer, error due to inadequately trained personnel, ...
		Others	...
	PROCEDURES	Same procedure	Incorrectness procedure, ...
		Others	
	PROCESS	Same process	Incorrect process, Inadequate manufacturing control, inadequate inspection, inadequate testing, ...
		Others	...
Installation/ Integration and Test	FITTER	Common fitter	Installation or error due to fitter, ...
		Others	...
	PROCEDURES	Installation phase	Common error due to phase, ...
		Others	
	LOCATION	Same zone	Local failure or event, ...
		Others	
	ROUTING	Same routing	Local event, ...
		Others	
Operation	STAFF	Common Staff	Error due to inadequately trained personnel, overstressed or disabled operator, ...
		Others	...
	PROCEDURES	Same procedure	Faulty operation procedures, misdiagnosis (following wrong procedure), Omission of action, incorrect or inadequate commission of action, ...
		Others	...
Maintenance	STAFF	Common Staff	Error due to inadequately trained personnel, Incorrect action, ...
		Others	...
	PROCEDURES	Same procedure	Failure to follow repair procedures defective repair procedure. lack of repair procedure, ...
		Others	...
Test	STAFF	Common Staff	Error due to inadequately trained personnel, Incorrect human action, ...
		Others	...
	PROCEDURES	Same procedure	Faulty test procedure, ...
		Others	...

The table continues on the next page.

COMMON MODE TYPE	COMMON MODE SUB-TYPE	EXAMPLES OF COMMON MODE SOURCES	EXAMPLES OF COMMON MODE FAILURES/ERRORS
Calibration	STAFF	Common Staff	Error due to inadequately trained personnel, ...
		Calibration Tools	...
		Others	...
	PROCEDURES	Same procedure	Inadequate tools adjustment, ...
		Others	...
Environmental	MECHANICAL AND THERMAL	Temperature	Fire, lightning, welding etc., cooling system faults, electrical short circuits, ...
		Grit	Airborne dust, metal fragments generated by moving parts with inadequate tolerances, ...
		Impact	Pipe whip, water hammer, missiles, structural failure, ...
		Vibration	Machinery in motion, earthquake, ...
		Pressure	Explosion, out of tolerance system changes (pump overspeed, flow, blockage), ...
		Humidity	Steam pipe breaks, ...
		Moisture	Compensation, pipe rupture, rainwater, ...
		Stress	Thermal stress at welds of dissimilar metals, thermal stresses, ...
		Others	...
	ELECTRICAL AND CORROSION	Electromagnetic	Welding equipment, rotating electrical machinery, lightning, interfaces power supplies, ...
		Radiation	Gamma radiation, charged particle radiation, ...
		Conducting Medium	Moisture, conductive gases, ...
		Out-of-tolerance	Power surge voltage, short circuit, power surge, current, ...
		Others	...
	CHEMICAL AND MISCELLANEOUS	Corrosion (acid)	Leak of acid used in maintenance for removing rust and cleaning, ...
		Corrosion (oxidation)	Failure leading to a water medium or around high temperature metals (ex filaments), ...
		Other chemical reactions	Galvanic corrosion, complex interactions of fuel cladding, water, oxide fuel, ...
		Biological	Poisonous gases, animate causes (mussels in heat exchanger), ...
		Others	...

Table 2-15: Checklist with Common Mode Types, Sources, and Failures/Errors

2.3.7 Software domain

So far we have surveyed common cause failure analysis in the nuclear and aerospace fields. Now we move to consider common cause failures in computer based systems. To our knowledge, no formalised methods exist to study dependencies amongst software components, even if there are efforts to build software "common mode failure free". The Airbus company built the first passenger aircraft with a computer-based flying control

system (A320) using several precautions to avoid any kind of coupling [Dorsett & Mellor, 1993]. They used:

- Computer systems developed by separate companies using 80186 & M68000 processors;
- Separate teams (only the requirements specifications were available for communication);
- For each computer: the control channel was written in Pascal, and the monitor in C; or the control channel in assembler and the monitor in Pascal;
- Particular care has been taken to ensure independence in command and monitoring development teams;
- Each team used different compilers;
- The voting logic was different in each computer.

Even if this is indeed a starting point, we cannot say to what extent such efforts are appropriate for the task they are asked to deal with, whether designers have been "more" or "less" effective than might reasonably be expected in avoiding couplings. Additionally, this comment does not reflect on the A320 *per se*, it simply indicates a lack of understanding of root causes and couplings affecting software.

2.3.8 Defences against common cause failures

The policy to prevent common cause failures starts early in the lifecycle. It involves engineers being aware of sources of common cause failures and possible defences against them. It is by eliminating sources of common cause failures from early in the design phase that saves expensive remedies later. However when it is not feasible to reduce causes of common cause failures, *ad hoc* defences against them based on specific features of each plant can usually be set up. Defences against common cause failures are careful project administration, planning, functional diversity, equipment diversity, protection and segregation of equipment, barriers, equipment derating and simplicity, quality control, preventive maintenance, monitoring etc. To check whether plant defences have been considered for each potential cause of common cause failure, some techniques have been conceived. The development of one such technique that lists plant defences against potential causes of common cause failures has been sponsored by the US Nuclear Regulatory Commission and presented under the name of "*Cause Defense Matrix*" in [Paula and Parry, 1990; Mosleh et al., 1993].

Table 2-16 displays an example of a cause defence matrix focused on environmental factors. A cause-defence matrix is a formal way to make sure that in a plant some defences have been considered for each potential cause of common cause failure. The first column from the left of the table lists causes of common cause failures, i.e. groups of failure causes. The remaining columns list the measures that are taken in the plant against each failure cause mechanism, that is the root cause (i.e. trigger event + conditioning event) and the coupling factor.

Failure Cause Group	Defence Against		
	Root Cause		Coupling Factor
	Conditioning Event	Trigger Event	
Internal environmental effect (corrosion, biofouling, etc.)	Ensure internal environment is "pure" Preventative maintenance	Surveillance testing/condition monitoring (slowly developing only)	Functional diversity Equipment diversity Barrier between inputs to redundant trains Staggered maintenance
External environmental effects Shock (fast acting)	Barriers at the component (equipment hardening) Equipment qualification	Barrier between source of shock and component Inspection of potential sources of shocks	External barriers between redundant trains
Slow acting	Barriers at the component (equipment hardening) Equipment qualification	Barrier between source of shock and component Surveillance testing/ condition monitoring for cumulative effects of environments	Functional diversity Equipment diversity External barriers between redundant trains

Table 2-16: Cause-Defence matrix for environmental-related causes

2.3.9 Common cause failures quantitative assessment

The contribution of common cause failures to the likelihood of critical events is estimated by using parametric models. These models were introduced in the late 1960's [Marshall, 1967] when the need to evaluate common cause failures arose. The most widely used parametric models are named from the parameters they use. The *Beta factor* model [Marshall and Olkin, 1967; Fleming, 1975] names the parameter it uses with the second letter of the Greek alphabet. While the *Multiple Greek Letter* model [Fleming and Kalinowski, 1983] uses many parameters (the order of redundancy minus one) called β, γ, δ, etc. In the case of two redundant components the multiple Greek letter model reduces to the Beta Factor model. Parameters can be thought of as representing the strength of the coupling among redundant components, but also as conditional probabilities as will be shown later. They range between 0 and 1. The lower bound represents complete lack of coupling whereas the upper bound represents complete coupling. In many cases it is difficult to estimate common cause failure parameters due

to lack of statistical data, therefore analysts use conservative values. Experience has shown that a conservative value for beta is 0.1[10] [Mosleh, et al., 1988]. Additionally, if some care is taken to ward off common cause failure, the beta for a redundant system can easily be reduced by one or two orders.

Parametric models take into account the contribution of common cause failures by modifying the value for the likelihood of events. They split this up into two or more contributions of which one is the likelihood of the independent occurrence of the event, the other(s) are probabilities of the common cause failures. Hence if we wish to represent common cause failures in a fault tree we have to add some events. To see how a fault tree is modified to consider common cause failures we produced a simple example based on a system the function of which is to arise oil from one tank to another. The system architecture consists of three redundant pumps. However only two of them are required to run at any one time to assure the system functionality. Figure 2-10 displays the architecture of the system. The system fails when any two pumps fail. The fault tree for the system is in Figure 2-11. If failures of the three pumps were completely independent, the fault tree would consist of only one level that is represented by the darker part. Since failures of pumps are not considered independent, the failure probability of each pump is divided into four contributions representing the random occurrence (i.e. A_I, B_I and C_I), the occurrence because of a shared cause with one other pump only (i.e. C_{AB}, C_{AC} and C_{BC}) or because of a shared cause with both other pumps (C_{ABC}). Parametric models assign probabilities to all the contributors to the pump failure probability both independent events (i.e. A_I, B_I, C_I) and dependent events (i.e. C_{AB}, C_{AC}, C_{BC}, C_{ABC}).

---

[10] Up to 0.18 for diesel generator sets.

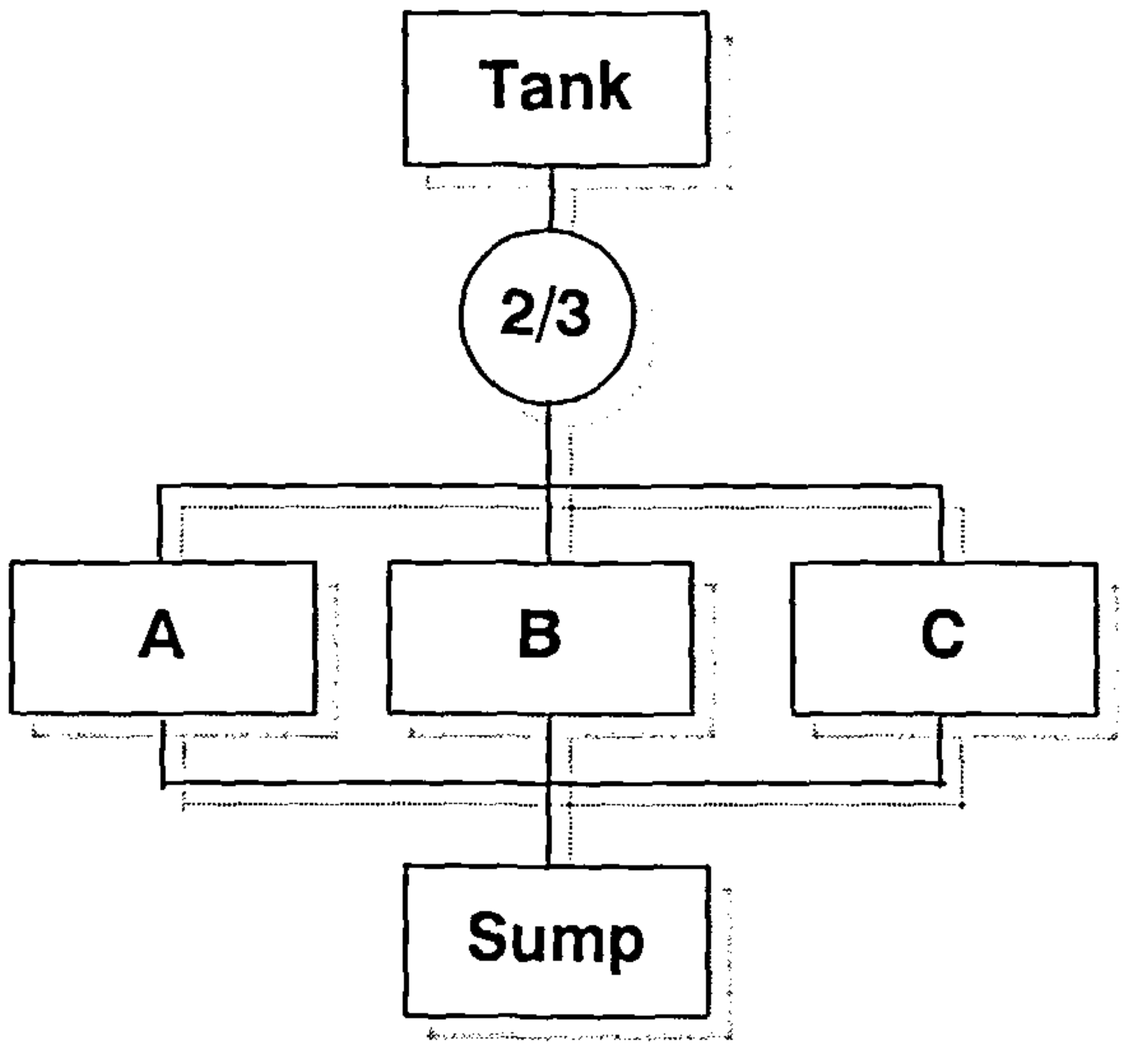


Figure 2-10: Triple redundant system raising oil from the sump to the tank

All the existing parametric models are based on the *Symmetry Hypothesis* [Mosleh, et al., 1988]. This hypothesis relies on the common practice in safety and reli ability analysis to assume that the probabilities of similar events involving similar types of components are the same. Hence if there are three events, i.e. A, B, and C, the symmetry hypothesis assumes that the probability of any one of them occurring independently is the same and is equal to a value called *"Q_1"*. Further it assumes that the probability of any two events occurring simultaneously is identical and equal to *"Q_2"*. Additionally it assumes that the probability that all the three events occurring simultaneously is equal to *"Q_3"*. This is represented by the following equation 2-2.

$$\begin{cases} P(A_1) = P(B_1) = P(C_1) = Q_1 \\ P(C_{AB}) = P(C_{AC}) = P(C_{BC}) = Q_2 \\ P(C_{ABC}) = Q_3 \end{cases} \qquad (2\text{-}2)$$

The symmetry hypothesis certainly holds in the many cases in which identical components are used in redundancies, but that cannot be taken for granted when fault tolerance is achieved by any mixture of software, hardware, or information and timing redundancy, e.g. for computer based fault tolerant systems. However before discussing this issue, we continue presenting the different features of the Beta factor and the Multiple Greek letter parametric models.

The Beta-factor model is the simplest of the parametric models. It considers the independent likelihood of each event in the MCS and the likelihood of all the events happening simultaneously because of a common cause failure. This is achieved by assuming the likelihood of common cause failures not affecting all the components to be zero, see equation 2-3 and 2-4.

$$P(C_{ab}) = P(C_{ac}) = P(C_{bc}) = Q_2 = 0 \qquad (2\text{-}3)$$

$$P(C_{abc}) = \beta * P(A_I) = Q_3 \qquad (2\text{-}4)$$

The Beta factor model is normally used with low orders of redundancy (maximum three channels) since it becomes conservative as the order of the redundancy increases. In these situations the Multiple Greek Letter method comes into place.

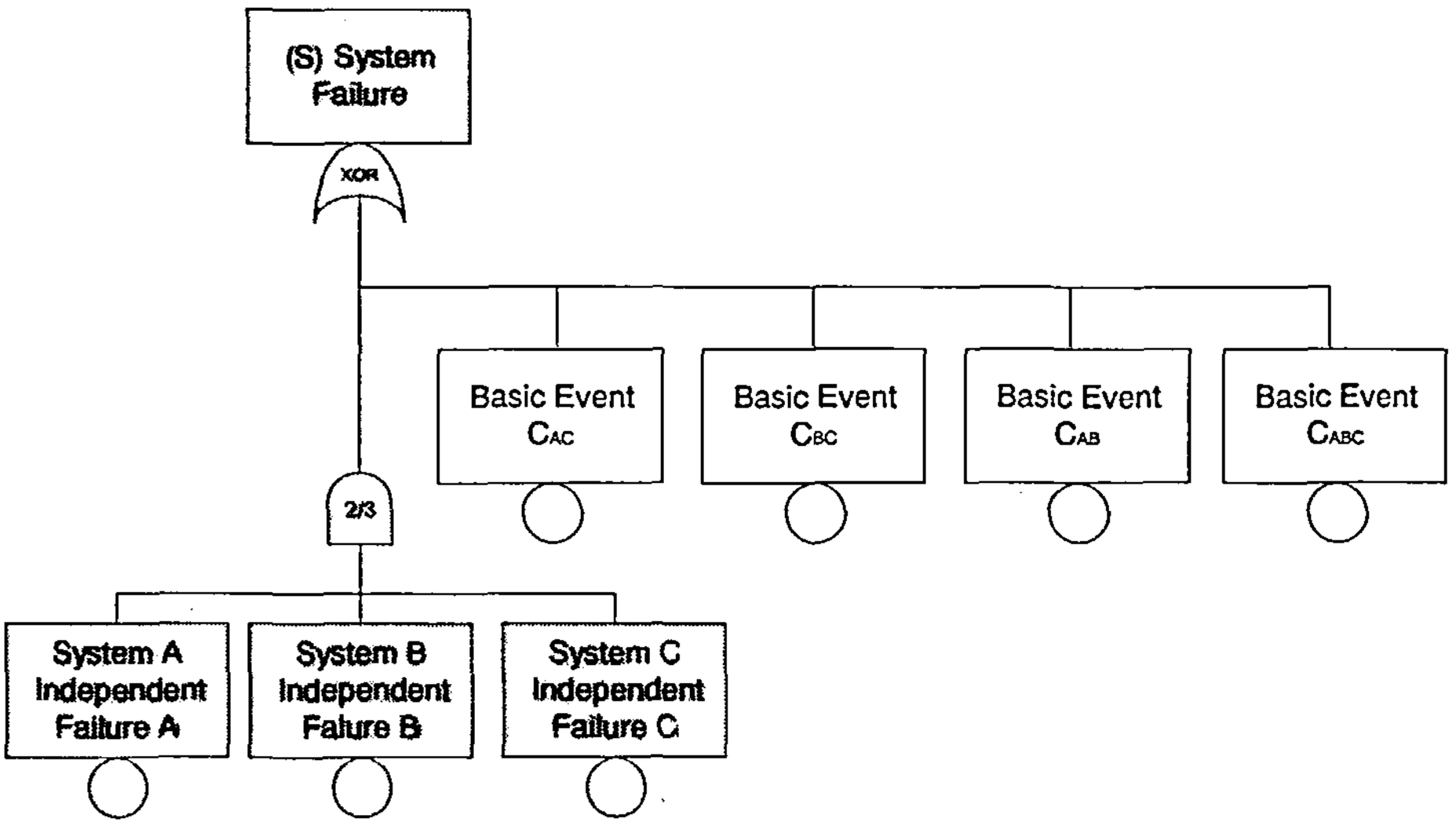


Figure 2-11: Tree for the system in Figure 2-10

The Multiple Greek Letter model [Fleming and Kalinowski, 1983] is an outgrowth of the Beta factor model that can consider systems with any degree of redundancy. Values for probabilities are assigned according to equations in 2-5. It is out of the scope of this thesis to explain how that equation is obtained, the explanation can be found in [Mosleh, et al., 1988]. We only say that *m* represents the number of redundant components in the system, *k* represents the order of the generic subset of components that can be created inside the common mode failure component group. It ranges between 1 and *m*. Q_k

represents the failure probability of a generic subset of events of the minimal cut set. Finally, ρ_i represents these generic parameters (i.e. $\rho_1=\beta$; $\rho_2=\gamma$; $\rho_3=\delta$; $\rho_4=\phi$; etc.).

$$Q_k = \frac{1}{\binom{m-1}{k-1}} \left(\prod_{i=1}^{k} \rho_i \right) (1-\rho_{k+1}) Q_t \qquad (2\text{-}5)$$

If we consider the system in Figure 2-10, i ranges between 1 and 3. Thus:

$$m = 3$$

$$\rho_1 = 1$$

$$\rho_2 = \beta$$

$$\rho_3 = \gamma$$

$$\rho_4 = \delta$$

Where:

β = Conditional probability that the cause of a component failure will be shared by one or more additional components, given that a specific component has failed.

γ = Conditional probability that the cause of a compo nent failure that is shared by one or more components will be shared by two or more additional components, given that two specific components have failed.

δ = Conditional probability that the cause of a component failure that is shared by two or more components will be shared by three or more additional components, given that three specific components have failed.

For the system in Figure 2-10, equations (2-5) becomes:

$$Q_1^3 = \frac{1}{\binom{2}{0}} \rho_1 (1-\rho_2) Q_t = \quad (1-\beta) Q_t$$

$$Q_2^3 = \frac{1}{\binom{2}{1}} \rho_1 \rho_2 (1-\rho_3) Q_t = \quad \frac{1}{2} \beta (1-\gamma) Q_t \qquad (2\text{-}6)$$

$$Q_3^3 = \frac{1}{\binom{2}{2}} \rho_1 \rho_2 \rho_3 (1-\rho_4) Q_t = \quad \beta\gamma \; Q_t$$

After a Boolean simplification of the tree in Figure 2-11, MCS are obtained, and the system failure probability Q_s is evaluated by using equation 2-7.

$$Q_S \equiv Q_1^3 + 2Q_2^3 + Q_3^3 \qquad (2\text{-}7)$$

Giving values to parameters

Methods have been developed to estimate values for parameters used by parametric models. These methods are partly based on statistics on common cause failure events recorded in databases, and partly on empirical considerations [Mosleh, et al., 1988]. Most of the time they estimate boundaries, i.e. max. and min. for each parameter. For instance, if a set of parameters for a plant is known, and a similar plant is reckoned more robust to common cause failures (but for which no statistics are available as it is a new plant), it will be assigned a set of parameters of slightly smaller values. An example of such an empirical method is given in [Humphreys, 1987]. This method is very field specific and concerns programmable electronic systems. It basically allows the estimation of the parameter β used in the Beta factor model by giving a weight to eight *sub-factors* as Table 2-17 shows.

FACTOR	*SUB-FACTOR*	*WEIGHT*
DESIGN	Separation	8
	Similarity	6
	Complexity	6
	Analysis	6
OPERATION	Procedures	10
	Training	5
ENVIRONMENT	Controls	6
	Tests	4

Table 2-17: Factor, sub-factor and sub-factor weight

Different sets of these sub-factor weights can also be assigned to account for different degrees of couplings in different plants. In Table 2-18, column 'a' and 'e' represent respectively the highest and the lowest possible sub-factor weight, thus the highest and the lowest possible coupling. One column need not be a multiple of another since the

sub-factor might not grow linearly with the strength of the coupling. Since the beta is a probability and it ranges between 0 and 1 it is obtained by a proper normalisation.

SUB-FACTOR	a	b	c	d	e
Separation	2400	580	140	35	8
Similarity	1750	425	100	25	6
Complexity	1750	425	100	25	6
Analysis	1750	425	100	25	6
Procedures	3000	720	175	40	10
Training	1500	360	90	20	5
Controls	1750	425	100	25	6
Tests	1200	290	70	15	4

Table 2-18: Possible sub-factor weights

2.4 Discussion

In this chapter we saw that a number of different techniques are used for safety analysis as the design evolves in the course of the lifecycle. Furthermore we saw that those techniques are not formally linked to each other and as a consequence the consistency of the analysis cannot be assured throughout the design development process. In a complex design it is, therefore, often difficult to trace (using the results of the safety assessment) the causes of critical malfunctions of the system in the hierarchy of subsystems and components that compose the design. We have also noted the trade-off between techniques providing a graphical and tabular representation of results.

The second part of the chapter focused on common cause failures. We discussed the mechanisms leading to common cause failures, and based on this discussion we showed that there are two possible ways to avoid common cause failures, either by eliminating root causes or removing coupling factors. We saw that there are methods that help to consider defences for each potential cause of common cause failure in a plant. However there are no methods that measure (or at least map) couplings among redundant components in a system. We also noticed problems related to the estimation of parameters for the quantitative evaluation of common cause failures. To address the limitations and shortcomings of classical techniques that we have highlighted here, this thesis will attempt to answer the following questions:

a) Is it possible to develop a technique that encompasses the different safety analyses typically performed across the lifecycle?
b) Can the application of this technique result in a meaningful and easy way to perform a collection of safety analyses which can assist the design of the system?
c) Can we ensure the consistency of the results within the assessment?
d) Can those results be represented both graphically and in tables, so that we can combine the benefits of both representations?
e) Finally, is it possible to use this technique to systematise the identification of common cause failures?

Chapter three

Preliminary work

In Chapter 2 we surveyed techniques for safety analysis. We saw that there are many techniques for tackling specific needs, however little has been done to integrate those techniques that are typically used in cascade across the lifecycle. That causes several problems that were highlighted. In addition, we found a lack of formalised methods to consider common cause failures in computer based safety critical systems. Causes of common failures have to be sought across the lifecycle so if a method has to be built to relate techniques typically used across the lifecycle the issue of common cause failures must be considered.

In this chapter we present the work that was done at the beginning of our research and that brought about (through many refinements) the formulation of the technique, known as Failure Logic Analysis for System Hierarchies (FLASH), that is presented in chapters 4 and 5. We think this preliminary work is important because it explores some original approaches and shows the reasons for developing FLASH.

3.1 Template based approach

The research started looking for a notation capable of showing *how hardware and software elements are dependent and support each other* in safety critical computer based systems. It was thought that this notation was needed to support top-down study of a system: the functional level first, then the architectural level and finally the component level. Therefore the functional representation of the system was addressed first. Functional failures were studied independently from the implementation of the function (i.e. hardware, software components or both). Malfunctions were represented as top events in fault trees whose basic events were either software or hardware failures. We perceived the importance of having formalised trees so we tried to systematise their construction by proposing *mini-trees* to represent failures of sub-systems and sub-functions. Figure 3-1 displays an example of a fault tree built using mini-trees for the system in Figure 3-2. These, which are very similar to the ones in [Leveson, 1983], represent the most common causes of failure. Additionally, they have undeveloped

events for considering faulty inputs and failures from other functions, sub-systems or components. The idea was that, once fault trees were built for system malfunctions, they could be assessed for repeated branches, which clearly are sources of dependent failures. These repeated branches were shared by both software and hardware components, hence they were identifying software-hardware dependencies.

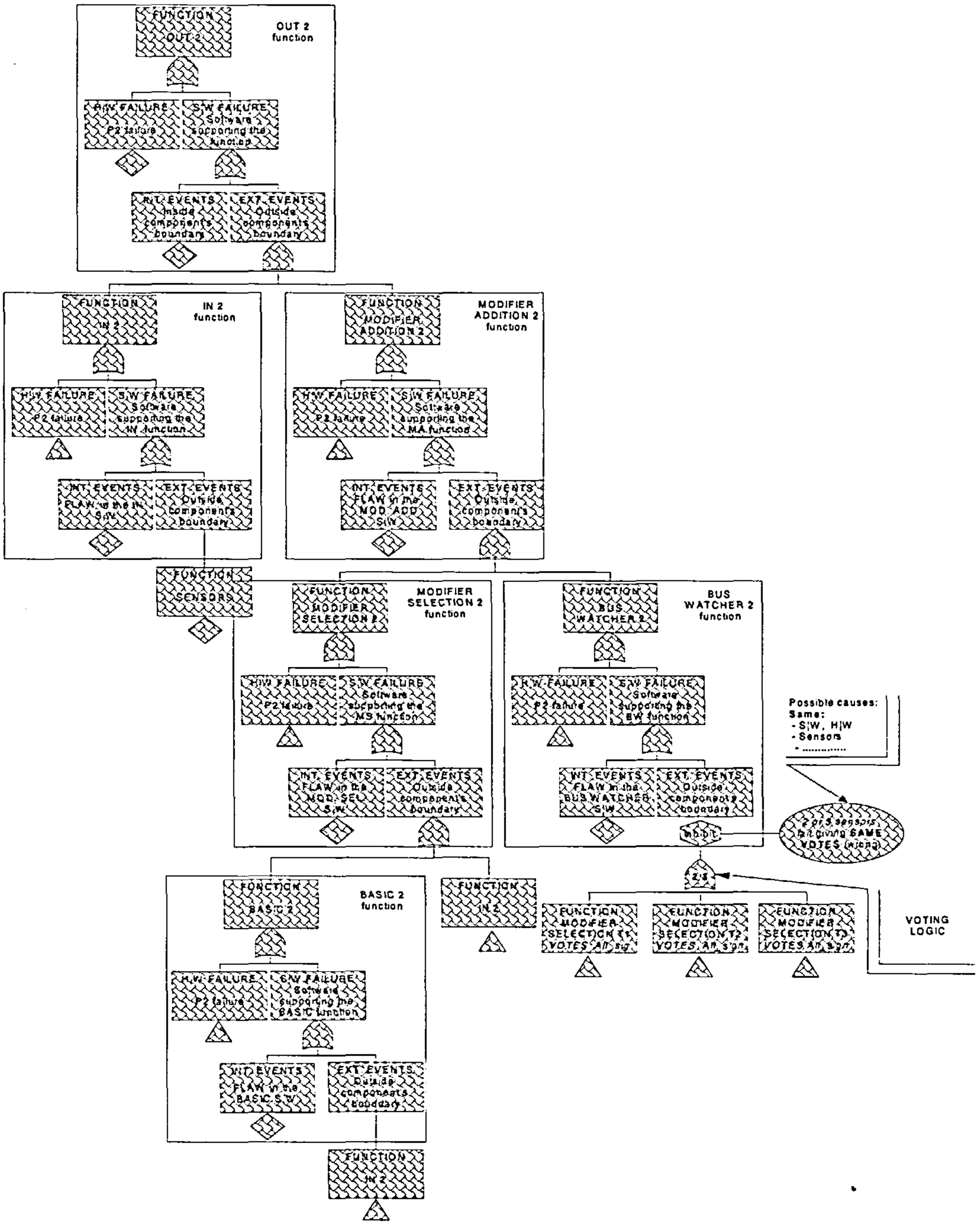


Figure 3-1: Fault tree built using mini-trees

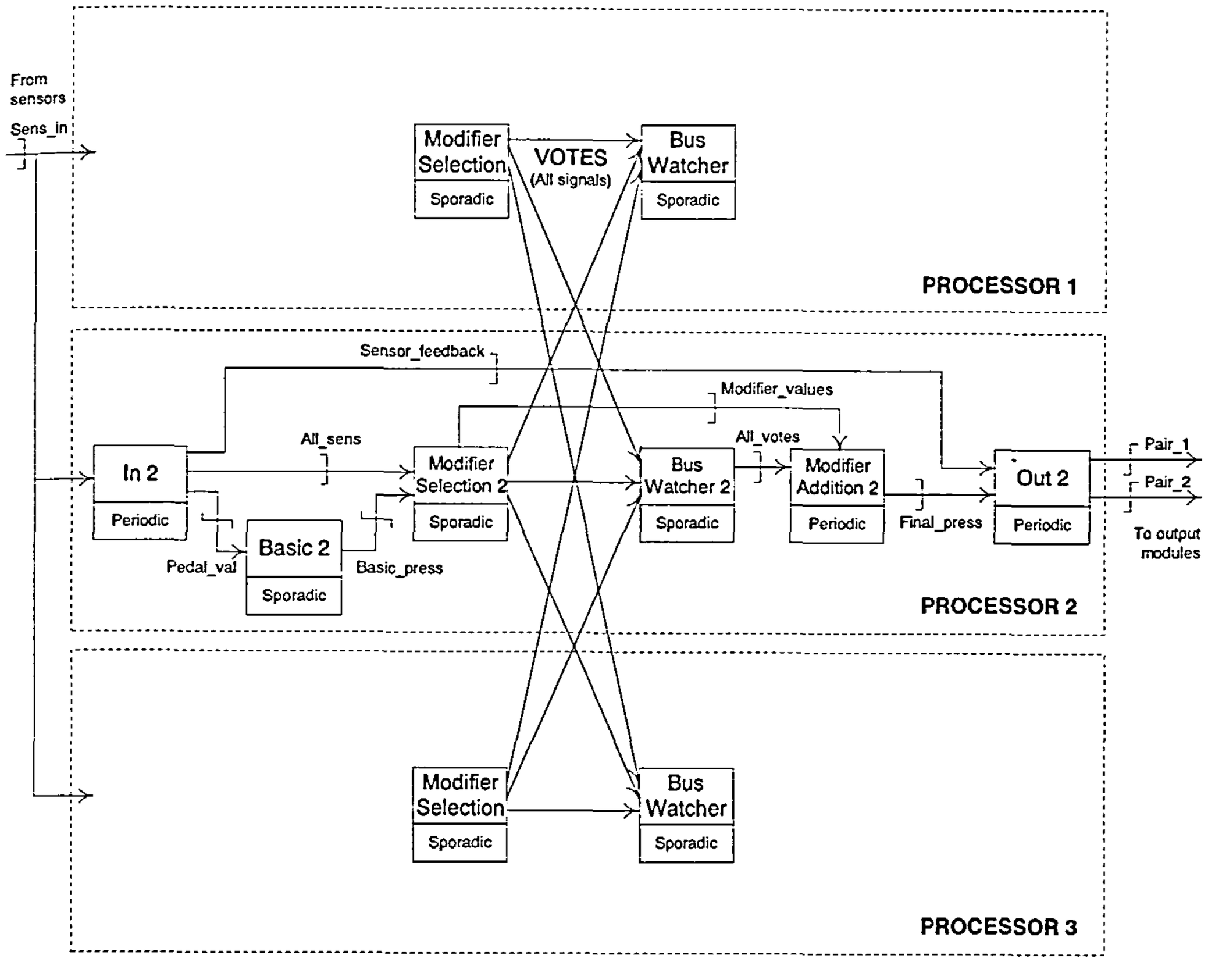


Figure 3-2: Fragment of a functional block diagram of a Computer Braking System

However, this notation was only deductive and it was not suitable for representing recovery actions taking place in fault tolerant systems. So it was thought desirable to represent recovery by using some of the gates used in cause consequence analysis. An example of such a representation is shown in Figure 3-3. In this notation a function, a component or a task (e.g. the sub-function OUT in the graph) is represented as a box with inputs, entering from the bottom, and outputs departing from the top. The ones leaving from the top left corner of the box are *intended outcomes*, whilst outputs leaving from the top right half are *fault outcomes*. Whilst a component can have only one correct functional mode[11], it can have many failure modes. Hence, it was thought useful to represent failure modes by using an *event tree style graph* placed near the top right

[11] It is recognised that, in principle, there can be many functional modes. However, for safety analysis, we can group them together.

corner of the gate and connected to its failure outcome. Then, each path through the "event tree" would represent a failure outcome of the function/component/task in the box. However, this notation posed additional problems: for instance it was not clear where to put the many fault trees representing failure modes of the component represented by the box. We decided to put them underneath the event tree so we ended up with the *Event Tree Output* notation that is presented next.

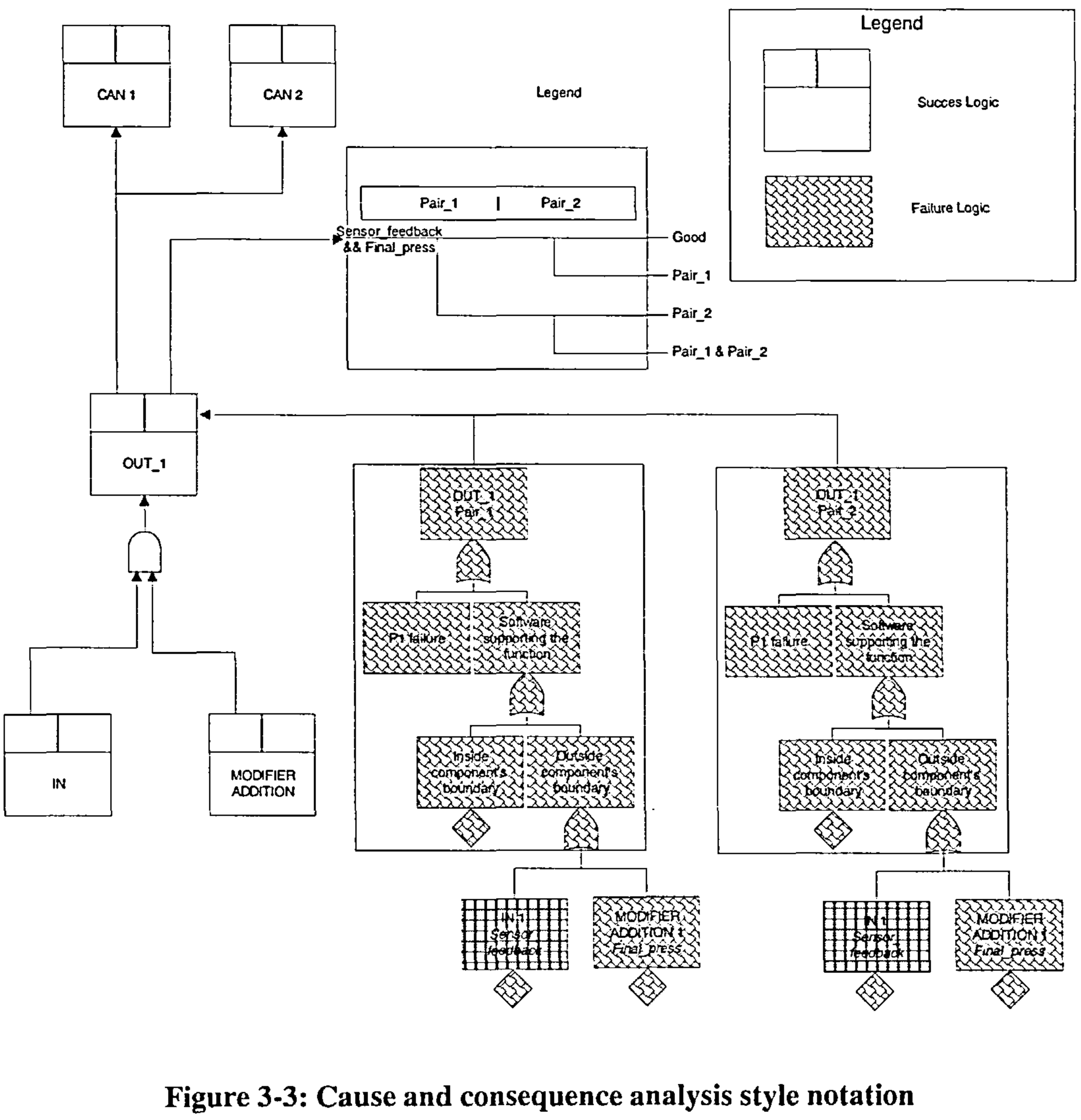


Figure 3-3: Cause and consequence analysis style notation

3.2 Event Tree Output Notation

Fault trees representing causes of system malfunctions were gathered below a sort of event tree providing a different path for each failure mode or combination of failures. Figure 3-4 shows an example of such a representation. The upper part of the graph

shows the outputs provided by the box, either good (i.e. YES) or faulty (i.e. NO). However, whilst at any time there can be only one good output (i.e. path on the top of the event tree), there can be many non-straight paths representing the presence of faults (i.e. when mitigation or recovery took place) or failures (i.e. where mitigation or recovery failed). Out of these outputs, some may compromise the safety of the system, hence be *critical*, whereas others may produce less serious consequences. Fault trees showing causes of each functional failure are represented below the event tree. Repeated branches in fault-trees identified couplings among functions.

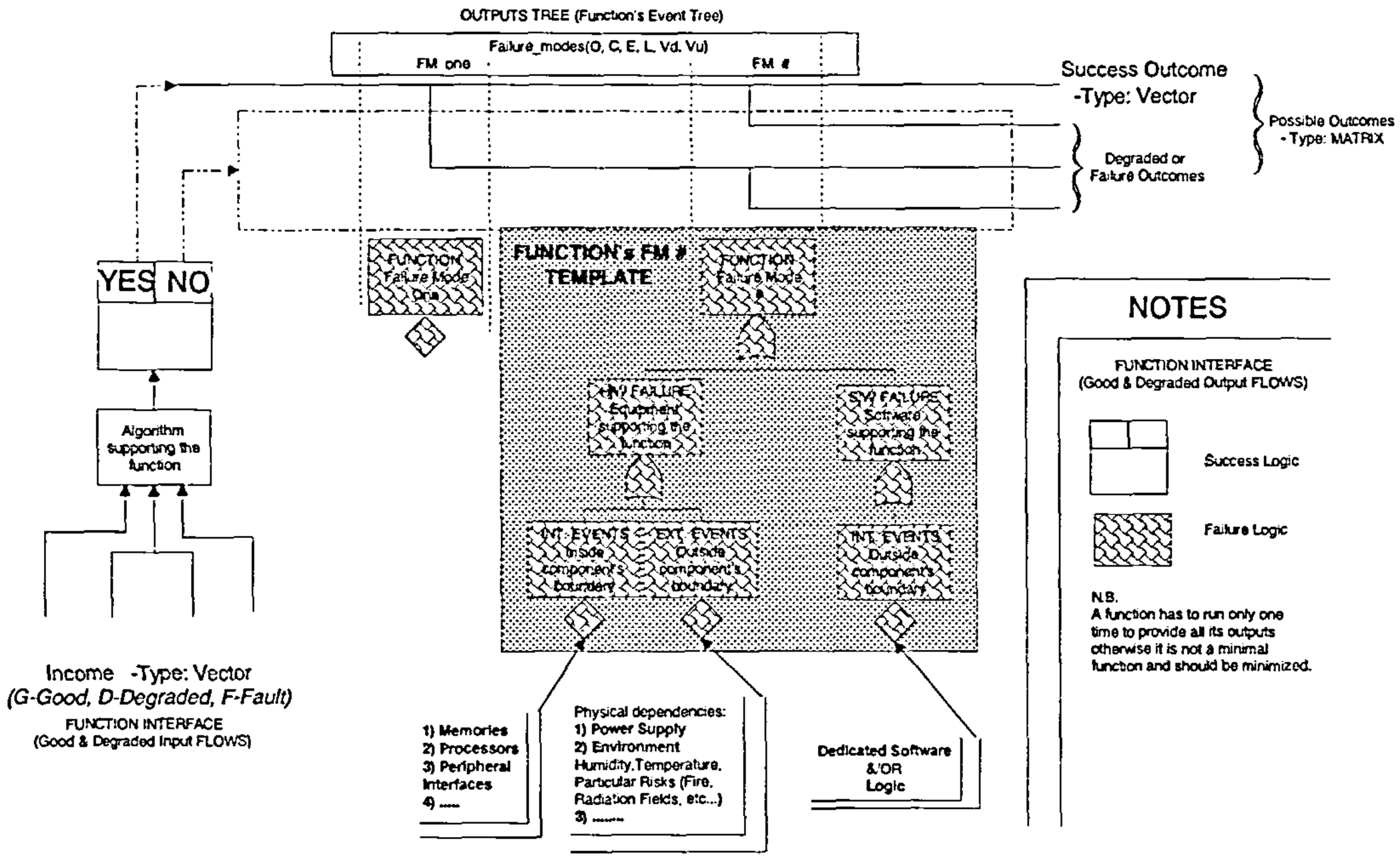


Figure 3-4: Fault trees are shown below an event tree

This notation provides a graphical representation of dependencies between software and hardware components, however it has some limitations. First of all it provides a huge number of outputs for each component. Hence a sort of filtering mechanism on the output to avoid propagating non-critical outputs should have been developed, but that would have complicated further the method. Additionally shared fault tree branches and shared components are represented in different places in fault trees and, in some cases, it is not easy to identify repeated branches. At this point it was thought practical to try to improve the representation by exploiting a variant of another notation: the Master Plant Logic Diagram.

3.3 Master Plant Logic Diagram approach

The next attempt to provide a notation able to merge the analysis of software and hardware components, considering common cause failures in complex computer based safety critical systems, was an extension of the Master Plant Logic Diagram notation in [Modarres, 1992]. As we said in the second chapter, the MPLD notation effectively represents the interrelationships amongst various components, and can model relationships between functions and systems, so it already has some of the characteristics that we were aiming for in our method. But MPLD as it is defined in [Modarres, 1992] does not allow the mapping of couplings which originate common cause failures. Consequently we extended this notation to include this additional category of couplings. The extended notation was called MPLD*.

MPLD* is a logic diagram that shows how functional, equipment and component failures combine to cause a system malfunction. An MPLD* diagram is constructed for each failure mode of the system that is represented as top event in the MPLD* graph. Combinations of function, sub-system and component failures, which cause the top event, are represented in a *fault-tree-like* structure. However, an MPLD* graph differs from a fault tree since basic events are not represented as *leaf events* in the tree, but they are listed in the lower left part of the graph and connected to gates through a sort of matrix. Lines that originate at basic events and those that end at each gate make this matrix. Small blobs mark active intersections of those lines. Therefore AND or OR gates can be connected to a number of primary events through a vertical line that intersects horizontal lines originating from events. Figure 3-5 shows the MPLD* for a functional failure (i.e. complete lack of braking) in the computerised braking system described in 6.2 and represented in Figures 6-9 and 6-10. It is possible to see that this failure (top event in the MPLD*) is caused by failures of both actuators (i.e. output modules 1 and 2). The failure of any of these components can be caused either by an internal failure (hardware failure) that is represented in the list at the bottom left, or by failures in both the redundant busses i.e. Bus 1 and Bus 2 (see the *AND* gate on the left immediately below the *OR* gate). Similarly, the failure of anyone of the busses is caused either by internal failures (i.e. hardware wear out or software implementation) or by the simultaneous failure of all of the three output tasks (i.e. output 1, output 2 and output 3). Failure of these tasks can be caused by other functional or hardware failures (e.g. Modifier addition, Bus Watcher, Modifier Selection, Basic, In, etc.) and ultimately, by wrong inputs from sensors (i.e. fault data from sensors) or supports (e.g. processors).

Critical Failure Modes A: **Complete lack of braking**

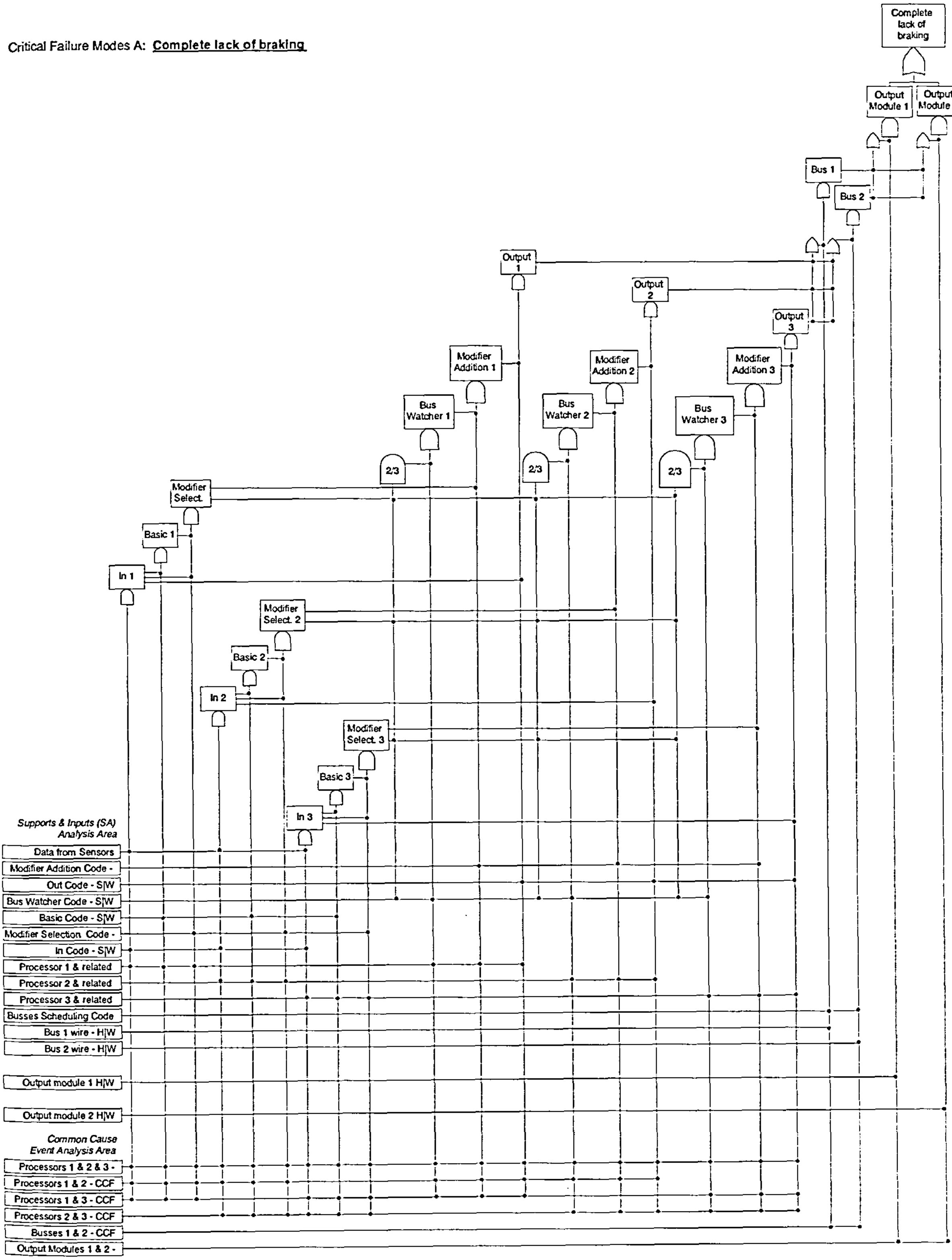
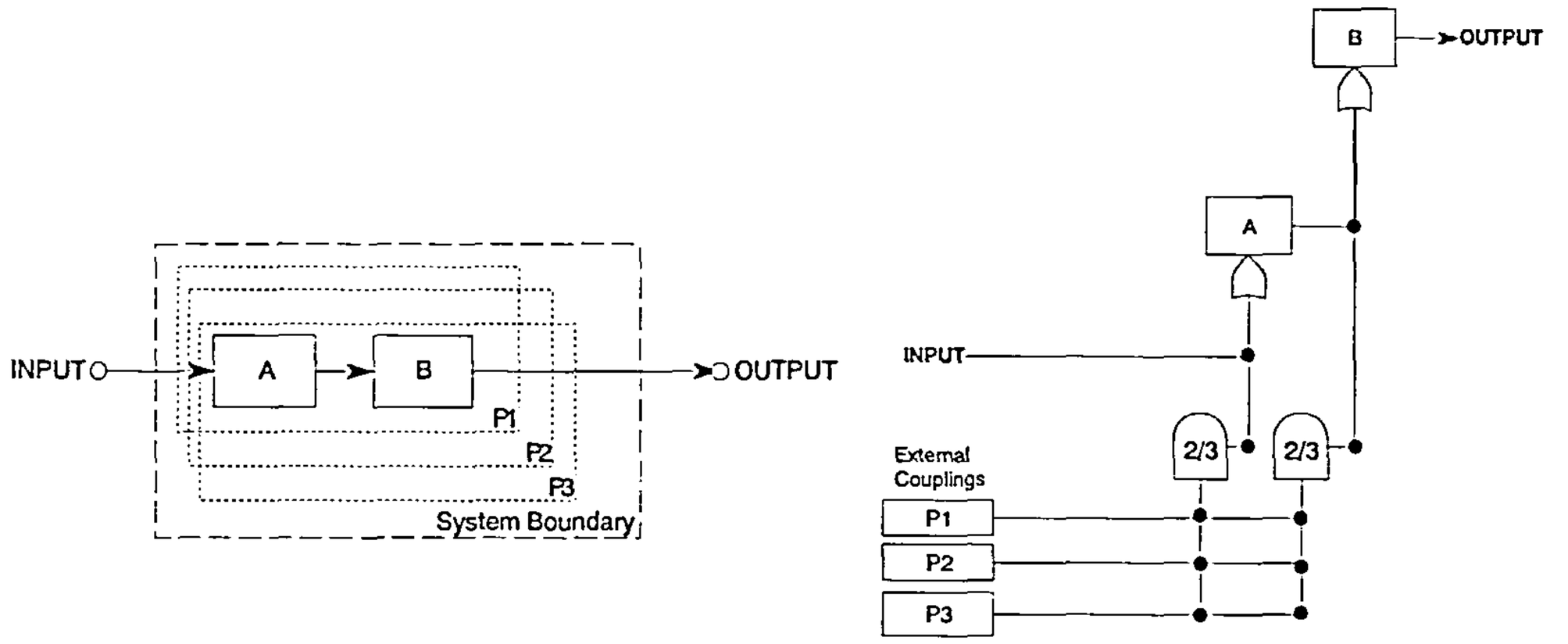


Figure 3-5: MPLD* for complete lack of braking in a braking system

However, what we have described so far is, actually, nothing other then the failure logic underneath a functional failure in a different format than a fault tree or an MPLD. We now show how, additionally, the MPLD* notation represents couplings among different sub-functions, sub-systems and components that may give rise to common cause failures. For this purpose the MPLD* notation reserves the *common cause events area* immediately below the input and support area. This area lists common cause events affecting two or more sub-functions, sub-systems, tasks or components. As for basic events, common cause events are graphical linked to intermediate events that are arisen by them and represented into the upper part of the graph. The dependency matrix at the right side of the list of common cause events identifies couplings.

It has to be noticed that in the MPLD* graph there is not a clear distinction between the functional, architectural, and component level, and there are sub-function failures as well as subsystem and component failures. This is because the analysis is not hierarchically represented, but flat. It is performed deductively moving backward from unwanted effects to causes. In addition, the MPLD* has two other limitations. It does not clearly represent the mapping of software to hardware (or vice versa) and does not allow recording of detailed information for basic components, for example component failure rates, mean time to failure, mission time, etc. Hence we had to define two other notations to apply at higher and lower levels of detail.

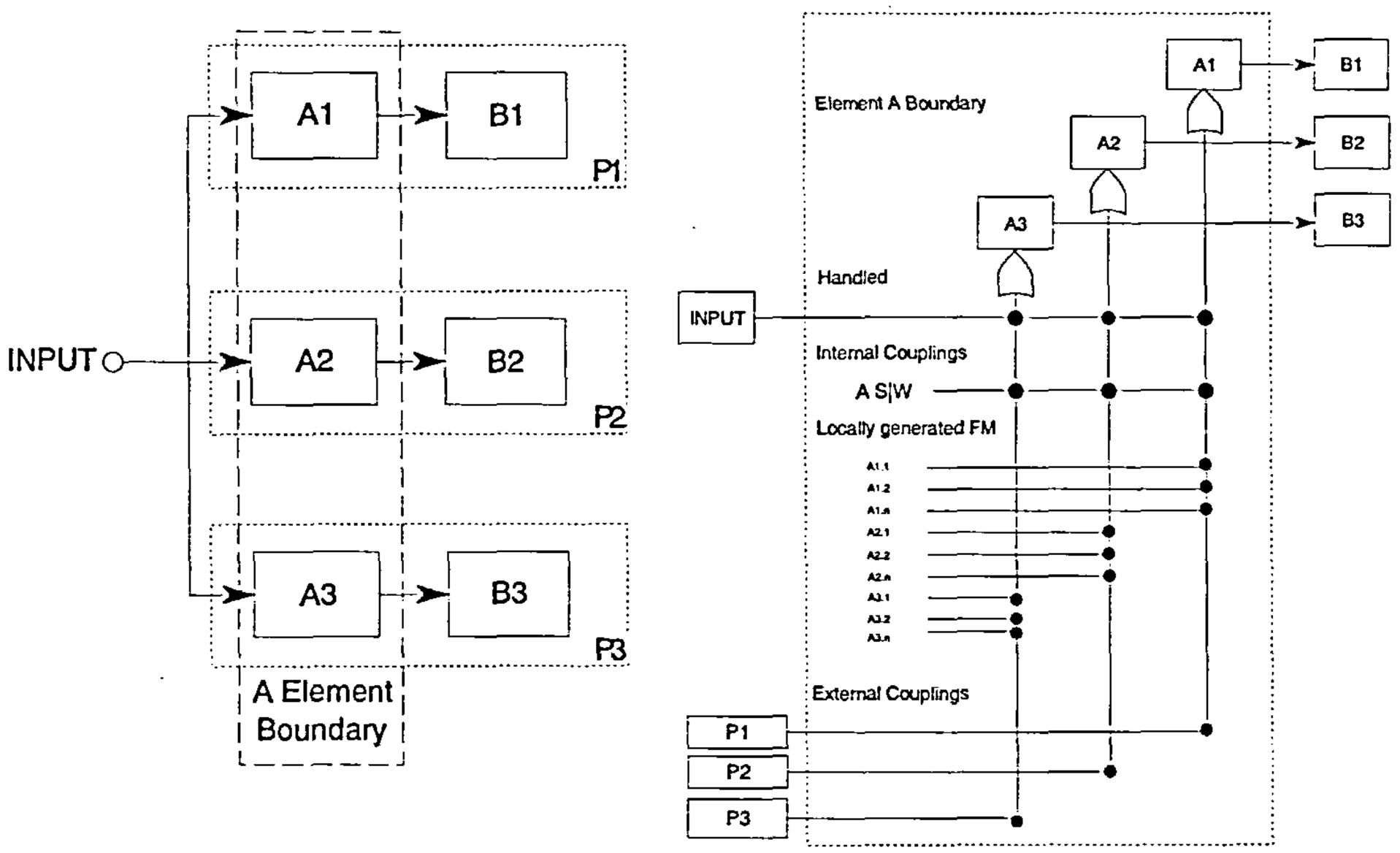
At the higher level of detail, we proposed a sort of *block diagram* notation displaying the mapping of functions to hardware and software components. An example is in Figure 3-6a that shows a cascade of two functions (i.e. boxes A and B) that are mapped onto three processors (i.e. boxes P1, P2 and P3). Figure 3-7a displays the breakdown of function A and its redundant architecture. Function A is actually achieved by three sub-functions (A1, A2 and A3) each mapped to a different processor. A breakdown of sub-function A1 is displayed in Figure 3-8a. It shows that this sub-function is achieved by three tasks called A1.1, A1.2 and A1.3 all running on processor P1. Master plant logic diagrams associated with these architectures are represented in Figure 3-6b, Figure 3-7b and Figure 3-8b.



a) Mapping of functions upon hardware b) MPLD* representation

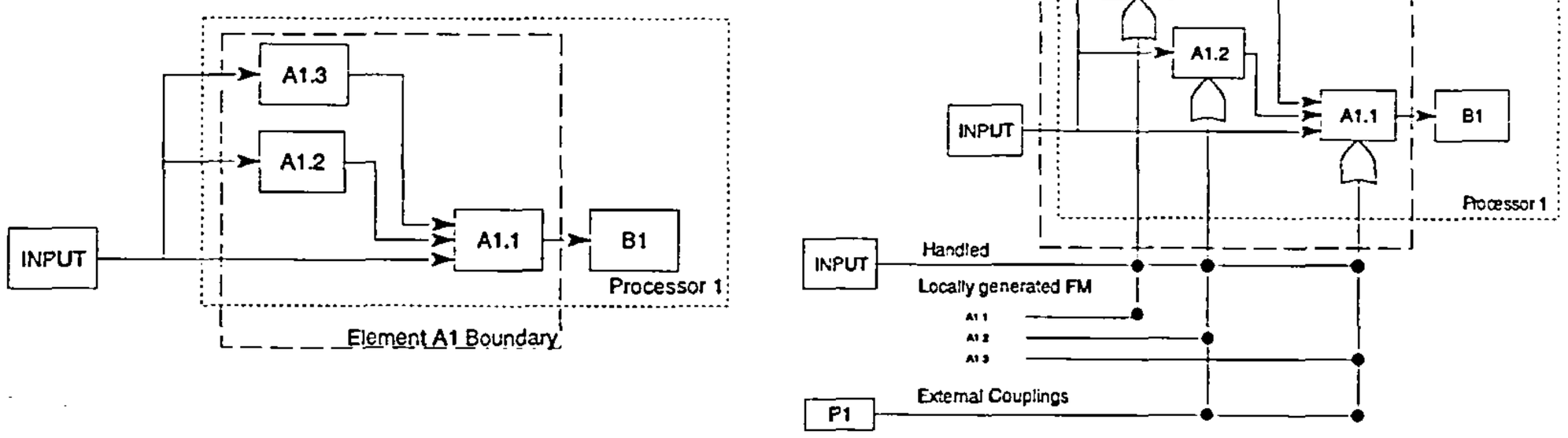
Figure 3-6: High functional level

Figure 3-6b shows how functions A and B are supported by processors P1, P2 and P3 and that function B has one only input that is from function A. At this representation level nothing is shown about their failure modes. Figure 3-7b shows the relationship among the three tasks, A1, A2 and A3 constituting function A and similarly for function B. The two functions are apparently independent, since they run on different processors and have different failure modes (i.e. A1.x, A2.x and A3.x) but they all share the same software. The coupling matrix shows that the three tasks are not coupled except for the software that is common to all of them.



a)Mapping of functions upon hardware b) MPLD* representation

Figure 3-7: Medium functional level



a) Mapping of functions upon hardware b) MPLD* representation

Figure 3-8: Detailed functional level

At lower levels of detail we defined a *table-based* notation placed along side the graphical ones. The tables record the information needed for a probabilistic analysis. This notation requires association of a table with each component in the design, contour information describing internal and external failures influencing the component's output, whether correct or faulty. In these tables there are various areas describing, for example, locally generated failure modes and couplings, externally generated failure modes and couplings, and the mechanism underneath the transformation and propagation of failures. Figure 3-9 shows the table that is associated with component A1 represented in Figure 3-7. The table is divided into five areas (i.e. Laws, Handled Couplings, Internal Couplings, Locally Generated Failure Modes and External Couplings); the Laws area is further divided into sub-areas. Failures that arise inside the component may appear either in the *internal coupling* or in the *locally generated failure mode* area. That depends on whether they are shared by one or more elements at the component level architecture. In a similar way, failures that happen outside the component boundary may appear either in the *external coupling area* or in the *input area*. This depends on whether they come from support systems shared by one or more other components or they are actually the input of the component i.e. data (or analog variables) that have to be processed by the function. The *Laws area* shows the failure logic describing how failure modes and couplings, listed in the previously mentioned areas, combine to provide either faulty or good outputs. This area is further divided into three headings that are *Failure Conditions* describing faulty outputs; *Normal Output* describing the (results of the) process that recovers recoverable failures and the *Conservative/Default outputs* describing when the

system handles failures and goes into a safe state. These are the cases in which full recovery is not possible, but the system can still deliver a safe output. A similar table can be written for the MPLD* in Figure 3-8b. It can be seen that the laws area in a table contains the information needed to draw part of the MPLD* diagram for the system.

The information in the *laws area* of the table is also the *causes-effect relationships* explaining the deviation of output flows (propagated by the component) from their expected values. This information is also the same that is needed to conduct a fault tree analysis. Hence, it can be used to draw the upper part of the MPLD* that is actually nothing else than a fault tree, as we have already said. Further, the information about couplings and support systems in the table is used to visually connect intermediate events representing component failures in MPLD* graph. Hence it can be used to draw the lower part of the MPLD*. Joining together the information in tables for all the component of the system it is possible to construct MPLD*s for each system failure modes. Ultimately, block diagrams mapping the software upon the hardware can be drawn with the information about software-hardware dependencies that can also be stored in the tables.

Drawing some conclusions, the MPLD* notation with the associated tables and *Block Diagram* notations are intended to fill the gap existing in the analysis of safety critical computer based system to account for the interactions of software and hardware components. They can be used in place of fault tree analysis but they do not substitute for Preliminary Hazard Analysis, HAZOP and FMEA. Table 3-1 compares the analysis of a safety critical computer based system from three different viewpoints: the hardware, the software and the integrated *software-hardware* viewpoints. The table shows that the MPLD* notation can be used at functional as well as at architectural and component level during the decomposition and design of safety critical computer based systems.

LAWS (TO THE OUTPUT POOL)

FAILURE CONDITION (END EFFECTS)
Value (2/3 Channels) :== A1.1 (V) AND A2.1 (V) OR A1.1 (V) AND A3.1 (V) OR A2.1 (V) AND A3.1 (V) OR O_Input OR V_Input
Omission (2/3 Channels) :== A1.2 (Late) AND A2.2 (Late) OR A1.2 (Late) AND A3.2 (Late) OR A2.2 (Late) AND A3.2 (Late)

NORMAL OUTPUT (GOOD INPUT TO NEXT DOWNSTREAM COMPONENT(S))
A1 & A2 AND INPUT OR A1 & A3 AND INPUT OR A2 & A3 AND INPUT

CONSERVATIVE DEFAULT OUTPUT (NEXT HIGHER LEVEL EFFECT)
(Not in this example)

INPUTS

Input **Description**
O_Input:== Input (Pool) gives Omission failure then A1 & A2 & A3 read the previous value
V_Input:== Input (Pool) gives Value failure then A1 & A2 & A3 read a wrong value

INTERNAL COUPLINGS

Internal S|W A is shared by A1 & A2 & A3 => Need Expert Justification
Note: Expert may say that in this circumstances S|W does not constitute a coupling to be further analysed

LOCALLY GENERATED FAILURE MODES

A1.1 = Value;
A1.2 = Late (Scheduler failure in A1)
A2.1 = Value;
A2.2 = Late (Scheduler failure in A2)
A3.1 = Value;
A3.2 = Late (Scheduler failure in A3)

EXTERNAL COUPLINGS

Processor P1 support A1
Processor P2 support A2
Processor P3 support A3

Note: Although A1, A2, A3 are not coupled by the same hardware, they might share the same life cycle (i.e. same type, same manufacturer, etc.) => Need further analysis

Figure 3-9: Table associated with component A1 represented in Figure 3-7

		Viewpoint		
		Hardware	***Software***	***Hardware-Software***
Levels	***Functional***	PHA or FFA + FTA	PHA or FFA + FTA	PHA or FFA + MPLD*
	Architectural	*HAZOP* + FTA	SHARD + software FTA	SHARD + MPLD*
	Component	*FMEA* + FTA	FMEA + software FTA	FMEA + MPLD*

⇐ Increasing details

Table 3-1: Overview of the Safety analysis used to assess critical systems

3.4 Discussion

Our aim was to address some shortcomings and limitations of classical safety analysis techniques for the study of complex computer based safety critical systems. After endeavouring to use some existing template based notations and what we called "Event tree output notation" we came out proposing a variant of another technique, the Master Plant Logic Diagram. This notation was able to solve some inefficiencies of the classical techniques like the visual representation of dependencies between hardware and software, but it also left us with some unresolved problems. For instance, it was not possible to represent clearly the mapping of software to hardware and it was not feasible to store in an MPLD* graph all the information that is needed to calculate the likelihood of its top event. Hence we proposed a graphical notation to represent the mapping of software to hardware and a table based notation to store the detailed information that could not be stored into an MPLD*. Further the table notation was enriched to contain information to encompass both MPLD* and the representation of the mapping of hardware upon software (i.e. the table and block diagram notations).

However these techniques alone were not enough to achieve the targets we were aiming for in the thesis. In fact, though they were applicable at any stage of the development phase, they were not integrated with other analyses (i.e. FHA, HAZOP and FMEA) that are performed in parallel. Additionally, they did not solve the problem of ensuring the consistency of results of analyses performed during the lifecycle by common safety analysis techniques, hence we decided to take a new approach in the rest of our research.

Out of these techniques, the table-based notation was the one that solved most of the problems we wanted to address. Such an approach allows information to be stored in a format which is easy to access. Hence we tried to extend further this notation to encompass other classical safety analysis techniques such as HAZOP, Functional Hazard Analysis and FMEA. We found that that was feasible. The starting point was the fact that flows and flow deviations propagated by a component to another contain information about the hypothetical deviations of flows from their expected value that are used to drive HAZOP. Hence we thought about a possible way to modify the table used to record specific information in the MPLD* notation to encompass also HAZOP analysis. This was the starting point that brought us additional further enhancements and finally to formalise the Failure Logic Analysis for System Hierarchies that is presented in the next chapter.

Chapter four

Failure Logic Analysis for System Hierarchies

This chapter presents the FLASH method, which enables the integrated assessment of complex hierarchical designs by helping analysts to identify potential functional failures of the system at the application level and then to systematically determine the causes of those failures in progressively lower levels of the design. The result of the assessment is a consistent collection of safety analyses which provides a meaningful picture of how low-level failures are stopped at intermediate levels of the design, or propagate and give rise to hazardous malfunctions.

FLASH is applied at two different stages of the lifecycle: a) system decomposition & design and b) integration & verification. In the first stage it checks the evolving design against higher-level safety requirements and supports the establishment of derived safety requirements for each sub-system and component. In the second stage it verifies whether the product as implemented and integrated meets its concept level and derived safety requirements.

The chapter begins giving a broad overview of the FLASH method, continues describing details of FLASH tables that are the core of the method, the process of conducting a FLASH analysis by compiling FLASH tables, presents the software tool that was developed to aid the analysis and finishes discussing some limitations.

4.1 FLASH Overview

FLASH enables the assessment of a hierarchically described system from the functional level down to the low levels of its hardware and software implementation. To ensure consistency of results, in FLASH, all safety analysis are performed on the same consistent hierarchical model of the system. The method places constraints on the notations used, and introduces some additional notation for describing levels of design. The notation allows complex systems to be modelled as system hierarchies (see Figure 4-1, left side). At each level of the hierarchy, flow diagrams are used to describe the

architecture of subsystems or components. At plant level these flow diagrams can be, for example, piping and instrumentation diagrams. At lower levels they can be derived from any form of structured design notation used for the architectural design of software or hardware components, for example Data-flow diagrams [Yourdon and Constantine, 1985] or MASCOT diagrams [Budgen, 1985].

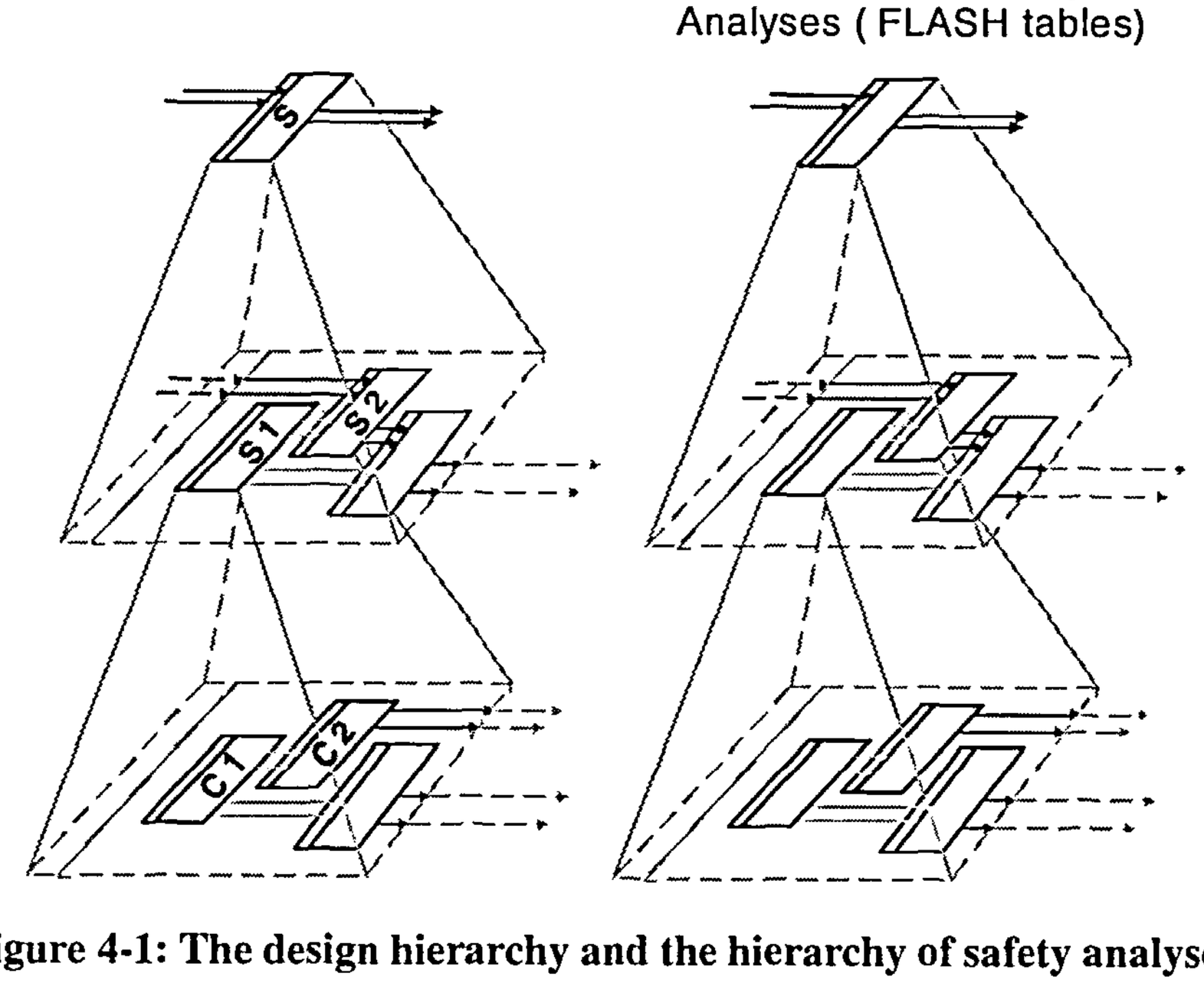


Figure 4-1: The design hierarchy and the hierarchy of safety analyses

The system hierarchy is created during the decomposition and design phase. The process involves the decomposition of the system into modules, and then further decomposition of each module into several more basic modules.

In the course of safety analysis, each module of the architecture (i.e. sub-system or basic component) is systematically examined for potential failure modes. One of the aims, here, is to identify the failure modes that the module propagates to other modules and the causes of those failures in lower levels of the design. The specific failure modes of each module are identified as the outputs of the module (functions, material flows, energy flows, data) are systematically examined for potential deviations from the expected normal behaviour. At the highest level of the design, failure modes represent functional failures. At lower levels they represent failures of subsystems as these can be

observed at the outputs of those subsystems. Finally at the lowest level, they represent root failures of the basic components of the architecture.

The results from the analysis of each module are recorded in a separate table, and the analysis is completed when we have created a table for each module in the design hierarchy. At the end of the assessment process, the results from the analysis of the system and its constituent parts form a hierarchy of FLASH tables (see Figure 4-1, right side). Table 4-1 illustrates a fragment from an example FLASH table for sub-system *S1* in the architectural decomposition depicted in Figure 4-1. The table records the analysis for one of the output failure modes of sub-system *S1* (*Output_Deviation_of_S1*). It can be seen that the analysis of a module is presented in six columns.

Sub-system S1					
Failure events	***Description***	***Causes***	**Contributing factors**	**Criticality, Handling Recommendations**	**Summary of FMEA Results**
Output_ Deviation _of_S1	The output of sub-system S1 deviates from the design intention. No effect on the system S	Failure_Mode_of_C1 AND Failure_Mode_of_C2	Excessively high temperature (T>max)	Criticality: Critical Handling: The failure is handled. The system detects the failure event and replaces the malfunctioning S1 with S2 Recommendations: a) Ensure that the failure detection mechanism is reliable b) The acceptable failure rate for this effect should be λ<1e-4 (f/h) Action required: Analyse the error detection mechanism for potential *failure modes*	The failure detection mechanism is reliable (pointer to the relevant analysis) The failure rate for this effect was calculated to be λ=1e-5 (i.e. within the acceptable limit) **Thus, the architecture of sub-system S1 is accepted**

Table 4-1: A fragment of an example FLASH table for sub-system S1

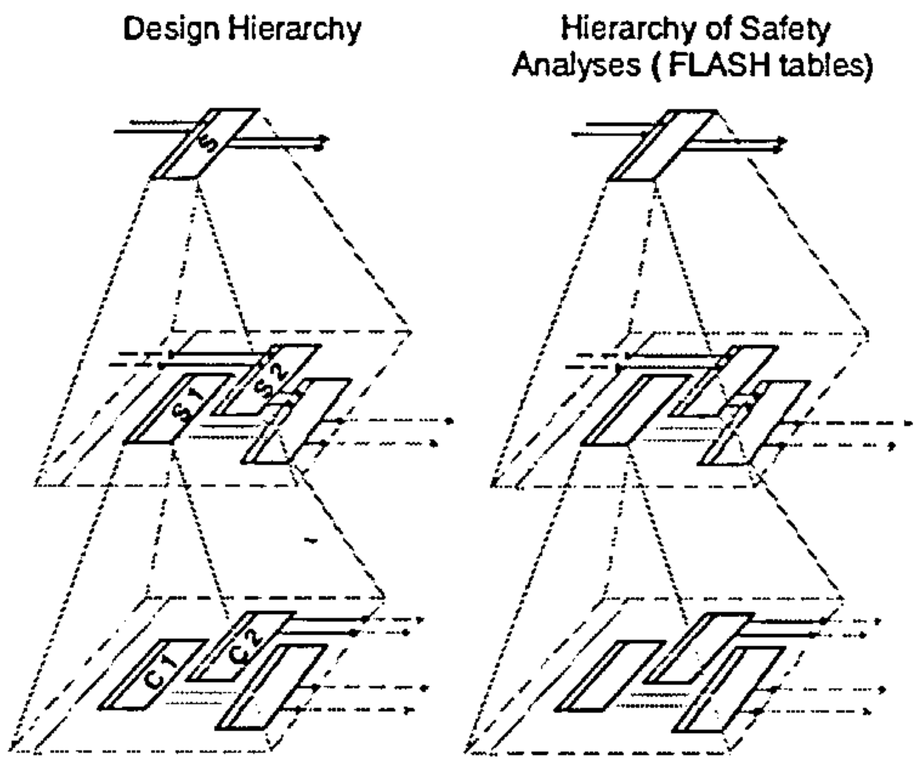
The first column (*Failure events*) lists the failure events generated by the module and propagated to other modules of the architecture. For each such *output failure event*, the second column (*Description*) provides in natural language a description of the event and its effect on the system. The third column (*Causes*) records a logical combination of lower level *failure events* that occur in the subordinate level of the architectural decomposition and cause the *output failure event* under examination. Table 4-1, for example, shows that the event *Output_Deviation_of_S1* can be caused by a simultaneous failure of components *C1* and *C2*.

The next column (*Contributing factors)* contains a set of qualifying conditions that are necessary for the given *output failure event* to occur. Such conditions typically

represent adverse environmental conditions (*temperature* > *max*, for example) or particular states that the system is in. The fifth column (*Criticality-Handling-Recommendations*) contains qualitative results from the analysis of the given event. Those results include the criticality of the event, information on whether it is handled or not, and requirements for ensuring that the event occurs with an acceptable frequency as well as that the system responds well to the occurrence of the event. During the development of the design this information can be used for a preliminary assessment of the architecture against qualitative safety requirements. Once the decomposition process has reached the lower possible level and we know the precise implementation of the system, we can use the reliability data contained in the FMEAs of basic (non-decomposed) components to calculate the frequency of each *output failure event* in the hierarchy. This information is recorded in the *Summary FMEA results* column and can be used to take a final decision on the suitability of the proposed architecture for the system or its constituent parts.

It is beyond the scope of this introduction to explain the precise role of the FLASH table in the development lifecycle. Here, we will just focus on the two most significant columns of the table, that listing the *output failure events* propagated by the module and that listing the causes of those failure events. Our aim is to illustrate how it is possible to achieve consistent linking of safety analyses within the framework of the proposed method. Figure 4-2 illustrates fragments of the analyses for our hypothetical system *S* at three successive levels of its architectural decomposition. It can be noticed that the causes considered at a certain level of the analysis become the failure events considered in subsequent levels. We can notice, for instance, that the output deviation of sub-system *S1* (*Output_ Deviation_of_S1*) appears as a cause of a functional failure at the highest level of the analysis. At the same time, this event also appears in the intermediate level of the analysis where it becomes the failure event under investigation. That consistent linking between the causes and effects of failure which occurs in the vertical axis of the hierarchy is a significant property of the proposed method. This property allows:

a) The implementation of automated checks that can verify the consistency of the analyses;
b) The implementation of hyperlinks between tables which would allow navigation from functional failure modes down to basic events;
c) The mechanical construction of fault trees from FLASH tables.



Failure modes	*Causes*
Functional_Failure_of_S	Output_Deviation_of_S1 AND Output_Deviation_of_S2 ...

Failure modes	*Causes*
Output_Deviation_of_S1	Failure_Mode_of_C1 AND Failure_Mode_of_C2 ...

Failure modes	*Failure rate (λ[1/h])*
Failure_Mode_of_C1	6.3e-7

Figure 4-2: Relationship between design hierarchy and hierarchy of FLASH tables

Figure 4-3 illustrates the fault tree that can be constructed for the high level *functional failure of S* from the information contained in the FLASH tables of Figure 4-2. It is apparent that this fault tree can be mechanically generated by simply traversing the FLASH tables and by progressively substituting the causes of failure at one level of the design with the corresponding failure modes at lower levels. To enable the automatic construction of fault trees in the framework of FLASH, we currently extend an existing algorithm for the mechanical synthesis of fault trees [Papadopoulos and McDermid, 1999b] which already operates on structural models of the system and tabular representations of failure behaviour. This algorithm is at the moment implemented in an experimental tool that supports hierarchical modelling of systems and the synthesis of fault trees for those systems. Figure 4-4 provides a distant view of an example fault tree that has been mechanically synthesised using this tool.

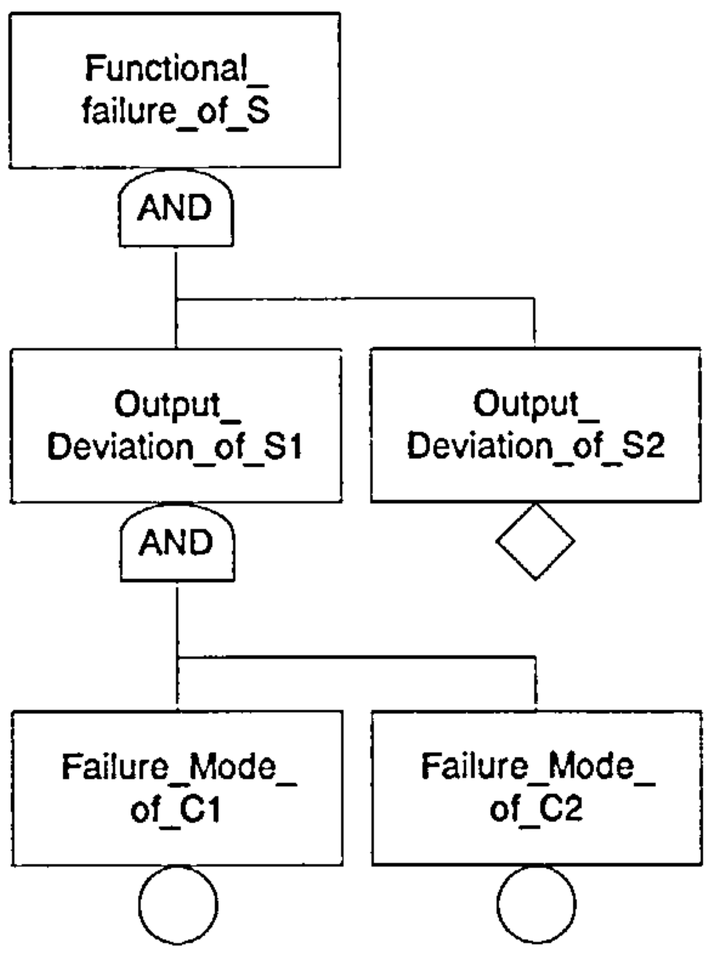


Figure 4-3: The top-level fault tree for the event "*Functional_failure_of_S*"

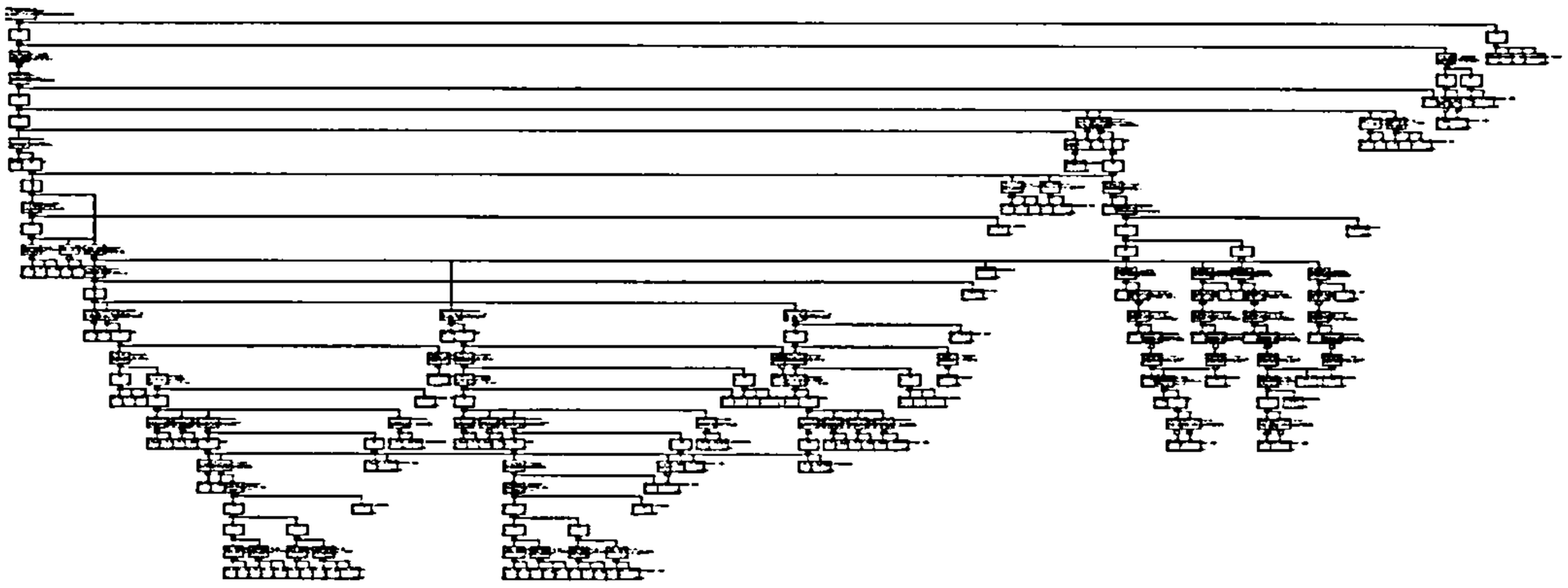


Figure 4-4: An example of a mechanically generated fault tree

It is important to point out that the synthesis algorithm would not be able to generate such fault trees if there are any inconsistencies among the safety analyses. In that case, the algorithm would simply point out the inconsistencies. The resultant *fault trees,* therefore, effectively link in a consistent manner the results from the various analyses to each other and back to the high level FLASH table for the overall system, and hence *guarantee the consistency of results.* At the end of the assessment process, those results (FLASH tables and synthesised fault trees) form an integrated collection of safety analyses which provides a consistent and meaningful representation of anticipated scenarios of the propagation or mitigation of failure in the system.

4.2 FLASH method: tables

FLASH analysis follows the decomposition and design of the system and produces a hierarchy of tables alongside the hierarchy of modules (see Figure 4-5). These tables contain the assessments of peer modules (i.e. functions, systems or components) in the system hierarchy. Whilst modules propagate *flows*, tables propagate *events* which may represent different entities according to whether the module bears the analysis of a function, a system or a component. Before we reach the lowest level of decomposition, events are failures as they appear at the output of the module propagating them, that is *flow deviations on outgoing flows*. At the lowest level of decomposition they represent *internal malfunctions of basic components* in the design.

The table for a function focuses on *functional failure modes* (i.e. loss of function, provision of function when not required, incorrect operation), associates with them a criticality level, lists their causes and where risk reduction is required. On the basis of the risk reduction needed, recommendations and derived safety requirements for the

architecture that achieves the function are given. Figure 4-6 a) summarises fields in a FLASH table used to assess a function.

The table for a system focuses on *system failure modes*. For each failure mode that can potentially be propagated by the system, it identifies causes and where risk reduction is required. On the basis of this risk reduction, derived safety requirements for each component are recorded. Figure 4-6 b) summarises fields in a FLASH table used to assess a system.

The table for a component focuses on *component failure modes* propagated to other components or systems. Causes are identified and for those that are failure modes of that component, reliability data (i.e. failure rates, repair rates, failure probability on demand etc.) and information about the lifecycle to be used for common cause failures analysis are recorded. For the causes that are external events, i.e. input or primary events, the module originating them is identified in the last part of the name of the failure mode itself, the tag. Figure 4-6 c) summarises fields in a FLASH table used to assess a component.

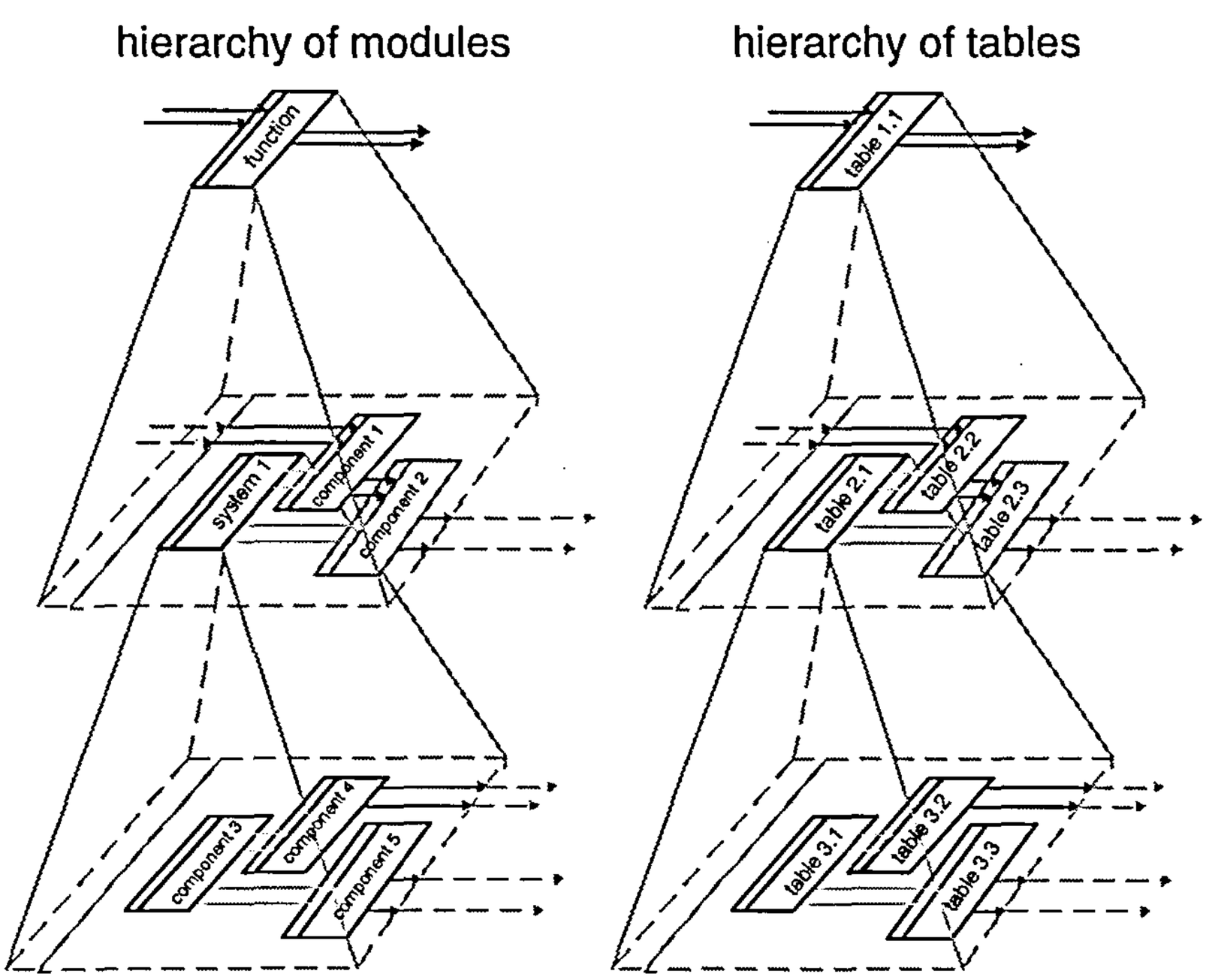
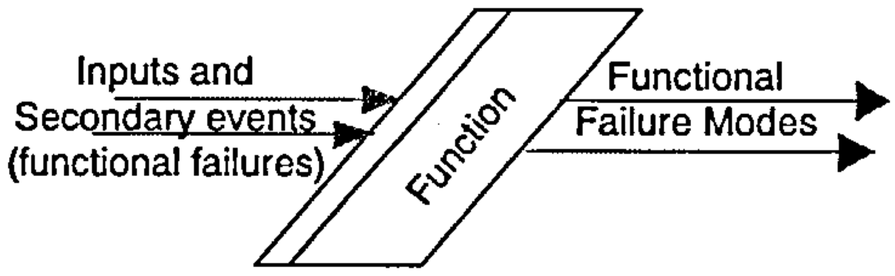


Figure 4-5: Hierarchy[12] of modules and tables

[12] This is a very generic decomposition. We do not mean this decomposition to accommodate any sort of systems

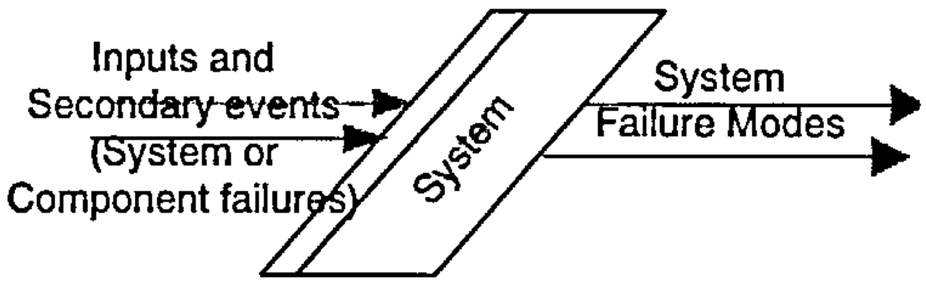
A single generic table template is proposed for these three analyses. The information recorded inside the table identifies whether it refers to a function, a system or a component.



a)

Fields used for the analysis of a function

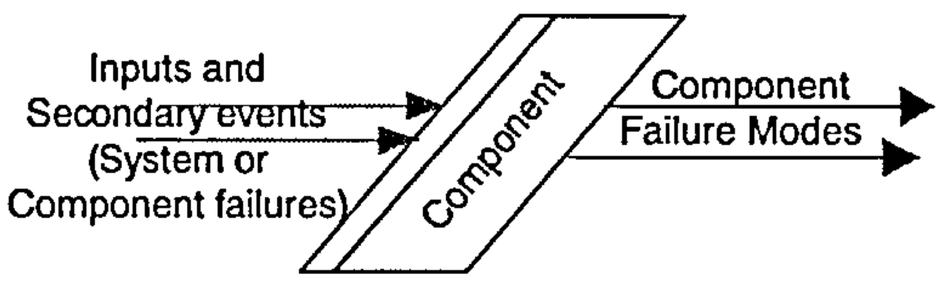
1) Effects (i.e. Functional failure modes)
2) Causes
3) Criticality
4) Description
5) Justification, design recommendation and derived safety requirements
6) FMEA results



b)

Fields used for the analysis of a system

1) Effects (i.e. Failure modes propagated)
2) Causes
3) Description
4) Justification, design recommendation and derived safety requirements
5) FMEA results



c)

Fields used for the analysis of a component

1) Effects (i.e. Failure modes propagated)
2) Causes
3) Description
4) Justification, design recommendation and derived safety requirements
5) Component's failure modes & reliability data (given by the manufacturer)

Figure 4-6: Fields in a FLASH table for a function, a system and a component

---

4.2.1 Events

The term event is used to designate a generic failure mode (or a success mode) that is propagated by a module (i.e. function, system or component) to another and any of its causes, either internal or external. Events are unique entities inside a FLASH analysis, consequently two failure modes of the same type propagated by two identical modules

are actually two different events. Events may represent malfunctions, module failure modes, the intended flow delivered by the module, its deviations from the correct value or tell whether data are delivered on time (i.e. early or late), not delivered at all or delivered when they were not supposed to be delivered. Events are used to link tables across the hierarchy. At the functional level events propagated by a table are *functional failure modes*. At the architectural level events propagated by a table are *system failure modes*. At the component level events propagated by a table are *component failure modes*. Events have their own syntax illustrated in Figure 4-7.

Event syntax

Events are identified by two pieces of information: an *entity* and a *tag*. The syntax is:

<*Event*>:= < *Entity*>.<*Tag*>

The *tag* identifies the module propagating the event (e.g. the name of the component or an acronym). The *entity* characterises the event. The entity may assume various meanings. In the FHA, it represents functional failure modes:

< *Entity*>:= <*Functional failure Mode*> (e.g.: OMISSION OF FUNCTION X)

In HAZOP, an *entity* consists of two pieces of information: a *flow* and a *deviation* that can be associated with that flow.

<*Entity*>:= <*Deviation*>.<*Flow*> (e.g.: LESS.OUTPUT PRESSURE)

In FMEA, an *entity* represents a component failure mode.

<*Entity*>:= <*Component failure mode*> (e.g.: SENSOR SHORT TO GROUND)

Figure 4-7: Syntax for events

Whether events represent functional, system, component failure modes or the intended value at the right time, they can be ranked using the taxonomy in Figure 4-8. Events are classified into three classes: events that are propagated by modules i.e. *Outgoing Events*

and are also called *Effects*; the ones that enter the component, i.e. *Incoming Events*; and the ones that are generated inside a module, i.e. *Generated Events*.

Outgoing events can be propagated either towards modules at the same hierarchical level, (i.e. *to the same level*) or to modules at a higher hierarchical level, (i.e. *enclosing level*).

Incoming events can either be *input* or *secondary* events. *Input events* are differentiated from *secondary events* since they represent deviations of the variables (e.g. flows, data etc.) that are processed by the module which the table refers to (i.e. the module was designed to process those variables). *Secondary events* represent deviations of flows supporting the function the module has to achieve. They provide what the module needs to carry out its task, e.g. power supply. Both input and secondary events may come from tables of the same hierarchical level, i.e. from *same level*[13], or from tables belonging to the higher hierarchical level, i.e. *enclosing level*.

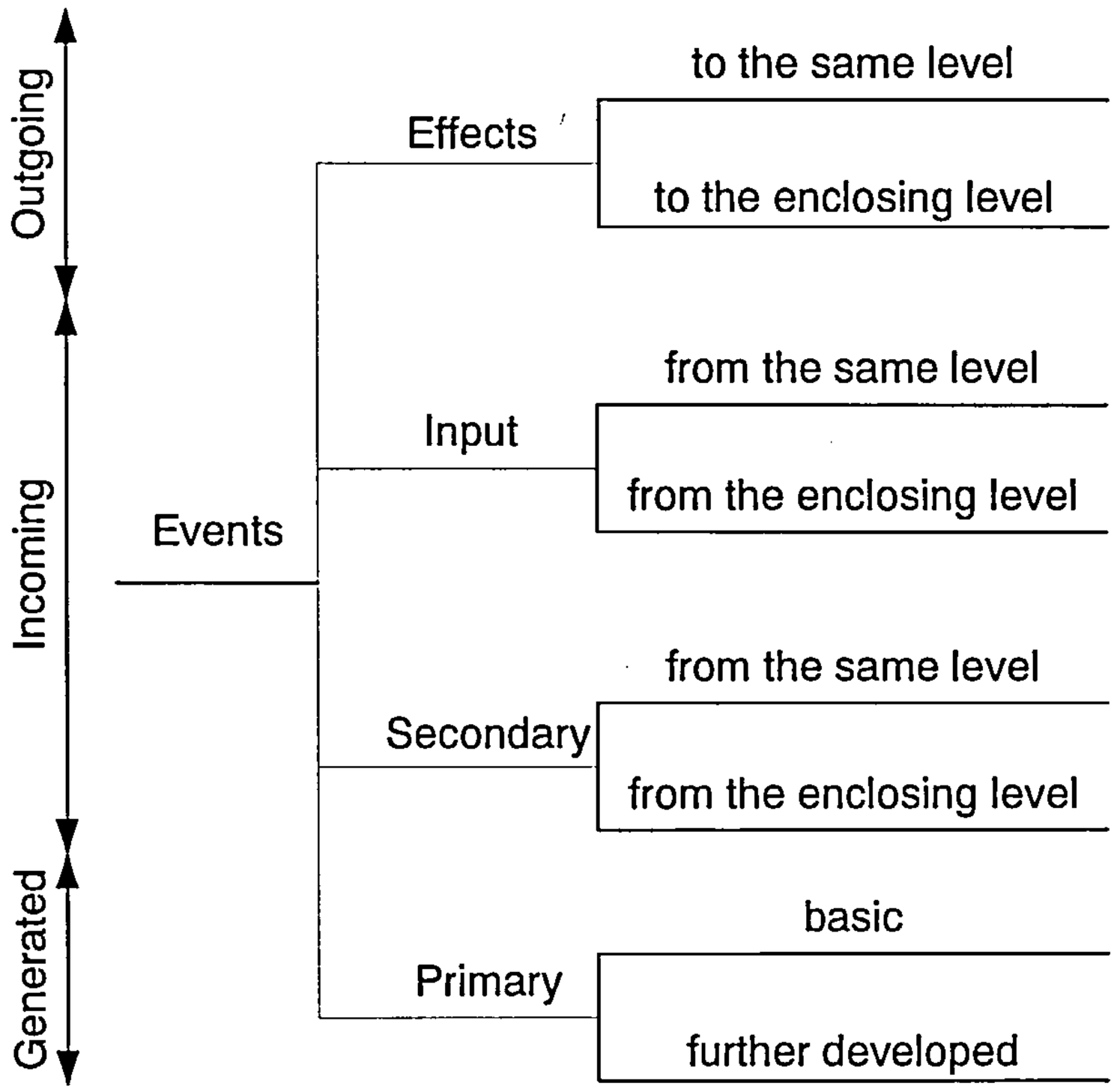


Figure 4-8: Taxonomy of events

[13] We found that the distinction between Input (and Secondary) events from the **same** level and from the **enclosing** level helps when parsing tables for the construction of fault trees. The algorithm may either follow, at first, links that come from the **same** level and then from the **enclosing** (or the other way round).

Events generated inside a module boundary are called *Primary events*. When the module represent a *basic component* (i.e. not further decomposed) they are called *Basic events*. For a basic event it is generally possible to give reliability data. When the module represents a system that is further decomposed into sub-modules or components they are called *Primary event further developed*. Causes of primary events further developed are investigated by analysing enclosed modules. Figure 4-9 summarises how effects, primary, secondary and input events relate each other. They refer to the highlighted module.

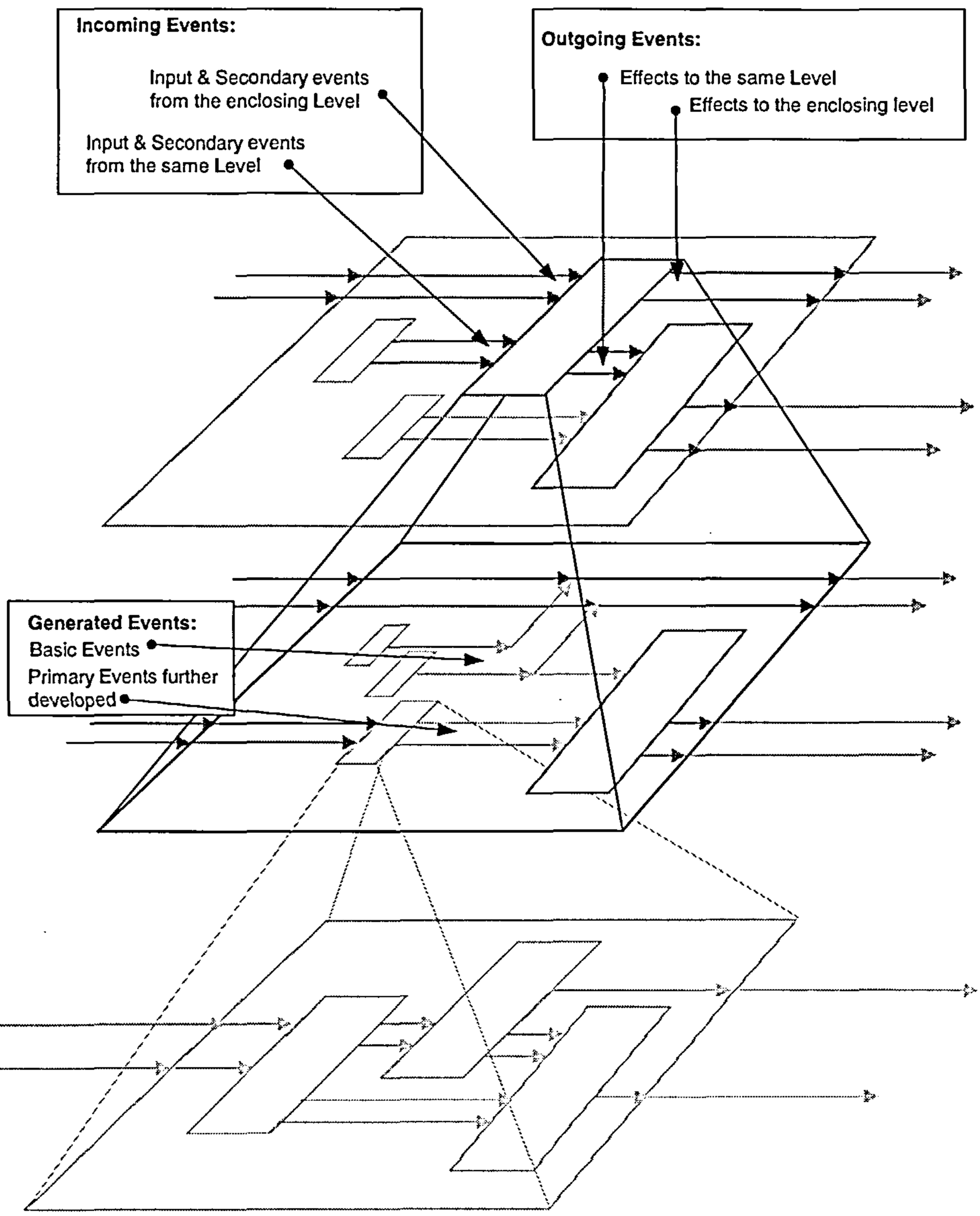


Figure 4-9: Incoming, Outgoing and Generated

4.2.2 Areas inside a table

FLASH tables are divided into three main areas that are used: to analy se outgoing events, to list incoming events and generated events (see Figure 4-10). The *Outgoing event area* is for the analysis of events propagated towards modules of the same or the enclosing level. The *Incoming event area* lists input and secondary events. The *Generated event area* lists primary events and records information about basic events.

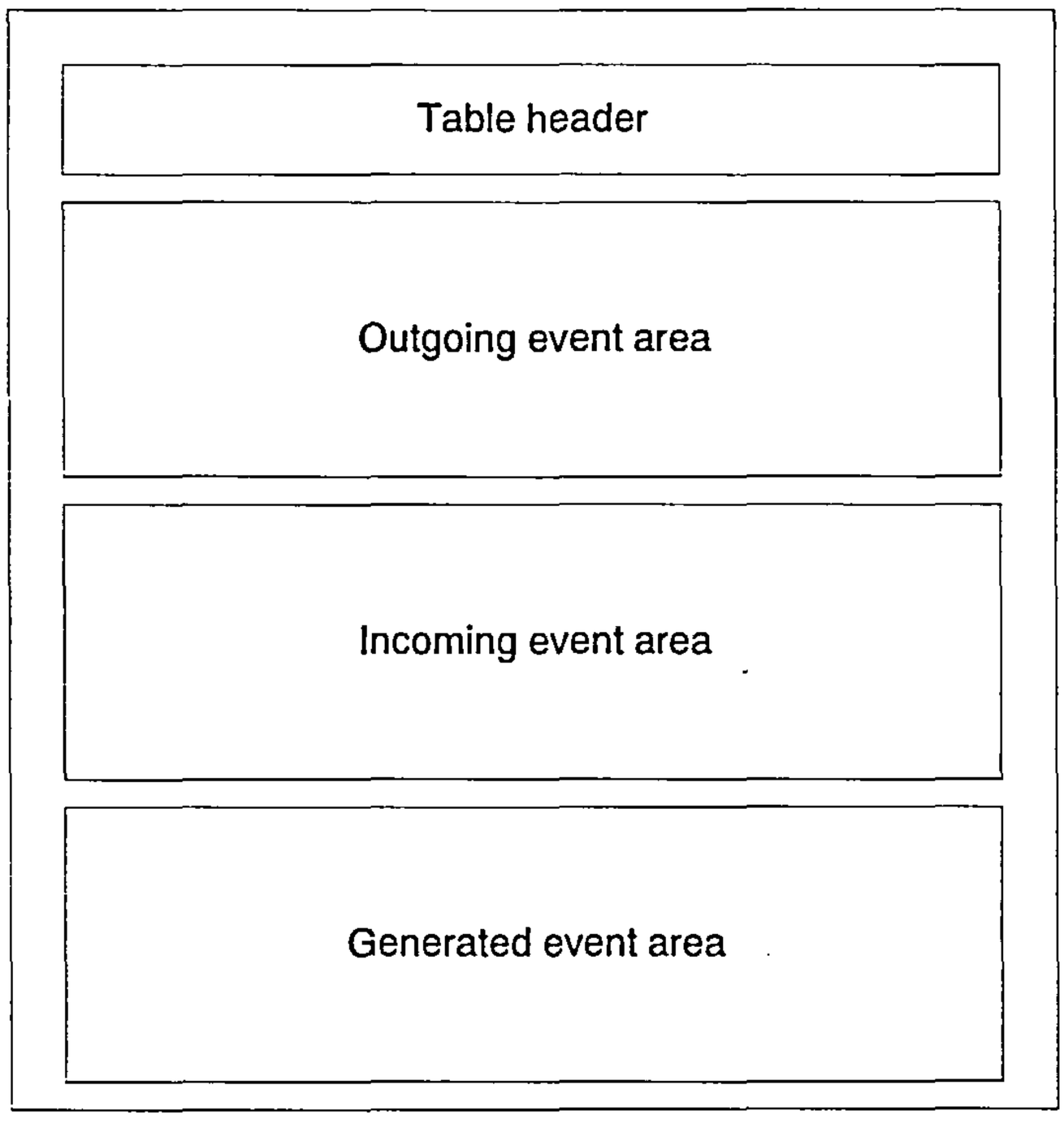


Figure 4-10: Main areas of a FLASH table

4.2.3 Outgoing event area: Effects

When referring to a module, events leaving the boundary are called *Effects*. A module *transforms* primary, secondary and input events into other events that are *propagated*. This may happen, for instance, when a timing error enters from an input and a value error is delivered by the output, or when a value error enters a module, but it is detected by a voting logic mechanism that allows the module to deliver a "good" event. The relation that models the transformation of events inside a module is written into the *Causes Column* of the FLASH table and (if it is made by only *AND* and *OR* gates) obeys the syntax in Figure 4-11. In the case additional gates are required to model the

transformation (e.g. *XOR*, *N out of M* or *dynamic fault tree gates*), more *composition rules* have to be added to the ones listed here. When an event leaves a module, it inherits the *tag* that identifies the father. The tag can be either the full name of the module or an acronym.

<expression> :: = <term> | <composition>
<composition> :: = <conjunction> | <disjunction>
<conjunction> :: = <term> "AND" <expression>
<disjunction> :: = <term> "OR" <expression>
<term> :: = <event> | "(" <expression> ")"
<event> :: = see Figure 4-7

Figure 4-11: Syntax of the Causes column of a FLASH table

For the construction of the hierarchy of tables, we found it useful to divide effects into two tables. Effects that are propagated directly towards the boundary of the enclosing module (i.e. to the higher hierarchical level) that are analysed in the table in which the first column is *Event to a higher level*; and effects that are propagated towards other enclosed modules, which appear in the table in which the first column is *Event to the same level*. Both these tables have six columns (see Table 4-2).

From left to right, the *Effects column* lists events propagated that have to be analysed. The *Causes* column records the logical combination of *events* (i.e. incoming and generated events) which cause the *Effect*. The *Description* column gives details of consequences of the event propagated. The *Criticality* column contains the criticality of the *Effect*s, however it is used only at the functional level. The 5th column (*Justifications, recommendations, derived safety requirements ...etc*) contains the result of the safety assessment of the proposed model of the module (i.e. against safety requirements). If the design satisfies such constraints, then the system decomposition proceeds, i.e. each sub-module is further decomposed. If the design is not satisfactory, a decision is taken on whether to modify the model of this module or the model of the enclosing module. In addition the table contains recommendations and derived safety requirements to develop enclosed modules. The *FMEA results* column is used in the integration and verification phase, that is when the decomposition process has reached

the lower possible level and rates becomes available for most of the basic events. At this stage, fault trees can be constructed for events propagated by parsing causes columns in FLASH tables. Summarised results from the probabilistic calculations of these fault trees are recorded in the *FMEA results column*. This information is then used to take a final decision on the suitability of the proposed model for the module.

Effects Events to the same level	**Causes**	**Description**	**Criti-cality**	**5th column Justification, Recommendations Derived Safety Requirements**	**Verifica-tion (FMEA results)**
Event_10	Event_1 AND/OR Event_2 AND/OR Event_5 AND ***(Event_6 OR Event_7)*** {AND/OR ...}	Description of the Event_10 and its consequences		Analysis of the Event_10 and derived safety requirements for the modules causing this event	Event_10 likelihood, as calculated from comp.'s data sheets
Event_20	Event_1 AND/OR Event_3 AND ***(Event_6 OR Event_7 OR Event_11)*** AND ***(Event_6 OR Event_7 AND Event_11)*** {AND/OR ...}	Description of the Event_20 and its consequences		Analysis of the Event_20 and derived safety requirements for the modules causing this event	Event_20 likelihood, as calculated from comp.'s data sheets
...					

Effects Events to the enclosing level	**Causes**	**Description**	**Criti-cality**	**5th column Justification, Design Recommendations Derived Safety Requirements**	**Verifica-tion (FMEA results)**
Event_30	Event_2 AND/OR Event_4 OR ***Event_6 OR Event_7*** AND/OR Event_11 AND/OR Event_14 {AND/OR ...}	Description of the Event_30 and its consequences		Analysis of the Event_30 and derived safety requirements for the modules causing this event	Event_30 likelihood, as calculated from comp.'s data sheets
Event_40	Event_6 AND/OR Event_7 AND/OR ***(Event_6 OR Event_7AND Event_11)*** AND/OR Event_12 AND/OR Event_15 {AND/OR ...}	Description of the Event_40 and its consequences		Analysis of the Event_40 and derived safety requirements for the modules causing this event	Event_40 likelihood, as calculated from comp.'s data sheets
...					

Table 4-2: Effects to the same and enclosing level

Table for groups of events

Sometimes the expression in the Causes column of a FLASH table is quite complicated. To simplify it we found it useful to take out of that column those groups of events that repeat in different rows (or that may have particular meanings). See, for example, groups of events that are highlighted in bold italic characters in Table 4-2: they can be taken out. The group of events "*Event_6 OR Event_7*" appears in rows for events: *Event_10, Event_20* and *Event_30*. In similar cases to make the table more neat and tidy, we suggest the substitution of repeated groups of events with one single event in this case it is called *GOE_1*. This new event and its causes are described into another table that is called *Group of events table*. Table 4-3 is an example, it has an identical structure to the effects table. A similar thing is done for the other group of events in Table 4-2 i.e. "*Event_6 OR Event_7 AND Event_11*", which is called *GOE_2*. After substituting groups of events that we have identified, Table 4-2 appears as in Table 4-4.

Group of events	Causes	Description	Criticality	5th column, Justification, Design Recommendations, Derived Safety Requirements	FMEA
GOE_1	Event_6 OR Event_7	Description of the GOE_1 and its consequences		Analysis of the GOE_1 and derived safety requirements for the modules causing this event	
GOE_2	Event_6 OR Event_7AND Event_11	Description of the GOE_2 and its consequences		Analysis of the GOE_2 and derived safety requirements for the modules causing this event	
...					

Table 4-3: Groups of events written for Table 4-2

4.2.4 Incoming event area: Input and Secondary events

Input and Secondary events are the only events that enter the module boundary. Each of them represents a flow with one of its deviations. Input events are differentiated from secondary events. Whilst Input events are processed by the module, Secondary events provide the module with what it needs to process input events. For example, take an Electronic Controller that receives signals from sensors, sends signals to actuators and needs a power supply to operate. Signals coming from sensors with any of the deviations that applies to them (i.e. omission, commission, early late, etc.) represent Input Events,

whilst the power supply with deviations from its expected value is a secondary event. Input and Secondary events are listed into two tables differentiating whether they come from modules on the same or the enclosing level.

Effects	Causes	Description	Criti-cality	5th column, Justification, Design Recommendations, Derived Safety Requirements	Verifica-tion (FMEA results)
Event_10	Event_1 AND/OR Event_2 AND/OR Event_5 AND **GOE_1***)* {AND/OR ...}	Description of the Event_10 and its consequences		Analysis of the Event_10 and derived safety requirements for the modules causing this event	Event_10 likelihood, as calculated from comp.'s data sheets
Event_20	Event_1 AND/OR Event_3 AND (**GOE_1** *OR Event_11)* AND **GOE_2** {AND/OR ...}	Description of the Event_20 and its consequences		Analysis of the Event_20 and derived safety requirements for the modules causing this event	Event_20 likelihood, as calculated from comp.'s data sheets
...					

Effects	Causes	Description	Criti-cality	5th column, Justification, Design Recommendations, Derived Safety Requirements	FMEA
Event_30	Event_2 AND/OR Event_4 AND/OR **GOE_1** AND/OR Event_11 AND/OR Event_14 {AND/OR ...}	Description of the Event_30 and its consequences		Analysis of the Event_30 and derived safety requirements for the modules causing this event	Event_30 likelihood, as calculated from comp.'s data sheets
Event_40	Event_6 AND/OR Event_7 AND/OR **GOE_2** AND/OR Event_12 AND/OR Event_15 {AND/OR ...}	Description of the Event_40 and its consequences		Analysis of the Event_40 and derived safety requirements for the modules causing this event	Event_40 likelihood, as calculated from comp.'s data sheets
...					

Table 4-4: Effects written using Groups of Event, defined in Table 4-3

4.2.5 Generated Events area: Primary events

Events generated inside the module boundary are called *Primary events*. There are two types of primary events: *Basic events* which are not developed any further, and *Primary events further developed* which are propagated by enclosed modules. When the module is a basic component, it is often possible to provide failure rates for its basic events.

When the module is further decomposed into other modules, its failure modes are the results of failures in its enclosed sub-modules or components, hence the analysis has to go further, investigating lower levels.

Basic Events

Basic events are component failure modes for which causes are not investigated any further. Often it is possible to collect reliability data for these events from the manufacturer of the component, but in some other cases (e.g. some software) it is not possible. The table for basic events is divided into two sub -areas. The upper part is used to collect *Reliability data*, the lower part to collect *Lifecycle information*, see *Table 4-5*. Reliability data (i.e. failure and repair rates, mean time to failure, failure probability on demand etc.) are calculated from manufacturer's data sheets and adapted to the environment where the component operates (i.e. temperature, vibrations, magnetic fields, humidity etc.). Lifecycle information is additional data regarding the component generating the event. It records information about the whole life of the component, going from the design, through the production, installation, testing, maintenance, and the environment where the component operates. It is collected from various sources including manufacturer, designers, experienced people working in maintenance and testing of similar installations and weighted using a multiple criteria decision analysis methods such as that in [Prasad, 1998]. Lifecycle information is actually a list of defects or errors that may occur during the component lifecycle that are likely to cause the component to fail in one of its failure modes. Defects can occur in the manufacturing process, in the materials employed or in the assembly line; errors can be in specifications, architecture, design, choice of materials, installation, test, operation, maintenance, etc. The list of defects and errors is supposed to be exhaustive[14], span the whole lifecycle and divide it into mutually independent categories called *Lifecycle Categories*. Lists in Table 2-12, Table 4-5 (taken from [SAE-ARP 4754, 1996]) and Table 5-1 obey such constraints. Lifecycle information is used to identify couplings among events and to estimate common cause failure probabilities[15]. For each *ith* Category of the lifecycle two data are given: the *Percentage* $\%_i$ and the *Coupling Code*.

The *Percentage* "$\%_i$" represents the contribution of the *ith* lifecycle Category to the likelihood of the basic event X. For example, take a generic component, manufacturer's

---

[14] In practice it can never be exhaustive, but it should be as extensive as practical.

[15] The issue of Common Cause Failure analysis is dealt in details in Chapter 5.

data show that the likelihood of one of its failure mode to be due to errors during maintenance is low e.g. below 1%. They justify this by saying that there is a very simple procedure that operators rarely get wrong. However the environment where the maintenance take place is quite harsh, according to experienced people working in maintenance of similar equipment and they say that this increases the likelihood of errors if compared with normal situations. Similar considerations are made for all other lifecycle categories. The safety analyst after hearing all the different viewpoints associates numbers to each of the lifecycle categories. Obviously the sum of all the *Percentage* "$\%_i$" has to be 100. It is not our intention to explain any formal method to arrive to those numbers, we just say that there are methods for the evaluation and consideration of the expert judgement that can be employed for this task [Prasad, 1998].

The *Coupling Code* specifies the actual source of the coupling. For example, take a group of valves of the same type. Several lifecycle categories may be responsible for failure modes in these valves, *maintenance procedures* is one of these. If it is known (i.e. from specifications) that the same maintenance procedure is used for the maintenance of all these valves, then failure modes of these valves will have the same coupling code for the *Lifecycle Category* "Maintenance Procedures". That coupling code is the identifier of that specific maintenance procedure.

Chapter 5 explains in details how FLASH uses coupling codes and lifecycle categories for common cause failure analysis.

Primary events further developed

Primary events further developed are generated by components or sub-modules of the module under examination, i.e. this module is decomposed into more units which generate these events. Reliability data for these events become available only when the system hierarchy has been decomposed into sufficient detail that there are sufficient data to build and evaluate fault trees with these events at the top.

4.2.6 Table template

In the FLASH method the same table template is used for the analysis of any module at any level in the system hierarchy. The template we propose for such analysis is presented in Table 4-6. The header identifies the module's instance, type, periodicity (in case it represents a real time software function), and the acronym that is used to identify the module in the hierarchy (i.e. tag). Areas for Outgoing events, Incoming events and Primary events follow below the header. However not all the fields of the table are

always used. The criticality column, in the outgoing event area, is considered only when the module represents a high level function, and incoming and generated events areas may not be used for some components.

Having said that, we add that the layout of a FLASH table is not strict, it can be modified to suit needs that may arise in the analysis of some systems. For instance, an additional column in the area for outgoing events may be necessary when causes of an event, which is propagated by the module, are function of a state (or mode) of the system. The additional column will made it possible to distinguish among different failure mechanisms that propagate the same failure mode but in different states of the system. If we do not have an additional column, which identifies the state in which the failure mechanism applies, the state has to be considered somewhere else, for instance in the causes column e.g. by using conditional or dynamic gates, however complicating the expressions.

			Basic Events			
			Event_1	Event_2	Event_3	...
Reliability Data		Failure Rate λ[1/h]	10^{-4}	-	-	...
		Repair Rate μ[1/h]	-	-	-	...
		Mean Time to Failure MTTF [h]	-	-	-	...
		Failure Probability on demand	-	10^{-3}	10^{-4}	...
		Mission time for the system [h]	50	50	50	...
Lifecycle Categories			Coupl. Code / %	Coupl. Code / %	Coupl. Code / %	Coupl. Code / %
	Concept and Design	Design Architecture	DCA1 / 2	DCA1 / 8	DCA2 / 4	... / ...
		Technological Materials Equipment Type	DTM1 / 3	DTM1 / 7	DTM2 / 2	... / ...
		Specifications	DS1 / 1	DS1 / 6	DS2 / 7	... / ...
	Manufacturing	Manufacturer	MM1 / 3	MM2 / 5	MM2 / 3	... / ...
		Procedures	MPD 1 / 5	MPD 2 / 4	MPD 2 / 6	... / ...
		Process	MPP 1 / 1	MPP 2 / 8	MPP 2 / 4	... / ...
	Installation/ Integration And Test	Fitter	IIF1 / 3	IIF1 / 5	IIF1 / 7	... / ...
		Procedures	IIP1 / 6	IIP2 / 4	IIP3 / 6	... / ...
		Location	IIL1 / 2	IIL2 / 6	IIL3 / 5	... / ...
		Routing	IIR1 / 5	IIR2 / 7	IIR3 / 4	... / ...
	Operation	Staff	OS1 / 4	OS1 / 8	OS2 / 7	... / ...
		Procedures	OP1 / 6	OP1 / 6	OP2 / 4	... / ...
	Maintenance	Staff	MS1 / 7	MS1 / 2	MS2 / 2	... / ...
		Procedures	MP1 / 8	MP1 / 3	MP2 / 6	... / ...
	Test	Staff	TS1 / 6	TS2 / 1	TS3 / 5	... / ...
		Procedures	TP1 / 8	TP2 / 3	TP3 / 4	... / ...
	Calibration	Staff	CS1 / 7	CS2 / 5	CS3 / 5	... / ...
		Procedures	CP1 / 6	CP2 / 1	CP3 / 5	... / ...
	Environmental	Mechanical and Thermal	EMT1 / 5	EMT2 / 3	EMT3 / 4	... / ...
		Electrical and Corrosion	EEC1 / 4	EEC2 / 6	EEC3 / 5	... / ...
		Chemical and miscellaneous	ECM1 / 8	ECM2 / 2	ECM3 / 5	... / ...

Table 4-5: Basic Events in a FLASH table

4.2.7 Programmable electronic modules

TABLE HEADER					
Instance = <Name>	**Component Type** = <Name>		**Periodicity** = <Periodicity>		**Tag** = <Name>
OUTGOING EVENTS					
EFFECTS					
Same level	**Causes**	**Description**	**Criticality**	**5th Column Justification, Design Recommendations, Derived Safety Requirements**	**Verification (FMEA results)**
NA | <Event>	NA | <expression>	NA | <description>		NA | <description>	NA | <description>
{<Event>}	{<expression>}	{<description>}		{<description>}	{<description>}
Enclosing Level	**Causes**	**Consequences**	**Criticality**	**5th Column Justification, Design Recommendations, Derived Safety Requirements**	**Verification (FMEA results)**
NA | <Event>	NA | <expression>	NA | <description>		NA | <description>	NA | <description>
{<Event>}	{<expression>}	{<description>}		{<description>}	{<description>}
GROUPS of Events					
Group of Events	**Causes**	**Consequences**	**Criticality**	**5th Column Justification, Design Recommendations, Derived Safety Requirements**	**Verification (FMEA results)**
NA | <Event>	NA | <expression>	NA | <description>		NA | <description>	NA | <description>
{<Event>}	{<expression>}	{<description>}		{<description>}	{<description>}
INCOMING EVENTS					
INPUTS					
Same level			**Enclosing level**		
NA | <Event> {< Event>}			NA | <Event> {< Event>}		
SECONDARY EVENTS					
From the Enclosing Level			**From Modules of the same level**		
NA | <Event> {< Event>}			NA | <Event> {< Event>}		
GENERATED EVENTS					
PRIMARY EVENTS					
Primary Events Further developed					
NA | <Event> {<Event>}					
Basic Events					

	Reliability data	<Event>	<Event>	<Event>	{<Event>}
	Failure Rate λ[1/h]	-	-	-	
	Repair Rate μ[1/h]	-	-	-	
	Mean Time to Failure MTTF [h]	-	-	-	
	Mission time [h]	-	-	-	

Lifecycle Category		**Coupl. Code**	**%**	**Coupl. Code**	**%**	**Coupl. Code**	**%**	**Coupl. Code**	**%**
Concept and Design	Design Architecture	...	...	...	...	...	...	...	...
	Technological Materials Equipment Type	...	...	...	...	...	...	...	...
	Specifications	...	...	...	...	...	...	...	...
Manufacturing	Manufacturer	...	...	...	...	...	...	...	...
	Procedures	...	...	...	...	...	...	...	...
	Process	...	...	...	...	...	...	...	...
Installation/ Integration And Test	Fitter	...	...	...	...	...	...	...	...
	Procedures	...	...	...	...	...	...	...	...
	Location	...	...	...	...	...	...	...	...
	Routing	...	...	...	...	...	...	...	...
Operation	Staff	...	...	...	...	...	...	...	...
	Procedures	...	...	...	...	...	...	...	...
Maintenance	Staff	...	...	...	...	...	...	...	...
	Procedures	...	...	...	...	...	...	...	...
Test	Staff	...	...	...	...	...	...	...	...
	Procedures	...	...	...	...	...	...	...	...
Calibration	Staff	...	...	...	...	...	...	...	...
	Procedures	...	...	...	...	...	...	...	...
Environmental	Mechanical and Thermal	...	...	...	...	...	...	...	...
	Electrical and Corrosion	...	...	...	...	...	...	...	...
	Chemical and miscellaneous	...	...	...	...	...	...	...	...

Table 4-6: Template for a FLASH table of a generic module

Programmable electronic modules are those modules that contain software. They may represent control units (made up of processors, memories, input and output circuits), Programmable Logic Controllers (PLC), smart sensors and smart actuators. In addition to hardware failures, these modules can suffer from software failures. Software failures are caused either by failure of the hardware upon which the software runs (e.g. memory and processor errors) or by flaws in software. The problem with the analysis of programmable electronic modules is due to the complexity of the hardware and the difficulty of mapping the software onto the hardware. The mapping depends on many factors, amongst these the compiler used to create the binary executable file and the chipset on which the binary file runs. In some cases the software dynamically allocates processes and variables to various hardware resources e.g. often variables are dynamically allocated in memories. Hence, a detailed mapping of the software onto hardware can be very complex.

The study of a system with both software and hardware components is usually performed considering all interdependencies. The analysis of programmable electronic modules in the FLASH method has a similar approach. A system is hierarchically decomposed regardless of the fact that functions are achieved by software, hardware, or by a mix of both of them. The decomposition proceeds until failure events propagated by a component are modelled as a combination of hardware and software basic events. Figure 4-12 shows the model of a programmable electronic module of such a kind. It encloses several sub-modules: input, output, processor, and the software. *Input* includes primary events that describe the failure and success of input circuits and registers. *Output* includes primary events that describe the failure and success of output circuits and registers. *Processor* contains primary events that describe the failure and success of processors and memories. *Software* contains primary events that account for requirements, specifications and implementation flaws of the executable file. Arrows that connect input, output and processor to the software module in Figure 4-12, represent success and failure events that can be transformed inside the programmable electronic module by the software. For example, the failure of *one out of n* redundant processors can be recovered by suitable software voting logic. Arrows that connect the software module to the output module represent success and failure events that can be transformed inside the output module by a suitable hardware. For instance the output module may have implemented hardware voting logic able to recover from some software failures. Arrows that connect input, output, software and processor straight to effects, represent

success and failure events that cannot be transformed any further inside the programmable logic module, e.g. undetectable software flaws, some requirements and specification errors, or register failures in the output module. They result in the propagation of failure events. All the events and combination of events that cause the same effect are linked by "OR" gates in the cause column of the relevant FLASH table. A full understanding of the system and how it works is necessary to build the model of the module and the corresponding FLASH table.

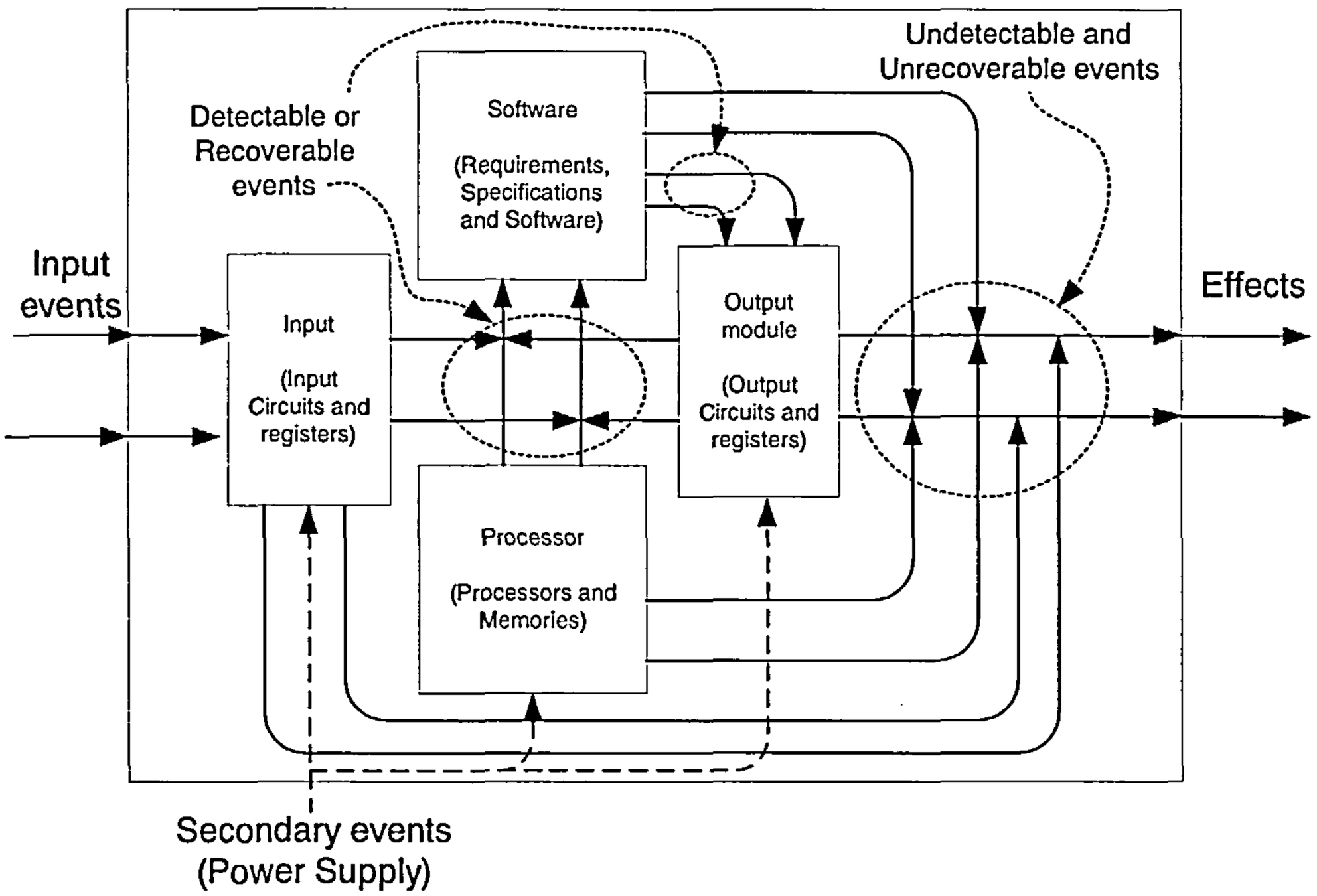


Figure 4-12: Propagation of events in a programmable electronic component

4.3 FLASH method: process

The previous section explained the *statics* of the FLASH method, that is the system hierarchy, the hierarchy of the safety analysis, and the FLASH table with all its entries. This section explains the *dynamics* of the FLASH method, that is the process of conducting a FLASH analysis by completing FLASH tables. First of all we explain how FLASH supports and drives the development of the system design. Then, we consider the ways in which FLASH supports the integration of different analysis and makes

possible overall system verification. A simple case study is used to show the process in practice.

4.3.1 Decomposition and Design

The FLASH process in the decomposition and design stage of the lifecycle is split into two phases, the first comes before the design of the internal model of the module takes place, the second comes after the design stage. During the first phase (see Figure 4-13), the analyst identifies events that are propagated by the module and consequences of such events on the whole system and environment. This information is stored in the Effects and the Description columns of the FLASH table. This preliminary part of the process makes safety analysts to focus on the severity of events propagated so that they can issue safety-related recommendations to designers for the development of the internal architecture of the model. These recommendations are written into the upper part of the 5th column. Once designers have produced the design of the module, safety analysts assess it by checking whether it meets the recommendations they gave before. For such analysis they consider hypothetical failure modes that may be propagated by components and sub-modules and write the mechanism underneath the propagation of failures in the causes column, in terms of generated, inputs, primary events and logical operators. The equation in the causes column is similar to the ones in FPTN [Fenelon et al., 1994], but in addition to the FPTN notation, in FLASH, the equation is later analysed and results recorded into the 5th column. Such analysis may bring safety analysts to accept the design proposal or to refuse it, giving some justifications. To help analysts in their decision making, the 5th column has been divided into sub headings: Detection, Recovery, Maximum accepted likelihood for critical events in the causes column, and Recommendations. Basically, the 5th column reports information on whether it is possible to detect or recover from the event propagated, the maximum accepted likelihood for critical events in the causes column and recommendations, either for choosing suitable components to place inside the module or for developing sub-modules. Table 4-7 shows how the 5th column is partitioned.

One of the concerns with FLASH is about the amount of information stored in the 5th column. Such information could be spread on multiple columns simply by changing the layout of the table. However, we prefer to keep this arrangement since during the integration and verification stage it is easier to compare recommendations and derived

safety requirements with what is actually achieved in the real system and recorded into the FMEA results column that is the 6th column.

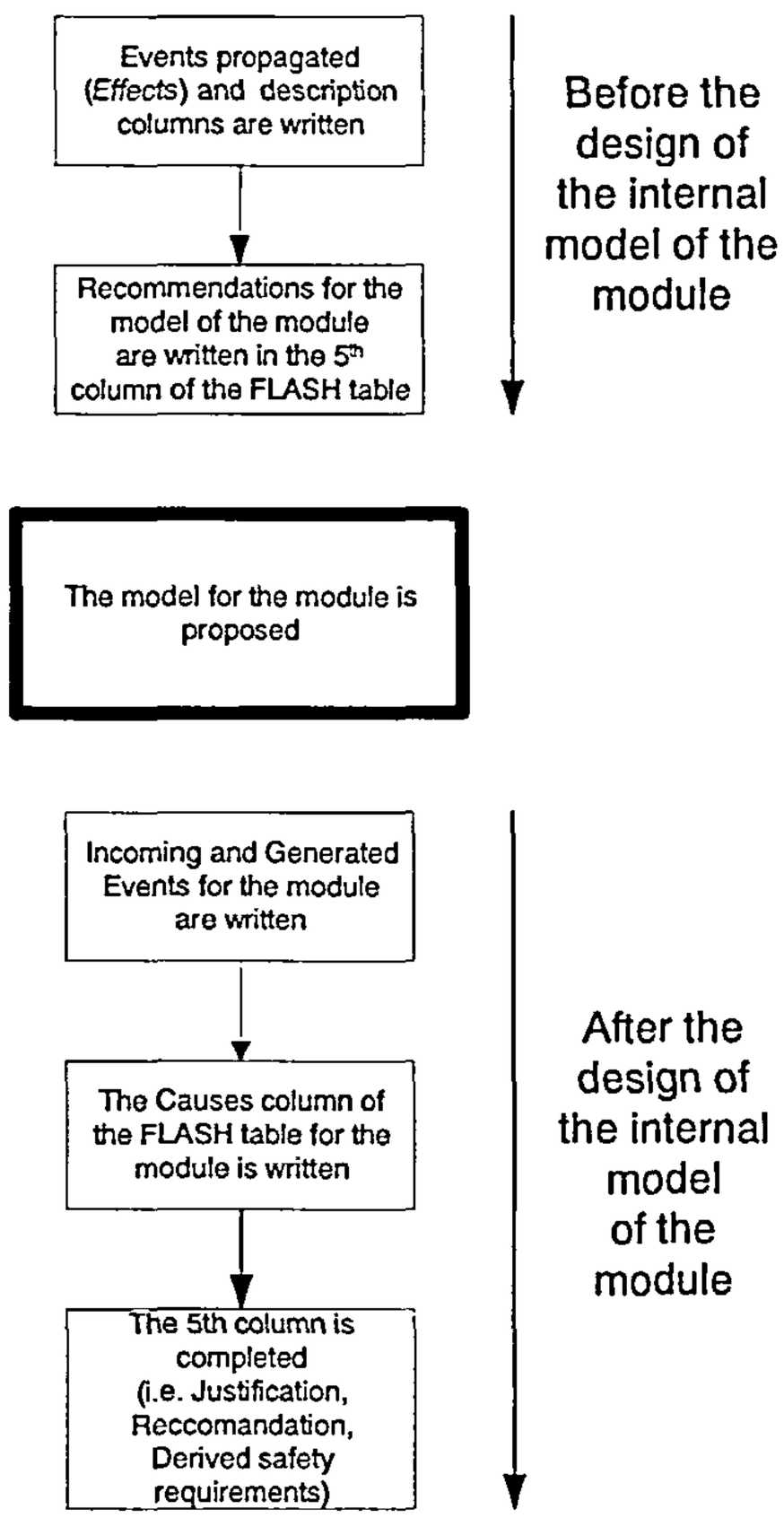


Figure 4-13: Process of creating a FLASH table

Effects	...	5th Column (Justifications, Recommendations, derived safety requirements,)	Verification (FMEA results)
Event propagated	...	*Before design* Recommendations ... Effect max accepted likelihood ... *After design* Detection ... Recovery ... Recommendation ... Max accepted likelihood for critical causes. ...	
...	...	...	...

Table 4-7: The 5th column is divided into many areas

The causes column

The expression modelling the transformation of events that goes into the Causes column of a FLASH table is constructed from the knowledge of the internal design of the module, of flows exchanged among its components and sub-modules, and their failure modes. Two approaches exist to write that equation. In the first approach, the fault tree for each effect is first constructed, then reduced to a logical expression, and finally this expression is written in the Causes column. The knowledge of the layout of the module, its components and sub-modules is used to draw the fault tree as suggested in [Vesely, 1981]. This approach is suitable for any modules of the system hierarchy. In the second approach, the correlation among causes and effects is written without the previous construction of the fault tree. This requires more effort from analysts since they have to make the effort to do all the steps for the construction of the fault tree before writing the actual expression. However this second approach can be much faster for experienced analysts than the first, and we recommend it for simple modules. These two approaches are shown for the system in Figure 4-14, which is made by four components and a control unit. These four components are divided into two groups. Components A1 and B1 make *Line 1*, components A2 and B2 make *Line 2*. Only one line is needed for the system to work. The task of the system is to regulate a flow going from left to right.

In the first approach the tree with the effect *No.Flow.Module* as top event is defined first, see Figure 4-15. Then it is reduced to a logical expression made up of events, logical operators (e.g. "AND", "OR") and parentheses, finally it is written into the Causes column like in Table 4-8 and Table 4-9.

In the second approach, causes of the critical effect "*No.Flow.Module*" are written as combinations of four incoming events (i.e. *No.Flow.Tank*, *C_.Stop_Signal.Stop*, *O_.Start_Signal.Start*, *No.Power.Busbar*) and two Groups of Events (GOE) (*No.Flow_Line_1.GOE* and *No.Flow_Line_2.GOE*) see Table 4-8. Incoming events describe the lack of incoming flow from the tank i.e. *No.Flow.Tank*, the omission or commission of start and stop signals i.e. *O_.Start_Signal.Start*, *C_.Stop_Signal.Stop* and the lack of power from the bus bar i.e. *No.Power.Busbar*. Groups of events describe the failure of *line one* (i.e. valves A1 and B1) and *line two* (i.e. valves A2 and B2). Table 4-9 shows that Causes of *No.Flow_Line_1.GOE* are component B1 generated events i.e. *No.Flow.B1*, or input events from the controller i.e. *No.Signal_B1.Ctr*. Similarly, component A1 may fail because of its generated events i.e. *No.Flow.A1* or because of the

lack of the signal from the controller i.e. *No.Signal_A1.Ctr*. In conclusion causes of *No.Flow_Line_1.GOE* can be written as:

(No.Flow.B1 OR No.Signal_B1.Ctr) OR (No.Flow.A1 OR No.Signal_A1.Ctr)

With similar reasoning it is possible to write causes for the second group of event, *No.Flow_Line_2.GOE*:

(No.Flow.B2 OR No.Signal_B2.Ctr) OR (No.Flow.A2 OR No.Signal_A2.Ctr)

These expressions are then written into the Causes column of the Group of Events table, see Table 4-9.

Redundancy of information

The hierarchy of FLASH tables may contain redundant information. For instance, incoming events for a module may appear both as causes in the table of that module and as causes in the table for the enclosing module. Once the hierarchy of tables is parsed for the fault tree construction these events are likely to originate two identical branches in the same fault tree. However, this is not a problem since cut set analysis will eliminate the duplication, additionally the algorithm for fault tree construction can be made sophisticated enough to draw fault trees avoiding repeating branches.

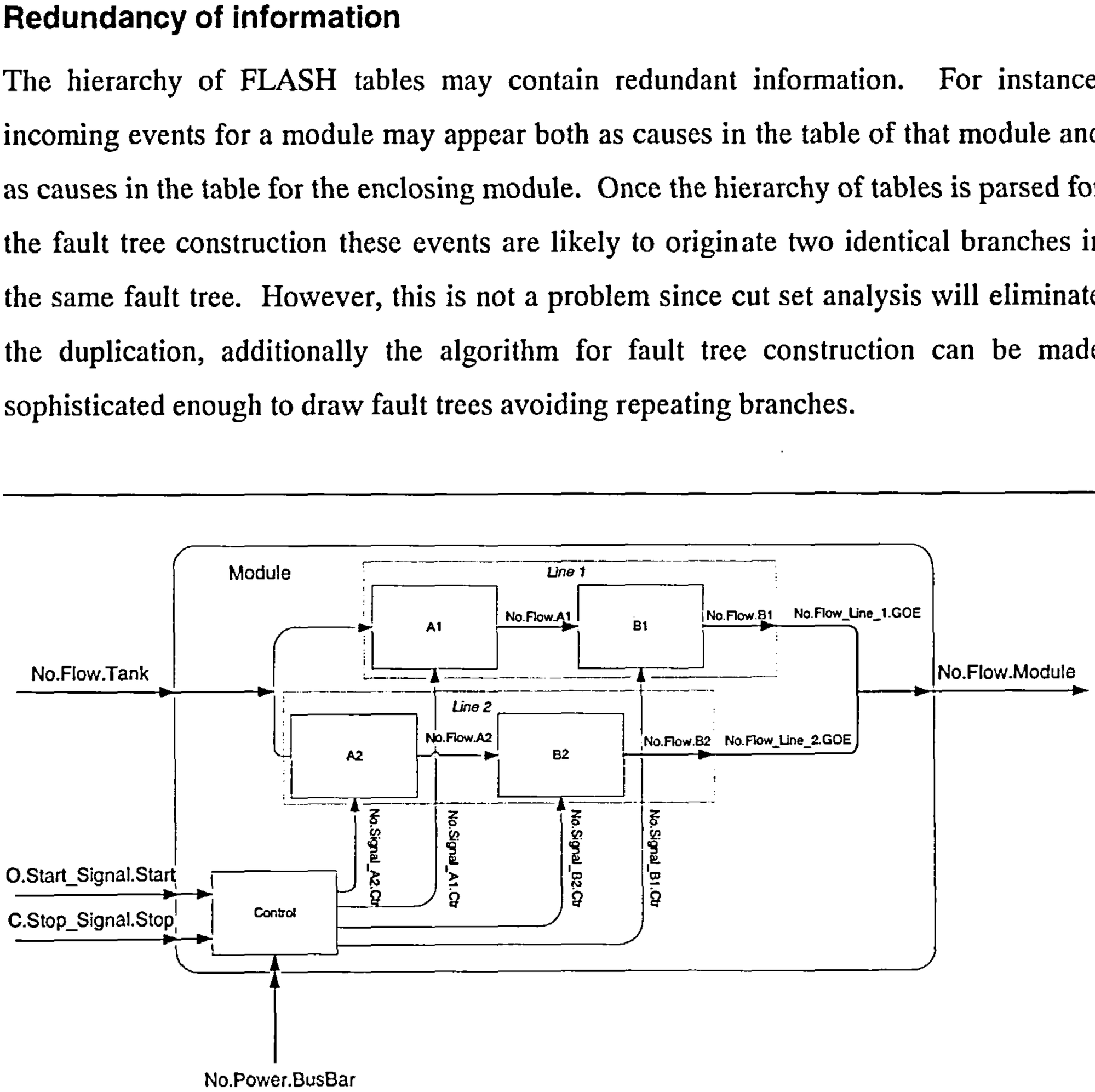


Figure 4-14: Model of fault tolerant flow controller

OUTGOING EVENTS					
Effects Events to the same level	**Causes**	**Description**	**Critica-lity**	**5th Column: Justification, Design Recommendations, Derived Safety Requirements**	**Verific a-tion (FMEA)**
No.Flow.Module	*No.Flow.Tank* *OR* *O_.Start_Signal.Start* *OR* *C_.Stop_Signal.Stop* *OR* *No.Power.Busbar* *OR* *No.Flow_Line_1.GŌE* *AND* *No.Flow_Line_2.GŌE*	No flow of fuel from the flow controller to the engine. The engine cannot start. No electric power is provided It can be caused by an omission of the start signal, a commission of the stop signal, lack of fuel from the tank or because there is no flow in the two possible paths that can be activated by the Controller	Cat.	*Before design* **Recommendations** The failure of the module cannot be handled. A fault tolerant architecture is needed to prevent that single failures in any of the valves cause a system failure The module has to be built with redundant components. **Effect max accepted likelihood** 10^{-4} on demand *After design* **Detection** A flow sensor after and external the module **Recovery** Possible for failure of one line **Recommendation** The second flow line has to be uncoupled with the first. CCF analysis is required. **Max accepted likelihood for critical events in the Causes column.** λ *(No.Flow.Tank)* $< 10^{-7}\ h^{-1}$ *P(No.Signal.Sensor)* $< 10^{-5}$ demand *P(O_.Start_Signal.Start)* $< 10^{-5}$ demand *P(C_.Stop_Signal.Start)* $< 10^{-5}$ demand λ *(No.Power.Busbar)* $< 10^{-7}\ h^{-1}$	It will be used during the verifica-tion stage

Table 4-8: Causes of the critical effects *No.Flow.Module*

Group of events	Causes	Description	Critica-lity	5th Column: Justification, Design Recommendations, Derived Safety Requirements	Verifica-tion (FMEA)
No.Flow_Line_1.GOE	*(No.flow.B1 OR No.Signal_B1.Ctr) OR (No.flow.A1 OR No.Signal_A1.Ctr)*	Line 1 is out of work, but flow may go through line 2. Action is needed to operate line 2	N/A	*Before design* **Recommendations** The failure can be handled. The system detects the failure event and replaces Line1 with Line 2 Ensure that the failure detection mechanism is reliable **Effect max accepted likelihood** The acceptable failure rate for this effect should be $\lambda<10^{-3}$ (1/h) *After design* **Detection** A flow sensor after and external the module **Recovery** Line 2 is activated upon failure of line 1 **Recommendation** The second flow line has to be uncoupled with the first. CCF analysis is required. **Max accepted likelihood for critical events in the Causes column.** *P(No.Signal_A1.Ctr)* < 10^{-5} on demand *P(No.Signal_B1.Ctr)* < 10^{-5} on demand *P(No.flow.A1)* < 10^{-5} on demand *P(No.flow.B1)* < 10^{-5} on demand	It will be used during the verifica-tion stage
No.Flow_Line_2.GOE	*(No.flow.B2 OR No.Signal_B2.Ctr) OR (No.flow.A2 OR No.Signal_A2.Ctr)*	Line 2 is out of work. Since line 2 is operated upon failure of line 1, which is already lost, then the whole system is lost.	N/A	*Before design* The failure cannot be handled. Line1 has already failed. Failure of line 2 causes the top event **Recommendations** Ensure that the failure detection mechanism is reliable **Effect max accepted likelihood** $5*10^{-5}$ on demand *After design* **Detection** Not possible **Recovery** Not possible **Recommendation** Minimise couplings with line 1, perform a common cause failure analysis **Max accepted likelihood for critical events in the Causes column.** *P(No.Signal_A2.Ctr)* < 10^{-5} on demand *P(No.Signal_B2.Ctr)* < 10^{-5} on demand *P(No.flow.A2)* < 10^{-5} on demand *P(No.flow.B2)* < 10^{-5} on demand	It will be used during the verifica-tion stage

Table 4-9: Group of Events for table in Table 4-8

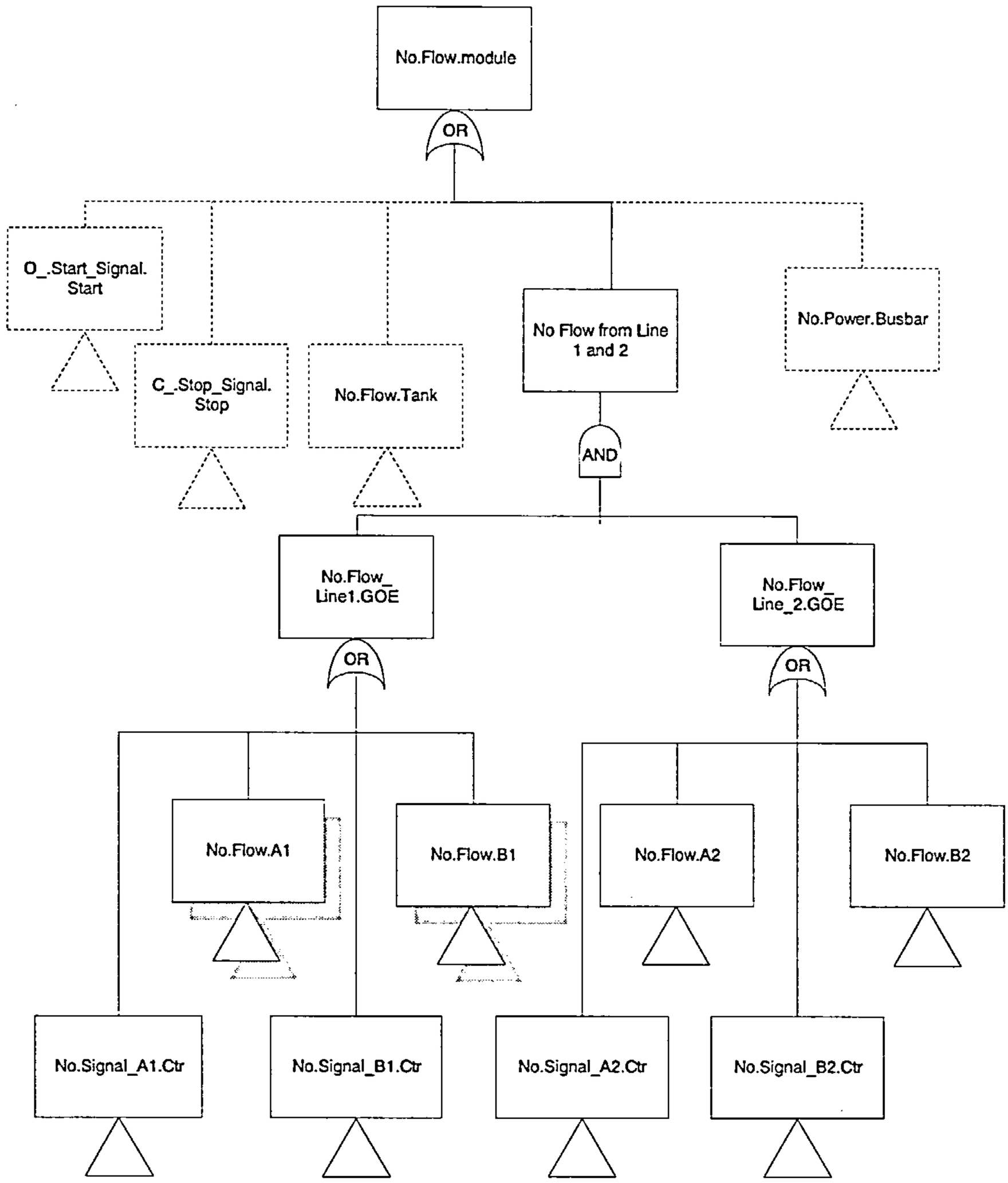


Figure 4-15: Tree for the event *No.Flow.Module* for the module in Figure 4-14

Analysis of basic components

Basic components are modules that are not further decomposed in the system hierarchy. They may represent hardware equipment, software functions or tasks. Causes of their failure modes can only be incoming events or basic events. The writing of relationships in the Causes column of FLASH tables for basic components proceeds in a similar way as for other modules. Figure 4-16 shows the failure model propagating the event *No.Flow.A1* (i.e. out of component A1) for the system in Figure 4-14. The corresponding FLASH table is Table 4-10. Causes of *No.Flow.A1* can be two primary events (i.e. *Fail to Open, Plugged*) and two incoming events (i.e. *No.Signal_A1.ctr* and *No.Flow.Tank*).

The tree built for the top event *No.Flow.A1* is in Figure 4-17. Given the simplicity of the failure model the tree comprises only one *OR* gate. As already said, the tree and the table are equivalent and the construction of the tree is not necessary for the construction of the table. Figure 4-18 shows the full model for component B1, which actually is a different instance of component A1. It has two incoming flows (i.e. *Flow.A1* and *Signal_B1.Ctr*) and one outgoing flow (i.e. *Flow.B1*). The outgoing event area in Table 4-11 considers all the events that can potentially be propagated by the module and their relations with Incoming and Generated events. Reliability data for generated events (taken from [OREDA, 1984]) are recorded into the Basic Events area. GOE are not used for the analysis of this component since relations between effects and causes do not need to be further simplified.

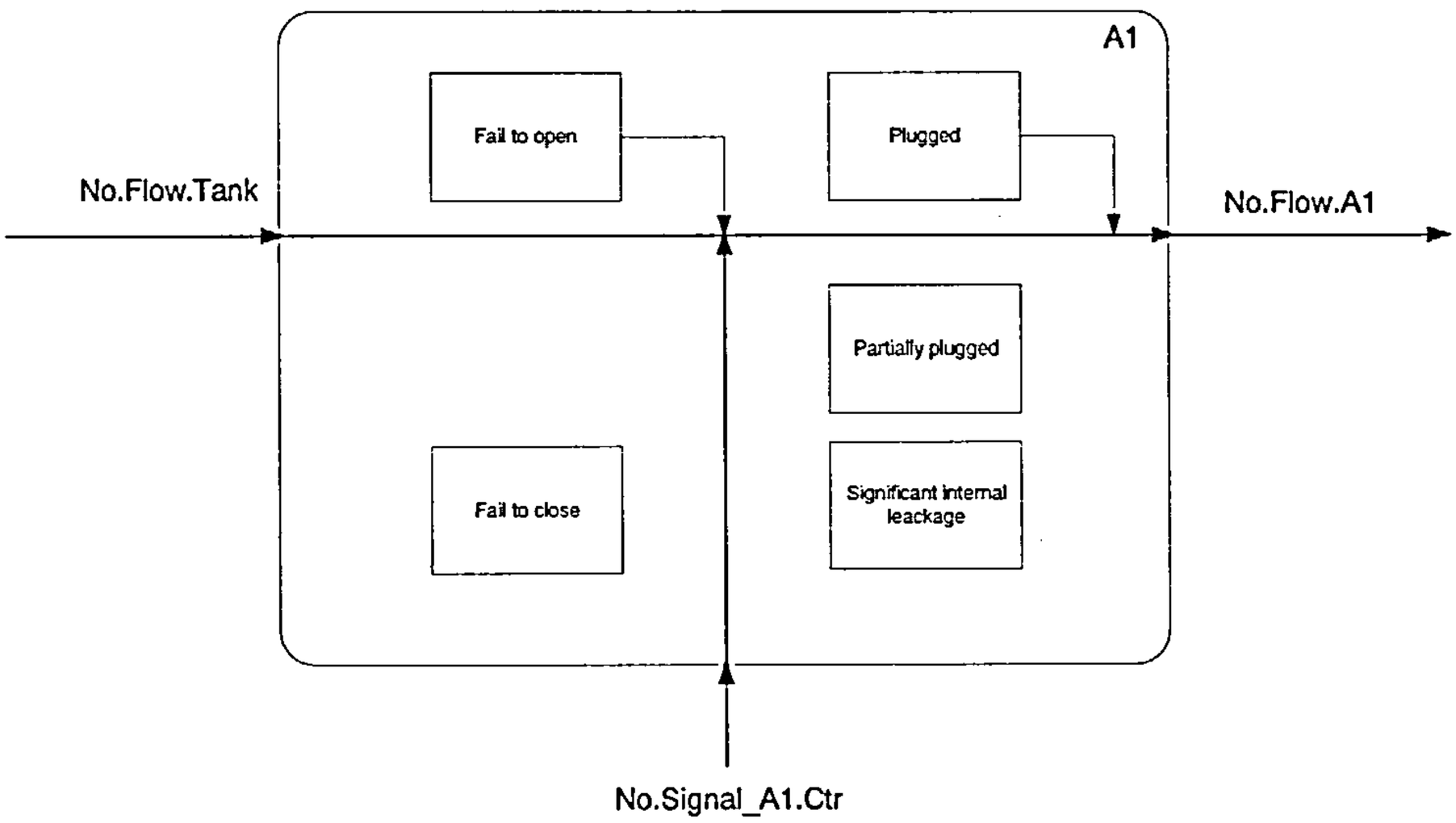


Figure 4-16: Failure model for component "A1"[16]

[16] For the completeness of the drawing all basic events of the component A are shown. However *Fail to Close, Significant Internal Leakage and Partially Plugged* are not causes of the effect *No.Flow.A1,* they contribute to other effects propagated by the module that are not analysed here.

OUTGOING EVENTS					
Effects Events to the same level	**Causes**	**Description**	**Critica-lity**	**5th Column: Justification, Design Recommendations, Derived Safety Requirements**	**Verification (FMEA results)**
No. Flow.A1	*No.Flow.Tank* *OR* *No.Signal_A1.Ctr* *OR* *Fail to Open.A1* *OR* *Plugged.A1*	The output of component A1 deviates from the design intention. There are no effects on the system if the failure is detected and recovered Line 1 is out of work, but flow can go through line 2. Action is needed to open line2	N/A	*Before design* **Recommendations** The failure have to be handled by detecting the failure event and activating Line 2 **Effect max accepted likelihood** 10^{-4} on demand *After design* **Detection** A flow sensor after and external the module **Recovery** Possible switching to line 2 **Recommendation** The line2 has to be uncoupled with the line 1 Ensure that the failure detection mechanism is reliable Analyse the error detection mechanism for potential failure modes **Max accepted likelihood for critical events in the Causes column.** Failure rate for *Fail to Open.A1* should be $< 10^{-3}\ h^{-1}$ Failure rate for *Plugged.A1* should be $< 10^{-3}\ h^{-1}$	It will be used during the verification stage

Table 4-10: FLASH table for the model in Figure 4-16

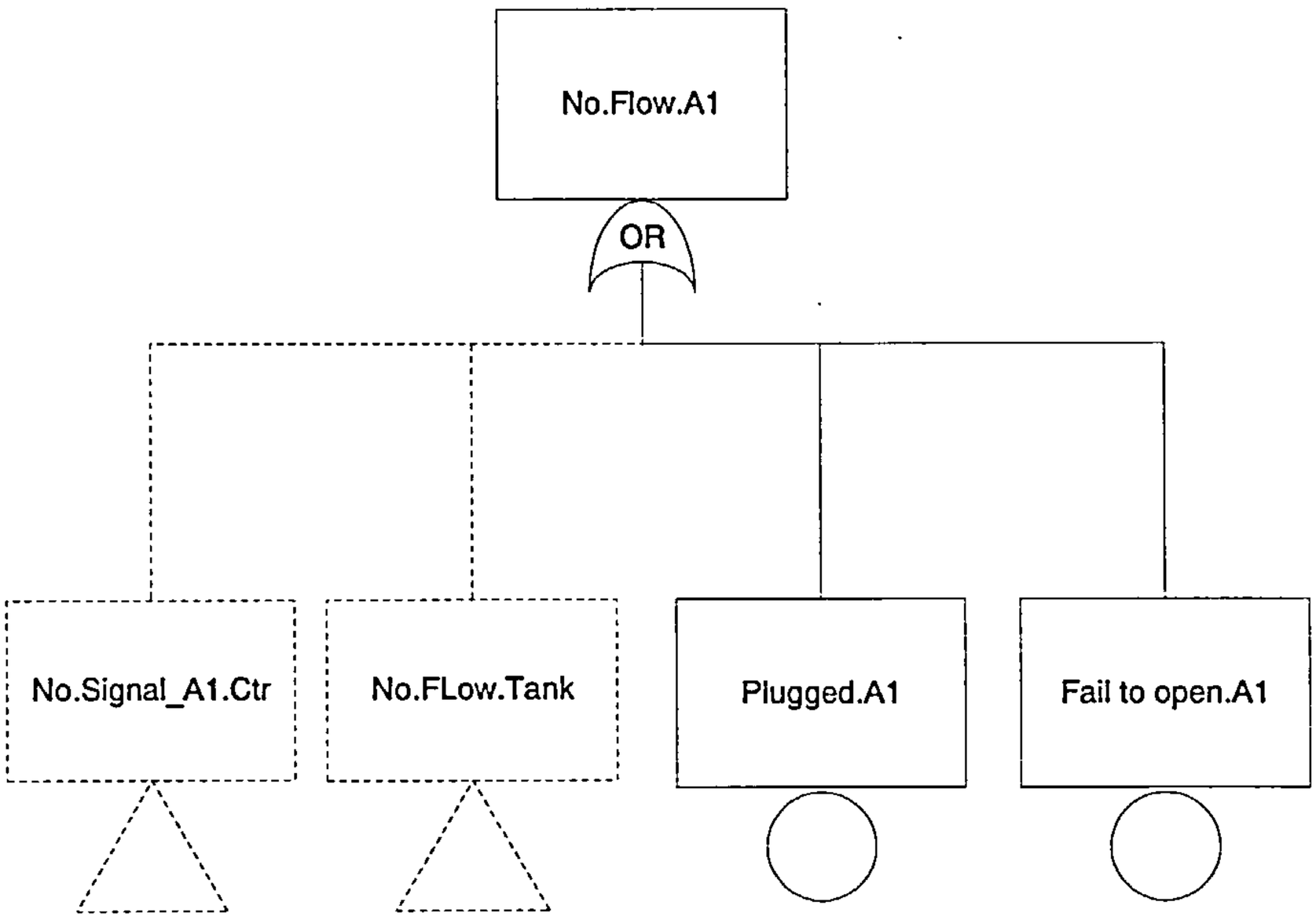


Figure 4-17: This tree for the effect *No.Flow.A1* in Figure 4-11

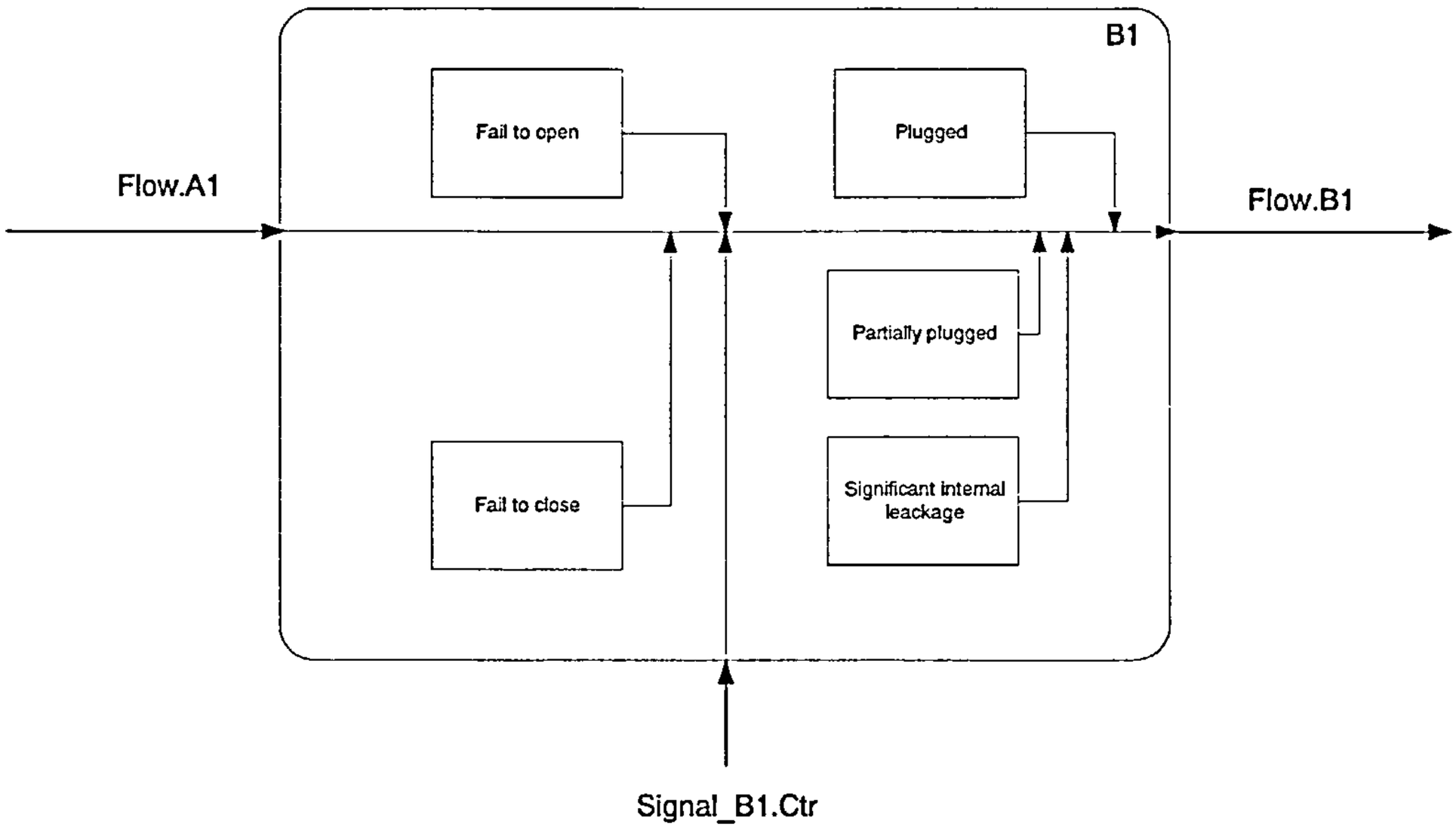


Figure 4-18: Model for Component B1

Analysis of sub-modules

The analysis of sub-modules proceeds in a similar way as the analysis of modules. Here we will see the analysis of the programmable electronic module *Control* in Figure 4-14. This sub-module has both software and hardware components. Events propagated can be caused either by incoming events, generated events that are further developed into lower level modules (i.e. Input, Output, Hardware and Software modules) and combinations of those. In fact this module has fault tolerant capabilities, hence the ability to recover from some single hardware or software failures. However, the recovery action may fail, either because of random failures or couplings, and cause failure events still to be propagated. The simultaneous occurrence of failure events with the redundant failure of the recovery action is represented by *AND* gates in the Causes column of the FLASH table. Figure 4-20 and Table 4-12 report the tree and the table for the effect *No.Signal_B1.Ctr* that is propagated by the Controller in Figure 4-19, the failure of any recovery function is represented by *AND* gates.

OUTGOING EVENTS					
Effects Events to the same level	**Causes**	**Effects & Consequences description**	**Criticality**	**5th Column: Justification, Design Recommendations, Derived Safety Requirements**	**Verification (FMEA results)**
No.Flow.B1	*No.Flow.A1* *OR* *No.Signal_B1.Ctr* *OR* *Fail to Open.B1* *OR* *Plugged.B1*	The output of component B1 deviates from the expected behaviour. There are no effects on the system if the failure is detected and recovered Line 1 is out of work, but flow can go through line 2. Action is needed to open line2	N/A	*Before design* **Recommendations** The failure have to be handled by detecting the failure event and activating Line 2 **Effect max accepted likelihood** 10^{-4} on demand *After design* **Detection** A flow sensor after and external the module **Recovery** Possible switching to line 2 **Recommendation** Line2 has to be uncoupled with line 1, CCF analysis is needed. **Max accepted likelihood for critical events in the Causes column.** λ (*Fail to Open.B1*)< 10^{-4} h^{-1} λ (*Plugged.B1*) < 10^{-4} h^{-1}	It will be used during the verification stage
More.Flow.B1	*More.Flow.A1* *AND* *(C_.Signal_B1.Ctr* *OR* *Fail to Close.B1* *OR* *Significant Internal Leakage.B1)*	The output of component B1 deviates from the design intention. There are no effects on the system if the failure is detected and recovered Too much flow is delivered by the B1. A1 has already failed. The system is lost	N/A	*Before design* **Recommendations** The failure cannot be handled since A1 has already failed **Effect max accepted likelihood** 10^{-4} on demand *After design* **Detection** A flow sensor after B1 **Recovery** Not possible **Recommendation** Valve A1 has to be uncoupled with Valve B1, CCF analysis is needed. **Max accepted likelihood for critical events in the Causes column.** λ (*Significant Internal Leakage.B1*)<$10^{-3}$$h^{-1}$ λ (*Fail to Close.B1*) <$10^{-3}$$h^{-1}$	It will be used during the verification stage
Less.Flow.B1	*Less.Flow.A1* *OR* *Partially Plugged.B1*	The output of component B1 deviates from the design intention. There are no effects on the system if the failure is detected and recovered Less flow than required is delivered by the component. Line 2 is likely to compensate. Action is needed to operate line2	N/A	*Before design* **Recommendations** The failure have to be handled by detecting the failure event, closing line 1 and activating Line 2 **Effect max accepted likelihood** 10^{-4} on demand *After design* **Detection** A flow sensor after and external valve B1 **Recovery** Possible switching to line 2 **Recommendation** Line2 has to be uncoupled with the line 1, CCF analysis is needed. **Max accepted likelihood for critical events in the Causes column.** λ (*Partially_Plugged.B1*) < 10^{-4} h^{-1}	It will be used during the verification stage
Normal.Flow.B1	*Normal.Flow.A1* *AND* *Normal.Signal_B1.Ctr*	Line 1 is working fine	N/A	Reliability must be > 0.999998	It will be used during the verification stage
Tagged.Flow.B1	*Normal.Flow.A1* *AND* *Tagged.Signal_B1.Ctr*	A recovery action took place in the controller. The module is still working, however the system will be lost for any additional failure	N/A	*Before design* **Recommendations** The system has fault tolerant capabilities, hence it is likely it is working though some failures has occurred **Effect max accepted likelihood** λ (*Partially_Plugged.B1*) < 10^{-3} h^{-1} *After design* **Detection** Already detected **Recovery** Already done **Recommendation** Correct the problem as soon as possible and not later than two hours after detection. **Max accepted likelihood for critical events in the Causes column.** *N/A*	It will be used during the verification stage

Basic Events

Reliability data	*Fail to Open.B1*	*Plugged.B1*	*Significant Internal Leakage.B1*	*Fail to Close.B1*	*Partially Plugged.B1*
Failure Rate λ[1/h]	1e-5	1e-5	1e-6	1e-5	1e-5
Repair Rate μ[1/h]	.25	.25	.25	.25	.25
Mean Time to Failure MTTF [h]	...	...	...	...	...
Mission time (of the system)[h]	8740	8740	8740	8740	8740

Table 4-11: FLASH table for B1

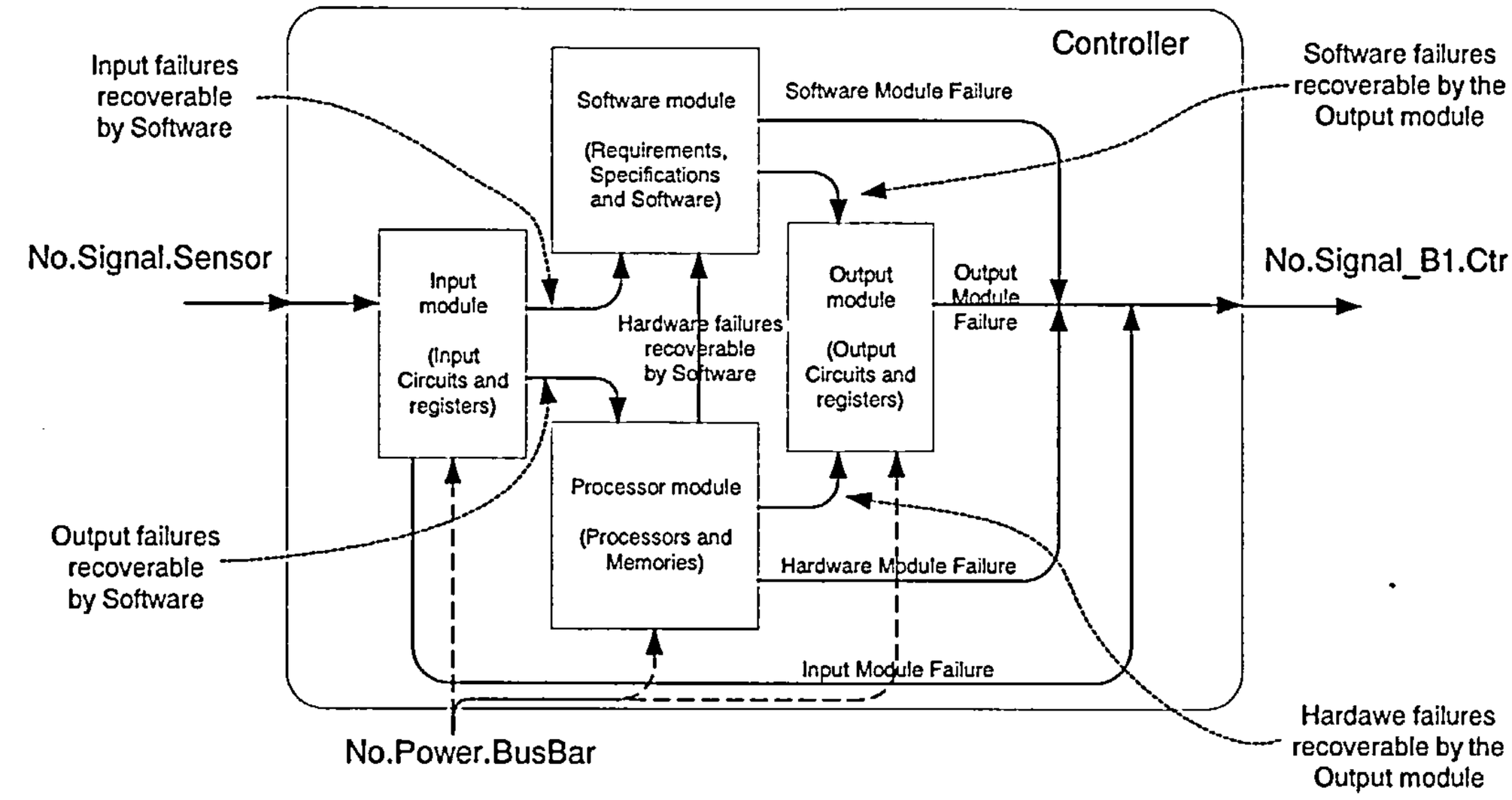


Figure 4-19: Controller with included modules

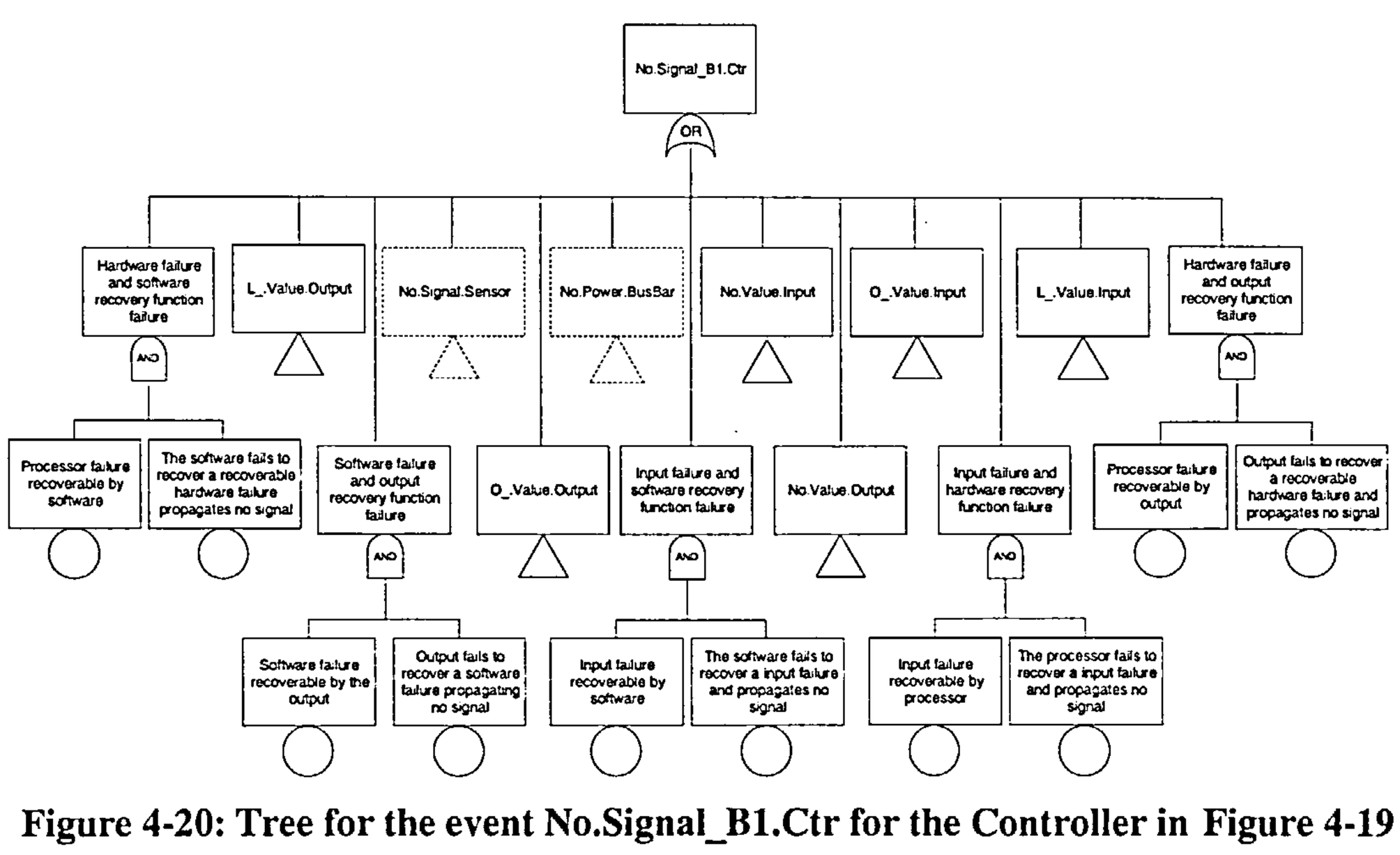


Figure 4-20: Tree for the event No.Signal_B1.Ctr for the Controller in Figure 4-19

OUTGOING EVENTS					
Events to the same level (Effects)	**Causes**	**Description**	**Criticality**	**5th Column: Justification, Design Recommendations, Derived Safety Requirements**	**Verification (FMEA results)**
No.Signal_B1.Ctr	*L.Value.Output OR No.Signal.Sensor OR O.Value.Output OR No.Power.Busbar OR No.Value.Input OR No.Value.Output OR O.Value.Input OR L.Value.Input OR Hardware_failure_and_Software_rec_funct_failure.GOE OR Software_failure_and_Hardware_rec_funct_failure.GOE OR Input_failure_and_Software_rec_funct_failure.GOE OR Input_failure_and_Software_rec_funct_failure.GOE OR Hardware_Failure_and_Output_rec_funct_failure.GOE*	Line 1 is out of work, but flow can go through line 2. The control should activate line2	N/A	***Before design*** **Recommendations** The failure cannot be handled. It has to be extremely unlikely **Effect max accepted likelihood** 10^{-5} on demand ***After design*** Some fault tolerance has been achieved, however some single failures of the Output and Input module cannot be recovered. **Detection** Not possible **Recovery** Not possible **Recommendation** Software must be developed to comply with safety integrity level four **Max accepted likelihood for critical events in the Causes column.** λ (*L.Value.Output*) $< 10^{-4}\ h^{-1}$ λ (*No.Signal.Sensor*) $< 10^{-4}\ h^{-1}$ λ (*O.Value.Output*) $< 10^{-4}\ h^{-1}$ λ (*No.Power.Bus bar*) $< 10^{-4}\ h^{-1}$ λ (*No.Value.Input*) $< 10^{-4}\ h^{-1}$ λ (*No.Value.Input*) $< 10^{-4}\ h^{-1}$ λ (*No.Value.Output*) $< 10^{-4}\ h^{-1}$ λ (*L.Value.Input*) $< 10^{-4}\ h^{-1}$	It will be used during the verification stage
Group of events	**Causes**	**Description**	**Criticality**	**5th Column: Justification, Design Recommendations, Derived Safety Requirements**	**Verification (FMEA results)**
Hardware_failure_and_Software_rec_funct_failure.GOE	*Processor failure recoverable by software AND Software fails to recover a recoverable Hardware failure and propagates no signal*	*Software fails to recover a recoverable hardware failure and propagates no signal*	N/A	***Before design*** **Recommendations** The failure cannot be handled. It has to be extremely unlikely **Effect max accepted likelihood** 10^{-7} on demand ***After design*** **Detection** Not possible **Recovery** Not possible **Recommendation** Ensure that evens in the causes column are uncoupled. Make a Co mmon Cause failure analysis **Max accepted likelihood for critical events in the Causes column.** 10^{-4} on demand for each of them	It will be used during the verification stage
Software_failure_and_Output_rec_funct_failure.GOE	*Software failure recoverable by the output AND Output fails to recover a recoverable software failure and propagates no signal*	*Output fails to recover a recoverable software failure and propagates no signal*	N/A	***Before design*** **Recommendations** The failure cannot be handled. It has to be ex tremely unlikely **Effect max accepted likelihood** 10^{-7} on demand ***After design*** **Detection** Not possible **Recovery** Not possible **Recommendation** Ensure that evens in the causes column are uncoupled. Make a Common Cause failure analysis. Software must be developed to comply with safety integrity level four. **Max accepted likelihood for critical events in the Causes column.** 10^{-4} on demand for each of them	It will be used during the verification stage
Input_failure_and_Software_rec_funct_failure.GOE	*Input failure recoverable by software AND Software fails to recover a recoverable input failure and propagates no signal*	*Software fails to recover a recoverable hardware failure and propagates no signal*	N/A	***Before design*** **Recommendations** The failure cannot be handled. It has to be extremely unlikely **Effect max accepted likelihood** 10^{-7} on demand ***After design*** **Detection** Not possible **Recovery** Not possible **Recommendation** Ensure that evens in the causes column are uncoupled. Make a Common Cause failure analysis. Software must be developed to comply with safety integrity level four. **Max accepted likelihood for critical events in the Causes column.** 10^{-4} on demand for each of them	It will be used during the verification stage
Input_failure_and_Hardware_rec_funct_failure.GOE	*Input failure recoverable by processor AND The processor fails to recover a input failure and propagates no signal*	*The processor fails to recover a input failure and propagates no signal*	N/A	***Before design*** **Recommendations** The failure cannot be handled. It has t o be extremely unlikely **Effect max accepted likelihood** 10^{-7} on demand ***After design*** **Detection** Not possible **Recovery** Not possible **Recommendation** Ensure that evens in the causes column are uncoupled. Make a Common Cause failure analysis. Software must be developed to comply with safety integrity level four. **Max accepted likelihood for critical events in the Causes column.** 10^{-4} on demand for each of them	It will be used during the verification stage
Hardware_Failure_and_Output_rec_funct_failure.GOE	*Processor failure recoverable by output AND Output fails to recover a processor failure and propagates no signal*	*Output fails to recover a processor failure and propagates no signal*	N/A	***Before design*** **Recommendations** The failure cannot be handled. It has to be extr emely unlikely **Effect max accepted likelihood** 10^{-7} on demand ***After design*** **Detection** Not possible **Recovery** Not possible **Recommendation** Ensure that evens in the causes column are uncoupled. Make a Common Cause failure analysis. Software must be developed to comply with safety integrity level four. **Max accepted likelihood for critical events in the Causes column.** 10^{-4} on demand for each of them	It will be used during the verification stage

Continue in the next page ...

Basic Events								
Reliability data	*Processor failure recoverable by the software*	Software fails to recover a recoverable hardware failure	*Software failure recoverable by the output*	*Output fails to recover a recoverable hardware failure and propagates no signal*	*Input failure recovera-ble by software*	*Software fails to recover a recoverable hardware failure and propagates no signal*	...
Failure Rate λ[1/h]	10E-5	...	...	10E-5	10E-5	...	...
Repair Rate μ[1/h]	...	...	...	...	...	...	...
Mean Time to Failure MTTF [h]	...	...	...	...	...	...	...
Mission time [h]	8740	...	...	8740	8740	...	...

Table 4-12: Piece of the FLASH table for the effect *No.Signal_B1.Ctr*

4.3.2 Integration and Verification

The aim of the FLASH analysis in the integration and verification phase is to confirm that the system with its real components meets requirements, specifications and recommendations produced during the decomposition and design. This verification cannot be done earlier, since the detailed information about components is not availab le until the end of the design process, which is when basic components are chosen. After this stage, we have the most detailed knowledge about the system and all the information we need to assess how good the system will be with real components. During the verification process, each component, sub-module and module in the hierarchy is individually verified to confirm that they meet requirements, specifications and recommendations. The process starts from basic components and proceeds towards higher levels of integration finishing at the top level. All the tables are considered and the likelihood of propagated events is evaluated by using fault tree analysis. Probabilities of these events (which are top events in fault trees) are recorded in the "*FMEA results*" column. Additionally, in this column evidence is given to show that the requirements and constraints defined during the decomposition and design are met. If some of the recommendations are not met they are reviewed or the system design enhanced. In the latter case, the architecture of the module that does not meet recommendations is modified. In some instances it may be sufficient to replace only one component, in others, a whole module may have to be re-engineered. Following those modifications, FLASH analysis has to be re-run for all new components and all the ones interfacing with them. Figure 4-21 shows the entire process of verification with the feedback given to the decomposition and design. In this figure n_{max} represents the number of levels in the hierarchy, n the current level, m the current module under analysis and E the effect for which the tree is built.

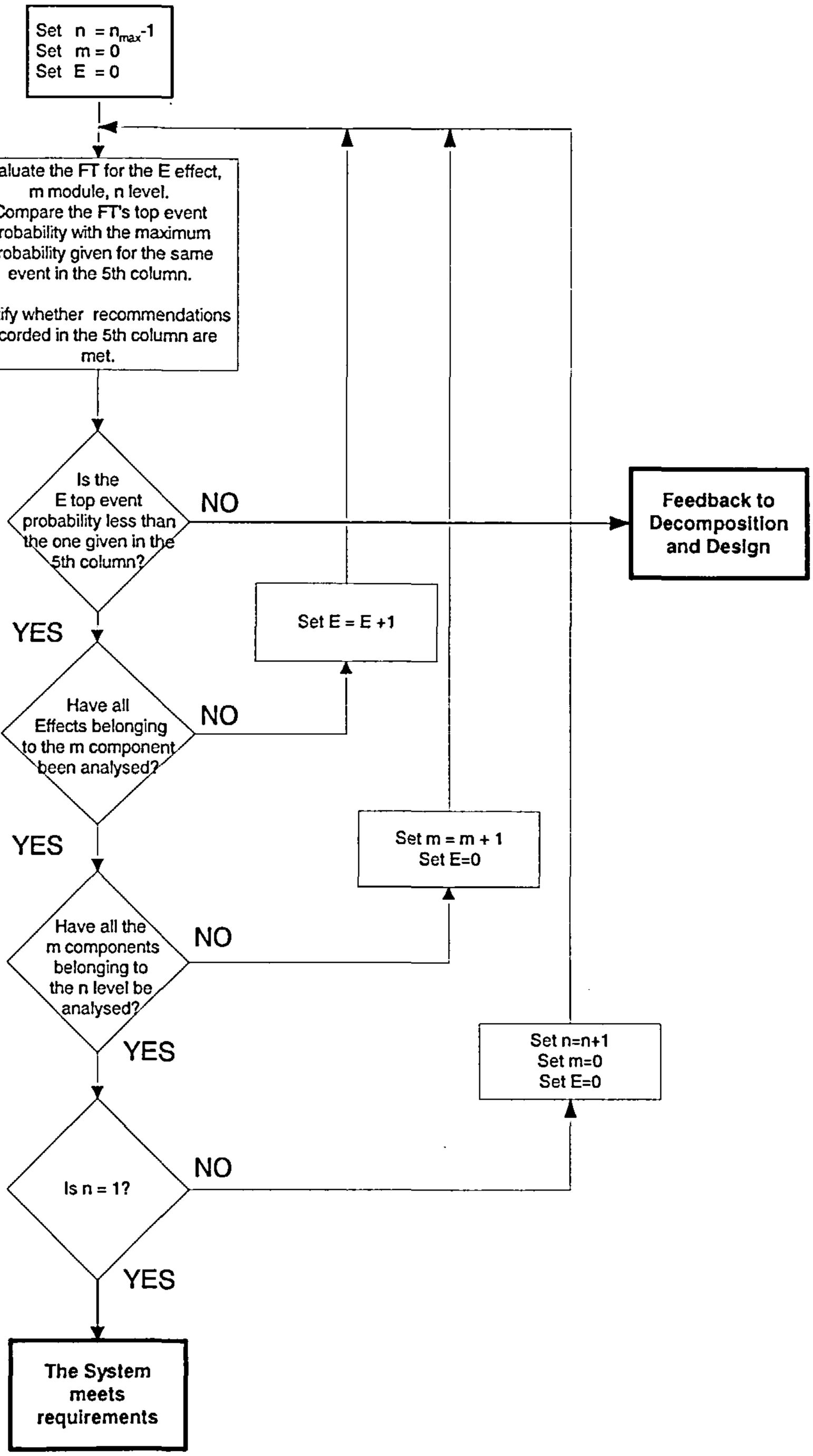


Figure 4-21: Feedback to decomposition and design

Trees for effects are constructed by parsing the hierarchy of tables. These trees link in a consistent manner results from the functional level analysis to low level FMEAs. The process of fault tree synthesis is mechanical. It is a simple parsing of tables, from the

current level down to the bottom. Figure 4-22 shows the tree drawn from the top event *No.Flow.Module* that is obtained by parsing tables for *Module*, *A1*, *B1* and *Controller*. This tree can also be obtained linking trees represented in Figure 4-15, Figure 4-17, Figure 4-20. Intermediate events *No.Signal_A2.Ctr*, *No.Signal_B2.Ctr*, *No.Flow.A2* and *No.Flow.B2* are not developed. Dashed branches represent incoming events of the *Module*. Table 4-13 is the FLASH table for the *Module* as it should be after the verification (i.e. the FMEA results column is completed).

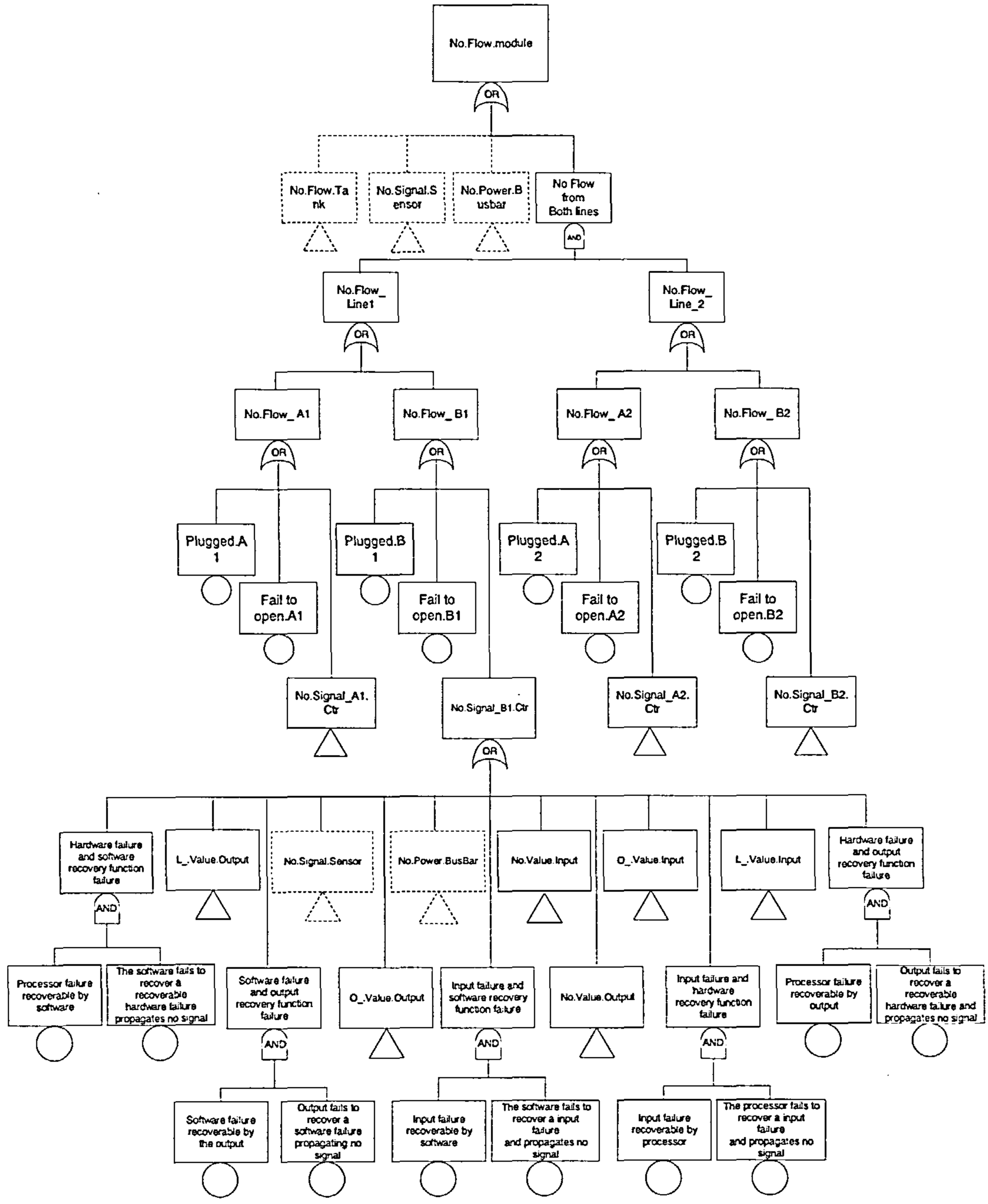


Figure 4-22: Tree for the top event *No.Flow.Module*

OUTGOING EVENTS					
Events to the same level (Effects)	**Causes**	**Description**	**Criticality**	**5th Column: Justification, Design Recommendations, Derived Safety Requirements**	**Verification (FMEA results)**
No.Flow.Module	*O_.Start_Signal.Start* *OR* *C_.Stop_Signal.Stop* *OR* *No.Flow.Tank* *OR* *No.Power.Busbar* *OR* *No.Flow_Line_1.GOE* *AND* *No.Flow_Line_2.GOE*	No flow of fuel from the flow controller to the engine. The engine cannot start. No electric power is provided It can be caused by an omission of the start signal, a commission of the stop signal, lack of fuel from the tank or because there is no flow in the two possible paths that can be activated by the Controller	Catastrophic	*Before design* **Recommendations** The failure of the module cannot be handled. A fault tolerant architecture is needed to prevent that single failures in any of the valves cause a system failure The module has to be built with redundant components. **Effect max accepted likelihood** 10^{-4} on demand --- *After design* **Detection** A flow sensor after and external the module **Recovery** Possible for failure of one line **Recommendation** The second flow line has to be uncoupled with the first CCF analysis is required **Max accepted likelihood for critical events in the Causes column.** λ *(No.Flow.Tank)* $< 10^{-7}\ h^{-1}$ *P (No.Signal.Sensor)* $< 10^{-5}$demand (during the mission) *P (O_.Start_Signal.Start)* $< 10^{-5}$demand (during the mission) *P (C_.Stop_Signal.Start)* $< 10^{-5}$demand (during the mission) λ *(No.Power.Busbar)* $< 10^{-7}\ h^{-1}$	**Likelihood of the event propagated** $5*10^{-5}$ on demand **Likelihood of critical causes** *P(No.Flow.Tank)* $= 5*10^{-4}$ *P(No.Signal.Sensor* $= 4*10^{-4}$ *P(No.Power.Busbar)*$= 10^{-4}$ **Justification** The module is built with redundant components. No single points to failure for mechanical components are present. At least two valves must fail to cause the event. Recovery is possible for single mechanical failure. Flow lines are sufficiently uncoupled to meet requirements. Programmable Logic Controller and software meet safety requirements for their integrity level.
Group of events	**Causes**	**Description**	**Criticality**	**5th Column: Justification, Design Recommendations, Derived Safety Requirements**	**Verification (FMEA results)**
No.Flow_Line_1 .GOE	*(No.flow.B1* *OR* *No.Signal_B1.Ctr)* *OR* *(No.flow.A1* *OR* *No.Signal_A1.Ctr)*	Line 1 is out of work, but flow may go through line 2. Action is needed to operate line 2	N/A	*Before design* **Recommendations** The failure can be handled. The system detects the failure event and replaces Line1 with Line 2 Ensure that the failure detection mechanism is reliable **Effect max accepted likelihood** The acceptable likelihood for this effect should be $<10^{-3}$ on demand (during the mission) --- *After design* **Detection** A flow sensor after and external the module **Recovery** Line 2 is activated upon failure of line 1 **Recommendation** The second flow line has to be uncoupled with the first. CCF analysis is required. **Max accepted likelihood for critical events in the Causes column.** *P(No.Signal_A1.Ctr)* $< 10^{-5}$on demand (during the mission) *P(No.Signal_B1.Ctr)* $< 10^{-5}$on demand (during the mission) *P(No.flow.A1)* $< 10^{-5}$ on demand (during the mission) *P(No.flow.B1)* $< 10^{-5}$ on demand (during the mission)	**Likelihood of the event propagated** $4*10^{-4}$ on demand **Likelihood of critical causes** *P(No.Signal_A1.Ctr)*$= 5*10^{-7}$ on demand *P(No.Signal_B1.Ctr)=*$5*10^{-7}$ on demand *P(No.flow.B1)* $= 5*10^{-4}$ on demand *P(No.flow.B1)* $= 5*10^{-4}$ on demand **Justification** The software activates Line 2 upon failure of line 1. Flow lines are sufficiently uncoupled to meet requirements. Couplings have been minimised. The likelihood for the event propagated include the contribution from CCF.
No.Flow_Line_2 .GOE	*(No.flow.B2* *OR* *No.Signal_B2.Ctr)* *OR* *(No.flow.A2* *OR* *No.Signal_A2.Ctr)*	Line 2 is out of work. Since line 2 is operated upon failure of line 1, which is already lost, then the whole system is lost.	N/A	*Before design* The failure cannot be handled. Line1 has already failed. Failure of line 2 causes the top event **Recommendations** Ensure that the failure detection mechanism is reliable **Effect max accepted likelihood** $5*10^{-4}$ on demand (during the mission) --- *After design* **Detection** Not possible **Recovery** Not possible **Recommendation** Minimise couplings with line 1, perform a common cause failure analysis **Max accepted likelihood for critical events in the Causes column.** *P(No.Signal_A2.Ctr)* $< 10^{-5}$on demand (during the mission) *P(No.Signal_B2.Ctr)* $< 10^{-5}$on demand (during the mission) *P(No.flow.A2)* $< 10^{-5}$ on demand (during the mission) *P(No.flow.B2)* $< 10^{-5}$ on demand (during the mission)	**Likelihood of the event propagated** $4*10^{-4}$ on demand **Likelihood of critical causes** *P(No.Signal_A2.Ctr)*$= 5*10^{-7}$ on demand *P(No.Signal_B2.Ctr)=*$5*10^{-7}$ on demand *P(No.flow.A1)* $= 5*10^{-4}$ on demand *P(No.flow.A1)* $= 5*10^{-4}$ on demand **Justification** The software activates Line 2 upon failure of line 1. Flow lines are sufficiently uncoupled to meet requirements. Couplings have been minimised. The likelihood for the event propagated include the contribution from CCF.

Table 4-13: Complete FLASH table for Module

4.4 Tool support

The FLASH method presented so far appears to be quite complex. However, it can be supported by a software tool that automates the most tedious and errors prone procedures. A software tool may help to navigate through the hierarchy of tables, generate trees, to calculate the likelihood of events propagated and to make consistency checks on the whole hierarchy. The navigation through tables is useful to trace the propagation and transformation of events from high level functional failures to low level component failure modes. The automatic fault trees generation and evaluation, allows

drawing trees for hazardous events and updating them any time the design is modified. Consistency checks are related to the FLASH hierarchy. Hierarchies, like the one produced by FLASH need to be consistent to be useful. It may happen that while writing tables for FLASH modules or modifying them, that consistency among tables in the hierarchy is lost. Hence consistency has to be checked following changes.

During our research, an existing software tool, the Safety Argument Manage (SAM), developed at the University of York [McDermid, 1994], has been adapted to support some phases of the FLASH method. At present, a new SAM module supports writing and updating of FLASH tables. In addition, it has been shown how generation of fault trees from FLASH tables is possible. Until now the automatic tree generation has not been implemented in the FLASH module of the SAM software. Here are two reasons: 1) lack of time; 2) it is believed to be possible to reuse part of existing code for the automatic fault tree generation already developed in [Papadopoulos and McDermid, 1999a] within the HiP-HOPS module of SAM.

FLASH tables are written using an editor very similar to widely used commercial table editors. When the table hierarchy is completed, the editor makes possible navigating from any table to lower or higher level tables by selecting an event and choosing to *Explore causes* or *Navigate back*. Causes of any event in the hierarchy can be traced to component failure modes. Events are chosen from the Causes column of a table. The tag of the event identifies the table that contains causes of this event. Events are sought in the Effects column of the table propagating them. Causes are in the corresponding box of the *Causes column*. Figure 4-23 displays the Outgoing area of the table for the top level of a fuel system. This table propagates three effects *No_.Fuel.fc*, *More_.Fuel.fc* and *Less_.Fuel.fc*. Causes of the event *No_.Fuel.fc* are:

O_.Fuel.bva AND O_.Fuel.bvb OR No_.PowerSupply.PS

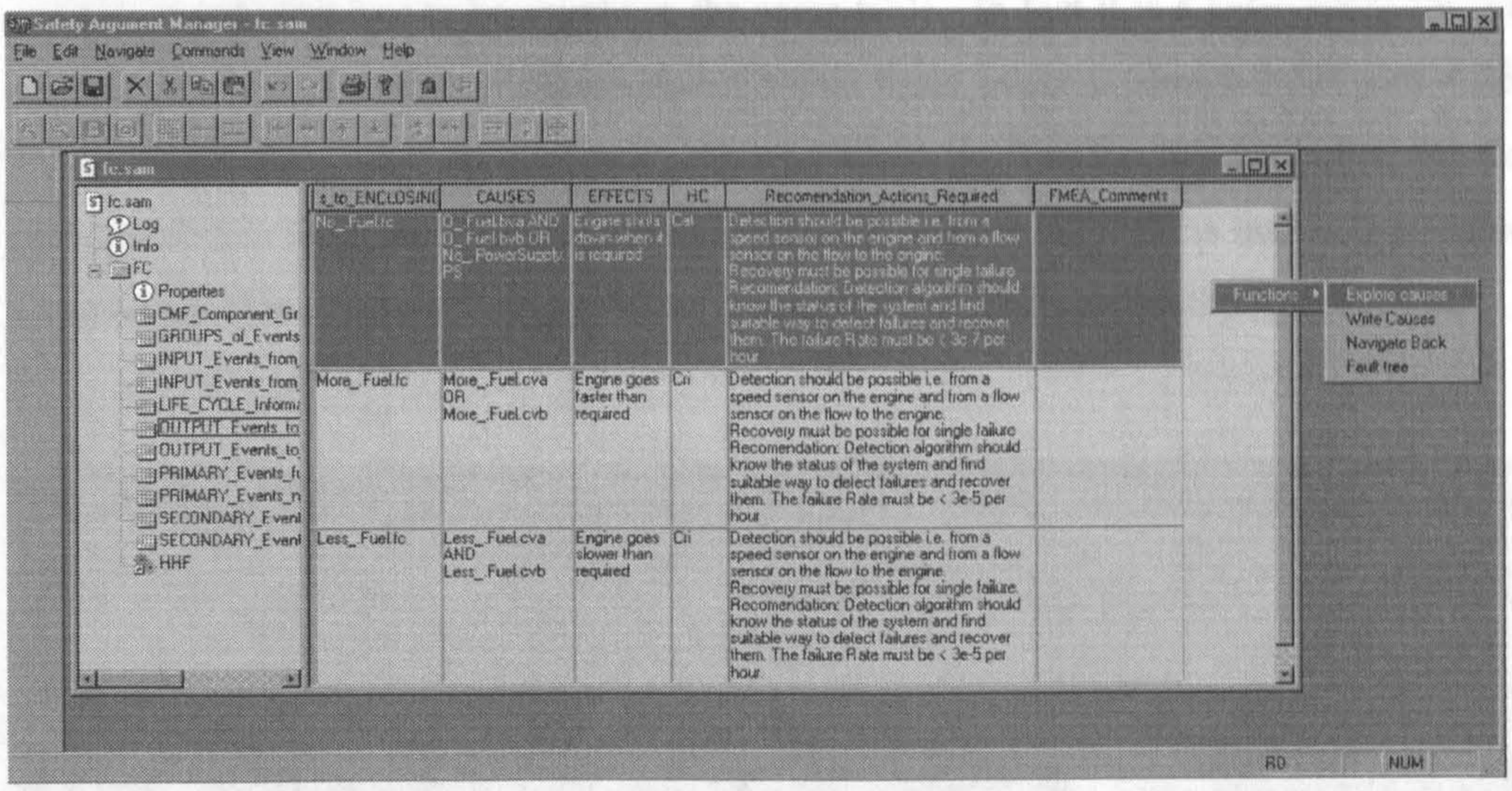


Figure 4-23: Outgoing area of the table for the top level

If the analyst requires to search for causes of the event *O_.Fuel.bva*, the software takes the *bva* table and seeks that event in output columns of this table (i.e. Output events to the "Same" or "Enclosing" level). As the Figure 4-24 shows causes of this event are the following:

Fail_to_open.bva OR O_.BVAi.ec OR Plugged.bva

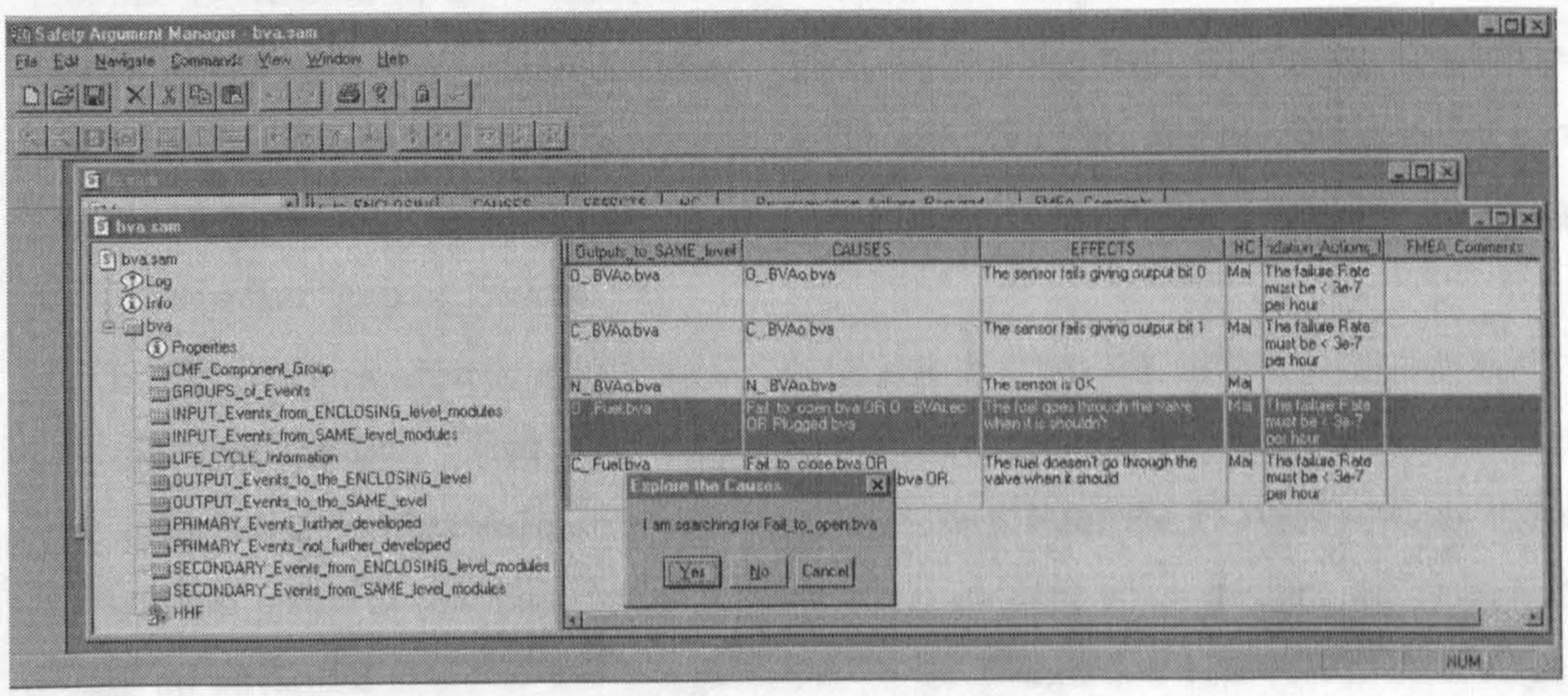


Figure 4-24: Causes of the effect O_.Fuel.bva

Once analysts have found causes of the event *O_.Fuel.bva* they may want to seek more details or the causes of the causes. For instance, they may want to investigate the event *Fail_to_open.bva*, that is one of the causes of the event *O_.Fuel.bva.* This event has the

tag *bva* therefore it has to be sought in the same table. In fact it is a basic event for the component *bva*. Reliability information for this basic event is shown in Figure 4-25. The function *Navigate back* that is highlighted at the right of the SAM window for the component *bva* leads back to the previous table i.e. Figure 4-24. This makes it possible to select another event and investigate its causes or find out its reliability information.

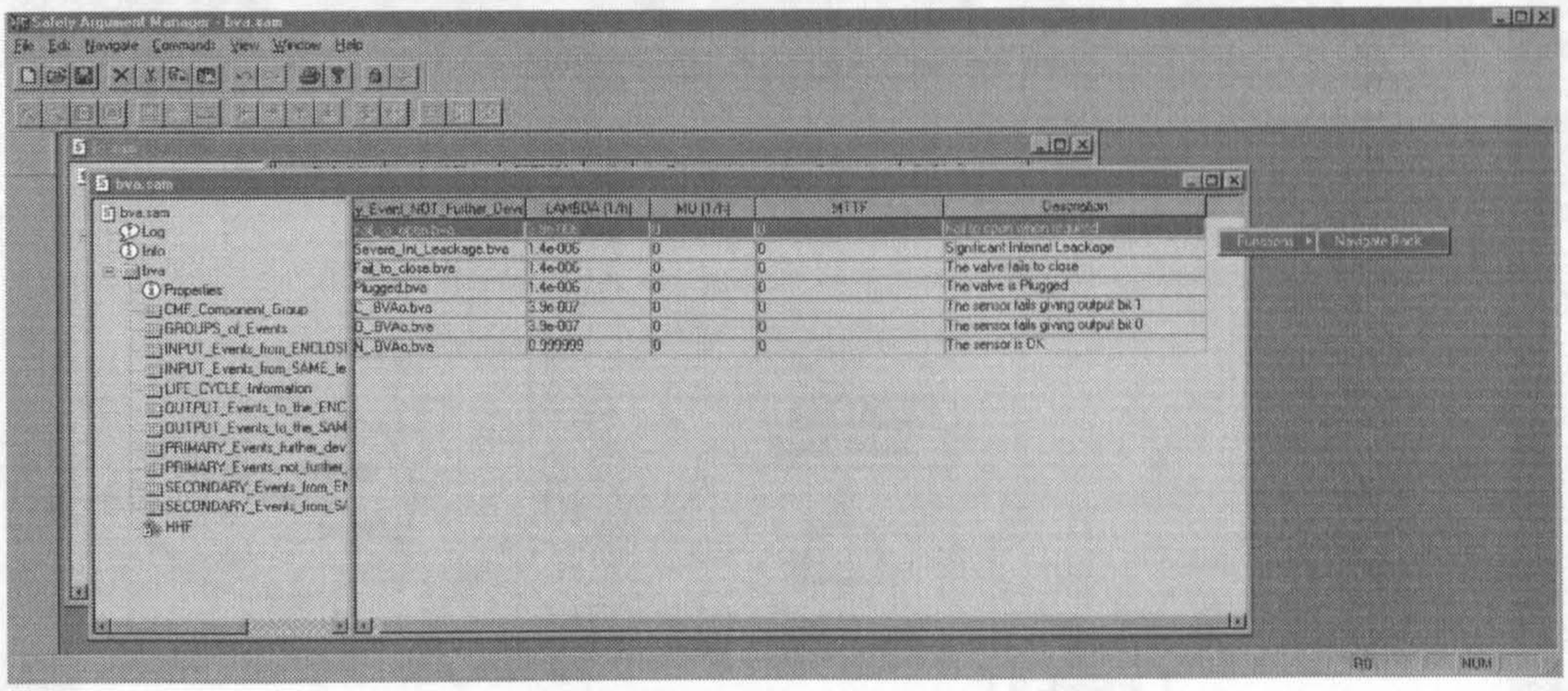


Figure 4-25: Basic events table for component bva

The FLASH module of SAM also supports the writing of expressions in the Causes column of the FLASH table by providing the analyst with the set of events and gates to form the expression in the Causes column. Once the analyst has chosen the function "write causes" and points on the Causes column, a menu appears. This menu shows the only set of incoming and generated events, *AND* and *OR* gates, and parentheses that can be used to build expressions in that table. Any expression is built only by selecting entries from that "pop-up" menu.

For the reasons already mentioned, we did not develop the software for automatic fault tree construction. That prevented us from running complex case studies. It has been discovered that manual construction of fault trees from the FLASH table is quite tedious, and prone to mistakes. This is particularly evident when designs are reviewed during an advanced stage of the lifecycle, i.e. in the integration and verification stage, and many trees are to be modified and re-evaluated. In those cases the automatic fault tree generation would be very helpful. Once an effect is selected in a FLASH table, the function *Fault Tree* should draw a tree with that event at the top. When sufficient reliability data are available also the top event likelihood should be given. Figure 4-26 displays the tree for the top event *No_.Fuel.fc* presented in previous tables as it should

appears for the event in our example. Additionally the automatic fault tree generation can potentially be used to check the consistency of the whole hierarchy. Building all the possible fault trees in the hierarchy would not be possible if there are inconsistencies in tables.

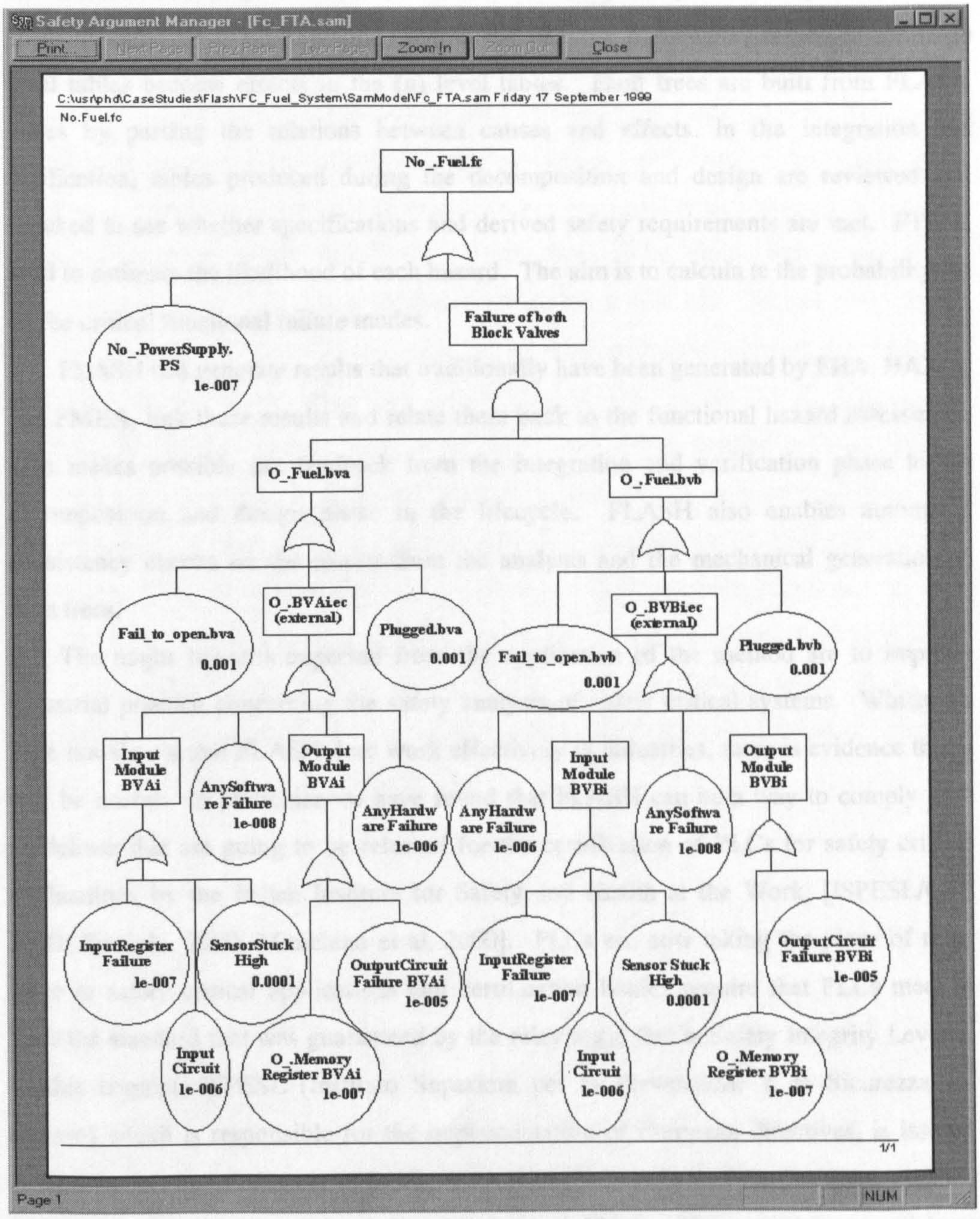


Figure 4-26: Fault tree for the top event *No_.Fuel.fc*

4.5 Discussion

This chapter presented the FLASH method, which aims to support the decomposition and design of a system and the integration and verification process. FLASH is performed in parallel to the design in a hierarchical fashion. It uses a common syntax that formalises the causal relationship that underlies traditional safety analysis techniques like FHA, HAZOP and FMEA information. Within the FLASH framework, causes in the (n-1) level tables become effects in the (n) level tables. Fault trees are built from FLASH tables by parsing the relations between causes and effects. In the integration and verification, tables produced during the decomposition and design are reviewed and checked to see whether specifications and derived safety requirements are met. FTA is used to estimate the likelihood of each hazard. The aim is to calcula te the probability for all the critical functional failure modes.

FLASH can generate results that traditionally have been generated by FHA, HAZOP and FMEA, link these results and relate them back to the functional hazard assessment. This makes possible the feedback from the integration and verification phase to the decomposition and design phase in the lifecycle. FLASH also enables automated consistency checks on the results from the analysis and the mechanical generation of fault trees.

The major benefits expected from the application of the method are to improve industrial practice concerning the safety analysis of safety critical systems. Whilst we have not shown that FLASH does work effectively in industries, there is evidence that it will be useful. In particular we have found that FLASH can be a way to comply with guidelines that are going to be released for the certification of PLCs for safety critical applications by the Italian Institute for Safety and Health at the Work, [ISPESL/CEI, 2000; Picciolo, 2000, Minichino et al. 2000]. PLCs are now taking the place of relay logic in safety critical applications and certification bodies require that PLCs meet at least the standard that was guaranteed by the relay logic that is Safety Integrity Level 2. In this context, ISPESL (Instituto Superiore per la Prevenzione E la Sicurezza sul Lavoro) which is responsible for the implementation of European directives, is issuing guidelines for the assessment of Safety Critical PLCs on behalf of the European Agency for Health and Safety and the Italian Ministry of Heath. These guidelines will be a national standard and will recommend a hierarchical decomposition and study of systems according to the SADT notation [Ross, 1985] that was considered by us prior developing

our method and is very similar to the FLASH hierarchical decomposition. In this context the FLASH method can potentially be seen as a way to meet those guidelines.

FLASH has been successfully applied to the analysis of small high integrity systems, among them a PLC and a computerised braking system, but it is a complex technique that can be heavy to apply without the assistance of suitable software to take charge of repetitive and error prone tasks. It is unclear that it will be applicable to support the design and verification of a very large system such as an aircraft, a helicopter, a chemical or nuclear installation. The design of those systems, though they appear to be hierarchically decomposable, is not usually approached hierarchically. Each subsystem is designed separately from given specifications and then they are assembled. Frequently it happens that significant changes have to be made later to put all the subsystems together. A real hierarchical top down design like the one proposed by FLASH is actually not done because it would take too much time, although it would save expensive modifications when the artefact is already in an advanced stage of construction. In this context FLASH could still be used to develop each sub-system. However we believe that if a FLASH analysis is available for each sub-systems then it may be possible to link all these FLASH analyses to produce the FLASH model for the full system. This might be possible if the technique were sufficiently highly automated.

The next chapter extends the FLASH formalism to Common Cause Failure analysis. The information about modules' lifecycle recorded into FLASH tables during the developing phases is used for a qualitative and a quantitative evaluation of common cause and the others dependent failures.

Chapter Five

Common Cause Failure

This chapter extends the FLASH formalism presented in chapter four to treat common cause failures. Here we show how the hierarchy of FLASH tables can be used to identify those minimal cut sets that need to be analysed for common cause failures. Additionally, we provide a novel method for quantitative estimation of the likelihood of minimal cut sets with coupled events that uses lifecycle information recorded in FLASH tables.

5.1 Overview

Common cause failures were extensively introduced in the second chapter. They are a particular kind of failure that occur within redundant devices and endanger fault tolerant systems by causing their redundant channels to fail at the same time or in a short time interval. They act like a single point of failure for these systems. If it were possible to have fault tolerant systems with uncoupled channels, there would be no need to investigate common cause failures in these systems since no single cause could give rise to system failure. However it is practically impossible to construct, maintain and operate completely independent redundant systems so there is always the need for common cause failure analysis in fault tolerant systems.

The easiest way to consider common cause failures is to study minimal cut sets of fault trees drawn for critical events of the fault tolerant system. Minimal cut sets exhaustively represent *all* the combinations of failures that, when occurring simultaneously, cause the system failure. In the case of common cause failures, it happens that the *root cause*[17] through the coupling factor causes all the events in the minimal cut set to occur within a very short time span. Consequently the fault tolerant system fails as if all the events in the minimal cut set had arisen randomly.

Typically, the likelihood of a minimal cut set occurring because of common cause failures is extremely small, at least one or two orders smaller than the smallest likelihood of events in the minimal cut set. However, it is always greater than the likelihood of the

---

[17] See chapter 2

whole minimal cut set if the events occur randomly, consequently it is the most important contribution to the total likelihood of the minimal cut set. Hence analysts have to consider common cause failures any time they want to use redundancies to pursue failure rates smaller than those of any component employed in the redundant configuration.

One purpose of common cause failure analysis is to evaluate the actual likelihood of minimal cut sets with coupled events. Without considering common cause failures, fault trees for fault tolerant systems underestimate, often by many orders, the likelihood of the top event.

FLASH supports the studies of common cause failures after fault trees have been drawn for critical failures and minimal cut sets obtained[18]. The process consists, first, in the identification of minimal cut sets that have to be analysed for common cause failures (i.e. with coupled events), then in the estimation of the likelihood of these minimal cut sets.

5.2 Identification of MCS with coupled events

The identification of minimal cut sets with coupled events is the first step for considering common cause failures in the FLASH method. In the second chapter we saw that common cause failures arise when there are couplings among redundant components. These couplings may be generated anywhere in the lifecycle of components making up redundant channels. Coupling may be the same person producing the design, the way components are manufactured, installed, tested, maintained etc. To formalise the identification of couplings, many checklists have been proposed, two of these are in [Mosleh et al., 1993] and [SAE-ARP 4754, 1996]. However, these are very general so we have developed a new checklist (reported in Table 5-1) to address software components. These checklists aim to be a reference for designers who try to construct fault tolerant systems with coupling-free redundant channels, but also to help safety analysts to unveil hidden couplings overseen by designers.

In addition to these two uses, we believe, checklists can be employed for an additional purpose, which is to collect information about potential couplings that may occur during the lifetime of components. The idea is that, in correspondence with each heading of a checklist, we can record information specific to each component, e.g. a code that identifies potential couplings. For example, in correspondence with the entry

---

[18] It is not intended in this chapter to show how minimal cut set can be obtained by reducing a Fault Tree. That can be found in [Vesely, 1981].

component manufacturer, the code may identify the company producing the component; in correspondence with the entry *component procedure*, it may identify the procedure adopted for manufacturing; and so on, for all the entries in the list. In this way we have unequivocally identified all[19] the potential couplings in which a component may be affected.

Development (Process)	• Requirements • Requirements team • Specifications • Specifications team • Implementation strategy • Implementation team • Design strategy • Design team
Test (Process)	• Criteria • Objectives • Requirement test specification • Integration test specification • Unit test specification • Test team
Tools (for Development and Test)	• Compiler • Link/Loaders • Code Generator • Design & Requirements Tools • Operating System • Test Stubs and Drivers • Test Monitoring • Test Management
Installation Procedure	• Production of PROMS • Loading a FLASH memory
Operating Environment	• Operating System • Device Drivers

Table 5-1: Checklist of potential couplings in Generic Software Modules

The list[20] of potential couplings is, then, inherited by basic events originating within the component. This makes it feasible to *compare basic events* on the same ground. For example, if two apparently different valves, a stop and a control valve, produced by two

---

[19] We aim to identify all the couplings of the component by using a list of attributes that spans the whole lifecycle and that is as exhaustive as practical.

[20] The list of potential couplings is what, in chapter 4, was called lifecycle information, or lists of lifecycle categories.

different manufacturers, are maintained by the same person[21], basic events[22] for both valves will record the name of that person in their corresponding lifecycle category. That person makes them coupled, so a minimal cut set in which there are basic events generated in those valves (e.g. *fail to control* for the control valve and *fail stuck* for the stop valve) is vulnerable to common cause failures, hence the analysts have to undertake common cause failure analysis.

To show a practical example of how the FLASH method works, we will analyse the minimal cut set in Figure 5-1. This figure provides a graphical representation of couplings that may exist in a minimal cut set of the third order. It shows that events *A*, *B*, and *C* share a number of couplings. First we see that all of the three events share code *IIF1* which represents the fact that the same people have installed components in which these events may arise. Then we see that events *A* and *B* are coupled by coupling codes i.e. *DCA1*, *DTM1*, *DS1*, *OS1*, *OP1*, *MS1*, and *MP1* which are the potential couplings generated during the *Concept and Design*, the *Operation* and the *Maintenance* stages. Additionally we see that events *B* and *C* share coupling codes *MM2*, *MDP2*, and *MPP2*, which are the couplings generated during manufacturing. Therefore this minimal cut set has to undergo common cause failure analysis.

When a minimal cut set like this is found there are actually three possibilities. The first and most obvious, is to try to remove couplings. For instance, if only we eliminate the couplings *IIF1* (i.e. *Installation - Fitter*) we prevent a potential *root cause* which could affect all the events in the minimal cut set. This can be done employing different staff[23] to fit components where *A*, *B* and *C* arise.

The second possibility is to assume that no root causes will spread through those couplings, then evidence has to be given. For instance it can be said that experience from similar systems has shown it to be extremely unlikely that conceivable root causes will spread through such coupling. Additionally, we can say that the remaining couplings (i.e. those coupled events *AB* and *BC*) do not endanger the system since they

---

[21] It can be also the same team or the subcontractor.

[22] In FLASH terminology: *all their generated events*.

[23] A practical example is an accident that happened to a British Aerospace aircraft (BAE 146) with four engines which had a *four-engine failure* due to common maintenance errors. Now they have changed the procedure: rules say that two teams have to be appointed for the maintenance of the four engines (i.e. they have reduced the coupling).

only reduce the degree of fault tolerance from three to two failures, and they alone cannot cause the minimal cut set.

The third option is to quantify the likelihood of the coupled minimal cut set with the method that we propose in the next section.

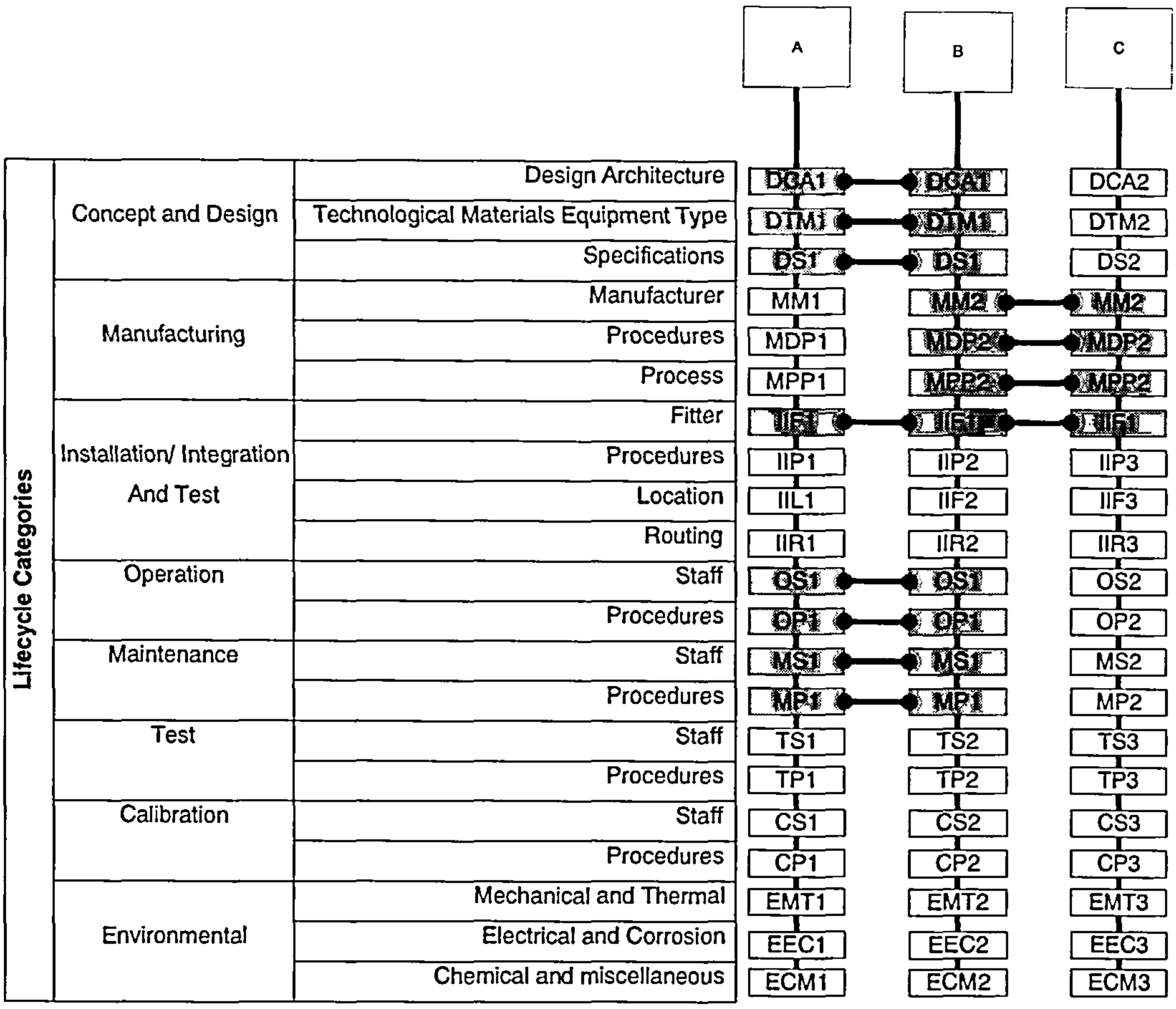


Figure 5-1: Couplings in minimal cut set ABC

5.3 Likelihood of MCS with coupled events

The likelihood of a minimal cut set with coupled events is always evaluated by using parametric methods. These methods adopt parameters to represent conditional probabilities of an event arising in some circumstances. As was said in the second chapter, all parametric methods developed, so far, assume the symmetry hypothesis [Mosleh et al., 1988], which is based on the common practice in safety and reliability analysis to use the same likelihood for events involving similar types of components. Experience has actually shown that this is appropriate for systems where common cause

failures have been studied in the last thirty years (i.e. nuclear power and chemical plants). What happens is that failure rates for similar types of valves, pumps, diesel engines etc. (i.e. same size, activation etc.) operating in comparable environmental conditions are very similar, regardless of the manufacturer and the designer [T-Book, 1992; OREDA, 1984].

However we are addressing a different area. Fault tolerant computer based systems are usually not constructed using similar hardware components, but by using a mixture of different hardware, software, information and time redundancies. Consequently events representing misbehaviours of their redundant channels are always bound to have very different probabilities. Therefore the symmetry hypothesis cannot be accepted when evaluating the likelihood of coupled minimal cut set for these systems.

Hence in this section we propose a novel method for the quantitative estimation of probabilities of minimal cut set with coupled events that does not assume the symmetry hypothesis. First, we present a new perspective to look at the likelihood of a generic event. Then we show how to calculate the probability of a minimal cut set with coupled events considering only actual couplings.

5.3.1 Likelihood of a generic event

The likelihood of an event is the probability that it occurs within certain conditions that are described by some parameters. Among these parameters there are the failure probability on demand, and the frequency that the event occurs, which are functions of other parameters describing, for instance, environmental conditions in which the component originating the event is operating (e.g. environmental dependencies, etc.), maintenance, testing, etc. For our studies we assume that the probability of each event is already known. That is equivalent to say that contributions previously mentioned are already considered in the total likelihood of the event.

What we aim to do is partitioning the total likelihood of the event and associate each share to a lifecycle category. This is like assuming that basic events in components arise because of something that was not properly considered or that could have been done better in the lifecycle of the component, i.e. an error or a defect. The portion of the likelihood of each event that is associated with each lifecycle category is *"the Percentage $\%_i$"* that was introduced in chapter four.

In mathematical terms, the total likelihood of an event X can be written as $P(X)$. If event X is made up of n independent events $\underline{x}_i$, we can write $P(X)$ as in the following expression:

$$P(X) = P\left(\sum_{i=1}^{n} \underline{x}_i\right)$$

This is a very complex expression to expand[24]. For example, for $n = 3$ (which is equivalent to a minimal cut set of the third order) it can be written as:

$$\begin{aligned} P(X) &= P\left(\sum_{i=1}^{3} \underline{x}_i\right) \\ &= P(\underline{x}_1 + \underline{x}_2 + \underline{x}_3) \\ &= P(\underline{x}_1) + P(\underline{x}_2 + \underline{x}_3) - P(\underline{x}_1(\underline{x}_2 + \underline{x}_3)) \\ &= P(\underline{x}_1) + P(\underline{x}_2) + P(\underline{x}_3) - P(\underline{x}_2\underline{x}_3) - P(\underline{x}_1\underline{x}_2 + \underline{x}_1\underline{x}_3) \end{aligned}$$

However, since in our study we are considering probabilities extremely small (i.e. almost always smaller that 10^{-3}, terms of the second (or greater) order, i.e. $P(\underline{x}_2\underline{x}_3), P(\underline{x}_1\underline{x}_2 + \underline{x}_1\underline{x}_3)$, are extremely small if compared with term of the first order, i.e. $P(\underline{x}_1), P(\underline{x}_2), P(\underline{x}_3)$. Therefore they can be neglected and we can write the likelihood $P(X)$, for $n = 3$, as:

$$\begin{aligned} P(X) &= P\left(\sum_{i=1}^{3} \underline{x}_i\right) \\ &= P(\underline{x}_1 + \underline{x}_2 + \underline{x}_3) \\ &\cong P(\underline{x}_1) + P(\underline{x}_2) + P(\underline{x}_3) \end{aligned}$$

---

[24] Except in the case events $\underline{x}_i$ are mutually exclusive, in which P(X) can be written as

$$P(X) = \sum_{i=1}^{n} P(\underline{x}_i)$$

$P(X)$ can be generalized for n generic causes as in equation 5.1 that represents $P(X)$ as the arithmetical sum of *ith* terms, each of them representing the likelihood $P(\underline{x}_i)$ of the cause $\underline{x}_i$ occurring and giving rise to event X.

$$P(X) \cong \sum_{i=1}^{n} P(\underline{X}_i) \tag{5.1}$$

When event X is part of a minimal cut set, the cause x_i can be a *potential coupling*, consequently $P(\underline{x}_i)$ will be the likelihood that the potential coupling x_i will give rise to event X.

$P(\underline{x}_i)$ is obtained by multiplying the total likelihood $P(X)$ of the event X times *the Percentage* $\%_I$.

$$P(\underline{x}_i) = \%_I * P(X) \tag{5.2}$$

5.3.2 Likelihood of coupled events

In this section we see how to estimate the likelihood of a minimal cut set with coupled events by using expression 5.1. First we consider a very simple example, a minimal cut set made up of two events, for which the lifecycle is decomposed into two categories only. Then we will examine a minimal cut set with three events, finally we will extrapolate an expression for a minimal cut set of order n with l lifecycle categories.

We know that the likelihood of an uncoupled minimal cut set of the second order is the product of the likelihood of each event in the minimal cut set occurring randomly. That likelihood can be written for l lifecycle categories by using equation 5.1. It appears as in 5.3.

$$P(XY) = P(X)\,P(Y) \cong \sum_{i=1}^{l} P(\underline{X}_i) \sum_{i=1}^{l} P(\underline{Y}_i) \tag{5.3}$$

However, as we said in the second chapter, when the minimal cut set is coupled this expression does not hold, so we have to consider the product of the likelihood of every single potential cause of each event with the likelihood of every single potential cause of all the other events. Therefore, the likelihood of a minimal cut set of the second order ($n=2$) with two lifecycle categories ($l = 2$) has to be written as in 5.4.

$$P(XY) = P[(\underline{x}_1+\underline{x}_2)(\underline{y}_1+\underline{y}_2)] \qquad (5.4)$$
$$= P[\underline{x}_1\underline{y}_1 + \underline{x}_1\underline{y}_2 + \underline{x}_2\underline{y}_1 + \underline{x}_2\underline{y}_2]$$

To further expand 5.4 we must know whether or not potential causes are mutually exclusive. Actually they are not mutually exclusive. A *potential cause* that gives rise to an event does not exclude another potential cause of that event. For example, the fact that an actuator fails because it was wrongly manufactured (i.e. it wears out too quickly), does not exclude the same actuator failing, at exactly the same time, because it was also wrongly tested during maintenance. However, the likelihood of both events happening simultaneously is quite small. Therefore, since potential causes are not mutually exclusive, expression 5.4 can be expanded as follows in 5.5.

$$\begin{aligned} P(XY) &= P[(\underline{x}_1+\underline{x}_2)(\underline{y}_1+\underline{y}_2)] \\ &= P[\underline{x}_1\underline{y}_1 + \underline{x}_1\underline{y}_2 + \underline{x}_2\underline{y}_1 + \underline{x}_2\underline{y}_2] \\ &= P(\underline{x}_1\underline{y}_1) + P(\underline{x}_1\underline{y}_2) + P(\underline{x}_2\underline{y}_1) + P(\underline{x}_2\underline{y}_2) \qquad (5.5) \\ &\quad - P(\underline{x}_1\underline{y}_1\underline{y}_2) - P(\underline{x}_2\underline{y}_1\underline{y}_2) - P(\underline{x}_1\underline{x}_2\underline{y}_1) - P(\underline{x}_1\underline{x}_2\underline{y}_2) \\ &\quad + P(\underline{x}_1\underline{x}_2\underline{y}_1\underline{y}_2) \end{aligned}$$

This expression appears quite complex, however we can make some considerations to simplify it. Since we deal with probabilities which are extremely small, terms of the third and fourth order (i.e. $P(\underline{x}_1\underline{y}_1\underline{y}_2)$, $P(\underline{x}_2\underline{y}_1\underline{y}_2)$, $P(\underline{x}_1\underline{x}_2\underline{y}_1)$, $P(\underline{x}_1\underline{x}_2\underline{y}_2)$ and $P(\underline{x}_1\underline{x}_2\underline{y}_1\underline{y}_2)$) are negligible when compared to terms of the second order (i.e. $P(\underline{x}_1\underline{y}_1)$, $P(\underline{x}_1\underline{y}_2)$, $P(\underline{x}_2\underline{y}_1)$ and $P(\underline{x}_2\underline{y}_2)$), therefore they can be safely ignored. Hence the probability of the same minimal cut set can be written as:

$$P(XY) \cong P(\underline{x}_1\underline{y}_1) + P(\underline{x}_1\underline{y}_2) + P(\underline{x}_2\underline{y}_1) + P(\underline{x}_2\underline{y}_2) \qquad (5.6)$$

or in a more compact form as:

$$P(XY) \cong \sum_{i=1}^{2}\sum_{j=1}^{2} P(\underline{X}_i, \underline{Y}_j) \tag{5.7}$$

Now, if we suppose that events X and Y are coupled for the first of the two lifecycle categories (see the Figure 5-2), we know that $P(\underline{x}_1\underline{y}_1)$ has to be evaluated with methods for common cause failure analysis, whilst the remaining probabilities, i.e. $P(\underline{x}_1\underline{y}_2)$, $P(\underline{x}_2\underline{y}_1)$ and $P(\underline{x}_2\underline{y}_2)$, can be simply calculated as products of independent terms. Hence, if we use subscript letters I and C to indicate *Independent* and *Coupled* likelihood (i.e. P_I and P_C) of a generic event we can represent the value of each of the terms in 5.6 as in equation 5.8.

$$\begin{aligned}
P(\underline{x}_1\underline{y}_1) &= P_I(\underline{x}_1)\, P_I(\underline{y}_1) + P_C(\underline{x}_1\, \underline{y}_1) \\
P(\underline{x}_1\underline{y}_2) &= P(\underline{x}_1)\, P(\underline{y}_2) \\
P(\underline{x}_2\underline{y}_1) &= P(\underline{x}_2)\, P(\underline{y}_1) \\
P(\underline{x}_2\underline{y}_2) &= P(\underline{x}_2)\, P(\underline{y}_2)
\end{aligned} \tag{5.8}$$

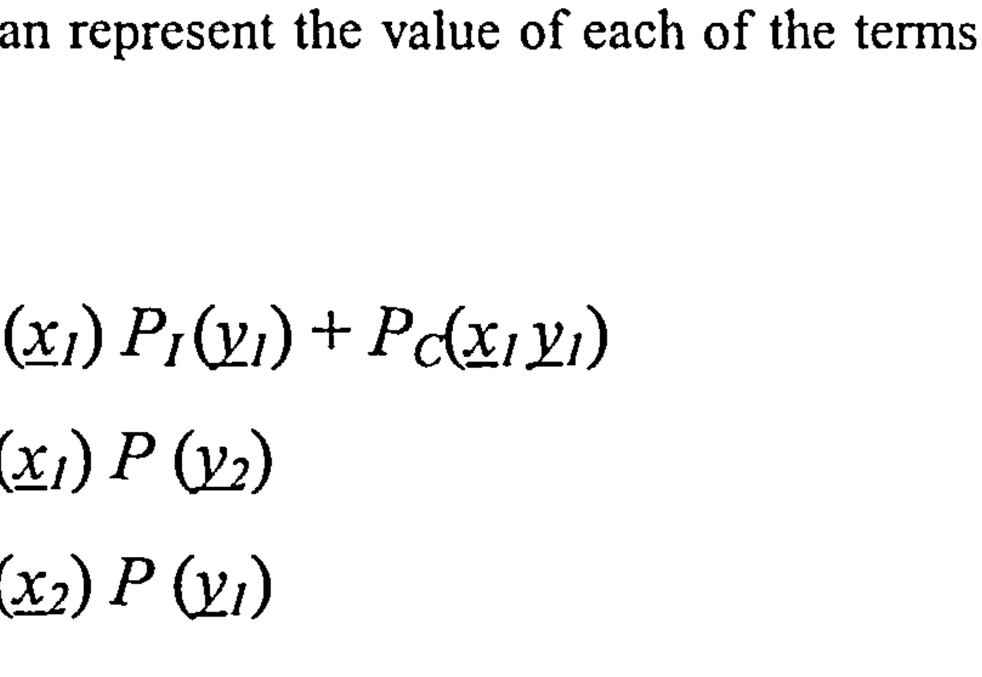


Figure 5-2: Minimal cut set of the second order with two lifecycle categories

Therefore, estimation of the likelihood of a minimal cut set of the second order, with two lifecycle categories can be handled quite easily, as we have seen. However, if we increase the order of the minimal cut set and the number of lifecycle categories, the complexity scales up. Actually, the number of terms in the expression for the likelihood of a minimal cut set of the third order with five lifecycle categories (like the one represented in Figure 5-3) has 125 terms that can be compactly represented by expression 5.9.

$$P(XYZ) \cong \sum_{i=1}^{5}\sum_{j=1}^{5}\sum_{k=1}^{5} P\left(\underline{X}_i, \underline{Y}_j, \underline{Z}_k\right) \tag{5.9}$$

Now, if we look at Figure 5-3, we see that some of the couplings related to only a sub-group of events in the minimal cut set. For instance, the first potential coupling concerns events X and Y, while the second concerns events Y and Z; the third involves events X and Z; and only the last one touches all the three events in the minimal cut set. Therefore if we highlight in bold coupled events in equation 5.9 we will have mixed likelihood terms like $P(\underline{x}_1, \underline{y}_1, \underline{z}_1)$, $P(\underline{x}_2, \underline{y}_2, \underline{z}_2)$, $P(\underline{x}_3, \underline{y}_3, \underline{z}_3)$, $P(\underline{x}_1, \underline{y}_1, \underline{z}_2)$, $P(\underline{x}_1, \underline{y}_2, \underline{z}_2)$, etc. Probabilities of these terms can be evaluated as products of an independent likelihood times a common cause failure likelihood. For example $P(\underline{x}_1, \underline{y}_1, \underline{z}_1)$ can be seen as product of common cause failure term $P(\underline{x}_1, \underline{y}_1)$ times the independent term $P(\underline{z}_1)$.

$$P(\underline{x}_1, \underline{y}_1, \underline{z}_1) = P(\underline{x}_1, \underline{y}_1) * P(\underline{z}_1) \tag{5.10}$$

Where:

$$P(\underline{x}_1\underline{y}_1) = P_I(\underline{x}_1)P_I(\underline{y}_1) + P_C(\underline{x}_1\underline{y}_2)$$

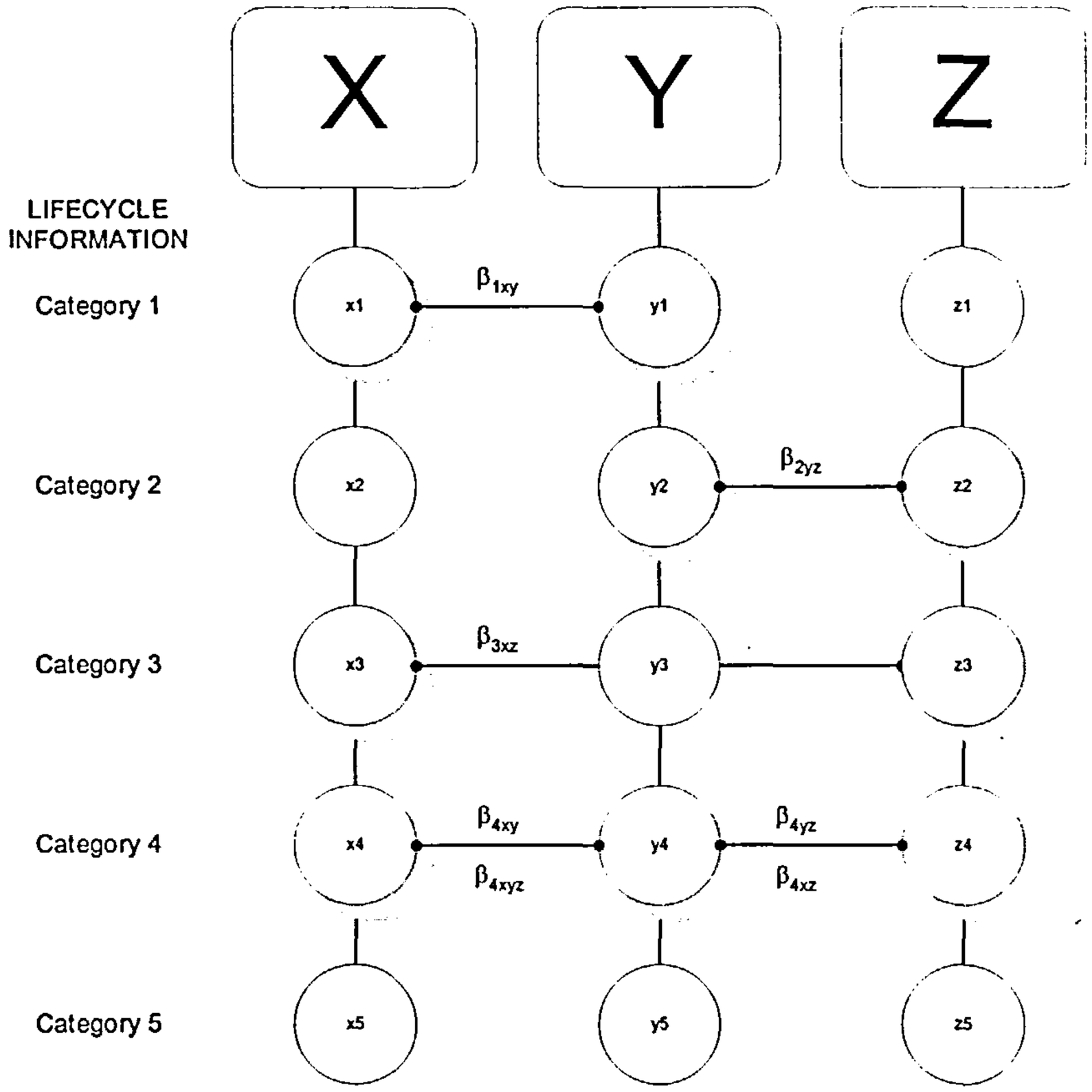


Figure 5-3: Minimal cut set of the third order with five lifecycle categories

Probabilities for the other mixed terms with two coupled events i.e. $P(\underline{x}_1, \underline{y}_1, \underline{z}_2)$ $P(\underline{x}_1, \underline{y}_1, \underline{z}_3)$, ...,$P(\underline{x}_1, \underline{y}_1, \underline{z}_5)$; $P(\underline{x}_1, \underline{y}_2, \underline{z}_2)$ $P(\underline{x}_2, \underline{y}_2, \underline{z}_2)$, ..., $P(\underline{x}_5, \underline{y}_2, \underline{z}_2)$ and $P(\underline{x}_3, \underline{y}_1, \underline{z}_3)$, $P(\underline{x}_3, \underline{y}_2, \underline{z}_3)$,..., $P(\underline{x}_3, \underline{y}_5, \underline{z}_3)$ can be similarly evaluated. While the likelihood for the term $P(\underline{x}_4, \underline{y}_4, \underline{z}_4)$ is calculated on the basis that $\underline{x}_4$, $\underline{y}_4$ and $\underline{z}_4$ are fully coupled. Therefore it will be as in expression 5.11.

$$
\begin{aligned}
P(\underline{x}_4\,\underline{y}_4\,\underline{z}_4) = \quad & P_I(\underline{x}_4)\,P_I(\underline{y}_4)\,P_I(\underline{z}_4) \\
& + P_I(\underline{x}_4)\,P_C(\underline{y}_4\,\underline{z}_4) \\
& + P_I(\underline{y}_4)\,P_C(\underline{x}_4\,\underline{z}_4) \\
& + P_I(\underline{z}_4)\,P_C(\underline{x}_4\,\underline{y}_4) \\
& + P_C(\underline{x}_4\,\underline{y}_4\,\underline{z}_4)
\end{aligned}
\qquad (5.11)
$$

Table 5-2 reports all the terms that have to be substituted in expression 5.9 to consider common cause failures in the minimal cut set in our example. The right column has

terms that consider common cause failures which are used to replace terms in the left column for the calculation of the likelihood of the minimal cut set in Figure 5-3.

Terms to be replaced	*Likelihood of coupled terms*
$P(\underline{x}_1 \underline{y}_1)$	$P_I(\underline{x}_1)\, P_I(\underline{y}_1) + P_C(\underline{x}_1\, \underline{y}_1)$
$P(\underline{y}_2\, \underline{z}_2)$	$P_I(\underline{y}_2)\, P_I(\underline{z}_2) + P_C(\underline{y}_2 \underline{z}_2)$
$P(\underline{x}_3\, \underline{z}_3)$	$P_I(\underline{x}_3)\, P_I(\underline{z}_3) + P_C(\underline{x}_3 \underline{z}_3)$
$P(\underline{x}_4\, \underline{y}_4\, \underline{z}_4)$	$P_I(\underline{x}_4)\, P_I(\underline{y}_4)\, P_I(\underline{z}_4) + P_I(\underline{x}_4)\, P_C(\underline{y}_4\, \underline{z}_4)$ $+ P_I(\underline{y}_4)\, P_C(\underline{x}_4\, \underline{z}_4) + P_I(\underline{z}_4)\, P_C(\underline{x}_4 \underline{y}_4)$ $+ P_C(\underline{x}_4 \underline{y}_4\, \underline{z}_4)$

Table 5-2: Likelihood of coupled terms

Now we go back to expressions 5.6 and 5.9. They were written for a minimal cut set of the second order with two lifecycle categories, and for a minimal cut set of the third order with five lifecycle categories respectively. They can be extended to a generic minimal cut sets of order n with l lifecycle categories. The extended expression is given in 5.12. We can easily see that the number of terms in that expression is equivalent to the order n of the minimal cut set raised to the number l of lifecycle categories.

$$P(X^1, X^2, \ldots X^n) \cong \sum_{i=1}^{l} \sum_{j=1}^{l} \ldots \sum_{z=1}^{l} P\left(\underline{x}_i^1, \underline{x}_j^2, \ldots, \underline{x}_z^n\right) \qquad (5.12)$$

This expression may have terms representing coupled probabilities up to order n. We have already seen expressions for coupled probabilities of second and third order (i.e. $P(\underline{x}_1 \underline{y}_1)$ and $P(\underline{x}_4 \underline{y}_4 \underline{z}_4)$ in 5.8 and 5.11). Similar expressions can also be written for greater order minimal cut sets, however it is not the aim of this thesis to show the form of these terms.

Drawing some conclusions, in this section we have seen that it is theoretically possible to analyse common cause failures at the level of potential couplings. Hence, we have transferred the problem of common cause failure analysis from the minimal cut set level to a lower, more detailed, level. Additionally, we have proposed a systematic way to identify minimal cut sets with coupled events and showed how common cause likelihood can be evaluated considering only contributions from real couplings and not with a summary analysis at minimal cut set level, as it is usually done. We have actually

approached the problem of common cause failure analysis the other way round. Instead of leaving common cause failure analysis as a final analysis we propose to start collecting data already at the beginning of the lifecycle then to use these data for a systematic identification of potential couplings hence for the evaluation of the common cause likelihood.

However we have not yet seen how, practically, we can estimate *Independent* and *Coupled* probabilities that are in the expression of the likelihood of a coupled minimal cut set.

5.3.3 Independent and coupled probabilities

In this section we show a way to estimate the independent and coupled probabilities that are in the expression of the likelihood of a coupled minimal cut set. The method we propose is based on the β factor parametric model that was introduced in the second chapter. However, instead of applying this model at minimal cut set level, we apply it at the level of potential couplings. This way to proceed can be laborious since it should use as many β parameters as the number of actual couplings, however some considerations based on the experience gained in the past thirty years of studying common cause failures will help us in setting reasonable values for these parameters.

The β factor parametric model was introduced in the second chapter. It is the simplest of all the parametric methods, since it considers only the independent likelihood of each event in the minimal cut set and the likelihood of all the events happening simultaneously because of a common cause failure. For a minimal cut set of the second order it is equivalent to more complicated methods, like the multiple Greek letter model. It becomes more and more conservative with the increasing of the order of the minimal cut set, however it is the model used the most to consider common cause failures.

The β factor parametric model is quite simple to apply. The coupled likelihood for a minimal cut set is obtained by multiplying the likelihood of any of the events in the minimal cut set (that for the symmetry hypothesis have the same likelihood) times the β parameter. The independent likelihood is then obtained by subtracting the coupled likelihood from the total. Therefore, if we could use the symmetry hypothesis we would need to know only two terms: the total likelihood, that is the likelihood of the event as if

it was a common, isolated basic event[25], and the β parameter, that is related only to the degree of coupling existing among events in the minimal cut set.

In case of a minimal cut set of the second order the study proceeds as follow. The likelihood of each event in the minimal cut set (that for the symmetry hypothesis is the same) is given to the term Q_T (total likelihood).

$$P(X) = P(Y) = Q_T \qquad (5.13)$$

(*Symmetry hypothesis*)

Then, the coupled likelihood is obtained multiplying Q_T times the parameter β. The coupled likelihood of two events $P_C(X, Y)$ is indicated by Q_2. The subscript "2" represents the number of events that are coupled in the minimal cut set.

$$P_C(XY) = \beta Q_T = Q_2 \qquad (5.14)$$

The independent likelihood is, then, calculated by subtracting the coupled contribution Q_2 from the total likelihood Q_T. Since the independent likelihood refers to one event, it has subscript "1" i.e. Q_1.

$$P_I(X) = P_I(Y) = (1-\beta)\, Q_T = Q_1 \qquad (5.15)$$

Therefore the common cause failure probability for a minimal cut set of the second order is written as in 5.16.

$$P(X\,Y) = P_I(X)\, P_I(Y) + P_C(XY) = Q_1^{\,2} + Q_2 \qquad (5.16)$$

Or as function of β and Q_T:

$$P(X\,Y) = Q_1^{\,2} + Q_2 = [(1-\beta)Q_T]^2 + \beta Q_T \qquad (5.17)$$

---

[25] The likelihood obtained from the manufacturer data sheets adapted to the condition where the component operates.

This expression shows also that for $\beta \to 0$ the likelihood of the minimal cut set tends to the likelihood of the two events happening independently. That is an important property, that we have to maintain in the expression of the likelihood of a coupled minimal cut set that we are going to propose.

$$\lim_{\beta \to 0} \left\{(1-\beta)Q_T\right]^2 + \beta Q_T\right\} = Q_T^{\,2}$$

$$\lim_{\beta \to 0} P(X\,Y) = P(X)\,P(Y) \tag{5.18}$$

The proper way to proceed is to write the coupled likelihood of two events as in 5.19, which says that the P_C for a minimal cut set XY can be thought as the fraction β_{xy} of the total likelihood of event X, or as a fraction β_{yx} of the total likelihood of event Y:

$$P_C(X\,Y) = \beta_{xy}\,P(X) = \beta_{yx}\,P(Y) \tag{5.19}$$

In this way, we would write independent probabilities $P_I(X)$ and $P_I(Y)$ as follow:

$$\begin{aligned} P_I(X) &= [P(X) - \beta_{xy}\,\mathrm{P}(X)] = P(X)\,(1-\beta_{xy}) \\ &= [P(Y) - \beta_{yx}\,P(Y)] = P(Y)\,(1-\beta_{yx}) \\ P_I(Y) &= [P(Y) - \beta_{yx}\,P(Y)] = P(Y)\,(1-\beta_{yx}) \\ &= [P(X) - \beta_{xy}\,P(X)] = P(X)\,(1-\beta_{xy}) \end{aligned}$$

and we would obtain the likelihood for a coupled MCS of the second order as in 5.20:

$$P\,(XY) = [P(X)(1-\beta_{xy})]\,[\,P(Y)(1-\beta_{yx})] + \beta_{xy}\,P(X) \tag{5.20}$$

However, this expression is quite complex and, additionally, we would have to estimate a lot of β parameters. In case of a minimal cut set of greater order this expression would become even more complex. For a third order minimal cut set it involves the estimation of twelve betas i.e. β_{xy}, β_{xz}, β_{yx}, β_{yz}, β_{zx}, β_{zy}, β_{xyz}, β_{xzy}, β_{yxz}, β_{yzx}, β_{zxy}, β_{zyx}. That is not practical, therefore we propose another way to proceed. We still believe that using

expressions like 5.14 to calculate the likelihood of a group of coupled events is a good choice, therefore in following sections we will propose to put all our β parameters equal to a very conservative value (*i.e.* $\beta=1$). Additionally we propose to assign to the term Q_T, the smallest of the probabilities of events in the minimal cut set, as explained in the next section.

The β parameter

In the β factor parametric method the β parameter represents the conditional probability that the cause of a component failure will be shared by one or more additional components, given that a component has already failed. As β is a probability, it may range between *0* and *1*, though, practically, it usually ranges between 10^{-4} and $2x10^{-1}$, where the first value is for extremely weakly coupled systems and the second one is for highly coupled ones. Additionally, β can also be seen as the strength of the coupling among events, the greater the β factor, the greater is the coupling.

In our approach, the strength of the coupling among events is represented by the existence or not of shared coupling codes in corresponding (peer) lifecycle categories weighted with their *Percentage* $\%_I$. The greater the number of shared coupling codes and the higher the *Percentage* $\%_I$ associated with each lifecycle category, the greater is the coupling among events. Basically, what in the β factor parametric method is considered in the β parameter is, in our approach, considered in the combination of the *Percentage* $\%_I$ and the existence of shared coupling codes for corresponding lifecycle categories. This is actually what we wanted to achieve since we have introduced lifecycle categories, coupling codes and the *Percentage* $\%_I$ to model explicitly the strength of couplings (i.e. the β parameter in the β factor parametric method) among events. Therefore the β that appears in our method, does not represent the same conditional probability as it does in the β factor parametric method. In our method we expect β to be very near to one. To be conservative, we propose to put $\beta = 1$ for the analysis of any unknown or new systems. However, if there are sufficient data, β should be statistically estimated with methods similar to those proposed in [Mosleh, et al., 1988], but we won't discuss this argument any further. Examples that follow in this and the next chapter will show that the failure probability for the same coupled minimal cut set evaluated with the β factor parametric method and the approach proposed in this thesis (with $\beta=1$) are very similar (at least within the same order).

Value for the total likelihood Q_T

As far as Q_T is concerned, since we cannot assume the symmetry hypothesis (equation 5.13) and we wish to use expression 5.14 to calculate P_C, we have to find another way to assign a value to Q_T. We have at least three alternatives. We could either put Q_T equal to the average, the smallest or the biggest value among probabilities of events in the minimal cut set. However, since probabilities involved span various orders of magnitude, the average value will coincide with the likelihood of the most likely event. Additionally, if Q_T is equal to the most likely event, we may come to the absurd conclusion that the coupled likelihood is bigger than the most unlikely event in the minimal cut set, which is absolutely unrealistic. Let us see an example. We have two events X and Y, the likelihood of X is $P(X)=10^{-3}$, the likelihood of Y is $P(Y)=10^{-5}$. If they were independent the likelihood of them occurring simultaneously would be their product i.e. 10^{-8}. Since they are coupled, the likelihood that they occur simultaneously should be bigger than the likelihood of the theoretically uncoupled minimal cut set (i.e. 10^{-8}), but smaller than the likelihood of the most unlikely single event (i.e. $P(Y)=10^{-5}$). If we have $\beta=.1$ and we put Q_T equal to the average likelihood among the two events in our minimal cut set, we would have the value $Q_T \cong 10^{-3}$, which practically coincides with the likelihood of the most likely event (i.e. X). After applying 5.14, we would have that $P_C(X,Y) \cong 10^{-4}$, which is absurd, since this figure is bigger than the upper bound of the likelihood that can realistically be associated with the coupled minimal cut set that we said is $P(Y)=10^{-5}$. Hence, if we cannot use the average or the biggest value among probabilities of events in the minimal cut set, we are left with the smallest one. Coming back to our example. If we put Q_T equal to the smallest likelihood (i.e. 10^{-5}) we would have that $P_C(XY)=10^{-6}$, that is inside the boundary we were expecting. Hence we propose to put the coupled probability for a minimal cut set of the second order equal to the smallest among the probabilities of events in the minimal cut set. For a second order minimal cut set we indicate that as in the following 5.21.

$$P_C(X\,Y) = \beta \min [\mathrm{P}(X); P(Y)] \tag{5.21}$$

Consequently, independent probabilities are written as in 5.22:

$$P_I(X) = P(X) - \beta \min [P(X); P(Y)] \tag{5.22}$$
$$P_I(Y) = P(Y) - \beta \min [P(Y); P(Y)]$$

The complete formula for the likelihood of the coupled minimal cut set of the second order obtained under these conditions, is stated in 5.23. This expression satisfies limit 5.18 that says that, for $\beta \to 0$, the likelihood of the minimal cut set tends to the likelihood of the two events to occur independently, as shown by the 5.24.

$$P(X\,Y) = P_I(X)\,P_I(Y) + P_C(X\,Y) =$$

$$=\{P(X)\text{-}\beta \min[P(X);\,P(Y)]\}\,\{P(Y)\text{-}\beta \min[P(X);\,P(Y)]\}$$

$$+\,\beta \min[P(X);\,P(Y)] \qquad (5.23)$$

$$\lim_{\beta \to 0} \{P(X)\text{-}\beta \min[P(X);\,P(Y)]\}\,\{P(Y)\text{-}\beta \min[P(X);P(Y)]\}$$

$$+\,\beta \min[\mathrm{P}(X);\,\mathrm{P}(Y)] = \mathrm{P}(X)\mathrm{P}(Y) \qquad (5.24)$$

It can be demonstrated that the expression for the likelihood of a coupled minimal cut set of the third order is represented by the following expression:

$$P(X,Y,Z) = P_i(X)P_i(Y)P_i(Z)+P_i(X)[P_c(Y;\,Z)\text{-}P_c(X;\,Y;\,Z)]+P_i(Y)[P_c(X;\,Z)$$
$$\text{-}P_c(X;\,Y;\,Z)]+P_i(Z)[P_c(X;\,Y)\text{-}P_c(X;\,Y;\,Z)]+P_c(X;\,Y;\,Z)$$

and, within our hypotheses about β and Q_T, it can be reduced to the following:

$$P(X,Y,Z) =\{P(X)\text{-}\beta \min[P(X);P(Y);P(Z)]\}\,\{P(Y)\text{-}\beta \min[P(X);\,P(Y);P(Z)]\}\,\{P(Z)\text{-}$$
$$\beta \min[P(X);P(Y);P(Z)]\} + \beta \mathrm{Min}[P(X);P(Y);P(Z)]$$

Expressions for greater order minimal cut sets can also be written, but it is not intended in this thesis to investigate all the statistics related to this matter.

Final considerations

Using expressions for the likelihood of coupled events obtained in previous sections at the level of lifecycle categories, produces better, more realistic results than applying them at minimal cut set level. At the level of lifecycle categories, we individually

consider each cause of coupling. Then, we perform the analysis only on the causes that are shared among events in the minimal cut set by applying proposed formula for estimating the coupled likelihood. Since each cause is responsible for only a fraction of the total likelihood of each event (i.e. the *Percentage* $\%_l$ which refers to the corresponding lifecycle category), the use of a conservative value for β does not produce too conservative a figure for the likelihood of the minimal cut set as a whole. Let us see an example.

If we take the minimal cut set in Figure 5-2 and we suppose that:

- The first lifecycle category is responsible for a 5 percent share of the total likelihood of each event (i.e. *percentage* $\%_1=5$);
- The second lifecycle category is responsible for the remaining 95 percent (i.e. *percentage* $\%_2=95$);
- The total likelihood of event X is $P(X)=10^{-3}$;
- The total likelihood of event Y is $P(Y)=10^{-5}$.

After applying 5.2 and 5.23 to equation 5.8, we obtain probabilities in expression 5.6. These are the passages:

$$P(\underline{x}_1) = \%_1 * P(X) = .5 * 10^{-4}$$
$$P(\underline{x}_2) = \%_2 * P(X) = .95 * 10^{-3}$$
$$P(\underline{y}_1) = \%_1 * P(Y) = .5 * 10^{-6}$$
$$P(\underline{y}_2) = \%_2 * P(Y) = .95 * 10^{-5}$$
$$\beta = 1$$

$$P(\underline{x}_1\underline{y}_1) = \{ P(\underline{x}_1) - \beta \min[P(\underline{x}_1);P(\underline{y}_1)]\} \{P(\underline{x}_1) - \beta \min[P(\underline{x}_1);P_T(\underline{y}_1)]\} + \beta \min[P(\underline{x}_1);P(\underline{y}_1)]$$

$$\cong .25 * 10^{-9} + .5 * 10^{-6} \quad \cong .5 \quad * 10^{-6}$$

$$P(\underline{x}_1\underline{y}_2) = P(\underline{x}_1)\, P(\underline{y}_2) \quad = .25 \quad * 10^{-9}$$
$$P(\underline{x}_2\underline{y}_1) = P(\underline{x}_2)\, P(\underline{y}_1) \quad = .475 \quad * 10^{-8}$$
$$P(\underline{x}_2\underline{y}_2) = P(\underline{x}_2)\, P(\underline{y}_2) \quad = .9025 * 10^{-7}$$

$$P(XY) \cong P(\underline{x}_1\underline{y}_1) + P(\underline{x}_1\underline{y}_2) + P(\underline{x}_2\underline{y}_1) + P(\underline{x}_2\underline{y}_2) \cong .5 * 10^{-6} \qquad (5.25)$$

If we wanted to obtain the same value for that likelihood by applying the same study at minimal cut set level, we would have had to put $\beta = .5*10^{-1}$ as shown in the 5.26. Being that a low value for β, we would have to introduce arguments to justify it. That would have required using expert judgement.

$$P(X,Y) = P_I(X)\,P_I(Y) + P_C(X,Y) \cong$$
$$\cong 10^{-8} + .5*10^{-1} * 10^{-5} \cong .5 * 10^{-6} \qquad (5.26)$$

What we have actually done is to screen individual causes that may give rise to events and be responsible for couplings in a minimal cut set. This has been done on the basis of a checklist that spans the whole lifecycle of each component in the system. This assumes that lifecycle information was collected during the decomposition and design stages, when this procedure is more economic and practical. In addition to the screening of couplings, we have assigned a share of the likelihood of each event to causes of events. Then we have provided mathematical support for evaluating the likelihood of coupled minimal cut sets that considers individually the contribution of each single cause of each event. As the weight of each cause we took the share of the total likelihood of each event that is associated to the correspondent lifecycle category.

5.4 Discussion

In this chapter we have extended the FLASH formalism to consider common cause failures. We have used some of the information stored into FLASH tables for two purposes: investigation of minimal cut sets to find ones with coupled events and estimation of their likelihood.

The identification of minimal cut sets with coupled events was achieved by analysing all minimal cut sets responsible for critical failures in the system under investigation. Events in each minimal cut set were scanned to see whether they were sharing one or more causes of coupling (i.e. coupling codes defined in Chapter four). If any sharing was found, the minimal cut set was considered coupled. Since the method identifies exactly those categories of the lifecycle responsible for actual couplings, it makes it easy to investigate feasible remedies for those couplings. The analysts can then give evidence why those couplings cannot give rise to common cause failures or, in the last resort, calculate the likelihood of minimal cut sets considering only contributions

from those couplings that were identified. The innovative contribution of the method lies in the systematic identification of the actual couplings. They come out of a mechanical process that is the comparison of lifecycle information of events in the minimal cut set. This process can also be easily automated. Therefore the liability for the identification of couplings is transferred from expert judgement summarily estimating the likelihood at minimal cut set level, to choosing the most convenient lists to use as base for the identification of couplings among events.

The estimation of the likelihood of minimal cut sets with coupled events, is the natural step forward, after the identification of actual couplings. We provided a mathematical framework to calculating the likelihood of minimal cut sets considering the contribution of each actual coupling. Since the expression for that likelihood was quite complex we made some approximations by eliminating terms which influence was negligible for the final results.

Drawing some conclusion, we have shown that common cause likelihood can be evaluated considering only contributions from real couplings and not with a summary analysis at minimal cut set level, as is usually done. We have transferred the problem of common cause failure analysis from the minimal cut set level to a lower (more detailed) level, systematised the identification of couplings, and obtained more realistic results.

The next chapter presents two case studies to illustrate the overall FLASH process as described in this and the previous chapter. The first case study is a Fuel System adapted from [Vesely, 1981], the second case study is a computerised braking system.

Chapter Six

Case studies

This Chapter presents two case studies illustrating the FLASH process during the decomposition and design, and during the integration and verification stages of the lifecycle. The first case study is based on a Fuel System adapted from an example in the fault tree handbook [Vesely, 1981], the second case study is based on a computerised braking system developed at the University of York, but based on realistic industrial data.

6.1 The Fuel System

The system that we examine in this section is a (hypothetical) fuel system (FS) whose task is to provide emergency supply of fuel to an engine (a generator of electrical power, for example) when the main supply to that engine is out of order. The system is automatically activated when the primary supply fails, however, it can also be manually activated and interrupted. As Figure 6-1 shows, the fuel system draws fuel resources from a tank and provides fuel supplies to the engine.

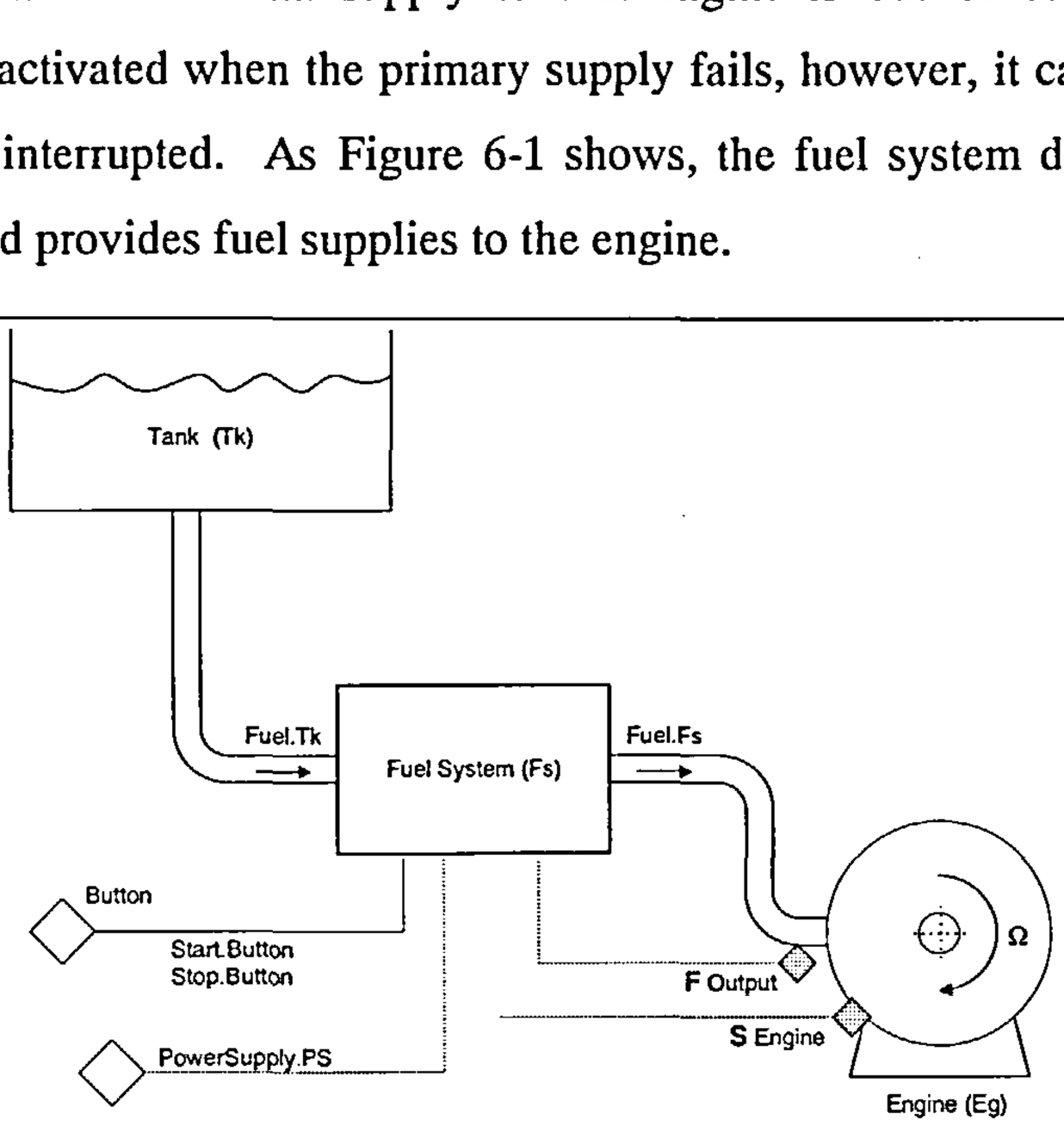


Figure 6-1: The Fuel System

Safety specifications for the fuel system require single point failures, which can give rise to hazards, to be avoided. Real-time specifications require that the engine speed Ω be limited at $\Omega=\Omega_0=constant$. Moreover, the engine speed cannot deviate from the nominal

value Ω_0 for more than *5* seconds. Only a failure may cause such a deviation for a longer period. Figure 6-2 illustrates the hierarchical decomposition of the Fuel System. At the first level of the decomposition (functional), we can see the whole equipment encapsulated in a box that receives fuel from the tank, electrical power, start/stop command signals and delivers the fuel to the engine. At the second level, we can see the architecture of the system, in other words basic components and their connections. The diagram shows three block valves (*BVA*, *BVB* and *BVX*), two control valves (*CVA* and *CVB*) and an Electronic Controller (*PLC*) which sets the position of those valves to regulate the path and rate of flow between the tank and the engine. Finally at the lowest level of the decomposition, we see a high level representation (*GRAFCET*) of the control sequence executed by the controller.

The control sequence shows that, in normal conditions of operation, the controller sets valve *BVX* to the closed position and lets the fuel flow through the path that connects valves *BVA* and *CVA*. While the system is in this state (*Path1*), the controller manages the position of valve *CVA* and ensures that the flow of fuel through the valve always equals the current demand by the engine. When the controller detects a disturbance of that flow (caused, for example, by a failure or blockage of a valve) it activates a new path in the system to restore the flow of fuel at the output. The new path (*Path2*) is the one connecting valves *BVB* and *CVB*. Finally, the control sequence shows that a failure (or blockage) of valve *BVB* while the system is in that state will trigger further action by the controller, namely the activation of a third path (*Path3*) in the system, that between valves *BVA* and *CVB*[26].

However, there are two cases in which *Path3* is not available. The first case is when *CVA* fails open (i.e. *Stuck Open* or *Significant Internal Leakage*), *BVA* closes and the flow goes through *Path2* (i.e. *Path3* is not available since *BVA* has to stay closed to avoid the fuel going through *CVA* that is failed open). The second case arises when *BVA* fails open (i.e. *Fail to close* or *Significant Internal Leakage*), *CVA* reduces the flow to a minimum, but the supply is not completely shut off. Both of them are critical incidents and the likelihood has to be less than 10^{-3} during the mission. To activate the remaining possible path (i.e. *Path4*, through valve *BVB*, *BVX* and *CVA*), *CVB must* be plugged and *BVB*, *BVX* and *CVA must* be still operating. This is an extremely unlikely circumstance therefore *Path4* was not implemented.

---

[26] There is actually a fourth possible path, that connecting valves *BVB* and *CVA*, which for simplicity we do not consider in this discussion.

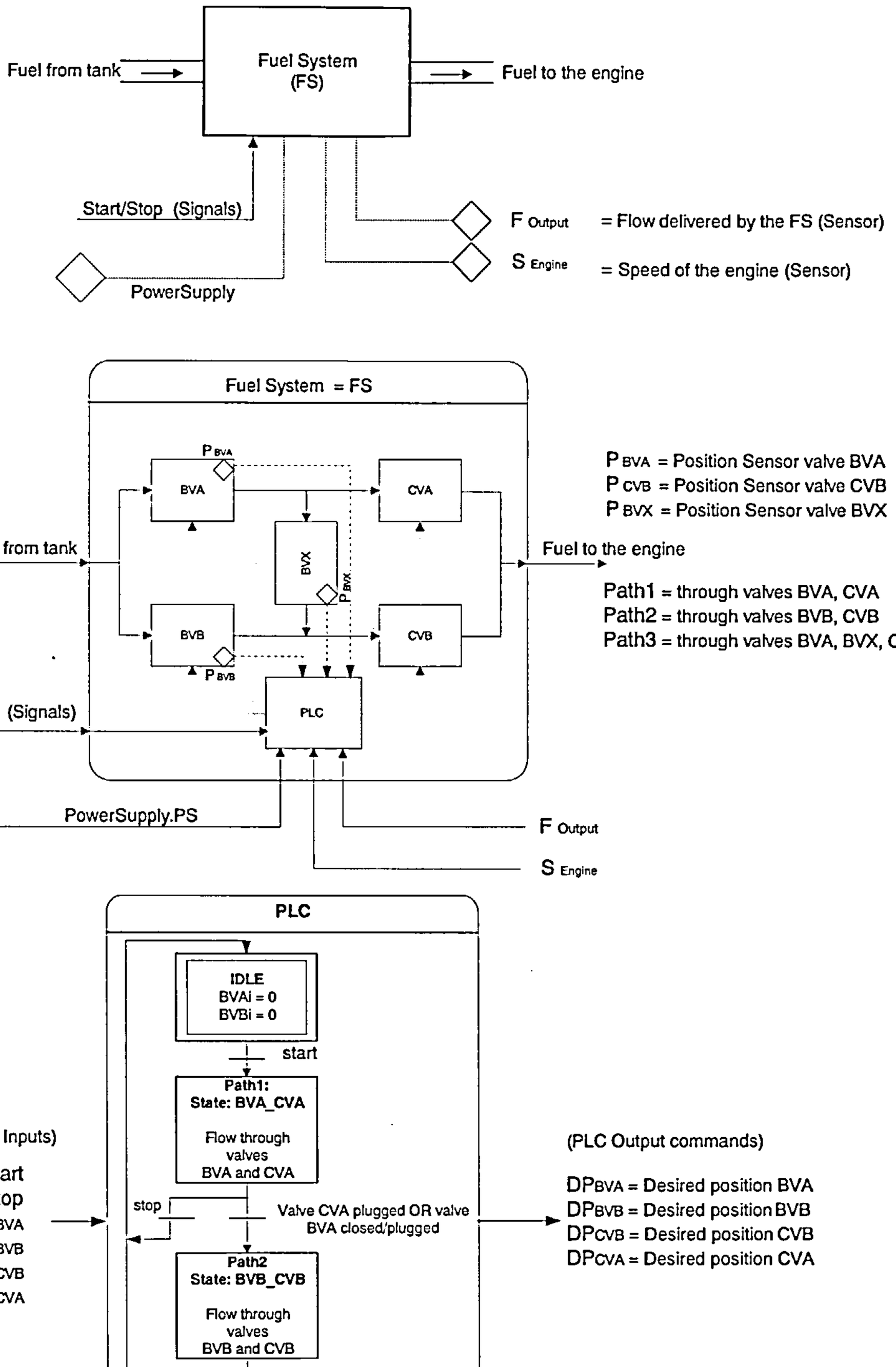


Figure 6-2 :Hierarchical Decomposition of the Fuel System

6.1.1 Analysis in the Decomposition and Design Stage

The analysis of the Fuel System (FS) starts at functional level. The fuel system is studied as a function that has to provide fuel to the engine when it is required. As such the system has three failure modes: a) *fuel is required, but not provided*, b) *fuel is not required, but provided* and c) *fuel is provided when required, but the system is not fully functional*[27] (i.e. a recovery action took place to by-pass a faulty component). Hence the FLASH table for the fuel system at functional level propagates three effects as represented in Table 6-1. These effects have different criticality levels that are indicated in the *Criticality* column (i.e. Catastrophic, Critical and Negligible). The 5th column reports recommendations that have to be considered by designers for the development of the fuel system internal architecture. For example, recommendations for the first effect states that the system should be fault tolerant for single failures in mechanically activated components, additionally they require that the likelihood of this effect be smaller than 10^{-6} during the mission time. For the second and third effect requirements are less demanding, being events not as critical as the first. However their likelihood is requested to be smaller than 10^{-3} during the mission.

After the functional hazard analysis is completed, the FLASH method requests the architecture achieving the function to be proposed. The architecture has to take into account specifications, recommendations and derived safety requirements into the 5th column. Such architecture is shown in Figure 6-3. It requires the failure of at least two valves to cause the failure of the system. Figure 6-4 represents the correspondent failure model of the fuel system drawn according to the FLASH notation. Names are given to flows delivered by components according to the syntax defined in Chapter 4.

---

[27] In this case a fault occurred. The system is still working properly, however any additional fault may cause a system failure i.e. failure modes *a)* or *b)*.

FUEL SYSTEM					
Failure event	**...**	**Description**	**Criticality**	**5th Column: Justification, Design Recommendations, Derived Safety Requirements**	**Comments (FMEA)**
Fuel is required, but not provided .fs (No Flow – fuel from the FS to the engine)		No Flow of fuel on the line that feeds the engine. The engine cannot start. No electric power is provided.	Catastrophic	*Before design* Recommendations The failure of the module cannot be handled. A fault tolerant architecture is needed to prevent that single failures in mechanically activated component will cause a system failure. The module has to be built with redundant components. **Effect max accepted likelihood** 10^{-6} during the mission *After design* Detection ... Recovery ... Recommendation ... **Max accepted likelihood for critical events in the Causes column.** ...	
Fuel is not required, but provided .fs		The engine does not stop – Electric power is wasted	Critical	*Before design* Recommendations Single point to failures are allowed **Effect max accepted likelihood** 10^{-3} during the mission *After design* Detection ... Recovery ... Recommendation ... **Max accepted likelihood for critical events in the Causes column.** ...	
Fuel is provided when required, but the system is not fully functional .fs		The engine performs properly, but a failure has occurred – Electric power is provided	Negligible	*Before design* Recommendations Single point to failures are allowed **Effect max accepted likelihood** 10^{-3} during the mission *After design* Detection ... Recovery ... Recommendation ... **Max accepted likelihood for critical events in the Causes column.** ...	

Table 6-1: FLASH table for the FS function, before the architecture is drawn

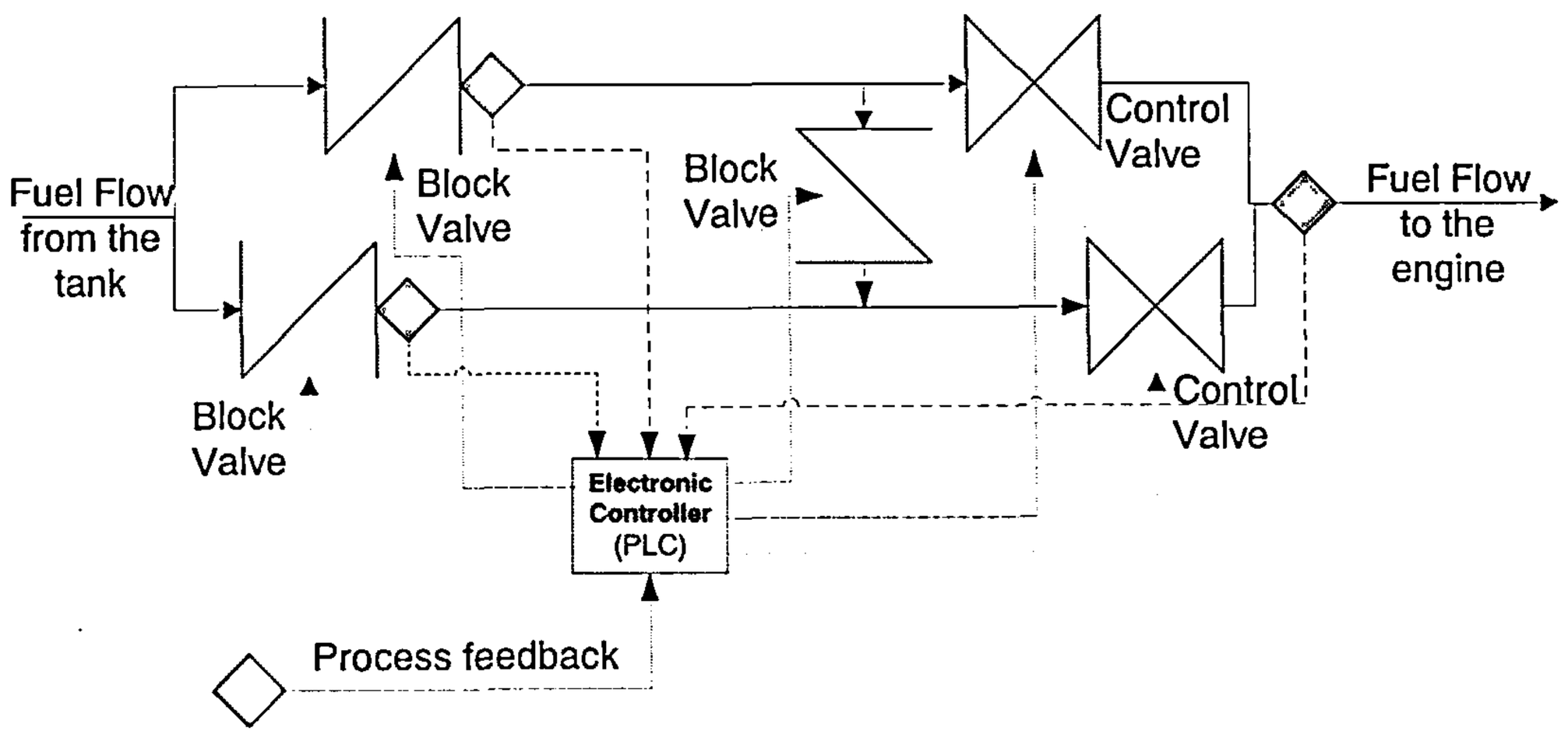


Figure 6-3: Architecture for the fuel system

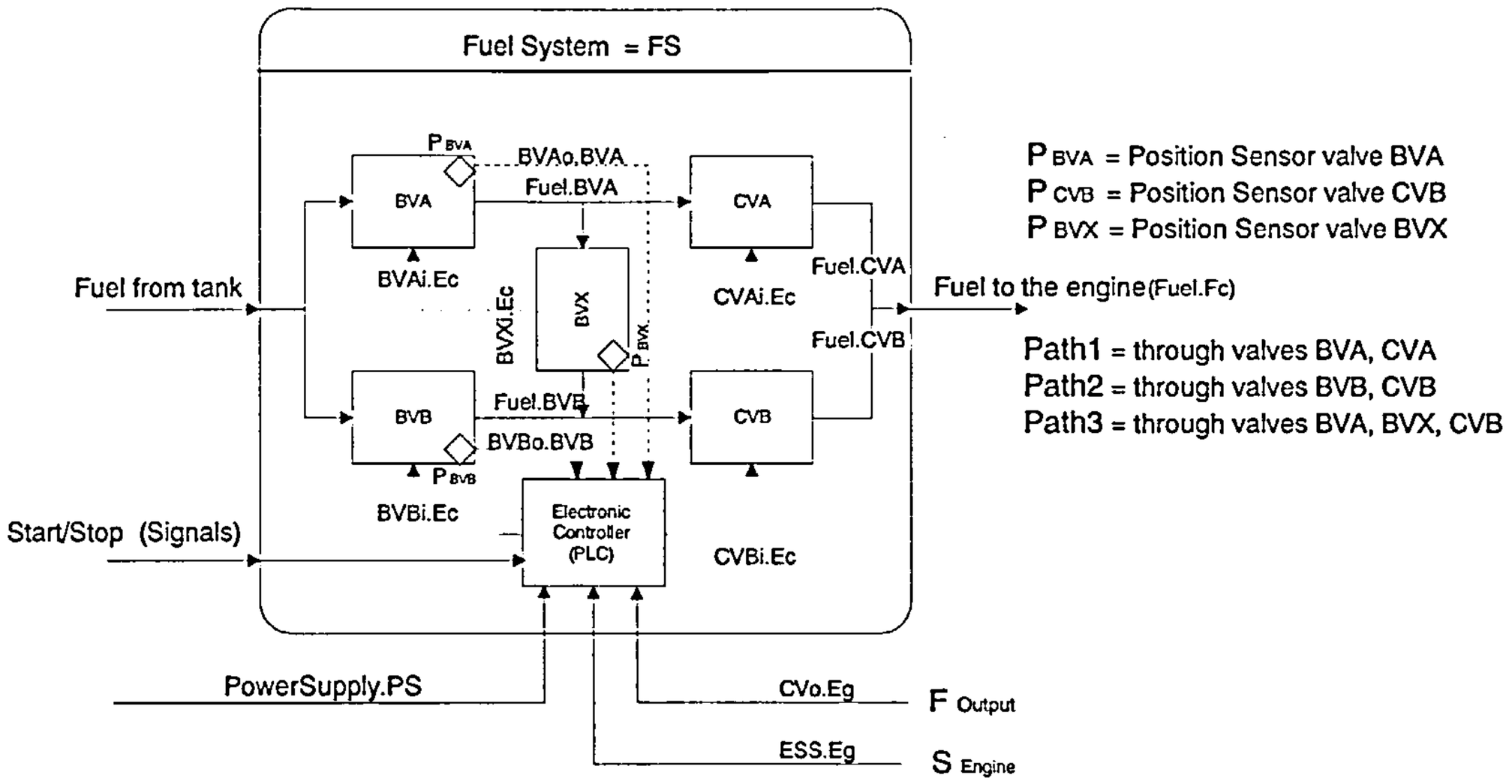


Figure 6-4:Details for the fuel system

Table 6-2 illustrates a fragment of the high-level FLASH analysis for the fuel system after the causes column has been completed. The table records one (and perha ps the most critical) of the functional failure modes of the system, the absence of flow in the line that feeds the engine (*No Flow – fuel to the engine*). According to the analysis, this event can be caused by an omission of the start signal (which causes a failure to start the system), a commission of the stop signal (which causes inadvertent shut-down of the system), or a combination of component failures that block all the available paths in the system (*No Flow-Path1, No Flow-Path2, No Flow-Path3*). For simplicity and economy of space, the table that we present here determines only the causes of failure in the third path (*No Flow – Path3*). The analysis shows that the flow in this path is disrupted either by internal failures of valves *BVA*, *BVX*, *CVB* or omissions of the signals that are continuously sent by the electronic controller to maintain block valves *BVA* and *BVX* open (*Omission – DP_{BVA}* , *Omission – DP_{BVX}*). The root causes of those events are further explored in the FLASH tables for the corresponding components (i.e. the valves and electronic controller).

FUEL SYSTEM		
Failure event	**Causes**	**Description**
No Flow – fuel from the FS to the engine	Omission – Start OR Commission – Stop OR (No flow – Path1 AND No flow – Path2 AND No flow – Path3) Where: No flow – Path3 = BVA failed closed OR BVX failed closed OR CVB failed closed OR Omission – DP_{BVA} OR Omission – DP_{BVX}	No Flow of fuel on the line that feeds the engine. The engine cannot start. No electric power is provided. It can be caused by an omission of the start signal, a commission of the stop signal or because there is no flow in the three possible paths that can be activated by the PLC. For simplicity, the causes of failure in one path only (Path3) are further explored only

Table 6-2: Fragment of the high-level FLASH analysis

Table 6-3 presents a fragment of the analysis for valve *BVA*. Here we can see that the condition *BVA failed closed* can be caused either by an electromechanical failure of the valve (*BVA failed to open*) or because the aperture of the valve is blocked (*BVA Plugged*).

Block Valve A		
Failure Event	**Description**	**Causes**
BVA failed closed	Valve *BVA* is inadvertently closed due to an internal hardware failure which causes it to fail to open or because it is plugged.	*BVA* failed to open OR *BVA* plugged

Table 6-3: Fragment of the FLASH table for *BVA* failed closed

Table 6-4, on the other hand, contains fragments of the analysis for the electronic controller (*PLC*). There we can see that the failure of the electronic controller to deliver the valve open signal to *BVA* (*Omission* – DP_{BVA}) can arise from different root failures in the two states of the system that the valve is active (i.e. in states: *Path1* and *Path3*). In both states, the event is caused by a number of low level internal hardware failures of the controller (*electronic controller output circuit stuck at zero; electronic controller register BVA_CVA stuck at zero; electronic controller register BVA stuck at zero; electronic controller logical operation negated*). However, in the first state and while the system delivers fuel through the initial path, the analysis shows that the event can also be caused by a failure of the sensor that monitors the position of valve *BVA*. Indeed, if the output of that sensor is stuck at zero (*Omission* -P_{BVA}), the controller will wrongly perceive this as an indication that *BVA* is closed. This in turn will trigger a inadvertent

transition of the control sequence to the second state (*Path2*), the deactivation of the initial path (including *BVA*) and the activation of the second path in the system.

On the other hand, if the system is in a transition towards the third state (for example because valve *BVB* has failed closed) the electronic controller may fail to open valve *BVA* simply because sensor P_{BVB} has failed to detect that *BVB* is failed closed (*Commission* – P_{BVB}). Here, we can observe that the analysis provides some useful pointers to particular subtle failures that may confuse the controller, corrupt the control sequence and eventually compromise the failure detection and recovery mechanisms of the system.

Electronic Controller (PLC)			
Failure event	**Description**	**Causes**	**Contributing Factor**
Omission – DP_{BVA}	The PLC fails to deliver the valve open signal to valve BVA, while the system is in state Path1 (in other words while it delivers fuel through valves BVA and CVA. It can be caused by a number of low level PLC hardware failures, or because there is a commission of the P_{BVA} (sensor) signal which causes an inadvertent exit from the *Path1* state.	PLC output circuit stuck at zero OR PLC register BVA_CVA stuck at zero OR PLC register BVA stuck at zero OR PLC logical operation negated OR Omission – P_{BVA}	Path1
Omission – DP_{BVA}	The PLC fails to deliver the valve open signal to valve BVA, while the system is in state Path3 (in other words while it delivers fuel through valves BVA, BVX and CVB. It can be caused by a number of low level PLC hardware failures, or because there is a commission of the P_{BVB} (sensor) signal which prevents the system of entering the *Path3* state.	PLC output circuit stuck at zero OR PLC register BVA_CVB stuck at zero OR PLC register BVA stuck at zero OR PLC logical operation negated OR Commission – P_{BVB}	Transition from Path2 to Path3

Table 6-4: Fragment of the FLASH table for the *PLC*

After writing causes for events propagated, it is possible to finish off the 5th column by considering the possibility to detect, recover from, or mitigate the effect of the event propagated, and eventually to issue recommendations for further developing lower level components and the *Maximum accepted likelihood* for critical causes. Table 6-5 represents a fragment of the FLASH table for the electronic controller after the completion of the 5th column.

Electronic Controller (PLC)				
Failure event	**Description**	**Causes**	**Contributing Factor**	**5th Column: Justification, Design Recommendations, Derived Safety Requirements**
Omission – DP_{BVA}	The PLC fails to deliver the valve open signal to valve BVA, while the system is in state Path1 (in other words while it delivers fuel through valves BVA and CVA. It can be caused by a number of low level PLC hardware failures, or because there is a commission of the P_{BVA} (sensor) signal which causes an inadvertent exit from the *Path1* state.	PLC output circuit stuck at zero OR PLC register BVA_CVA stuck at zero OR PLC register BVA stuck at zero OR PLC logical operation negated OR Omission – P_{BVA}	Path1	*Before design* **Recommendations** The failure cannot be handled. It has to be extremely unlikely **Effect max accepted likelihood** 10^{-5} on demand *After design* This failure cannot be recovered. **Detection** Sensor off valve BVA **Recovery** Unlikely **Recommendation** Software must be developed to comply with safety integrity level four **Max accepted likelihood for critical events in the Causes column.** P(PLC output circuit stuck at zero t) $<10^{-6}h^{-1}$ P(PLC register BVA_CVA stuck at zero) $<10^{-6}h^{-1}$ P(PLC register BVA stuck at zero) $<10^{-6}h^{-1}$ P(PLC logical operation negated) $<10^{-6}h^{-1}$ P(Omission – P_{BVA}) $<10^{-6}h^{-1}$
Omission – DP_{BVA}	The PLC fails to deliver the valve open signal to valve BVA, while the system is in state Path3 (in other words while it delivers fuel through valves BVA, BVX and CVB. It can be caused by a number of low level PLC hardware failures, or because there is a commission of the P_{BVB} (sensor) signal which prevents the system of entering the *Path3* state.	PLC output circuit stuck at zero OR PLC register BVA_CVB stuck at zero OR PLC register BVA stuck at zero OR PLC logical operation negated OR Commission – P_{BVB}	Transition from Path2 to Path3	*Before design* **Recommendations** The failure cannot be handled. It has to be extremely unlikely **Effect max accepted likelihood** 10^{-5} on demand *After design* This failure cannon be recovered. **Detection** Sensor off valve BVA **Recovery** Unlikely **Recommendation** Software must be developed to comply with safety integrity level four **Max accepted likelihood for critical events in the Causes column.** P(PLC output circuit stuck at zero t) $<10^{-6}h^{-1}$ P(PLC register BVA_CVB stuck at zero) $<10^{-6}h^{-1}$ P(PLC register BVA stuck at zero) $<10^{-6}h^{-1}$ P(PLC logical operation negated) $<10^{-6}h^{-1}$ P(Commission – P_{BVA}) $<10^{-6}h^{-1}$

Table 6-5: Fragment of the FLASH table after completion of the 5th column

Figure 6-5 shows a fragment of the fault tree that is mechanically generated from the hierarchy of FLASH tables for the fuel system. It can be seen how the interruption of flow in the line that feeds the engine (*No Flow – fuel to the engine*) can be caused by a combination of lower level malfunctions and basic component failure modes. Additionally, since the tree is constructed by parsing the hierarchy of FLASH tables, its construction validates whether the hierarchy is consistent and the required information is in the hierarchy of tables.

The following section presents the FLASH analysis of the fuel system in the integration and verification phase of the lifecycle.

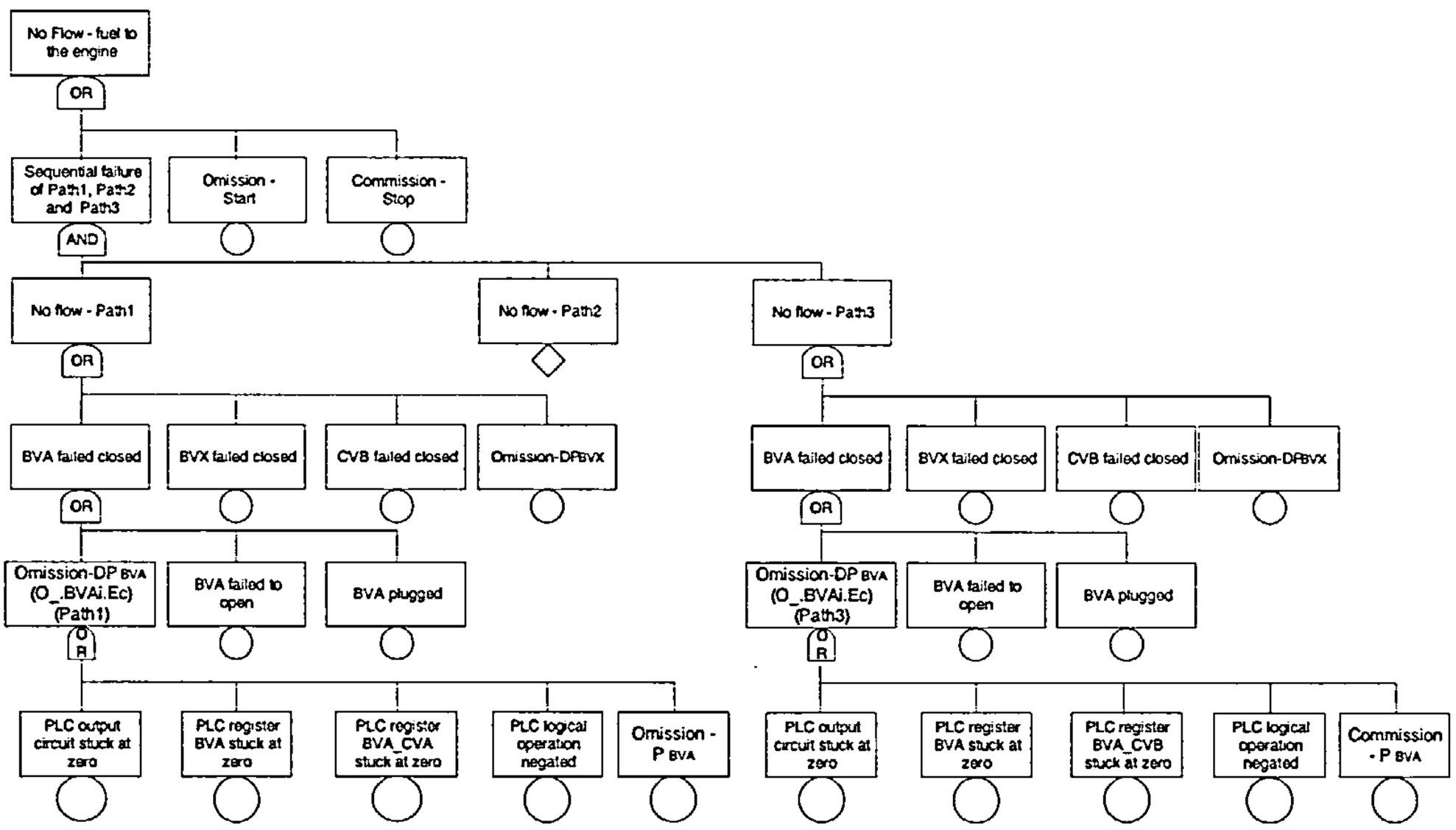


Figure 6-5: The fault tree for the failure event "*No Flow – fuel to the engine*"

6.1.2 Analysis in the Integration and Verification

The aim of the FLASH analysis in the integration and verification is to confirm that each module and component of the hierarchy meets the requirements, specifications and recommendations entered into the FLASH tables (i.e. the 5th column). The process of verification starts from modules at the lowest hierarchical level and proceeds towards the top functional level. Fault trees are built for each effect and evaluated using fault tree analysis. The structure of each tree is taken from the Causes column and by parsing tables of included modules. The likelihood of the top event is recorded in the "*FMEA results*" column as "*Likelihood of the effect*".

Among modules here considered, block valve A (*BVA*) and Electronic Controller (*EC*) are basic components. Hence they are analysed first. The likelihood of each event propagated by *BVA* can be calculated from information into Table 6-6. For example, *O_.Fuel.bva*, which tree is shown in Figure 6-6, is caused by two basic events (i.e. *Fail_to_open.bva* and *Plugged.bva*) and one incoming event (i.e. *O_.BVAi.Ec*). The

likelihood[28] of each basic event is calculated considering the mission time, the failure probability on demand and the failure rate as reported in Table 6-6. The likelihood of incoming event *O_.BVAi.Ec*, is obtained either, developing and evaluating the fault tree with that top event (i.e. represented in Figure 6-7) or, taken from the *Summary FMEA result* column in the table for the Electronic controller (i.e. Table 6-7). Hence, from Table 6-6 and Table 6-7 we determine that, in the transition between *Path2* and *Path3*, $P(Plugged.bva)=5*10^{-4}$, $P(Fail\_to\_open.bva)=4.13*10^{-4}$ and $P(O\_.BVAi.Ec)= 7*10^{-5}$. Consequently, the likelihood of event *O_.Fuel.BVA* (when the system is in transition between *Path2* and *Path3*) is $P(O\_.Fuel.bva) \cong 9.8*10^{-4}$.

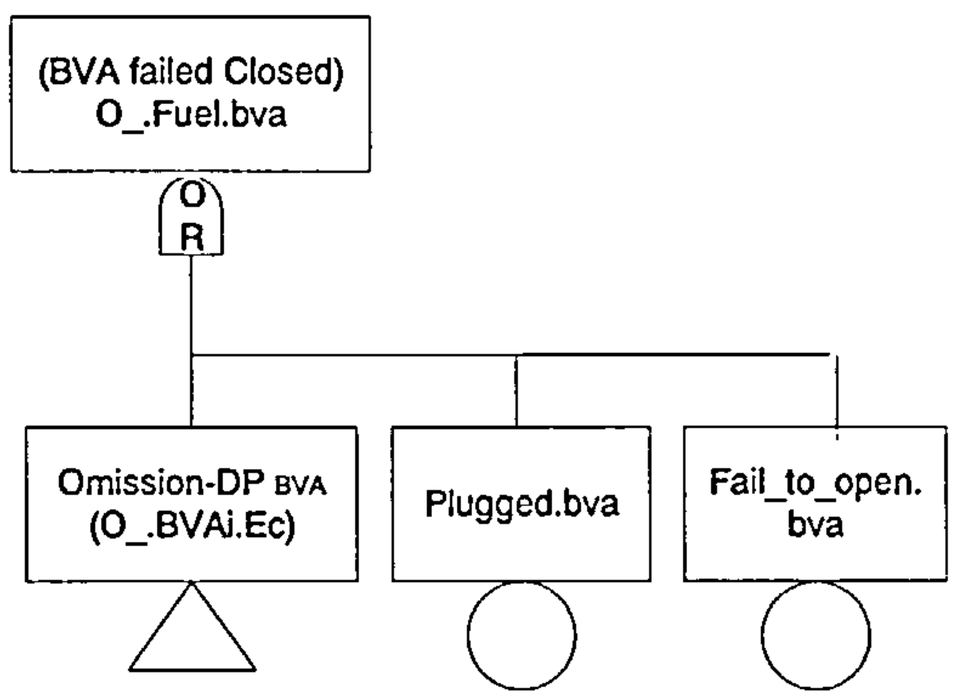


Figure 6-6: Tree for the event omission Fuel from BVA

This likelihood is recorded in the FMEA result column for the event along with information demonstrating that recommendations and constraints in the 5th column are met. We can see that in Table 6-7, the FMEA results column reports that the detection for the event *O_.Fuel.BVA* is possible from a speed sensor on the engine and from a flow sensor on the flow to the engine. Additionally, it says that the likelihood of this event is $9.8*10^{-4}$ in both states in which it may a rise. Since recommendations and constraints are actually met and the likelihood for the event *O_.fuel.bva* is less then the acceptable value into the 5th column the analysis moves further, another event in the same table is

[28] The likelihood that the event "*E*" happens during the mission time "Δt" is equal to the sum of the likelihood the event happens on demand "q_o" plus the likelihood the event happens during the mission time. If it is assumed an exponential distribution with constant rate "λ_E" for the event to happen, the equation for likelihood of the event becomes $P(\mathrm{E}, \Delta \mathrm{t}, \lambda_E) = qo + \left[1 - \frac{\lambda_E}{\Delta t} e^{-\lambda_E \Delta t}\right]$.

analysed. When all the events in that table have been considered, the analysis moves to another table at the same or higher hierarchical level. If all of the recommendations and constraints are met the analysis will eventually reach the highest functional level and validate the overall design. In any other case some modifications in the design will be necessary.

Same level	Causes	Effects & Consequences	Critica-lity	5th Column Justification, Design Recommendations, Derived Safety Requirements	Summary FMEA results
O_.Fuel.bva	Fail_to_open.bva OR O_.BVAi.ec OR Plugged.bva	The fuel goes through the valve when it should not	NA	***Before design*** **Recommendations** .... ***Effect max accepted likelihood*** The likelihood must be less than 10^{-5} during the mission time (100 hours) Detection: should be possible. i.e. from a speed sensor on the engine and from a flow sensor on the flow to the engine. Recovery: must be possible for single failure. Recommendations: Detection algorithm should know the status of the system and find suitable way to detect failures and recover them. Accepted Likelihood: λ (Fail_to_open.bva) < 10^{-3} h^{-1} λ (O_.BVAi.ec) < 10^{-4} h^{-1} λ (Plugged.bva) < 10^{-3} h^{-1}	
C_.Fuel.bva	Fail_to_close.bva OR Severe_Int_Leackage.bva OR C_BVAi.ec	The fuel doesen't go through the valve when it should	NA	***Before design*** ***Recommendations*** .... **Effect max accepted likelihood** 10^{-4} during the mission Detection: should be possible. i.e. from a speed sensor on the engine and from a flow sensor on the flow to the engine. Recovery: must be possible for single failure. Recommendations: Detection algorithm should know the status of the system and find suitable way to detect failures and recover them. Accepted Likelihood: *The likelihood must be < $3*10^{-5}$ during the mission*	
O_.BVAo.bva	O_.BVAo.bva	The sensor fails giving *output bit 0*	NA	***Before design*** **Recommendations** .... **Effect max accepted likelihood** 10^{-4} during the mission Detection: not possible . Recovery: must be possible for single failure. Recommendations: Detection algorithm should know the status of the system and find suitable way to detect failures and recover them. Accepted Likelihood: The likelihood must be < $3*10^{-5}$ during the mission	
C_.BVAo.bva	C_.BVAo.bva	The sensor fails giving output bit 1	NA	***Before design*** ***Recommendations*** .... **Effect max accepted likelihood** 10^{-4} during the mission Detection: not possible Recovery: must be possible for single failure. Recommendations: Detection algorithm should know the status of the system and find suitable way to detect failures and recover them. Accepted Likelihood: The likelihood must be < $3*10^{-5}$ during the mission	

Basic Events

Reliability data	Fail_to_open.bva	Severe_Int_Leackage.bva	Fail_to_close.bva	Plugged.bva	C_.BVAo.bva	O_.BVAo.bva	...
Description	Fail to open when required	There is a major leakage inside the valve	The valve fails to close	The valve is Plugged	The sensor fails giving *output bit 1*	The sensor fails giving *output bit 0*	
Failure Rate λ[1/h]	3e-006	3e-006	1.4e-006	5e-005	2e-007	2e-007	
Repair Rate μ[1/h]							
Failure Probability on Demand [q_d]	4.1e-004		4.1e-004				
Mean Time to Failure MTTF [h]							
Mission time [h]	100	100	100	100	100	100	

Table 6-6: Table for the block valve BVA

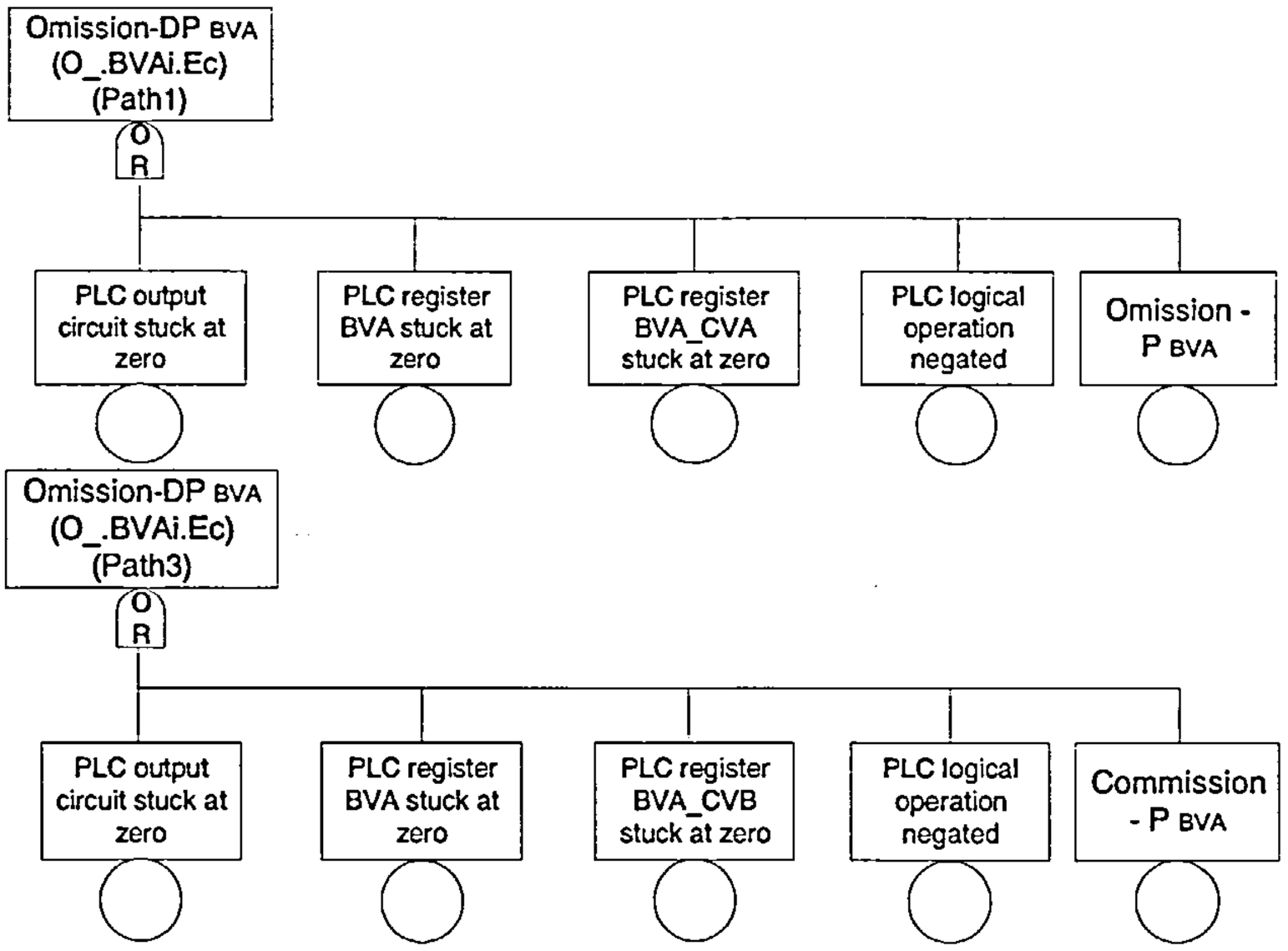


Figure 6-7: Trees for the event omission DP- BVA

Block Valve A				
Failure Event	**Causes**	**Description**	**Justification, Design Recommendation & Actions Required**	**Summary FMEA Results**
BVA failed closed (O_.Fuel .bva)	*BVA* failed to open OR *BVA* plugged OR O_.BVAi.ec	Valve *BVA* is inadvertently closed due to an internal hardware failure which causes it to fail to open or because it is plugged.	***Before design*** **Recommendations** .... **Effect max accepted likelihood** The likelihood must be less than 10^{-3} during the mission time (100 hours) **Detection**: should be possible. i.e. from a speed sensor on the engine and from a flow sensor on the flow to the engine. **Recovery**: must be possible for single failure. **Recommendations**: Detection algorithm should know the status of the system and find suitable way to detect failures and recover them. **Accepted Failure Rate:** λ (Fail_to_open.bva) $< 10^{-3}$ h^{-1} λ (O_.BVAi.ec) $< 10^{-4}$ h^{-1} λ (Plugged.bva) $< 10^{-3}$ h^{-1}	The detection is possible from a speed sensor on the engine and from a flow sensor on the flow to the engine. The average likelihood that the event O_.Fuel.bva has to be generated by internal events is $9.8*10^{-4}$ Likelihood is very near the upper bound for the acceptability.

Electronic Controller (PLC)					
Failure event	**Causes**	**Description**	**Contributing Factor**	**5th Column: Justification, Design Recommendations, Derived Safety Requirements**	**Summary FMEA Results**
Omission – DP_{BVA} (*O_.BVAi .ec*)	PLC output circuit stuck at zero OR PLC register BVA_CVB stuck at zero OR PLC register BVA stuck at zero OR PLC logical operation negated OR Commission – P_{BVB}	The PLC fails to deliver the valve open signal to valve BVA, while the system is in state Path3 (in other words while it delivers fuel through valves BVA, BVX and CVB. It can be caused by a number of low level PLC hardware failures, or because there is a commission of the P_{BVB} (sensor) signal which prevents the system of entering the *Path3* state.	Transition from Path2 to Path3	***Before design*** **Recommendations** The failure cannot be handled. It has to be extremely unlikely **Effect max accepted likelihood** 10^{-4} during the mission time ***After design*** This failure cannon be recovered. **Detection** Sensor off valve BVA **Recovery** Unlikely **Recommendation** Software must be developed to comply with safety integrity level four **Max accepted Failure Rate for critical events in the Causes column.** λ(PLC output circuit stuck at zero t)$<10^{-6}$ h^{-1} λ (PLC register BVA_CVB stuck at zero) $<10^{-6}$ h^{-1} λ (PLC register BVA stuck at zero) $<10^{-6}$ h^{-1} λ (PLC logical operation negated) $<10^{-6}$ h^{-1} λ (Commission – P_{BVA}) $<10^{-6}$ h^{-1}	The actual likelihood is $7*10^{-5}$ during the mission. The failure cannot be handled, however it is extremely unlikely. The software is developed to comply with safety integrity level four.

Table 6-7: BVA and Ec tables after the Integration and Verification

6.1.3 Common Cause Failures

Common cause failures are the subset of dependent failures that cannot be treated explicitly in the analysis. They arise when two or more events in a minimal cut set are coupled. FLASH addresses the study of common cause failures when there is all the information required for constructing fault trees and minimal cut sets for each tree can be obtained. Minimal cut sets susceptible to common cause failure are subsequently identified by comparing lifecycle information among their events. When events sharing the same coupling code in a peer lifecycle category are found, these events are considered coupled; hence that minimal cut set has to be considered for common cause failure analysis.

Table 6-8 reports the list of all the minimal cut sets of the fuel system responsible for the functional failure mode "*Fuel is required, but not provided .fs*". It can be seen that there are *96* minimal cut sets of the second order and *188* minimal cut sets of the third order. The FLASH method requires all these minimal cut sets to be analysed to find couplings. For example the first minimal cut sets represent the simultaneous failure of the first timer register (i.e. *T1*) and block valve A (i.e. *FTO_BVA = Fail to Open*). An accurate examination of Table 6-9 representing its couplings, reveals that these events are actually uncoupled hence the likelihood of this MCS is the simple product of the likelihood of each of its constituent events (i.e. P(*FTO_BVA)* * P(*FTO_BVA)= 1*10*$^{-8}$).

On the other hand, Table 6-10 for minimal cut set 81 (i.e. *C_BVAO_BVA*; *FTO_BVA*) clearly shows several couplings. This minimal cut set represents the simultaneous arising of two failure events inside the block valve *BVA*: a) the fail to open of the valve and the failure of the sensor monitoring the flow through the valve. These events are coupled since they share coupling codes in several lifecycle categories. For instance they have the same concept and design i.e. *Design architecture=DCA1*, *technological material equipment type=DTM1*, and *Specification=DS1*, additionally, they share the same installation fitter (i.e.*IIF1*), finally they the have same staff and procedures for operation (i.e. *OS1* and *OP1*) and maintenance (i.e. *MS1* and *MP1*). Hence, the likelihood for this minimal cut set has to be estimated using methods for common cause failure analysis.

LIST OF MINIMAL CUTSETS SORTED VS. ORDER

#	Minimal Cut Sets	#	Minimal Cut Sets	#	Minimal Cut Sets	#	Minimal Cut Sets
1	FTO_BVA T1	72	O O C O200 T2	143	C I005 O I010 O R403	214	C I001 C I C I005 O I010
2	FTO_BVA O I005	73	FTO_BVA O R405	144	C I C I005 O I010 O R403	215	C I C I001 C I C I005 O I010
3	FTO_BVA O I C I005	74	O R405 PLG_BVA	145	C ESSB1_EN O I010 O R403	216	C BVBO_BVB C I C I005 O I010
4	FTO_BVA O ESSB1_EN	75	O O200 O R405	146	O I005 O I010 O R403	217	C ESSB1_EN O I010 T3
5	C I C I006 FTO_BVA	76	O O C O200 O R405	147	O I010 O I C I005 O R403	218	0 R406 C ESSB1_EN O I010
6	C ESSB2_EN FTO_BVA	77	FTO_BVA O R402	148	O ESSB1_EN O I010 O R403	219	C ESSB1_EN C I001 O I010
7	FTO_BVA O I006	78	O R402 PLG_BVA	149	C I006 O I010 O R403	220	C ESSB1_EN C I C I001 O I010
8	FTO_BVA O I C I006	79	O O200 O R402	150	C I C I006 O I010 O R403	221	C BVBO_BVB C ESSB1_EN O I010
9	FTO_BVA O ESSB2_EN	80	O O C O200 O R402	151	C ESSB2 EN O I010 O R403	222	O I005 O I010 T3
10	PLG_BVA T1	81	C_BVAO_BVA FTO_BVA	152	O I006 O I010 O R403	223	0 R406 O I005 O I010
11	O R404 PLG_BVA	82	C_BVAO_BVA PLG_BVA	153	O I010 O I C I006 O R403	224	C I001 O I005 O I010
12	C I005 PLG_BVA	83	C_BVAO_BVA O O200	154	O ESSB2 EN O I010 O R403	225	C I C I001 O I005 O I010
13	C I C I005 PLG_BVA	84	C_BVAO_BVA O O C O200	155	O R401 O R403 T1	226	C BVBO_BVB O I005 O I010
14	C_ESSB1_EN PLG_BVA	85	C I000 FTO_BVA	156	O R401 O R403 O R404	227	O I010 O I C I005 T3
15	O I005 PLG_BVA	86	C I000 PLG_BVA	157	C I005 O R401 O R403	228	0 R406 O I010 O I C I005
16	C I006 FTO_BVA	87	C I000 O O200	158	C I C I005 O R401 O R403	229	C I001 O I010 O I C I005
17	O I C I005 PLG_BVA	88	C I000 O O C O200	159	C ESSB1_EN O R401 O R403	230	C I C I001 O I010 O I C I005
18	O ESSB1_EN PLG_BVA	89	C I C I000 FTO_BVA	160	O I005 O R401 O R403	231	C BVBO_BVB O I010 O I C I005
19	C I006 PLG_BVA	90	C I C I000 PLG_BVA	161	O I C I005 O R401 O R403	232	O ESSB1_EN O I010 T3
20	C I C I006 PLG_BVA	91	C I C I000 O O200	162	O ESSB1_EN O R401 O R403	233	0 R406 O ESSB1_EN O I010
21	C ESSB2_EN PLG_BVA	92	C I C I000 O O C O200	163	C I006 O R401 O R403	234	C I001 O ESSB1_EN O I010
22	O I006 PLG_BVA	93	FTO_BVA O R404	164	C I C I006 O R401 O R403	235	C I C I001 O ESSB1_EN O I010
23	O I C I006 PLG_BVA	94	C I005 FTO_BVA	165	C ESSB2 EN O R401 O R403	236	C BVBO_BVB O ESSB1_EN O I010
24	O ESSB2_EN PLG_BVA	95	C I C I005 FTO_BVA	166	O I006 O R401 O R403	237	C I006 O I010 T3
25	O O200 T1	96	C ESSB1_EN FTO_BVA	167	O I C I006 O R401 O R403	238	0 R406 C I006 O I010
26	O O200 O R404	97	C I C I001 C I C I005 O R401	168	O ESSB2 EN O R401 O R403	239	C I001 C I006 O I010
27	C I005 O O200	98	C BVBO_BVB C I C I005 O R401	169	FTO_BVB O I010 O R403	240	C I006 C I C I001 O I010
28	C I C I005 O O200	99	C ESSB1_EN O R401 T3	170	FTO_BVB O R401 O R403	241	C BVBO_BVB C I006 O I010
29	C ESSB1_EN O O200	100	0 R406 C ESSB1_EN O R401	171	O I010 O R403 PLG_BVB	242	C I C I006 O I010 T3
30	O I005 O O200	101	C ESSB1_EN C I001 O R401	172	O R401 O R403 PLG_BVB	243	0 R406 C I C I006 O I010
31	O I C I005 O O200	102	C ESSB1_EN C I C I001 O R401	173	O I010 O O201 O R403	244	C I001 C I C I006 O I010
32	O ESSB1_EN O O200	103	C BVBO_BVB C ESSB1_EN O R401	174	O O201 O R401 O R403	245	C I C I001 C I C I006 O I010
33	C I006 O O200	104	O I005 O R401 T3	175	O I010 O O C O201 O R403	246	C BVBO_BVB C I C I006 O I010
34	C I C I006 O O200	105	C BVBO_BVB O I C I005 O R401	176	O O C O201 O R401 O R403	247	C ESSB2 EN O I010 T3
35	C ESSB2 EN O O200	106	O ESSB1_EN O R401 T3	177	O I010 O R403 T2	248	0 R406 C ESSB2 EN O I010
36	O I006 O O200	107	0 R406 O ESSB1_EN O R401	178	O R401 O R403 T2	249	C ESSB2 EN C I001 O I010
37	O I C I006 O O200	108	C I001 O ESSB1_EN O R401	179	O I010 O R403 O R405	250	C ESSB2 EN C I C I001 O I010
38	O ESSB2 EN O O200	109	C I C I001 O ESSB1_EN O R401	180	O R401 O R403 O R405	251	C BVBO_BVB C ESSB2 EN O I010
39	O O C O200 T1	110	C BVBO_BVB O ESSB1_EN O R401	181	O I010 O R402 O R403	252	O I006 O I010 T3
40	O O C O200 O R404	111	C I006 O R401 T3	182	O R401 O R402 O R403	253	0 R406 O I006 O I010
41	C I005 O O C O200	112	0 R406 C I006 O R401	183	C BVAO_BVA O I010 O R403	254	C I001 O I006 O I010
42	C I C I005 O O C O200	113	C I001 C I006 O R401	184	C BVAO_BVA O R401 O R403	255	C I C I001 O I006 O I010
43	C ESSB1_EN O O C O200	114	C I006 C I C I001 O R401	185	C I000 O I010 O R403	256	C BVBO_BVB O I006 O I010
44	O I005 O O C O200	115	C BVBO_BVB C I006 O R401	186	C I000 O R401 O R403	257	O I010 O I C I006 T3
45	O I C I005 O O C O200	116	C I C I006 O R401 T3	187	C I C I000 O I010 O R403	258	0 R406 O I010 O I C I006
46	O ESSB1_EN O O C O200	117	0 R406 C I C I006 O R401	188	C I C I000 O R401 O R403	259	C I001 O I010 O I C I006
47	C I006 O O C O200	118	C I001 C I C I006 O R401	189	C I001 O I010 T1	260	C I C I001 O I010 O I C I006
48	C I C I006 O O C O200	119	C I C I001 C I C I006 O R401	190	0 R406 O I005 O R401	261	C BVBO_BVB O I010 O I C I006
49	C ESSB2 EN O O C O200	120	C BVBO_BVB C I C I006 O R401	191	C I001 O I005 O R401	262	O ESSB2 EN O I010 T3
50	O I006 O O C O200	121	C ESSB2 EN O R401 T3	192	C I C I001 O I005 O R401	263	0 R406 O ESSB2 EN O I010
51	O I C I006 O O C O200	122	0 R406 C ESSB2 EN O R401	193	C BVBO_BVB O I005 O R401	264	C I001 O ESSB2 EN O I010
52	O ESSB2 EN O O C O200	123	C ESSB2 EN C I001 O R401	194	O I C I005 O R401 T3	265	C I C I001 O ESSB2 EN O I010
53	FTO_BVA FTO_BVB	124	C ESSB2 EN C I C I001 O R401	195	0 R406 O I C I005 O R401	266	C BVBO_BVB O ESSB2 EN O I010
54	FTO_BVB PLG_BVA	125	C BVBO_BVB C ESSB2 EN O R401	196	C I001 O I C I005 O R401	267	O R401 T1 T3
55	FTO_BVB O O200	126	O I006 O R401 T3	197	C I C I001 O I C I005 O R401	268	0 R406 O R401 T1
56	FTO_BVB O O C O200	127	0 R406 O I006 O R401	198	C BVBO_BVB O I010 T1	269	C I001 O R401 T1
57	FTO_BVA PLG_BVB	128	C I001 O I006 O R401	199	O I010 T1 T3	270	C I C I001 O R401 T1
58	PLG_BVA PLG_BVB	129	C I C I001 O I006 O R401	200	0 R406 O I010 T1	271	C BVBO_BVB O R401 T1
59	O O200 PLG_BVB	130	C BVBO_BVB O I006 O R401	201	C I C I001 O I010 T1	272	O R401 O R404 T3
60	O O C O200 PLG_BVB	131	O I C I006 O R401 T3	202	O I010 O R404 T3	273	0 R406 O R401 O R404
61	FTO_BVA O O201	132	0 R406 O I C I006 O R401	203	0 R406 O I010 O R404	274	C I001 O R401 O R404
62	O O201 PLG_BVA	133	C I001 O I C I006 O R401	204	C I001 O I010 O R404	275	C I C I001 O R401 O R404
63	*O O200 O O201*	134	*C I C I001 O I C I006 O R401*	205	*C I C I001 O I010 O R404*	276	C BVBO_BVB O R401 O R404
64	O O201 O O C O200	135	C BVBO_BVB O I C I006 O R401	206	C BVBO_BVB O I010 O R404	277	C I005 O R401 T3
65	FTO_BVA O O C O201	136	O ESSB2 EN O R401 T3	207	C I005 O I010 T3	278	0 R406 C I005 O R401
66	O O C O201 PLG_BVA	137	0 R406 O ESSB2 EN O R401	208	0 R406 C I005 O I010	279	C I001 C I005 O R401
67	O O200 O O C O201	138	C I001 O ESSB2 EN O R401	209	C I001 C I005 O I010	280	C I005 C I C I001 O R401
68	O O C O200 O O C O201	139	C I C I001 O ESSB2 EN O R401	210	C I005 C I C I001 O I010	281	C BVBO_BVB C I005 O R401
69	FTO_BVA T2	140	C BVBO_BVB O ESSB2 EN O R401	211	C BVBO_BVB C I005 O I010	282	C I C I005 O R401 T3
70	PLG_BVA T2	141	O I010 O R403 T1	212	C I C I005 O I010 T3	283	0 R406 C I C I005 O R401
71	O O200 T2	142	O I010 O R403 O R404	213	0 R406 C I C I005 O I010	284	C I001 C I C I005 O R401

Table 6-8: List of the minimal cut sets generating the top event

For common cause failure analysis we will use the method proposed in chapter 5 as an extension of the FLASH method. First of all we calculate the total likelihood of each event during the mission, i.e. *P(FTO_BVA)=1.0e-3* and *P(C_BVAO_BVA)=1.0e-5*. Then, we calculate the likelihood of each lifecycle category to cause the event by applying expression 5-2 (the whole list of probabilities is reported in Table 6-11). For example, the likelihood of event *FTO_BVA* to be caused by an error in the lifecycle category *Manufacturer* (i.e. *MM1*) of valve *BVA* is:

$$P(FTO\_BVA_{MM1}) = P(FTO\_BVA)* 3/100 = 3*e\text{-}5$$

Then, we calculate the likelihood of each coupling cause by applying expression 5.23 to obtain the list of coupled probabilities shown in Table 6-12 (estimated by using $\beta = 1$).

Basic Events

		FTO_BVA	*TI*
Reliability Data	Failure Rate λ[1/h]	1e-6	1e-7
	Repair Rate μ[1/h]	-	-
	Mean Time to Failure MTTF [h]	-	-
	Failure Probability on demand	1e-3	-
	Mission time [h]	100	100

			Coupl. Code	%	Coupl. Code	%
Lifecycle Categories	Concept and Design	Design Architecture	DCA1	2	DCA3	8
		Technological Materials Equipment Type	DTM1	3	DTM3	7
		Specifications	DS1	1	DS3	6
	Manufacturing	Manufacturer	MM1	3	MM3	5
		Procedures	MPD1	5	MPD3	4
		Process	MPP1	1	MPP3	8
	Installation/ Integration And Test	Fitter	IIF1	3	IIF3	5
		Procedures	IIP1	6	IIP3	4
		Location	IIL1	2	IIL3	6
		Routing	IIR1	5	IIR3	7
	Operation	Staff	OS1	4	OS3	8
		Procedures	OP1	6	OP3	6
	Maintenance	Staff	MS1	7	MS3	2
		Procedures	MP1	8	MP3	3
	Test	Staff	TS1	6	TS3	1
		Procedures	TP1	8	TP3	3
	Calibration	Staff	CS1	7	CS3	5
		Procedures	CP1	6	CP3	1
	Environmental	Mechanical and Thermal	EMT1	5	EMT3	3
		Electrical and Corrosion	EEC1	4	EEC3	6
		Chemical and miscellaneous	ECM1	8	ECM3	2

Table 6-9: Coupling table for MCS 1 (FTO_BVA; TI)

		Basic Events	
		FTO_BVA	*C_BVAO_BVA*
Reliability Data	Failure Rate λ[1/h]	1e-6	1e-7
	Repair Rate μ[1/h]	-	-
	Mean Time to Failure MTTF [h]	-	-
	Failure Probability on demand	1e-3	-
	Mission time [h]	100	100

Lifecycle Categories			Coupl. Code	%	Coupl. Code	%
	Concept and Design	Design Architecture	DCA1	2	DCA1	8
		Technological Materials Equipment Type	DTM1	3	DTM1	7
		Specifications	DS1	1	DS1	6
	Manufacturing	Manufacturer	MM1	3	MM2	5
		Procedures	MPD1	5	MPD2	4
		Process	MPP1	1	MPP2	8
	Installation/ Integration And Test	Fitter	IIF1	3	IIF1	5
		Procedures	IIP1	6	IIP2	4
		Location	IIL1	2	IIL2	6
		Routing	IIR1	5	IIR2	7
	Operation	Staff	OS1	4	OS1	8
		Procedures	OP1	6	OP1	6
	Maintenance	Staff	MS1	7	MS1	2
		Procedures	MP1	8	MP1	3
	Test	Staff	TS1	6	TS2	1
		Procedures	TP1	8	TP2	3
	Calibration	Staff	CS1	7	CS2	5
		Procedures	CP1	6	CP2	1
	Environmental	Mechanical and Thermal	EMT1	5	EMT2	3
		Electrical and Corrosion	EEC1	4	EEC2	6
		Chemical and miscellaneous	ECM1	8	ECM2	2

Table 6-10: Coupling table for MCS 81 (C_BVAO_BVA FTO_BVA)

		Basic Events			
		FTO_BVA		*C_BVAO_BVA*	
Lifecycle Categories			Coupling Code	Likelihood that the Lifecycle category causes the basic event	Coupling Code	Likelihood that the Lifecycle category causes the basic event
	Concept and Design	Design Architecture	DCA1	2e-5	DCA1	8e-7
		Technological Materials Equipment Type	DTM1	3e-5	DTM1	7e-7
		Specifications	DS1	1e-5	DS1	6e-7
	Manufacturing	Manufacturer	MM1	3e-5	MM2	5e-7
		Procedures	MPD 1	5e-5	MPD 2	4e-7
		Process	MPP 1	1e-5	MPP 2	8e-7
	Installation/ Integration And Test	Fitter	IIF1	3e-5	IIF1	5e-7
		Procedures	IIP1	6e-5	IIP2	4e-7
		Location	IIL1	2e-5	IIL2	6e-7
		Routing	IIR1	5e-5	IIR2	7e-7
	Operation	Staff	OS1	4e-5	OS1	8e-7
		Procedures	OP1	6e-5	OP1	6^{e}-7
	Maintenance	Staff	MS1	7e-5	MS1	2e-7
		Procedures	MP1	8e-5	MP1	3e-7
	Test	Staff	TS1	6e-5	TS2	1e-7
		Procedures	TP1	8e-5	TP2	3e-7
	Calibration	Staff	CS1	7e-5	CS2	5e-7
		Procedures	CP1	6e-5	CP2	1e-7
	Environmental	Mechanical and Thermal	EMT1	5e-5	EMT2	3e-7
		Electrical and Corrosion	EEC1	4e-5	EEC2	6e-7
		Chemical and miscellaneous	ECM1	8e-5	ECM2	2e-7

Table 6-11: Probabilities that a coupling cause will rise an event in MCS 81

Terms that have to be substituted	Dependent likelihood expression	Likeli-hood
$P(FTO\_BVA_{DCA1})P(C\_BVAO\_BVA_{DCA1})$	{P(FTO_BVA$_{DCA1}$) -Min[(P(FTO_BVA$_{DCA1}$);P(C_BVAO_BVA$_{DCA1}$)]} *{P(C_BVAO_BVA$_{DCA1}$) -Min[(P(FTO_BVA$_{DCA1}$);P(C_BVAO_BVA$_{DCA1}$)]} + Min[(P(FTO_BVA$_{DCA1}$);P(C_ BVAO_BVA$_{DCA1}$)]	≅2e-7
$P(FTO\_BVA_{DTM1})P(C\_BVAO\_BVA_{DTM1})$	{P(FTO_BVA$_{DTM1}$) -Min[P(FTO_BVA$_{DTM1}$);P(C_BVAO_BVA$_{DTM1}$)]} *{P(C_BVAO_BVA$_{DTM1}$) -Min[(P(FTO_BVA$_{DTM1}$);P(C_BVAO_BVA$_{DTM1}$)]} +Min[(P(FTO_BVA$_{DTM1}$); P(C_BVAO_BVA$_{DTM1}$)]	≅3e-7
$P(FTO\_BVA_{DS1})P(C\_BVAO\_BVA_{DS1})$	{P(FTO_BVA$_{DS1}$) -Min[P(FTO_BVA$_{DS1}$);P(C_BVAO_BVA$_{DS1}$)]} *{P(C_BVAO_BVA$_{DS1}$) -Min[(P(FTO_BVA$_{DS1}$);P(C_BVAO_BVA$_{DS1}$)]} +Min[(P(FTO_BVA$_{DS1}$);P(C_BVAO_BVA$_{DS1}$)]	≅1e-7
$P(FTO\_BVA_{IIF1})P(C\_BVAO\_BVA_{IIF1})$	{P(FTO_BVA$_{IIF1}$) -Min[P(FTO_BVA$_{IIF1}$);P(C_BVAO_BVA$_{IIF1}$)]} *{P(C_BVAO_BVA$_{IIF1}$) -Min[(P(FTO_BVA$_{IIF1}$);P(C_BVAO_BVA$_{IIF1}$)]} +Min[(P(FTO_BVA$_{IIF1}$);P(C_BVAO_BVA$_{IIF1}$)]	≅3e-7
$P(FTO\_BVA_{OS1})P(C\_BVAO\_BVA_{OS1})$	{P(FTO_BVA$_{OS1}$) -Min[P(FTO_BVA$_{OS1}$);P(C_BVAO_BVA$_{OS1}$)]} *{P(C_BVAO_BVA$_{OS1}$) -Min[(P(FTO_BVA$_{OS1}$);P(C_BVAO_BVA$_{OS1}$)]} +Min[(P(FTO_BVA$_{OS1}$); P(C_BVAO_BVA$_{OS1}$)]	≅4e-7
$P(FTO\_BVA_{OP1})P(C\_BVAO\_BVA_{OP1})$	{P(FTO_BVA$_{OP1}$) -Min[P(FTO_BVA$_{OP1}$);P(C_BVAO_BVA$_{OP1}$)]} *{P(C_BVAO_BVA$_{OP1}$) -Min[(P(FTO_BVA$_{OP1}$);P(C_BVAO_BVA$_{OP1}$)]} +Min[(P(FTO_BVA$_{OP1}$);P(C_BVAO_BVA$_{OP1}$)]	≅6e-7
$P(FTO\_BVA_{MS1})P(C\_BVAO\_BVA_{MS1})$	{P(FTO_BVA$_{MS1}$) -Min[P(FTO_BVA$_{MS1}$);P(C_BVAO_BVA$_{MS1}$)]} *{P(C_BVAO_BVA$_{MS1}$) -Min[(P(FTO_BVA$_{MS1}$);P(C_BVAO_BVA$_{MS1}$)]} +Min[(P(FTO_BVA$_{MS1}$);P(C_BVAO_BVA$_{MS1}$)]	≅7e-7
$P(FTO\_BVA_{MP1})P(C\_BVAO\_BVA_{MP1})$	{P(FTO_BVA$_{MP1}$) -Min[P(FTO_BVA$_{MP1}$);P(C_BVAO_BVA$_{MP1}$)]} *{P(C_BVAO_BVA$_{MP1}$) -Min[P(FTO_BVA$_{MP1}$);P(C_BVAO_BVA$_{MP1}$)]} +Min[(P(FTO_BVA$_{MP1}$);P(C_BVAO_BVA$_{MP1}$)]	≅8e-7
Sum of all the dependent probabilities		*≅4e-6*

Table 6-12: Products that have to be substituted in equation 5-1

Substituting these values into expression 5-7, we obtain the likelihood of the minimal cut set, which is *4e-6* during the mission time. This value is almost two orders bigger than the likelihood calculated without considering couplings, that is *1.e-8.*

Under same conditions, if we had applied the Beta factor parametric method to MCS 81 we would have obtained a likelihood of the same order (only four time bigger than it was obtained with the FLASH method). This can be seen from the following expression.

$$P(FTO\_BVA)*P(C\_BVAO\_BVA)+Beta*Min[P(FTO\_BVA);P(C\_BVAO\_BVA)]=$$
$$\cong 1.0e\text{-}3 * 1.0e\text{-}5 + 0.1 * Min[1.0e\text{-}3,1.0e\text{-}5]$$
$$\cong 1.0e\text{-}6$$

Hence, the FLASH method contributes studying and evaluating inter-component dependencies since:

1) Only real couplings among events in a MCS are considered.
2) Quantitative estimation of the likelihood of MCS with coupled events is based only on real couplings hence the figure for the likelihood of MCS is more accurate and realistic.
3) For a given MCS and under same conditions, the FLASH method obtains a similar value for the likelihood of the MCS as the β factor parametric model.

6.2 Computer-Assisted Braking system

6.2.1 Description

The Computer-Assisted Braking (CAB) system that we address in this section is a model of a concept being considered for employment in modern cars to enhance braking performance and vehicle safety. It is meant to provide three functions in addition to traditional brakes. The *Anti-lock braking*, widely known as ABS, that detects the onset of wheel lock up (which would result in skidding) and momentarily release the brakes to allow the wheel to turn and regain grip. The *Emergency stop detection and enhancement* that detects the rapid pedal movement associated with an emergency stop, and automatically maximises the braking used. The *Load-compensated braking* that measures the weight on the vehicle's suspension to ensure that a given pressure on the brake pedal provides the same degree of braking, regardless of how heavily the vehicle is loaded, or how the load is distributed.

The braking system has to meet legal requirements therefore it must retain a direct hydraulic link from the brake pedal to the brakes so that, in the event of complete failure of the computerised parts of the system, the driver will still have minimal braking functionality. Additionally, to allow the system to control braking individually to each wheel, there must be four separate hydraulic lines, the pressure in each of which can be altered (reduced as well as increased) by computer controlled actuators. This means that, if the system fails, the actuators must be guaranteed to return to a "neutral" position, where they are neither increasing nor decreasing the driver's braking effort. Therefore, it was decided to fit each hydraulic line to each wheel with a feedback pressure sensor to allow closed-loop control. The brake pedal is to be fitted with two sensors, each returning a value indicating how hard the pedal has been pressed. Axles of the vehicle

are to be fitted with two pressure transducers to measure the load on the vehicle. Finally, each wheel is to be fitted with a rotation sensor to be used for lock-up detection for anti-lock braking functions. The braking system context diagram is shown in Figure 6-8.

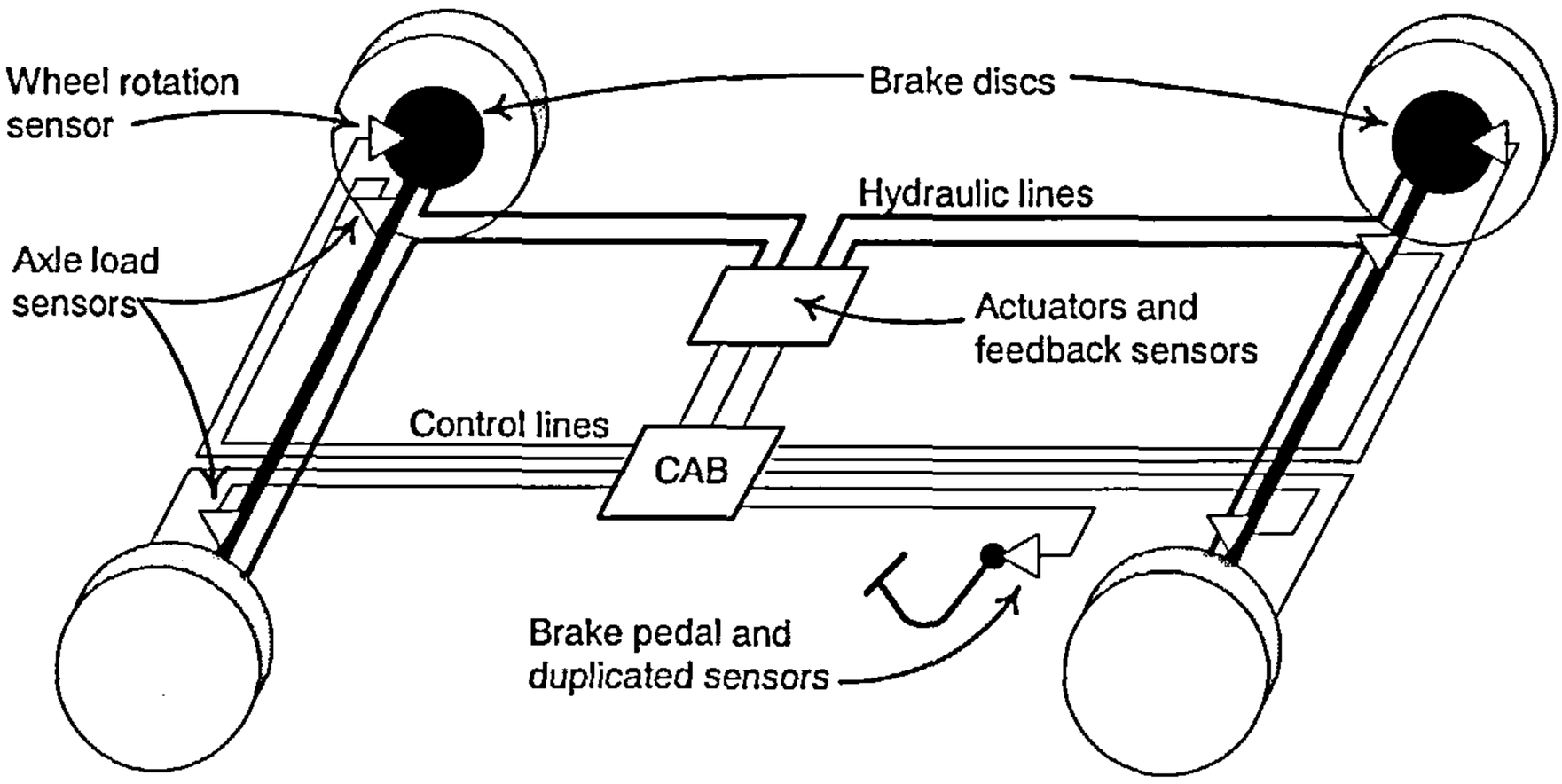


Figure 6-8: CAB system context diagram

Since the braking system has to be implemented in a vehicle it has to meet some basic performance requirements that are derived from the vehicle dynamics. It has to achieve the maximum available braking pressure in less than *400 ms*, additionally, it has to decrease the pressure delivered such that brakes are fully released from maximum pressure in less than *200 ms*. Finally, the maximum permissible latency from pedal movement to brake effect is *20 ms*.

Hence, it was decided to fit two output controllers to drive the hydraulic actuators, each controlling the actuators for a diagonal pair of wheels. These controllers are already designed (commercial of the shelf). Each takes required output commands over a duplicated Controller Area Network (CAN) bus[29] link, and converts these to the required

[29]Controller Area Network (CAN) is a high-speed local area network protocol designed to have predictable properties, and to be suitable for control applications. In CAN, data is transmitted as a *message* consisting of between 1 and 8 bytes. Messages are sent via *stations*, which police access to the bus. Typically, each station can buffer a maximum of 14 incoming or outgoing messages. Each message source is assigned a unique identifier, represented as an 11-bit number. This identifier is used to filter messages and assign priorities to the messages. Messages have a period, or minimum inter-arrival time, which they inherit from the sending task. If either the sender or receiver of a message detects an error, the sender station is signaled and re-transmits the message.

See ISO Draft International Standard *Road Vehicles - Interchange of Digital Information - Controller Area Network (CAN) for High Speed Communication*, ISO DIS 11898, 1992

or K. Tindell, A. Burns and A. Wellings "Calculating Controller Area Network (CAN) Message Response Times" in *Proceedings of the 1994 IFAC Workshop on Distributed Computer Control.*

electrical output to the actuators. Each controller works on a cyclic basis, and will only alter its outputs once per period of the cycle. If more than one output command message were received in a period, only the first will be used; subsequent messages will be ignored. If no command message is received during any period, outputs will be "frozen" at the last value received until a new command is received.

Because of the highly critical nature of the application, the manufacturer has imposed further design constraints. The *computer hardware implementing the design must have redundancy*. Additionally, the system must exhibit *graceful degradation* in case of failures. In particular, there must be a "fallback" algorithm which is capable of running independently on any of the redundant hardware units, and which is capable of providing minimum braking functionality (i.e. pressure simply proportional to pedal travel) on its own. Finally there must be redundancy in any communication system used between the hardware units.

6.2.2 Analysis in the Decomposition and Design

The safety analysis of the CAB started carrying out a preliminary hazard analysis (PHA) on the computerised braking system. The analysis concluded that any deviation from the specified behaviour is potentially hazardous. Seven specific failure modes were identified, and assigned criticalities as summarised in Table 6-13. Most of these failure modes are already present in a hydraulic braking system. They are caused by loss of hydraulic fluid or ingress of air, water or other contaminants into brake lines. Failure modes that the PHA identified as unique to a computer-assisted system are the *unexpected application of brake* (c) and *uneven braking* (f).

ID	Effect description	Risk class
A	Complete lack of braking	Catastrophic
B	Lock up (1-4 wheels, 1-2 axles)	Catastrophic
C	Unexpected application / release of brakes	Catastrophic
D	Braking response not proportional to demand	Major
E	Tardy response (time from demand to brake effect, slow rate of change in response to demand)	Major
F	Uneven braking (pressures vary "wildly" in response to constant demand)	Major
G	Unequal braking (1-3 wheels brake less or more than required)	Major

Table 6-13: CAB failure modes identified by PHA

Table 6-14 represents the FLASH table that corresponds to the same analysis level as the PHA. For economy of space we have put the complete table, as it looks after the completion of the FLASH analysis. However during the PHA only the first, third and fourth columns are completed. It can be noticed that these columns contain the same information as Table 6-13.

Instance = Brakes		Component Type = Brakes		Periodicity = Sporadic	Tag = Brakes
Event propagated	**Causes**	**Description**	**Criticality**	**5th Column: Justification, Design Recommendations, Derived Safety Requirements**	**Comments (FMEA)**
a_.Braking_func .brakes	V_.Sens_in.Sens OR (V_.FN_RO.cab AND V_.FO_RN.cab)	Complete lack of braking	Catastrophic	*Before design* **Recommendations** Sensors should be fail-safe and without SPF. Hardware redundancies must be implemented. **Effect max accepted likelihood** 10^{-7} during the mission time	According to the implementation and integration proposed/hypothesised, there are no SPFs and the expected rate for the event is 0.6E-7 [1/h] V_.Sens_in.Sens rate is < 0.4E-7 [1/h] V_.FN_RO.cab rate is < 0.1E-3 [1/h] V_.FO_RN.cab rate is < 0.1E-3 [1/h]
b_.Braking_func .brakes	V_.Sens_in.Sens OR (V_.FN_RO.cab AND V_.FO_RN.cab)	Lock-up (1-4 wheels, 1-2 axles)	Catastrophic		
c_.Braking_func .brakes	V_.Sens_in.Sens OR (V_.FN_RO.cab AND V_.FO_RN.cab)	Unexpected application/release of brakes	Catastrophic	*After design* **Detection** Not possible **Recovery** The driver will try to correct with the steering wheel **Recommendation** Software must be developed to comply with *safety integrity level four* **Max accepted likelihood for critical events in the Causes column.** The rate of V_.Sens_in.Sens must be < .5E-7 [1/h], each V_.FN_RO.cab and V_.FO_RN.cab rate must be <0.4E-3 [1/h].	
d_.Braking_func .brakes	V_.Sens_in.Sens OR (V_.FN_RO.cab OR V_.FO_RN.cab)	Braking response not proportional to demand	Major	*Before design* **Recommendations** Sensors should be fail-safe and without SPF. **Effect max accepted rate** <0.4E-3 [1/h]	Sensors are fail-safe and without SPF. d_.Braking_func.brakes rate is < 0.3E-3 [1/h]
e_.Braking_func .brakes	V_.Sens_in.Sens OR FN_RO.cab OR V_.FO_RN.cab	Tardy response (time from demand to brake effect, slow rate of change in response to demand)	Major		Sensors are fail-safe and without SPF. e_.Braking_func.brakes rate is < 0.3E-3 [1/h]
f_.Braking_func .brakes	V_.Sens_in.Sens OR FN_RO.cab OR V_.FO_RN.cab	Uneven braking (pressures vary "wildly" in response to constant demand)	Major	*After design* **Detection** Possible **Recovery** The driver will try to correct with the steering wheel **Recommendation** Software must be developed to comply with safety integrity level four **Max accepted failure rate for critical events in the Causes column.** Each rate of: E_.FN_RO.cab, E_.FO_RN.cab, L_.FN_RO.cab, and L_.FO_RN.cab must be < $0.1*10^{-3}$ [1/h].	Sensors are fail-safe and without SPF. f_.Braking_func.brakes rate is < 0.3E-3 [1/h]
g_.Braking_func .brakes	V_.Sens_in.Sens OR V_.FN_RO.cab OR V_.FO_RN.cab OR E_.FN_RO.cab OR E_.FO_RN.cab OR L_.FN_RO.cab OR L_.FO_RN.cab	Unequal braking. (1-3 wheels brake less or more than required)	Major		Sensors are fail-safe and without SPF. g_.Braking_func.brakes rate is $<0.6*10^{-4}$ [1/h] E_.FN_RO.cab rate$<0.7*10^{-4}$ [1/h] E_.FO_RN.cab rate$<0.7*10^{-4}$ [1/h] L_.FN_RO.cab rate$<0.9*10^{-4}$ [1/h] L_.FO_RN.cab rate$<0.9*10^{-4}$ [1/h]

Table 6-14: Top level FLASH table

After the preliminary hazard analysis a system design is proposed. It consists of three independent hardware "channels". Each comprises a processor, with necessary memory, hardware timer and counter registers for scheduling and accurate interval timing of sensor input, signal processing electronics to handle the inputs from the sensors and dual CAN bus stations. Sensor inputs are duplicated, and hard-wired to each board. The three boards communicate only via the duplicated CAN buses, which are also used to send output to the output controllers. Figure 6-9 shows the high level structure of the

proposed braking system hardware and flows delivered by some modules. With this information it is possible to complete the 2nd and 5th columns in Table 6-14 and begin development and analysis of lower levels components. Table 6-15 is a fragment of the FLASH table for the CAB device. It represents rows regarding the delivery of commands to brakes actuators of the front near side and rear offside wheels (*FN_RO*). The table shows how Omission (*O*), Commission (*C*), Early (*E*), Late (*L*) and Value (*V*) failures in the output module (*Output.Mod1*) directly result in similar failures to be propagated by the CAB. Whilst, value failure propagated form the CAB to the front near side and rear offside wheels of the car (i.e. *V_.FN_RO.CAB*) can be caused by a fault in both busses (*V_.Busses.2B*) or in the three processors (*V_.Pair_1.3P*). More specifically, the failure logic underneath propagation of *V_.Busses.2B* and *V_.Pair_1.3P* is further considered in Table 6-16 representing the so called *group of events* (4.2.3).

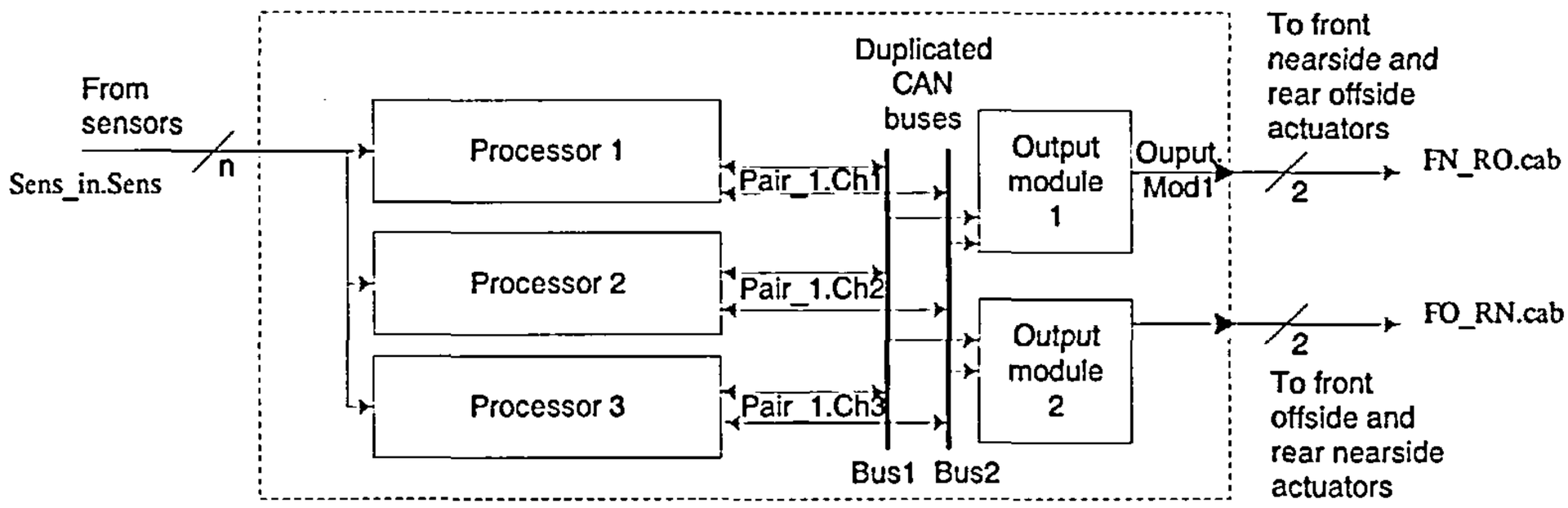


Figure 6-9: Structure of the proposed braking system hardware

The development of sub-modules proceeds. The control system of the CAB is developed to be cyclic, with some processes running at regular intervals to provide the necessary response characteristics. The scheduling of processes is *periodic* for processes that run at regular intervals (offsets is used to control the order and time interval between process running at the same periodic rate), while it is *sporadic* for tasks that run in response to some events e.g. the arrival of a signal from another process.

Instance = CAB	Component Type = CAB		Periodicity = Perodic	Tag = CAB
Event propagated	Causes	Description	5th Column: Justification, Design Recommendations, Derived Safety Requirements	Comments (FMEA)
O_. FN_RO.cab	O_.Output.Mod1	Omission braking for 2/4 wheels	This event is stopped by the output module	According to the implementation and integration proposed/hypothesised the expected rate for the event is $0.3*10^{-4}$ [1/h]
C_. FN_RO.cab	C_.Output.Mod1	Commission braking for 2/4 wheels	This event is stopped by the output module	The expected rate for the events is $0.3*10^{-4}$[1/h]
E_. FN_RO.cab	E_.Output.Mod1	Early braking for 2/4 wheels	The event must exhibit a rate < $0.3*10^{-3}$[1/h]. The E_.Output.Mod1 rate must be<$0.1*10^{-3}$[1/h]	The expected rate for the events is $0.9*10^{-4}$[1/h]
L_. FN_RO.cab	L_.Output.Mod1	Late braking for 2/4 wheels	The event must exhibit a rate < $0.3*10^{-3}$[1/h]. The L_.Output.Mod1 rate must be<$0.1*10^{-3}$[1/h].	The expected rate for the events is $0.9*10^{-4}$[1/h]
V_. FN_RO.cab	V_.Output.Mod1 OR V_.Pair_1.3P OR V_.Busses.2B	Wrong braking value for 2/4 wheels	The event must exhibit a rate < $0.4*10^{-3}$[1/h]. V_.Output.Mod1 and V_.Pair_1.3P must be<$0.2*10^{-3}$[1/h]; V_.Busses.2B must be <$0.1*10^{-7}$ [1/h]	The expected rate for the event is < $0.1*10^{-3}$[1/h]. V_.Output. Mod1 and V.Pair_1.3P rate is <$0.5*10^{-4}$[1/h]; V_.Busses.2B is <$0.9*10^{-8}$[1/h].

Table 6-15: Flash table for the CAB system

GROUPS OF EVENTS				
GOE	Causes	Description	Justification, Design Recommendations, Action required	Comments (FMEA)
V_.Pair_1.3P	V_.Pair_1.ch1 AND V_.Pair_1.ch2 AND V_.Pair_1.ch3	Wrong pressure value delivered to brakes (All three channels fail giving the same value to the output module)	The event must exhibit a rate<$0.5*10^{-4}$ [1/h] V_.Pair_1.ch1, V_.Pair_1.ch2 and V_.Pair_1.ch3 rate must be<$0.1*10^{-4}$ [1/h]	The expected rate for the events (considering common cause failures) is $0.3*10^{-7}$[1/h]
V_.Busses.2B	(Fail_silent.Bus.Bus1 AND V_.Bus.Bus2) OR (V_.Bus.Bus1 AND Fail_silent.Bus.Bus2) OR (V_.Bus.Bus1 AND V_.Bus.Bus2)	All data exchange are messed up	The event must exhibit a rate < $0.1*10^{-7}$	The expected rate for the events (considering common cause failures) is $0.9*10^{-8}$[1/h]

Table 6-16: Group of even table for FLASH Table 6-15

To meet requirement for a maximum *20 ms* latency from pedal movement to brake effect, a *10 ms* period has been selected for the main periodic tasks. This ensures that a complete iteration of the main control loop will always complete within the time limit. The proposed top-level functional decomposition of the CAB system is shown in Figure 6-10. The notation used to show the inter-process communications is based on DORIS /

DIA[30], a development of MASCOT. The communications protocols used are summarised in Table 6-17.

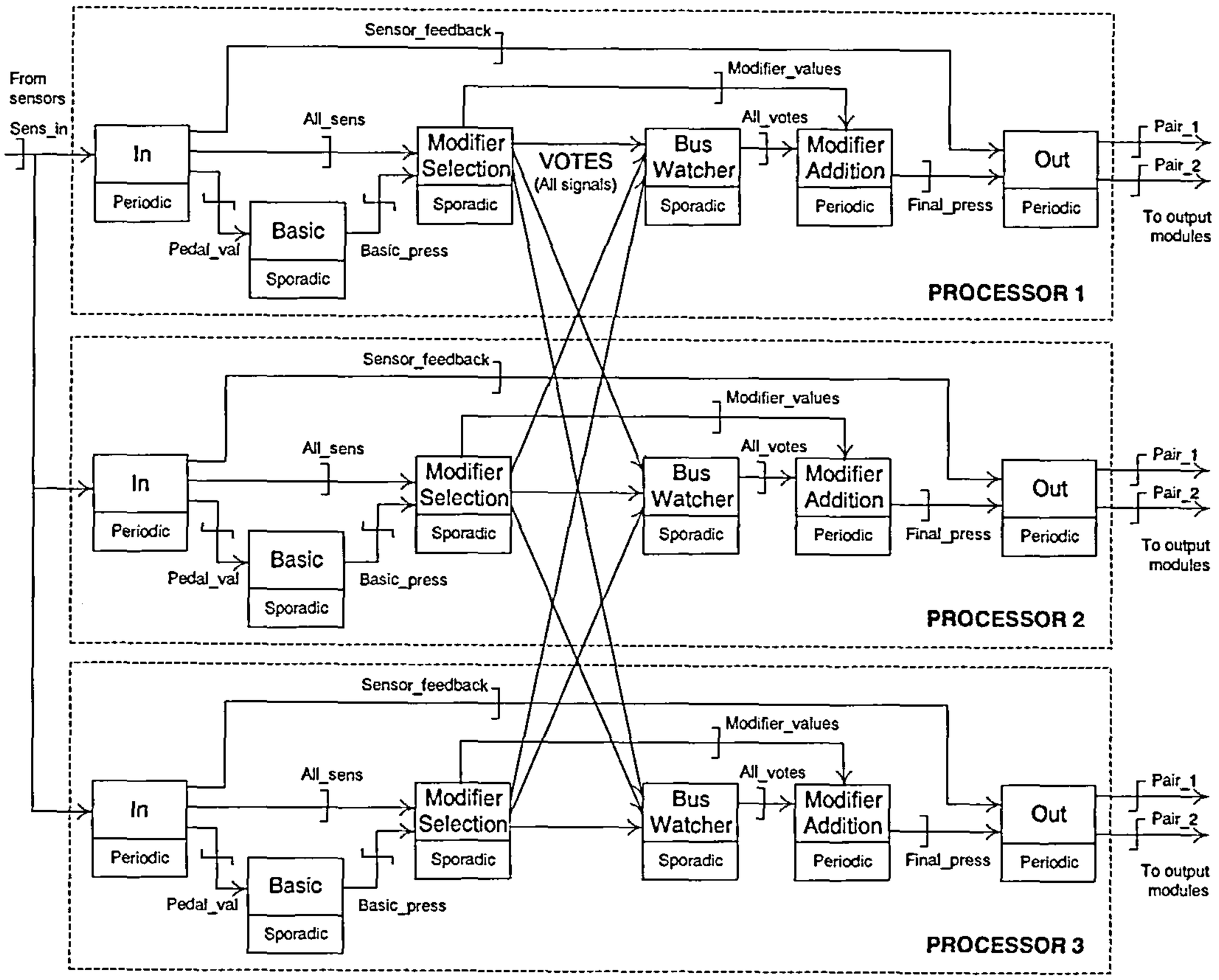


Figure 6-10: Functional block diagram of the CAB system

Interaction Name	Symbol	Inputs	Outputs	Writer can be held up	Reader can be held up
Signal		One	One	N	Y
Pool		One	One or more	N	N
Multicast Signal		One	Many, distributed	N	Y

Table 6-17: Communications protocols

[30] H. Simpson *Methodological and Notational Conventions in DORIS Real Time Networks* British Aerospace Dynamics Division, 1994

In each period of the cycle, each channel (processor) calculates a basic braking value from the pedal sensor value. The values of all the other sensors are then used to determine whether the three modifiers (i.e. Anti-lock, Emergency stop enhancement and Load compensation) are required in the current cycle, and calculate the necessary changes in braking for each wheel to implement these modifiers. The three channels then vote on which modifiers to add. For a modifier to be added, at least two of the channels must have determined that it is required. The actual amount by which braking at each wheel is to be increased or decreased to implement the modifiers is not communicated, as it is so dependent on the precise value of the sensors read by each channel. This means that, if one channel has not calculated a value for a modifier that has been voted necessary (i.e. the other two channels require it), then this channel must revert to the basic value initially calculated.

The system is scheduled so that the *OUT* processes on the three processors should always complete in the order *Processor 1 - Processor 2 - Processor 3*. In the case where a channel has had to revert to a basic value, this will be output later than the normal completion time of all three *OUT* processes, to ensure that the basic value is not used if an enhanced value is available from another channel. This ensures that *Processor 1* normally controls the braking, avoiding the possible fluctuations caused by switching between channels, as would be the case if output order were not pre-determined. Data types of all the flows in the design shown in Figure 6-10 are shown in Table 6-17. The functionality of each process is reported in Table 6-19.

The use of a pre-emptive, priority-based scheduler is proposed for the system. This means that, if a low-priority process is executing, and a higher priority task becomes runable, the low-priority task will be suspended until the high priority task has completed. Figure 6-11 shows roughly what the timing of processes in one cycle of the CAB system is expected to be.

Flow name	Source	Destination	Protocol	Data Type
Sens_in	Sensors	IN	Pool	14 Individual sensor values: 2 pedal 4 wheel rotation 4 axle load 4 pressure feedback
Sensor_ Feedback	IN	OUT	Pool	Record containing 4 pressure feedback values
All_sens	IN	MODIFIER_ SELECTION	Pool	Record containing pedal value 4 wheel rotation sensor values 4 axle load values
Pedal_val	IN	BASIC	Signal	Pedal value
Basic_press	BASIC	MODIFIER_ SELECTION	Signal	Record containing 4 basic braking values (1 per wheel)
Modifier_ Values	MODIFIER_ SELECTION	MODIFIER_ ADDITION	Pool	Record containing 4 basic braking values (1 per wheel) 4 ABS modifier (1 per wheel) 4 Load compensation modifiers (1 per wheel) 4 Emergency stop modifiers (1 per wheel)
Votes	MODIFIER_ SELECTION	BUS_ WATCHER (On all 3 processors)	Signal	Record containing 3 flags, indicating whether each of the 3 modifiers is required
All_votes	BUS_ WATCHER	MODIFIER_ ADDITION	Pool	Record containing 3 sets of 3 votes (i.e. one from each processor)
Final_press	MODIFIER_ ADDITION	OUT	Pool	Record containing 4 Braking values (1 per wheel) Flag indicating whether modifiers have been added successfully
Pair_1	OUT	Hardware	Signal	Braking actuator drive values for front nearside and rear offside wheels
Pair_2	OUT	Hardware	Signal	Braking actuator drive values for front offside and rear nearside wheels

Table 6-18: Data types of all flows

IN (Periodic process run at the start of each cycle)
Read all sensors
Use data from both pedal sensors to form single pedal value
Output pedal value to BASIC
Output pedal, wheel revolution and load values to pool for use by MODIFIER SELECTION
Output actuator feedback to pool for use by OUT

BASIC (Sporadic process triggered by arrival of pedal sensor data)
Calculate a basic braking pressure for each wheel based on the pedal sensor only

MODIFIER SELECTION (Sporadic process triggered by arrival of basic)
Use all sensor information to determine which modifiers are required in this cycle
Calculate modifier values and place record containing basic and modifier values in pool for use by MODIFIER **ADDITION**
Broadcast votes to all BUS WATCHER processes to identify which modifiers are required

BUS WATCHER (Sporadic process triggered by arrival of votes)
Build up record of votes (i.e. which processors have determined a need for each of the modifiers)

MODIFIER ADDITION (Periodic process, with offset from start of cycle)
From record of all votes assembled by BUS WATCHER, determine which modifier(s) to add to the basic braking value
If no value is available for a required modifier, revert to basic

OUT (Periodic process, with offset from start of cycle)
If modifiers successfully added output calculated pressure immediately (adjusted according to sensor Feedback) otherwise wait until end of period, and output basic value (adjusted according to sensor feedback)

Table 6-19: The functionality of each process

1 OUT 2 MODIFIER ADDITION 3 IN 4 BUS WATCHER 5 BASIC 6 MODIFIER SELECTION

Table 6-20: Order of priority tasks (1 is high)

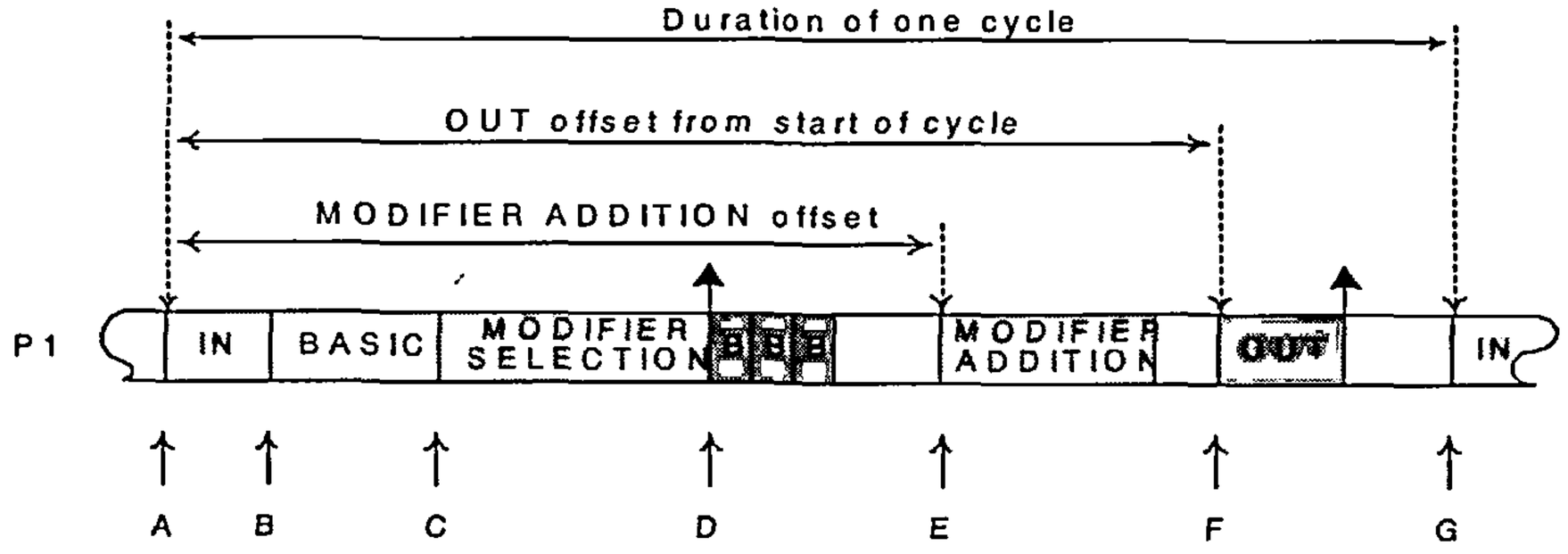


Figure 6-11: Timing of 1 cycle of the CAB system on processor 1

Table 6-21 represents the fragment of the FLASH analysis that regards the propagation of *Pair_1* from *Channel_1*. We can see that some failure events (i.e. *O*, *C*, *E*) are stopped by the output module, while value failure (i.e. *V_*.) is propagated to brake shoes. A value failure of *Pair_1* out of *Channel_1* can be due to a single failure that arise inside *Channel_1*, (i.e. *V_.Pair_1.Out1*, *V_.Sensor_feedback.In1*, *V_.Modifier_values.MS1*, *V_.Final_press.MA1*, *V_.All_votes.BW1* or *V_.Processor.P1*), a failure of both busses (i.e. *V_.Busses.2B*) or some internal failure of *Channel_1* (i.e. *V_.Pedal_Val.In*, *V_.Basic_Press.BS1*, *V_.All_sens.In*, *V_.Votes.MS1*) combined to a failure in an output of another channel (i.e. *V_.Votes.Ch2* or *V_.Votes.Ch3*).

The study of the CAB has continued further. However we are not reporting tables that have been made for all the other modules in Figure 6-10 since we believe we already have illustrated enough the FLASH process in practice.

Instance = Channel_1	Component Type = Channel	Periodicity = Signal	Tag = Ch1
Event propagated	**Causes**	**Description**	**5th Column: Justification, Design Recommendations, Derived Safety Requirements**
O_. Pair_1.ch1	O_.Pair_1.Out1 AND O_.Busses.2B OR O_.Processor.P1	Omission for value to output mod 1	This event is not propagated. In case of omission the output module will use the previous value delivered by the system.
C_. Pair_1.ch1	C_.Pair_1.Out1 OR C_.Busses.2B OR C_.Processor.P1	Commission for value to output mod 1	This event is not propagated. In case of commission the output module will use the previous value delivered by the system.
E_. Pair_1.ch1	E_.Pair_1.Out1 OR E_.Busses.2B OR E_.Processor.P1	Early for value to output mod 1	This event is not propagated. In case of omission the output module will use the previous value delivered by the system.
L_. Pair_1.ch1	L_.Pair_1.Out1 OR L_.Busses.2B OR L_.Processor.P1	Late for value to output mod 1	This event is not propagated. In case of omission the output module will use the previous value delivered by the system.
V_. Pair_1.ch1	V_.Pair_1.Out1 OR V_.Sensor_feedback.In1 OR V_.Modifier_values.MS1 OR (V_.Pedal_Val.In OR V_.Basic_Press.BS1 OR V_.All_sens.In OR V_.Votes.MS1) AND (V_.Votes.Ch2 OR V_.Votes.Ch3) OR V_.Final_press.MA1 OR V_.All_votes.BW1 OR V_.Busses.2B OR V_.Processor.P1	Wrong value to output mod 1	The event must exhibit a rate < $0.4*10^{-3}$ h^{-1}

Table 6-21: FLASH table for Channel 1

6.2.3 Integration, verification and Common Cause Failures analysis

FLASH analysis in the integration and verification stage confirmed that the design meets specifications, recommendations and derived safety requirements issued during the decomposition and design. FLASH tables presented in the previous section (i.e. Table 6-14, Table 6-15 and Table 6-16) already showed the column *Summary FMEA results* with results from the verification stage. Common cause failure analysis has not been performed on the CAB system. It would have followed the same path as the analysis of the Fuel System. Once minimal cut sets for critical failure modes are obtained, they are searched for couplings. If couplings are not found, their likelihood is calculated as product of likelihood of each event in the minimal cut set; if couplings are found, the extension of the FLASH method presented in chapter 5 has to be applied as we did in the first case study presented in this chapter.

6.3 Discussion

The overview of these case studies has highlighted how the FLASH analysis can be integrated with the decomposition and design, and the integration and verification, stages of the lifecycle. The FLASH process was able to provide the results expected from FHA, HAZOP, FMEA, and FTA. Chapter 2 discussed the main problems arising in the assessment of complex systems, when classical safety analysis techniques are applied. These are the inconsistencies between the various analyses and the difficulty in linking the results back to the functional hazard assessment. It is believed that FLASH has overcome these issues. Additionally, the way FLASH decomposes a system and collects information about events is a valuable help for considering common cause failures. It has been shown how data gathered into FLASH tables during the lifecycle can be u sed to identify minimal cut sets vulnerable to common cause failures. Coupled events and coupled components are identified and when reliability data are known, quantitative figures can also be derived. Hence, on the basis of the FLASH analysis, analysts and designers can decide whether to accept a proposed system design or ask for improvements.

FLASH can be compared with classical analysis methods on the basis of the support it gives to development, formality, speed of analysis, keeping the analysis updated with design and providing immediate feedback. In addition, the traceability of functional failures to basic events becomes useful when it has to be shown that the system meets specifications and safety requirements.

The FLASH approach is quite formal and it becomes heavier as the complexity of the system increases. It becomes laborious and tedious when tables are compiled by hand. However a software tool can speed up the FLASH process and make it a competitor (we at least hope) of classical methods in terms of results obtained and overall economy of the safety analysis. A suitable software tool will reduce to instants the most tedious phases of the FLASH process that is consistency checking of tables and FT construction and evaluation. We have experienced that the "*FLASH schema*" we implemented into the SAM platform [McDermid, 1994], extensively helps in navigating the FLASH hierarchy by following links among tables. This tool had helped considerably in the writing and updating of tables for our case studies.

Though FLASH is based on classical, widely diffused safety analysis techniques it is a completely new approach and its results and benefits have to be compared with ones achieved with the best industrial practice. The application of FLASH to a variety of industrial cases will prove the practicability of this technique in an industrial environment. It is believed that some changes might have to be made to the FLASH approach to adapt this technique to some complex systems (e.g. adding new columns to accommodate additional information). However, it is also believed that the basic FLASH process will not need to change significantly.

Chapter Seven

Conclusions

This thesis has made a step towards integration of safety analysis techniques ordinarily performed in sequence during the lifecycle of computer based safety critical systems. Integration was made possible by the identification of several links amongst such analysis techniques. We exploited those links to provide a unifying analysis technique, supporting common cause failure assessment and continuous feedback to designers so that we can substantiate also what is often called "design for safety". The result is to improve the quality of the safety analysis of moderately complex computer based safety critical systems.

7.1 Review of Research Objectives

The research addressed some limitations and shortcomings of classical safety analysis techniques that were pointed out in Chapter 2. These were presented in terms of the following questions:

a) Is it possible to develop a technique that encompasses the different safety analyses typically performed across the lifecycle?
b) Can the application of this technique result in a meaningful and easy way to perform a collection of safety analyses which can assist the design of the system?
c) Can we ensure the consistency of the results within the assessment?
d) Can those results be represented both graphically and in tables, so that we can combine the benefits of both representations?
e) Finally, is it possible to use this technique to systematise the identification of common cause failures?

To answer these questions we investigated different approaches, which resulted in a method called FLASH. This was described in Chapter 4 and was extended to enable

common cause failure analysis in Chapter 5. The method was applied in two case studies presented in Chapter 6.

7.2 Contribution of the thesis

We believe that the material presented in this thesis substantiates to a large extent the central proposition that was expressed in Chapter 1:

> *"It is possible to produce an integrated safety analysis framework which can be used to produce a complete and consistent safety analysis, including treatment of common cause failure and which can be used to drive "a design-for-safety" process."*

The proposed safety analysis framework is founded on current industrial practice for safety analysis and on our understanding of the similarities shared among analysis techniques. The framework supports common cause failure analysis by providing an in-depth and detailed screening of the real couplings amongst components and quantitative estimation of the common cause failure contribution. Finally, design for safety is achieved by the continuous feedback between designers and analysts that is furthered by the proposed analysis process.

7.2.1 Theoretical Contribution

Need for Formality and Traceability throughout the design process

We started our research studying current industrial practice and saw that a number of different techniques are used for safety analysis as the design evolves. However these techniques are not formally linked to each other and, as a consequence, the consistency of the analysis cannot be assured throughout the design and development process (2.2). In a complex design it is, therefore, often difficult to trace (using the results of the safety assessment) causes of critical malfunctions of the system in the hierarchy of subsystems and components which comprises the design (2.4).

Need for Careful Screening of Couplings among events in minimal cut sets

We showed that there are two possible ways to prevent common cause failures, either by eliminating (effects of) root causes or by removing coupling factors. In the first case

defences against potential root causes are considered and put in place (2.3.4). In the second case, a high degree of diversity among components is required (2.3.5). However, whilst there are methods that assess defences against the causes of common cause failures (e.g. the cause defence matrix in 2.3.8), there are no methods that provide a careful screening of couplings among events in the same minimal cut set.

Need for Quantification of the Common Cause Failures Likelihood

We saw that, whilst there are a number of methods for making a quantitative evaluation of common cause failures in pure hardware systems, these methods cannot be used for evaluating the likelihood of common cause failures in computer based safety critical systems. The reason is that all of these methods are based on the *symmetry hypothesis* which, unfortunately, cannot be accepted for minimal cut sets in computer based safety critical systems, since probabilities of events in the same minimal cut set may span various orders (2.3.9).

Extending Recently Proposed Techniques

To address the limitations and shortcomings that were pointed out, we investigated various approaches. First we tried to extend software fault tree notation [Leveson, 1983] (3.1). We merged this technique with the Cause Consequence Analysis notation formulating what we called "*Event Tree Output notation*" (3.2). Then, we attempted to extend the Master Plant Logic Diagram [Modarres, 1992] producing the MPLD* notation (3.3). Finally, at the last stage of our early work, we introduced two other notations: a graphical one to represent the mapping of software to hardware and a table based one to store the detailed information that was not practical to store in the MPLD*. However, we were not yet able to achieve the targets that we had set out for our thesis. None of the above notations was integrated with classical techniques (i.e. FHA, HAZOP, FMEA and FTA) nor could they be used to link those techniques (3.4).

Links amongst Classical Safety Analysis Techniques

The identification of links amongst classical safety analysis techniques and the decision to develop an approach which encourages the top-down study of computer based safety critical systems were the major turning points that led to the proposed method. Firstly, we realised that the causes of failure events considered at a certain level of the analysis can become the failure events considered in subsequent levels (4.1). Secondly, a common syntax that formalised such causal relationships was introduced (4.2.3) and a

table suitable for supporting FHA, HAZOP, FMEA and containing information to construct fault trees was outlined (4.2.6). Finally, the process of the FLASH analysis was established both as it should be approached in the decomposition and design (4.3.1) and in the integration and verification (4.3.2) stages of the lifecycle.

Systematic Screening of Couplings

The FLASH framework is also designed to enable the identification of couplings among components, and to support common cause failure analysis, i.e. by means of screening of minimal cut sets to find ones with coupled events and by quantitative evaluation of such cut sets. In this process, events in each minimal cut set are scanned to see whether or not they share one or more potential couplings (5.2). If any sharing is found, minimal cut sets are considered coupled, and they are designated to undertake common cause failure analysis. By exactly identifying actual couplings, FLASH makes it easy to propose effective remedies.

Likelihood of Minimal Cut Sets with Coupled Events

We also provided a mathematical approach for calculating the likelihood of minimal cut sets that considers the contribution of each actual coupling cause (5.3) as identified by the FLASH method. We have transferred the problem of common cause failure analysis from the minimal cut set level to a lower, more detailed, level systematising the identification of couplings and obtaining more realistic figures. As a consequence, the basis for the identification of couplings is transferred from "expert's judgement" to the list of lifecycle categories used for identifying couplings among events and its weighting factors.

7.2.2 Practical Contribution

Integration of Safety Analyses Techniques

As we discussed in Chapters 2 and 4, the integration of safety analysis techniques performed throughout the lifecycle offers possibilities for checking the consistency of results of safety analysis, guaranteeing continuous feedback to designers, providing a compact and accessible format to present safety analysis results (e.g. to certification authorities), and finally the advantage of applying one technique instead of several. The

greater cost due to the utilisation of a more complex and articulated technique[31], should be paid back by the fact that less techniques have to be taught to the personnel and that the results from one level analysis are immediately available to start the following level of analysis. Additionally, all safety analyses are immediately accessible to anyone with the knowledge of that single technique without any conversion cost. Furthermore, costs for the infrastructure (e.g. software packages) necessary to support the overall safety analysis process may also be reduced, although there are issues of developing tool support (see below).

Results both Graphical and Tabular

The provision of safety analysis results, both in tabular and graphical format, combines the advantages of the ability to provide detailed tabular information that is easy to access[32] and an intuitive[33] graphical representation. In FLASH the graphical representation of results is taken from the tabular one. Fault trees can actually be built starting from tables at any level of the hierarchy by parsing relationships between causes and effects.

Industrial Practice

It is expected that FLASH will improve the industrial practice of safety analysis of computer based safety critical systems (at least of moderate complexity[34]). Whilst we have not shown that FLASH does work effectively in industry, there is evidence that it will be useful. In particular we have found that FLASH may offer a way to comply with guidelines that are to be released for the certification of Programmable Logic Controller (PLC) for safety critical applications by the Italian regulatory authorities (4.5).

Automation

FLASH can be supported by a software tool that eases repetitive and error prone tasks, helps to navigate through the hierarchy of tables, generate trees, calculate the likelihood

---

[31] FLASH is more complex than any single technique it intends to replace.

[32]Though tabular representation is perhaps little intuitive and difficult to understand at a glance, it can be detailed down to any granularity.

[33]Graphical representation can be intuitive and easy to understand at a glance, however when graphs extend to multiple pages, they are no longer intuitive.

[34]At present, FLASH is quite complex to apply, hence expensive. Therefore we reckon it is justifiable at least for highly critical systems.

of events propagated and make consistency chècks on the whole hierarchy. Suitable software may automate also the scanning of minimal cut sets to find those sha ring one or more causes of coupling (5.2). We believe also that the quantitative evaluation of the total failure probability for each coupled minimal cut set can be automated to a certain extent. A preliminary software prototype with basic features has been developed and presented in (4.4).

Providing Support to Design for Safety

The case studies discussed in the sixth chapter showed how FLASH analysis is integrated with the decomposition and design, and the integration and verification stages of the lifecycle. The application of this technique was able to provide the results that we would expect from FHA, HAZOP, FMEA, and FTA. Additionally, we have shown how the data gathered into the FLASH tables during the lifecycle can be used to identify minimal cut sets that are vulnerable to common cause failures and to estimate the likelihood of those cut sets. Hence, on the basis of the FLASH analysis, analysts and designers are provided with a comprehensive and detailed view on system safety implementation, hence they may use this information to decide whether to accept a proposed system design or ask for improvements.

7.3 Suggestions for Further Work

We believe that this thesis is only a starting point towards integration of safety analysis techniques. As such we can see several areas in which further work can be done. At present there are some limitations that we believe are easy to remove, and some others that probably require more thought. Further work on this method can be divided into two main domains: the consolidation of the present technique and a theoretical extension.

7.3.1 Consolidation of the Technique

This work has to be done to have the technique accepted and employed as best practice by the industry.

Automation

Often in this thesis we have stressed the problem of automation. Though FLASH can be performed by hand, the process becomes heavier for the analyst as the complexity of the system increases. The analyst is actually asked to perform a lot of repetitive and error prone tasks. Our software prototype has considerably helped us to run our case studies,

but its limited potential has prevented us considering big industrial applications. Therefore, there is the need of a more powerful tool, which automates the process further.

Industrial Case Studies

Once a suitable software tool is available, it will be possible to run industrial case studies. These should demonstrate whether or not a FLASH approach is feasible in complex systems. At present, there is no conclusive evidence that the method supports the design and verification of a very large system such as an aircraft, a helicopter, a chemical or nuclear installation. The design of those systems (though they appear to be hierarchically decomposable) is not usually approached hierarchically. As it was p ointed out in Chapter 4, FLASH could still support the development at sub-system level. However, it still remains to be demonstrated that if a FLASH analysis is available for each sub-system then it may be possible to link all these FLASH analyses to each other in order to produce a FLASH model for the full system.

Validation

Though FLASH is based on classical, widely diffused, safety analysis techniques it is a completely new approach and its results and benefits have to be compared with those achieved with the best industrial practice. Availability of industrial case studies performed using FLASH will allow benchmarking the method against current safety analysis techniques, and enable to access the extent of its limitations and benefits.

FLASH can be compared with classical analysis methods on the basis of the support it gives to development, formality, speed of analysis, keeping the analysis updated with design and providing immediate feedback. The traceability of functional failures to basic events, which FLASH provides, becomes useful when evidence is required that the system meets specifications and safety requirements. It is believed that, to adapt the technique to some complex systems, some changes might have to be made (e.g. adding new columns to accommodate additional information). However, it is also believed that the basic FLASH process will not change significantly.

Certification

Provided that the proposed framework will "survive" the test on industrial case studies, it might also be certified by regulatory authorities as a standard practice to conduct and present safety analyses. Hence industries may be asked to present safety analyses in a

"FLASH format". At present, we believe FLASH has a good chance to reach this point, since it can trace any event to its causes and it seems it fits the forthcoming Italian guidelines on PLCs for safety critical applications. Additionally FLASH tables have the potential to accommodate features that may be requested by certification authorities. For instance, FLASH tables may be modified to record motivations for any decision concerning safety made during the design. Names of people responsible for such decisions could also be recorded, hence, eventually, it could be possible to trace whoever is liable for any (good or wrong) design decision.

7.3.2 Theoretical Extension

This is the work that can be done to extend or improve the technique itself.

Sensitivity Analysis

Another extension of FLASH could be in improving the technique itself. For example, during the verification stage, a sensitivity analysis made on fault trees [Homma and Saltelli, 1996] drawn for critical events could show the impact of potential improvements even before proposing modifications to the design [Contini et al., 1999b] and performing again the FLASH analysis. This will indeed speed up the overall analysis process.

Merging with other Approaches and Techniques

The FLASH method can potentially be usefully extended to formalise the writing and updating of causes-effect relationships into FLASH tables. For instance it could be evaluated whether it is feasible to include a formal algorithm, which automates the writing of those relationships. Example of suitable algorithms are proposed in [Papadopoulos and McDermid, 1999a-b], in [Atkinson & Carpignano, 1996] and in [Sardella, 1995]. These enhancements may boost additional automation and formality.

7.4 Final Remarks

Closing the thesis we wish to say that our approach is a contribution towards improving *consistency*, *completeness* and *correctness* in safety analysis. We have focused our efforts towards the integration of well-established safety analysis techniques and found an interesting way to link several safety analysis techniques[35] ordinarily performed in

---

[35]Although integration of classical techniques is the route we selected, we are aware that perhaps other routes could be taken to achieve our goals.

series during the lifecycle. We were also able to consider common cause failures both qualitatively and quantitatively. The proposed method incorporates a number of principles that, in theory, could enable their application in a complex system. The two case studies we ran demonstrated the validity of the approach when applied to moderately complex systems. However a conclusive evaluation of the real value and scalability of this approach could only be achieved in a much wider and realistic context of application.

Bibliography

[Adelard, 1994] Adelard Ltd., *HAZOP Study Guidance for Equipment Containing Programmable Electronic System: Feasibility Study Report.* D/49/0076/2 v1.0, 1994.

[Amendola, 1986] Amendola A., Systems Reliability Benchmark Exercise: final report. Euratom Report EUR 10696/I EN, 1986.

[Atkinson & Carpignano, 1996] Atkinson M. & Carpignano A., Advances with STARS. Applications to Safety Problems. 4th Intern. Workshop on Functional Modelling of Complex Technical Systems, (ORA/PRO 40131), 6th September 1996.

[Avizienis, 1985] Avizienis A. et al., The N-Version Approach to Fault-Tolerant Software. *IEEE Transactions on Software Engineering*, SE-11(12):1491-1501, December 1985.

[Boeing, 1996] *Statistical Summary of Commercial Jet Aircraft Accidents: Worldwide operations 1959-1995*, Boeing Commercial Airplane Group, Seattle, WA, 1996.

[Bondavalli and Simoncini, 1990] Bondavalli A., Simoncini L., Failure Classification with Respect to Detection, in Predictably Dependable Computing Systems – First year Report, Task B, Volume 2, May 1990.

[Bourne et. al., 1981] Bourne A. J., Edwards G T, Hunns D.M. Poulter D.R., *Defences against common-mode failures in redundancy systems.* SRD-R-196, United Kingdom Atomic Energy Authority, Safety and Reliability Directorate, January 1981.

[Bowen and Stavridou, 1992] Bowen J. and Stavridou V., "Safety Critical Systems, Formal Methods and Standards", PRG-TR-5-92, Programming Research Group, Oxford University, 3rd March 1992.

[Budgen, 1985] Budgen D., *Combining MASCOT with Modula-2 to Aid the Engineering of Real-time Systems*, Software Practise and Engineering, 15(8): 767-793, 1985.

[Burns and Pitblado, 1993] Burns D. J., Pitblado R. M., A modified HAZOP methodology for Safety Critical System Assessment, 7th meeting of the UK Safety Critical Club, Bristol, February 1993.

[Bussolini, 1971] Bussolini J.J., High Reliability Design Techniques Applied to the Lunar Module, Lecture Series No. 47 on Reliability on Avionics System. A.G.A.R.D. Rome, 16-17 September. London, 20-21 September, 1971.

[Carpignano & Poucet, 1994] Carpignano A., Poucet A., Computer Assisted Fault Tree Construction: a Review of Methods and Concerns. *Rel. Engng. and System Safety*, Vol. 44-3, pp. 265-278, 1994.

[Chudleigh et al., 1995] Chudleigh M.F., Catmur J.R., Redmill F.A., A guideline for HAZOP studies on Systems which include Programmable Electronic Systems. Proceeding of the 14th International Conference on Computer Safety, Reliability and Security (SAFECOM 95). Belgirate, Italy, 11-13 October 1995.

[CISHEC, 1977] CISHEC, *A guide to Hazard and Operability Studies*. The Chemical and Health Council of the Chemical Industries Association Ltd, 1977.

[Cojazzi, et al, 1995] Cojazzi G., Mauri G., Sardella R., The Treatment of physical

dependencies and CCFs within the STARS environment: methodological framework and applications. PSA '95, Taejon, Korea, 1995.

[Contini et al., 1999a] Contini S., Scheer S., Wilikens M., De Cola G., Cojazzi G., *ASTRA: An Integrated Tool Set for Complex Systems Dependability Studies*, in: Tool Support for System Specification, Development and Verification, R. Berghammer, Y. Lakhnech (eds.), pp. 77 – 91. Advances in Computing Science, Springer, 1999.

[Contini, 1999b] Contini S., ASTRA, Advanced Software Tool for Reliability Analysis, FTA: Fault Tree Analysis module PTD: Time Dependent Analysis module EUR 18727/EN, 1999.

[Dhillon and Singh, 1981] Dhillon B S, Singh C., *Engineering Reliability,* John Wiley and Sons, New York, 1981.

[Dorsett and Mellor, 1993] Dorsett, Mellor, The Airbus A320 electronic flight control system, Unpublished manuscript, 1993.

[Dugan et al., 1993] Dugan J B, S J Bavuso, M A Boyd., Fault trees and Markov models for reliability analysis of fault-tolerant digital systems. *Rel. Engng. and System Safety,* Vol. 39, Elsevier Science Publishers Ltd. England, pp291-307, 1993.

[Edwards and Watson, 1979] Edwards G T and I A Watson, *A Study of Common Mode Failures* SRD-R-146, United Kingdom Atomic Energy Authority, Safety and Reliability Directorate, July 1979.

[EPRI, 1985] EPRI NP-3837, *A study of common cause failures. Phase 2: A comprehensive classification system for component fault analysis*. Los Alamos Technical Associates, Inc. June 1985.

[Ezhilchelvan and Shrivastava, 1986] Ezhilchelvan P. D., Shrivastava S. K., A Characterisation of Faults in Systems, in Proceedings of the 5th Symposium on Reliability in Distributed Software and Database Systems, pages 215-22, LA United States, 1986.

[Fenelon and McDermid, 1993] P Fenelon, J A McDermid, An Integrated Toolset For Software Safety Analysis. *Journal of Systems and Software*, 1993.

[Fenelon et al., 1994] P Fenelon P, J A McDermid, D J Pumfrey, M Nicholson, *Towards Integrated Safety Analysis and Design*, ACM Applied Computing Review, Vol. 2 N°1, pp21-32, August 1994.

[Fleming and Kalinowski, 1983] Fleming K. N., Kalinowski A. M.. *An Extension of the Beta Factor Method to Systems with High Levels of Redundancy*. Pickard, PLG-0289, Lowe and Garrick, Inc., USA, June 1983.

[Fleming, 1975] Fleming K. N., A Reliability Model for Common Mode Failure in Redundant Safety Systems. *Proceedings of the Sixth Annual Pittsburgh Conference on Modelling and Simulation,* GA-A13284, General Atomic Report, Pittsburgh, PA, April, 23-25, 1975.

[Friedman, 1995] Friedman M A and J M Voas, Software Assessments: Reliability, Safety, Testability; John Wiley and Sons, inc., 1995.

[Fussell, 1973] Fussell J. B., Synthetic Tree Model. Report ANCR 1098 Aerojet Nuclear Company. March 1973.

[Haasl, 1965] Haasl, *Advanced Concept in Fault Tree Analysis System Safety Symposium*,

University of Washington Library, Seattle, 1965.

[Hecht and Hecht, 1986] Hecht H. and Hecht M., Fault-tolerant Software. In D.K.Pradhan, editor, *Fault-Tolerant Computing: Theory and Techniques,* volume II, pages 658-696. Prentice-Hall, 1986.

[Henley and Kumamoto, 1981] Henley J., Kumamoto H., *Reliability Engineering and Risk Assessment,* Prentice Hall, Englewood Clift, NJ, 1981.

[Homma and Saltelli, 1996] Homma, T., Saltelli, A., Importance Measures in Global Sensitivity Analysis of Nonlinear Models. *Reliability Engineering & System Safety*. Vol 52, N. 1, 1996.

[Hu, Modarres, 1997] Hu Y-S, Modarres M., "Ensuring and Automating Software Assurance Based on A Logic Model". Center for Reliability Engineering Department of Materials and Nuclear Engineering University of Maryland College Park, MD 207442, USA, 1997.

[Humphreyes, 1987] Humphreyes R A., Assigning a numerical value to the Beta Factor Common Cause Evaluation. Rolls Royce Associates Limited, Derby, 1987.

[Humphreys and Johnston, 1987] Humphreyes P., Johnston B.D., Dependent Failure Procedure Guide SRD-R-418, United Kingdom Atomic Energy Authority, Safety and Reliability Directorate, March 1987.

[IEC 61508, 1997] International Electrotechnical Commission 65A/179-185, IEC-61508: Functional Safety of Electrical / Electronic / Programmable Electronic Safety-related Systems, IEC, 3 rue de Varembé CH 1211 Geneva Switzerland, 1997.

[IEEE, 1975] Guide for General Principles of Reliability Analysis of Nuclear Power Generation Station Protection Systems (An American National Standard) ANSI N41-4-1976, IEEE Std 352-1975 (Revision of IEEE WStd 352-1972). 1975.

[ISO, 1992] Draft International Standard *Road Vehicles -Interchange of digital information-Controller Area Network (CAN) for High Speed Communication,* ISO DIS 11898, 1992.

[ISPESL-CEI, 2000] Draft "PLC Safety Critical" Linea Guida Assessment, GDL-ISPESL – Report, 2000.

[Johnston and Crackett, 1985] Johnston B. D. and Crackett J., Common cause failure reliability Benchmark exercise. SRD-R-383 (GD/PE-N/1329, United Kingdom Atomic Energy Authority, Safety and Reliability Directorate, August 1985.

[Kaplan & Garrick, 1981] Kaplan S, Garrick J. "On qualitative Definition of Risk" *Risk Analysis*, vol. 1, No.1., 1981.

[Kemeny, 1969] Kemeny J. G., *Report of the President's commission on the accident at Tree Mile Island*, 1969.

[King and Rudd, 1971] King C.F. and Rudd D.F., Design and maintenance of economical failure-tolerant processes. *American Institute Chemical Engineering Journal*, 18, 257-69, 1971.

[Kletz, 1992] Kletz T., HAZOP and HAZAN: Identifying and Assessing Process Industry Standards, 3rd Edition, Hemisphere Publishers, ISBN: 1-56032-276-4, 1992.

[Laprie, 1993] Laprie J., "Dependability: From Concepts to Limits", *Proc. SAFECOMP*, Poznan, Poland, pp. 157-168, 1993.

[Lawley, 1974] Lawley H.G., Operability Study and Hazard Analysis, *Chemical Industry Progress,* 70, 45-56, 1974.

[Lawley, 1976] Lawley S, Size up plant hazard this way, *Hydrocarbon Processing*, 247-61, April, 1976.

[Lees, 1980] Lees P., Loss Prevention in the Process Industries, Butterworth, London, 1980. *American Institute Chemical Engineering Journal*, 18, 257-69, 1971.

[Leveson, 1983] Leveson N G, P R Harvey, Software Fault Tree Analysis *Journal of System and Software*; 3:173-81, 1983.

[Leveson, 1991] Leveson N G, T J Shimeall, Safety Verification of Ada Programs using Software Fault Trees; *IEEE Software*, 8(4):48-59, July 1991.

[Lievens, 1976] Lievens, *Sécurité des Systèmes*, Cepadeus Editions, Toulouse 1976.

[Littlewood, 1993] Littlewood B., *The need for evidence from disparate sources to evaluate software safety*, 7th meeting of the UK Safety Critical Club, Bristol, February 1993.

[Maier, 1995] Maier T., FMEA and FTA to support safe design of embedded software in safety-critical systems. Proceeding of the 12th CSR workshop and 1st ENCRESS conference, Bruges, September 1995.

[Malhotra, 1993] Malhotra M., *Specification of Dependability Models for Fault-Tolerant Systems.* PhD thesis, Graduate School of Duke University, USA, 1993.

[Marshall and Olkin, 1967] Marshall A. W., Olkin A., Multivariate Exponential Distribution. *Journal of American Statistic Association,* Vol. 62, pp. 30-44, 1967.

[Mauri, 1995] Mauri G., Le dipendenze e i guasti da causa comune nella analisi di affidabilità di un impianto complesso. Special Publication, No. I.95.15, European Commission-ISEI/IE, Ispra, Italy, 1995.

[Mauri, 1996] Mauri G., Dependency and Common Cause/Mode Failures Analysis for Safety Critical Systems: Field Survey and Review. First Year Qualifying Dissertation. University of York, York, June, 1996.

[Mauri, 1997a] Mauri G., Dependency and Common Cause/Mode Failures Analysis for Safety Critical Systems: Field Survey and Review. Thesis Prposal. University of York, York, June, 1997.

[Mauri, 1997b] Mauri G., Integrated Hardware and Software Safety Assessment for Safety Critical Systems. Final Report. University of York and University of Pisa, York, December 1997.

[Mauri et al, 1998] Mauri G., McDermid J. A., Papadopoulos Y., *Extension of Hazard and Safety Analysis Techniques to Address Problems of Hierarchical Scale*, in Proceedings of IEE Colloquium on Systems Engineering of Aerospace Projects, IEE Digest no: 98/249, pages. 4.1/4.6, London, 1998.

[McCord, Moroney, 1964] McCord J R, Moroney R M, Introduction to Probability Theory. Collier MacMillan, London, 1964.

[McDermid, 1994] McDermid, J.A., Support for safety cases and safety arguments using SAM, Reliability engineering and System Safety, Vol. 43, pp. 111-127, 1994.

[McDermid and Pumfrey, 1994] McDermid and Pumfrey D., A Development of Hazard Analysis to aid Software Design. Proceeding of the 9th annual conference on

Computer Assurance (COMPASS 94). Gaithersburg, MD, Pp. 17-25, July 1994.

[McDermid et al., 1995] McDermid, J.A. Nicholson, Pumfrey D. & Fenelon, P., Experience with the application of HAZOP to Computer-Based Systems. Proceeding of the 10th annual conference on computer assurance (COMPASS 95). Gaithersburg, MD, Pp. 37-48, June 1995.

[MIL-STD-1629a, 1980] Department of Defense, Military Standard: Procedure for Performing a Failure Mode and Effect Analysis. MIL-STD-1629a, Washington D.C., 1980.

[MIL-STD-882, 1969] Department of Defense, Military Standard: System Safety Program Requirements. MIL-STD-882, United States of America. 1969.

[MIL-STD-882c, 1993] Department of Defense, Military Standard: System Safety Program Requirements. MIL-STD-882c, United States of America. 19 January 1993.

[MIL-STD-882d, 1999] Department of Defense, Military Standard: System Safety Program Requirements. MIL-STD-882d, United States of America. July 1999.

[Minichino et al., 2000] Minichiono M., Ciancamerla E., Chiaradonna S., Bandovalli A., *An Experience of Dependability assessment of a typical Industrial safety critical Programmable Logic Controller*, 4th Symposium of Programmable Electronic Systems in Safety Related Applications. TUV Rheinland, Cologne, 3-4 May, 2000.

[MoD 00-56, 1996] MoD., Draft Interim Defence Standard 00-56/1: a Guideline to HAZOP studies on systems which include a programmable electronic system, UK Ministry of Defence, 1996.

[MoD 00-58, 1995] MoD., Draft Interim Defence Standard 00-58/1: a Guideline to HAZOP studies on systems which include a programmable electronic system, UK Ministry of Defence, 1995.

[Modarres, 1987] Hunt R N, Modarres M., "Performing a Plant Specific PRA by Hand – A Practical Reality". 14th Inter-RAM Conference, Minneapolis, 1987.

[Modarres, 1992] Modarres M., "Application of the Master Plant Logic Diagram in Risk-Based Evaluations". Amer. Nucl. Society Topical Mtg. on Risk Management, Boston, MA, 1992.

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

[Mosleh et al., 1993] Mosleh A., Fleming K., Parry G., Paula H., Warledge D., Rasmusson D., *Procedures for Analysis of Common-Cause Failures in Probabilistic Safety Analysis*. NUREG/CR-5801, Vol. 1. Office of Nuclear Regulatory Research. Washington, DC, 1993.

[Mosleh, et al., 1988] Mosleh A. et al., *Procedures for Treating Common Cause Failures in Safety and Reliability Studies*. NUREG/CR-4780, Vol. 1 & 2. Office of Nuclear Regulatory Research, Washington, DC, Cap. 2, pp. 4-8, January 1988.

[Neumann, 1987] Neumann P.G., Index for computer related risks. *ACM Software Engineering Notes,* 12(1): 22-28, 1987.

[Norris, 1998] Norris J. R., Markov Chains (Statistical & Probabilistic Mathematics Series No. 2). Cambridge Univ Pr (Pap Txt); ISBN: 0521633966- 253 pages 1 Pbk Ed edition, August 1998.

[NUREG 2300, 1983] American Nuclear Society, IEEE. *PRA procedure guide; A Guide to*

the Performance of probabilistic Risk Assessments for Nuclear Power Plants. NUREG/CR-2300, Vol. 1 & 2, Office of Nuclear Regulatory Research, Washington, DC, 1983.

[NUREG 2815, 1985] American Nuclear Society, IEEE. *Probabilistic Safety Analysis Procedure Guide*. NUREG/CR-2815, Vol. 1 & 2, Office of Nuclear Regulatory Research, Washington, DC, 1985.

[NUREG 5993, 93] NRC, Brookhaven National Laboratory. *Methods for Dependency Estimation and System Unavailability Evaluation Based on Failure Data Statistics*. NUREG/CR-5993, Vol. 1 & 2, Office of Nuclear Regulatory Research, Washington, DC, July 1993.

[NUREG 74/014, 1975] *Reactor Safety. An assessment of accident risk in US commercial nuclear power plant*. Wash-1400, NUREG 74/014, US NRC, 1975.

[OREDA, 1984] Offshore Reliability Data. Oreda Participants, Harvic, Norway, 1984.

[Palady, 1995] Palady P., *Failure Modes and Effects Analysis*, PT Publications, ISBN: 0-94545-617-4, 1995.

[Papadopoulos and McDermid, 1998] Papadopoulos Y., McDermid J. A., *A Harmonised Model for Safety Assessment and Certification of Safety Critical Systems*, Requirements Engineering Journal, 3(2):143-150, Springer-Verlang, 1998.

[Papadopoulos and McDermid, 1999a] Papadopoulos Y., McDermid J. A, *Hierarchically Performed Hazard Origin and Propagation Studies*, in Lecture Notes in Computer Science, 1698:139-152, Proceedings of SAFECOMP'99, the 18th International Conference on Computer Safety, Reliability and Security, Toulouse France, Springer Verlag, 1999.

[Papadopoulos and McDermid, 1999b] Papadopoulos Y., McDermid J.A, *A new method for Safety Analysis and the Mechanical Synthesis of Fault Trees in Complex Systems*, in Proceedings of ICSSEA '99, 12th International Conference on Software and Systems Engineering and their Applications, 4(13):1-9, Paris, 1999.

[Papadopoulos, Mauri, McDermid, 2000] Papadopoulos, Y., Mauri, G., McDermid, J. A., Systematic Anticipation and Validation of Scenarios of Failure Propagation and Mitigation in PLC Controlled Processes, in: Proceedings of the PLC Conference, Cologne, 2000.

[Paula and Parry, 1990] Paula H.M., Parry G. W., *A Cause Defense Approach to the Understanding and Analysis of Common Cause Failures*. NUREG/CR-5460, Vol. 1. Office of Nuclear Regulatory Research, Washington, DC, 1990.

[Peyton and Peebles, 1987] Peyton Z. Peebles J., Probability, Random Variables and Random Signal Principles. McGraw Hill, New York, 1987.

[Picciolo, 2000] Picciolo G., *Guideline for Assessing PLCs Safety-Related Reliability*, 4th Symposium of Programmable Electronic Systems in Safety Related Applications. TUV Rheinland, Cologne, 3-4 May, 2000.

[Poucet et al., 1987] Poucet A., Amendola A., Cacciabue P.C., Common Cause failure Relibility Benchmark Exercise, EUR 110554 EN, JRC Commission of the European Communities, April 1987.

[Poucet, 1990] Poucet A, STARS: Knowledge Based Tools for Safety and Reliability Analysis, *Rel. Engng. And System Safety*, Vol. 31, *Elsevier Science Publishers Ltd.*, England, pp65-90, 1990.

[Poucet et al., 1993 a] Poucet A., Carpignano A., Scheer., *STARS Project: Fault Tree Editor, version 1.1 – X11.5*. Technical Notes, No. I.93.111, European Commission ISEI, Ispra, Italy, 1993.

[Poucet et al., 1993 b] Poucet A., Carpignano A., Scheer., *STARS Project: Fault Tree Analyser, version 1.1 –X11.5*. Technical Notes, No. I.93.110, European Commission ISEI, Ispra, Italy, 1993.

[Poucet et al., 1993 c] Poucet A, J Wand & M A Wilikens., The Licensing of Safety Critical Systems Containing Software, EUR 15341 EN, JRC Commission of the European Communities, July 1993.

[Prasad, 1998] Prasad D. K., Dependable System Integration Using Measurement Theory and Decision Analysis. Dphil Thesis. The University of York. United Kingdom. Novenber, 1998.

[Rasmussen, 1992] Rasmussen Dale M., A comparison of the small and large event tree approaches used in PRAs. *Rel. Engng and System Safety*, Num. 37, Elsevier Science Publishers Ltd., England, pp. 79-90, 1992.

[Recht, 1966] Recht J.L., *Failure Mode and Effect Analysis*, National Safety Council, 1966.

[Ross, 1985] Ross T., Application and Extension of SADT. *IEEE Transaction of Software Engineering*, April, 1985.

[SAE-ARP 4754, 1996] EUROCAE ED-79/ SAE-ARP 4754, Aerospace Recommended Practice. Certification Considerations for Highly Integrated or Complex Aircraft Systems. Society of Automotive Engineers, Inc., 1996.

[SAE-ARP 4761, 1996] Society of Automotive Engineers, *ARP-4761: Aerospace Recommended Practice: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*, 12th edition, SAE, 400 Commonwealth Drive Warrendale PA United States, 1996.

[Sardella, 1995] Sardella R., A Review of Knowledge-Based Systems for Fault Tree/Event Tree Construction. European Commission, Tech. Note No. I.95.21, 1995.

[Simpson, 1994] Simpson H., Methodological and Notational Conventions in DORIS Real Time Networks British Aerospace Dynamics Division, 1994

[Taylor, 1982] Taylor J. R., *An Algorithm for Fault tree construction. IEEE Transactions on Reliability*, R-29(1):2-9, IEEE, 1982.

[T-Book, 1992] The ATV Office, *Reliability Data of Components in Nordic Nuclear Power Plants*. 3rd Edition. The ATV Office, Vattenfall AB, Sweden, 1992.

[Thisdell, 1994] Thisdell D., "The quick route to an emergency stop", New *Scientist*, 5 November 1994, p.20), 1994.

[Toy, 1987] Toy W. N, Fault-Tolerant computing. M.C. Yovits, editor, *Advances in Computer-Vol26*, pages 201-279. Academic Press, 1987.

[Vesely, 1981] Vesely W. E, *Fault Tree Handbook*, US Nuclear Regulatory Committee Report NUREG-0492, US NRC Washington DC United States, 1981.

[Villemeur, 1991] Villemeur A., Reliability, Availability, Maintainability and Safety Assessment; John Wiley and Sons, inc., ISBN 0 471 93 048 (v1), ISBN 0 471 93 049 (v1), 1991.

[Villemeur, 1992] Villemeur A., Reliability, Availability Maintainability and Safety Assessment, John Willey and Sons Ltd, ISBN 0-471-93048-2, 1992.

[Virolainen, 1993] Virolainen R K., State of the Art of Level-1 PSA Methodology, Committee on the Safety of Nuclear Installations (CSNI) OCDE Nuclear Energy Agency, Issy-les-Moulineaux, France, January 1993.

[Wang et al., 1988] Wang j., Modarres M., Hunt R. N. M., Probabilistic Risk Assessment: a look at Role of Artificial Intelligence. Nuclear Engineering and Design, Vol. 106-3, pp. 375-387, 1988.

[Wilson & McDermid, 1995] Wilson S.P. and McDermid J.A., "Integrated analysis of Complex Systems", *The computer Journal, Special Issue on Engineering System*, 1995.

[Yamada, 1977] Yamada K., Reliability Activities at Toyota Motor Company, Rep. Stat. App. Res., JUSE, 24(3), 1977.

[Yourdon and Constantine, 1986] Yourdon E., Constantine L., Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design, Prentice Hall, September 1986, ISBN: 0-13854-471-9, 1986.