Institute of Computer Architecture and Computer Engineering
University of Stuttgart
Pfaffenwaldring 47
D–70569 Stuttgart

Diplomarbeit Nr. 3069

Simulation Framework for Built-In Diagnosis of Self-Checking Circuits

Laura Rodríguez Gómez

Course of Study:	Computer Science
Examiner:	Prof. Dr. Hans Joachim Wunderlich
Supervisor:	Dipl.-Inf. Melanie Elm
Commenced:	July 19, 2010
Completed:	January 18, 2011
CR-Classification:	B.8.1, G.3

Contents

1 Introduction 7

2 Theory 9
2.1 Faults in Digital Circuits . . . 9
2.1.1 Classification of faults . . . 9
2.1.2 Behavior of the circuit . . . 10
2.1.3 Importance of distinguishing among different types of faults . . . 11
2.2 Diagnosing Digital Circuits . . . 11
2.2.1 Cause-effect Analysis . . . 11
2.2.2 Effect-cause Analysis . . . 12
2.3 Self-Checking Circuits . . . 12
2.4 Bayesian Networks . . . 13
2.4.1 Basics of Probability . . . 14
2.4.2 Structure of a Bayesian Belief Network . . . 15
2.4.3 Bayesian Inference . . . 17
2.5 Applications of Bayesian Networks and a Related Approach . . . 21

3 Circuit Diagnosis with Bayesian Belief Networks 23
3.1 Identification of the fault . . . 23
3.1.1 Construction of the network . . . 24
3.1.2 Inference . . . 28
3.1.3 Interpretation . . . 31

4 Experiments and Results 35
4.1 Output of the program . . . 35
4.2 Experiments . . . 36
4.2.1 Permanent faults . . . 36
4.2.2 Intermittent faults . . . 37
4.2.3 Transient faults . . . 38
4.2.4 Impact of faults memory size on diagnosis accuracy . . . 39

5 Conclusion and Further Work 41

Bibliography 43

List of Figures

2.1 Example of a fault in a circuit 10
2.2 Structure of a self-checking circuit 12
2.3 Structure of a self-checking circuit 12
2.4 Example of a basic Bayesian belief network 15
2.5 Division of the net for exact inference with message passing. 18
2.6 First step of inference: a query about X is received. 19
2.7 Second step of inference: reaction of adjacent nodes. 20
2.8 Third step of inference: adjacent nodes of successor send information 20
2.9 Phases of method for transient-permanent fault distinction. 22

3.1 Architecture capable of error logging in fail memory. 23
3.2 Example of a simple diagnosis net 24
3.3 First iteration of inference: first detected fault is created with its possible causes as predecessors. 25
3.4 Second step of inference: second detected fault is added as child of its cause. . 25
3.5 Third step of inference: third fault is added as child. 25
3.6 Fourth iteration: a fault is added as child. Its second cause, which did not belong to the net yet, is created. 26
3.7 Final iteration: fifth output is added, making the net complete and ready for inference. 26
3.8 Initial probabilities for the first level nodes. 27
3.9 Tables of detection probabilities. 28
3.10 Pi and lambda message passing to infer belief in node F1. 29
3.11 Lambda messages from O2 and O3 to F1. 30
3.12 Intersection of the identifying pattern sets of different faults. 32
3.13 Occurring faults distributed over the patterns space. 33

4.1 Common detecting patterns for many faults in c17. 38

List of Tables

2.1 Example of fault dictionary 11
2.2 Probabilities of catching the first bus. 16
2.3 Probabilities of catching the second bus (conditioned to if he caught the first one). 17
2.4 Probabilities of being late for work (conditioned to if he caught the second bus). 17

4.1 Results of diagnosis of permanent faults 36
4.2 Results of diagnosis of intermittent faults 37
4.3 Diagnosis accuracy of permanent faults depending on fail memory size. 39
4.4 Diagnosis accuracy of intermittent faults depending on fail memory size. . . . 39

1 Introduction

In some industries such as the automotive, reliability is a necessity to meet safety requirements. Self-checking circuits are one option to monitor the reliability of circuits on the field.

Self-checking circuits indicate an error whenever a wrong output occurs. The root cause of this wrong output, however, can necessitate different counter measures. Hence, it is important to analyze root causes of errors thoroughly.

In this thesis, an analysis method for in-field errors of self-checking circuits is developed along with a simulation framework for validation of the method. The method itself is based on Bayesian networks.

Bayesian networks are widely used in problems of classification that need to deal with uncertainty, a category to which fault analysis belongs. The goal is to differentiate between permanent, intermittent and transient faults.

An example scenario for the application of the diagnosis method presented in this thesis could be the following: a self-checking circuit in a car, which suddenly registers faulty outputs. The system would show an error message, but it is important to identify which kind of fault is occurring in the hardware. There could be a permanent, intermittent or transient fault. The first two are solved by replacing the faulty component, while transients generally require to restart the circuit or set it back to its last correct state. Thus, depending on the classification of the fault in the system, a different decision is to be made. The diagnosis method presented in this thesis can support the decision making by locating and classifying the fault.

2 Theory

The theory basics for understanding this thesis are presented in this chapter. First, faults in circuits and their behavior are discussed. Then, we will have a look at error detection with self-checking circuits, explaining afterwards the different techniques for fault diagnosis. Finally, an explanation about Bayesian belief networks is included, followed by a brief overview of some related work.

2.1 Faults in Digital Circuits

The distinction between different faults occurring in a circuit is crucial for safety and economic reasons. In this section the different kinds of faults to be found in a circuit are described, as well as their differences in behavior and the reason why it is important to distinguish among them.

2.1.1 Classification of faults

Faults are generally classified into types depending on the defect mechanism causing the faulty response and the timing behavior [BGJ+09]. According to these criteria, we have three types of faults. The first of them are the permanent faults. Their cause and behavior do not change with time. Note that this does not mean that every output will be faulty if a permanent fault occurs, only the ones affected by the fault. As an example, in figure 2.1 we can see that the output of the function g is stuck at 0. This may be caused by an unwanted permanent connection of g to ground. This will not affect the result when the output of g is actually supposed to be 0, or when the output of function f is 0, because in either case the final output O of the circuit should be 0. The fault can only be detected when both outputs f and g are 1, but this does not mean that the fault does not occur in the other cases[BGJ+09][BA00].

Let us imagine now that the fault in the previous figure is not permanent. It is not stuck at 0, but sometimes a 0 is read as input of the and gate when the output of g is supposed to have a different value. This affects the output of the circuit in irregular intervals. It is an example of an intermittent fault, which has a fixed cause but changes with the time [BGJ+09][BA00]. It is due to unstable hardware, and because of aging, sometimes these faults end up evolving into permanent faults [SGM+08].

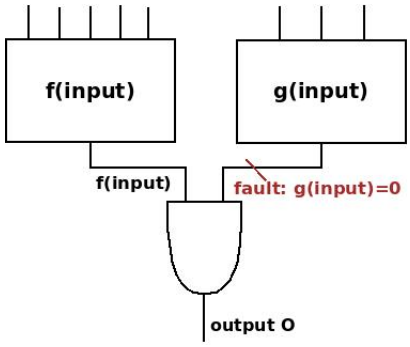

Figure 2.1: Example of a fault in a circuit

The last type of faults has a different nature to the previous two. They are called transient faults, and are generally caused by external conditions. The most common cause for transient faults are alpha particles. If an alpha particle strikes on a memory or storage element, it can induce data corruption. If it strikes on a combinational element, it can change the value of a signal and propagate the resulting error to the outputs, making the circuit return a faulty output. These errors can only be detected and/or corrected using redundancy [WWW06] [BGJ+09].

2.1.2 Behavior of the circuit

Because of the different nature of these faults, the effects they have on the outputs of the circuit or component are different. They all imply wrong outputs, but with different timing behavior. This characterization is extremely important to discriminate among all three types of faults.

Transient faults generally occur in bursts in different locations, but disappear rapidly and are solved by masking the error or by restarting the system if possible [BCGG97]. Characterization of intermittent faults is described in [Con07]. Errors induced by an intermittent fault appear in bursts and at the same location, which is a reasonable behavior to be expected from the description of intermittent faults introduced in the previous section. Permanent faults also occur always at the same location, but because they are more regular, they tend to provide faulty outputs with a higher frequency than intermittent faults. Generally, thresholds are used to differentiate among permanent, intermittent and transient faults. The more frequent the errors are, the more probable a permanent fault is assumed. Subsection 2.5 will provide further information on this matter by explaining some developed methods to distinguish among them.

2.1.3 Importance of distinguishing among different types of faults

The reason for which the identification of the type of fault in a circuit is important is the influence it has on the decision to be made once the fault is detected. Generally, in the case of an intermittent of permanent fault, the faulty component has to be replaced by a new one. In the case of a transient fault, however, there is not much to be done, since the fault is caused by external causes and does not exist within the circuit, once the conditions are not there anymore, the fault will disappear. Confusing a permanent or intermittent fault with a transient fault could have disastrous consequences. In the case, for example, of a transient fault diagnosed as a permanent or intermittent fault, the faulty component would be replaced without a real need. The economic cost of replacing every component under the temporarily effects of transient faults is too high to be afforded, and completely unnecessary. The opposite situation, in which a permanent or intermittent fault is taken for a transient, might mean even worse consequences. Not replacing the faulty component could result in a lack of safety or in premature failure of the whole system [PSBDG98].

2.2 Diagnosing Digital Circuits

In logic circuits, diagnosis is the process of narrowing down the location of a defect detected in the circuit. There are two different approaches for fault diagnosis: cause-effect and effect-cause. The first assumes a set of faults from which the effects have to be explained, while the latter goes from the effects to the causes. Information of this section is extracted from [WWW06].

2.2.1 Cause-effect Analysis

The cause effect analysis is based on the precomputation of a so-called dictionary of faults. The candidate faults are set from the beginning of the diagnosis. For every candidate fault, every input vector is simulated and the output is stored in the dictionary, resulting in a table like the one shown in 2.1. The output provided by the circuit when test patterns are provided is then compared to the information of the dictionary. Assuming that a certain faulty circuit provides "101100", fault F3 will be assumed.

	F1	F2	F3	F4	F5
T1	0	1	1	0	0
T2	1	1	0	1	1
T3	1	0	1	0	1
T4	0	1	1	1	0
T5	0	0	0	1	0
T6	1	1	0	0	1

Table 2.1: Example of fault dictionary

The table shows an example dictionary for a set of five candidates (F1 to F5) , for which up to six patterns are provided (T1 to T6). Note that faults F1 and F5 are not distinguishable, and that a fault that provides any other sequence for the six patterns cannot be diagnosed. This is one of the greatest limitations of fault dictionaries: a fault that is not modelled and included in the candidates set at the beginning cannot be diagnosed. Another problem is the size of the dictionary, since usually both circuits and sets of fault candidates are large.

2.2.2 Effect-cause Analysis

The effect-cause analysis is different from the previously presented one in that there is no set of candidates, but it is rather created when trying to find out the cause of the "symptoms", this is, the detected faults. The basic concept in effect-cause analysis is the fanin cone of an output, which is the collection of logic gates that affect this output. Depending on the number of faults assumed, a different part of the fanin cones of the inputs is discarded as candidates. Figure 2.2 shows two bits of the output that have an incorrect value (meaning that it is not the expected value). Each of them has their fanin cone drawn on the circuit. In the case of expecting one single fault, only the intersection of both fanin cones is taken into account. When multiple faults are expected, the union of all fanin cones of every faulty output are considered as candidates.

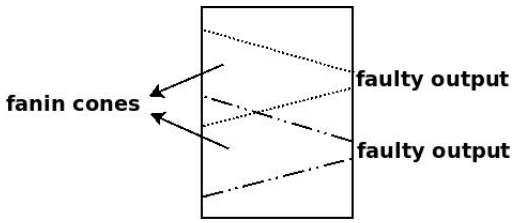

Figure 2.2: Structure of a self-checking circuit

2.3 Self-Checking Circuits

Self-checking circuits provide online fault detection, the ability to verify automatically if there is any fault in the logic without needing to apply additional external stimuli.

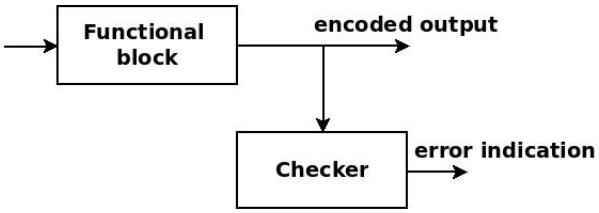

Figure 2.3: Structure of a self-checking circuit

They usually have a structure similar to the one shown in figure 2.3. The inputs are provided to the functional block, which returns the outputs encoded. The code is formed by the set of correct outputs. A circuit is referred to as totally self-checking[Lalo1] if it is self-testing (the occurrence of a fault from the assumed set is detected by at least one input word) and fault-secure (for any fault in the set, the circuit output is either a correct code word or a non-code word).

A circuit can also be strongly fault-secure, which means that for a fault f1 included in a set of faults F, either

1. the circuit is self-testing and fault-secure or
2. the circuit is fault-secure for f1, and if another fault f2 occurs in the circuit, then either the previous property or this one is true for the fault sequence f1 $\cup$ f2.

In a self-checking circuit, the checker is supposed to verify that the input word belongs to the chosen error detecting code. Otherwise, it should return an error indication. Obviously, the reliability of self-checking circuits depends on the behavior of the checker. Any of the techniques introduces redundant hardware and increasing power consumption, so there is a need to find a compromise between the complexity of the design, the overhead the checker implies and the fault coverage. Among the very different possibilities, parity checkers are very widely used because it is very effective under the classical assumption of single faults [KL04].

Fault tolerant techniques are frequently used in high reliability systems. Because self-checking design techniques provide on-line error detection, they are often combined in safety critical systems, especially in space, avionic or transport. An example of a fault tolerant system is shown in [SMM02], where the design and implementation of a controller for track-side railway systems are described.

2.4 Bayesian Networks

Probabilities are a very useful tool for problems in which we have to deal with uncertainty. When dealing with causality, there is the need to reflect that the cause makes the effect more probable, but not absolutely certain. For example, the proposition "if you drive too fast you may have an accident" does not mean that every fast driver has an accident, but only that driving fast can make the possibility of having an accident more likely. In this thesis we will use Bayesian belief networks, a graphical model that uses probabilities to infer the cause of provided evidence. Explanation of basic concepts of probability, needed to understand the inference, as well as descriptions of the model and the inference mechanism follow.

2.4.1 Basics of Probability

The very basic notions of probability of this section can be found in more detail in [Pea09]. We are just introducing the necessary concepts to understand the following sections. Let us start with the first important axiom of probability. Let A be an event from a certain set of events S. For any proposition A, it is verified that:

$$0 \leq P(A) \leq 1$$

being

$$P(sure\ proposition) = 1$$

For two events A and B, if they are mutually exclusive we can say that

$$P(A\ or\ B) = P(A) + P(B)$$

or, in other words, the belief assigned to a set of events is the addition of the beliefs of its non-intersecting components. For any given proposition A, its complementary not A covers the rest of the space, and it is mutually exclusive with it. Since it is sure that either A or not A will happen, we can also be sure that

$$P(A) + P(\not{A}) = 1$$

We may also have an event that gives us some background to estimate the belief in another event. The example of "someone having an accident", which we will now refer to as proposition A, has initially a relatively low probability, P(A). However, if we now know that the person in the proposition drives too fast (proposition B), the belief that he might have an accident increases. This probability is called a conditioned probability and is written $P(A|B)$. It can be calculated as

$$P(A|B) = \frac{P(A\ and\ B)}{P(B)}$$

Note that this is also true for B conditioned to A, which gets us to the formula

$$P(B|A) = \frac{P(B\ and\ A)}{P(A)}$$

so from the first equation we get that

$$P(A\ and\ B) = P(A|B)P(B)$$

and from the second

$$P(A\ and\ B) = P(B|A)P(A)$$

so

$$P(A|B)P(B) = P(B|A)P(A)$$

Now, if we imagine that A is a proposition (such as that of having an accident) and B is a hypothesis that has influence on A (such as "driving too fast"), what we have is the fundament of Bayesian inference. The formula

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

means that the belief we have in the hypothesis B upon some evidence A can be calculated by multiplying our previous belief in B being true by the probability that A will happen knowing that B is true. In other words, we are calculating the probability that someone was driving fast knowing that he had an accident. The P(A) in the denominator is only normalizing to make sure that $P(B|A)$ and its complementary $P(notB|A)$ sum to unity.

2.4.2 Structure of a Bayesian Belief Network

A Bayesian belief network is a directed acyclic graph that models the facts of a problem and their conditional dependencies. The facts are actually random variables represented in the graph by nodes [Lug97]. A certain node N being the origin of an arc means it is the cause, while the destination node N2 is the effect. It is also said that N is predecessor or parent of N2, and N2 may be referred to as the child or successor of N. To illustrate the structure of Bayesian belief networks more clearly, let us consider, for example, the following problem: a man must take two buses to get to work, one bus after the other. There is a chance that the man will not be able to catch the first bus (if he oversleeps, for example), and it could also happen that he misses the second bus (for instance if the first bus is trapped in a traffic jam). Also, what happens in the second bus has an effect on the man being late for work. We model this in a Bayesian net with a structure as the one shown in figure 2.4.

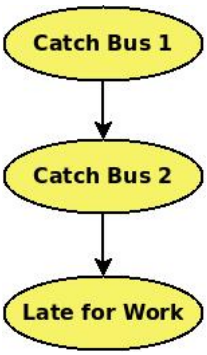

Figure 2.4: Example of a basic Bayesian belief network

Note that, although there is obviously an effect of the first node variable on the third (if the man does not catch the first bus, he will not be on time), it is not necessary to add an arc from "Catch first bus" to "Late for work" to explicitly represent this cause-effect relationship, since the arcs shown on the net already imply that there is an effect of "Catch first bus"

on ”Late for work”. ”Catch first bus” influences ”Catch second bus”, which at the same time influences ”Late for work”, so whatever effect the first bus has on the second, it will be reflected in the second node and thereby this information will be transfered to the third node.

Every variable can have many different values. In this example, all the three variables are binary, with both the bus nodes having {Caught, Missed} as their range of values, and the third node being {True, False}. However, it is possible that we are interested not only in if the man is late, but also how late he is, so the range for ”Late for work” could be {On_time, Late_less_than_30, Late_more_than_30}, to make it possible for us to represent if he is not late, or he comes more or less than half an hour late.

This representation is very intuitive and it gives a clear idea of the relationships between nodes. However, there is still some information missing for the network to be complete. A Bayesian belief network must also be annotated with conditional probabilities. Nodes can be of two types: discrete or continuous. The discrete ones generally have probabilities defined by tables (one entry for each value), while continuous nodes have their probability described by a function. Every node must have a table of probabilities containing the conditional probability of every outcome conditioned to every possible combination of outcomes for its parent nodes. Nodes in this example are discrete, so their probabilities are shown in tables. Table2.2 shows these probabilities for the first node, ”Catch bus 1”. Because it has no parents, ”Catch bus 1” is called a root node.

	Conditions: none (no parents)
P(Catch bus 1 = Caught)	0.95
P(Catch bus 1 = Missed)	0.05

Table 2.2: Probabilities of catching the first bus.

Node ”Catch bus 1”, since it has no predecessors, only shows the probability of the variable taking each of the values, which is given in the form of a vector: (0.95, 0.05). The first component of the vector corresponds to the probability of the first value of the range (”caught”), while the other corresponds to the second possible value (”missed”). The addition of both probabilities must be 1, for either the man will catch the bus or miss it, there is no third possibility. For nodes that have predecessors, the table of probability changes slightly: there are no longer absolute probabilities, but the probability of what will happen conditioned to what happens in the predecessors. In the case of ”Catch bus 2”, the probabilities needed are conditioned to ”Catch bus 1”, as shown in table 2.3.

The meaning of this table is clear: if the man caught the first bus, then it is very probable that he will catch the second. However, if he missed it, then it is very unlikely that he will catch the second bus. A similar table (2.4) serves for the third node, ”Late for Work”.

	Catch_bus_1 = Caught	Catch_bus_1 = Missed
P(Catch bus 2 = Caught)	0.95	0.02
P(Catch bus 2 = Missed)	0.05	0.98

Table 2.3: Probabilities of catching the second bus (conditioned to if he caught the first one).

	Catch_bus_2 = Caught	Catch_bus_2 = Missed
P(Late for Work = True)	0.95	0.02
P(Late for Work = False)	0.05	0.98

Table 2.4: Probabilities of being late for work (conditioned to if he caught the second bus).

In this example probabilities have been assigned "randomly". However, in a real problem probabilities would be either based on previous observations (if the buses are late frequently), and if no statistical data is available, then the data contained in the tables are actually subjective probabilities that the experts of the problem provide. The structure of the net must also be defined by an expert in the problem we are trying to model [FN07].

Depending on its characteristics, there are different types of Bayesian networks to be distinguished. They can be classified according to the following aspects:

1. structure: Bayesian networks can be singly-connected, if there is only one undirected path between any given pair of nodes (this is also called a poly-tree), or multiply-connected, if there is more than one path between any given set of nodes.
2. type of nodes: if every node is discrete, the net is also said to be pure discrete. If every node is continuous, the net is pure continuous, and in other case it is called a hybrid net.
3. temporal perspective: we can have static or dynamic networks. Static networks do not take time into account, and are generally used for classification. Dynamic networks, on the other hand, allow the modelling of temporal evolution by having the nodes of network t connected to those of network of time $t + \delta$

In our example, as well as in the developed diagnosis strategy, the networks are discrete and static.

2.4.3 Bayesian Inference

The structure of the network explained in the previous subsection allows us to ask about some variables upon some evidence. In the previous example, the network gives us the possibility to find out the probability of the man being late for work knowing that he missed the first bus, or the probability that he was late for the first bus if he was on time for work, or even the probability that he caught the second bus if he was late for work but caught the first bus on time. Inference is the computation of results for a query about the belief of

a node. In Bayesian networks, inference has been proved to be an NP-hard computational problem [GM05]. The very basic inference strategy is that of lambda-pi message passing, introduced by Pearl[KP83]. It is an exact inference query-oriented strategy, which means it allows the user to ask about the probability of one node, and the result returned by the net will be an exact number. There is also approximate inference, which returns boundaries in which the exact probability is assured to belong. In this work message passing is employed and will be described in the following paragraphs.

The fundamental idea of this strategy is to distinguish between the evidence "before" the node (its ancestors) and the evidence after it (the sucessors). The nodes are divided into these two categories for a query about a certain node X. The nodes in the cloud e+ are the evidences ”over” the node we are querying about, the ones in the cloud e- the evidences ”under” it. Because this algorithm is only applied in loop free graphs, any graph is easily divided into the scheme shown in figure 2.5.

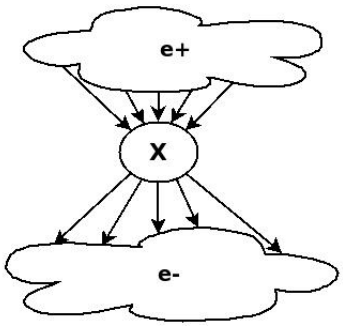

Figure 2.5: Division of the net for exact inference with message passing.

The computation of the updated belief of a certain node X based on the knowledge we now have in the net involves the following pieces of information:

1. $B(X)$: the belief we are looking for. It is computed using the following formula:

   $$B(X) = \alpha\Lambda(X)\Pi(X)$$

2. $\Lambda(X)$: the information received from the successors. It is calculated as the product of all the messages received from the successors, this is:

   $$\Lambda(X) = \prod_{X\_successors} \lambda_{y_j}(X)$$

3. $\Pi(X)$: summarizes the information received from predecessors. Its value is calculated as:

   $$\Pi(X) = \sum_{u_1}^{u_n} p(X|k) \prod_i \pi_X(i)$$

and as the reader can see, it is also based on the messages received from the predecessors.

4. $\pi_{y_j}(X)$: message sent from a predecessor towards node X. It contains the value

$$\pi_{yj}(X) = \alpha\Pi(X)\prod_{k\neq j}\lambda_{y_k}(X)$$

where α is only a normalizer.

5. and finally, $\lambda_X(u_i)$, the vectors sent from the successors towards node X. This vector is calculated as:

$$\lambda_X(u_i) = \beta\sum_x \Lambda(x)\sum_{u_k,k\neq i} p(X|u_1,..,u_n)\prod_{u_k,k\neq i}\pi_X(u_k)$$

with β working again as a normalizer.

When the inference starts, the direct successors and predecessors of node X have to compute their λ and π messages. It is easy to see that to do this, they actually need information from their own successors and predecessors. Every node will continue to do this until the root and leaf nodes are reached. When node X gets all messages, it will calculate its Π and Λ components and then update its belief. In figure 2.6 we have a possible net. A query about node X is done, so the inference starts.

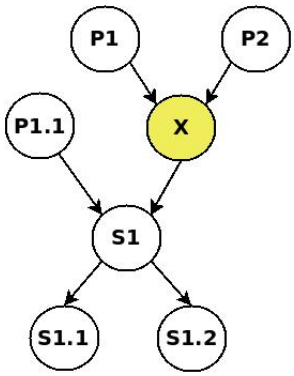

Figure 2.6: First step of inference: a query about X is received.

Predecessors P1 and P2 have to send their π messages. Because they are root nodes and have no successors, their π value is their probability table. They both send it to X. Successor S1, however, cannot send its λ value because it does not have all the information about its surrounding nodes(figure 2.7).

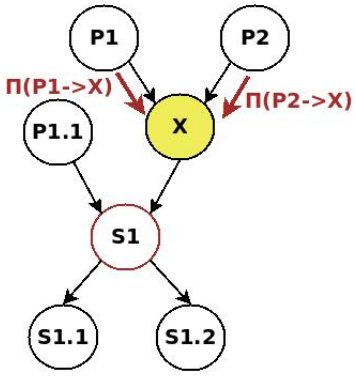

Figure 2.7: Second step of inference: reaction of adjacent nodes.

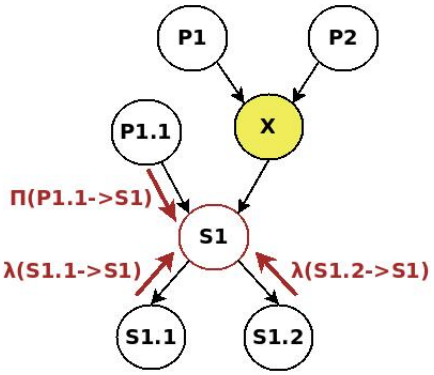

Figure 2.8: Third step of inference: adjacent nodes of successor send information

To calculate its value, it needs information about P1.1, S1.1 and S1.2, so the next step of the algorithm is to get this information (figure 2.8). Finally, S1 is ready to compute its λ value and send it to X, provided now with all the information to update its belief.

There are, however, some particularities to be taken into account when inferring with message passing. An evidence node is a node about which we already have evidence. It may be any node of the net. The propagation of the messages stops when an evidence node is hit. In the previous example, if we had had any information about node S1, there is no need to ask for information about its predecessor P1.1 or successors S1.1 or S1.2, since none of them will provide any more information than evidence about S1, so in this case, step 3 illustrated in figure 2.8 would not be necessary. Instead, S1 would send its information at the same time as P1 and P2.

2.5 Applications of Bayesian Networks and a Related Approach

Bayesian networks are widely used in domains where information is missing, because they deal very well with uncertainty. An example is shown in [BGH+02], where a model for troubleshooting in communication networks is shown. Troubleshooting was a manual process involving a series of checks and queries to applications and databases at the time of the development of this tool, and the idea was to make it faster and easier by building a tool that would do most of the inference.

Another field where Bayesian networks are used is the medicine domain, especially in illness diagnosis. Tools based on Bayesian networks are intended as medical decision support. The general Bayesian network scheme applies, only now the symptoms are observed in the patient, and the causes are sets of illnesses. In [Els10], a Bayesian network is used to predict the severity of breast masses. Some other real-world examples of Bayesian network applications include forecasting, automated vision and diagnosis, such as understanding and debugging of programs or troubleshooting applications, as well as system diagnosis.

Diagnosis is a very important part of an electronic product life cycle in industry, particularly during manufacture and maintenance. Generally, this task is performed by a human diagnostician. Although with low industrial acceptance, for the last three or four decades different approaches based on Artificial Intelligence have been developed by researchers to make the process more automatic. They include methods to deal with uncertainty. Fault models, causal models (where Bayesian networks belong to), models based on structure and behavior and diagnostic inference models are new techniques applicable for automated fault diagnosis. A more extense review is available in [FMM01].

Distinction between transient and permanent faults is an important target of research since it affects the decision to be made once the system is diagnosed. In [PSBDG98] an approach is presented in which Bayesian networks are used to distinguish between these two types of faults. To achieve this, the testing is divided into two phases: observation and adjudication (figure2.9). They correspond to the gathering of information produced by the monitorized module and to the determination if the produced output was right or not, respectively. This method is intended for circuits with some kind of fault detection, although no fault tolerance is needed. Because of the wide range of scenarios in which this diagnosis might be applied, these two phases vary depending on the case. For modular redundant, voted architectures, the observation is the production of data by the modules, while the adjudication is the voting, when the voter decides if the module gave a right or wrong output. For the second scenario, architectures using code signatures for low level error detection, the observation phase includes the generation of the output. The adjudication is the comparison between the generated signature and the precomputed one. Not fault-tolerant architectures that have some error detection, such as parity, include the gathering of outputs of the circuit in the observation and have a very trivial adjudication that merely consists of recording a failure as

a faulty output in the second phase. Finally, architectures in which modules are periodically taken off-line and tested provide the results of test programs for the observation phase. The adjudication is the analysis of the results.

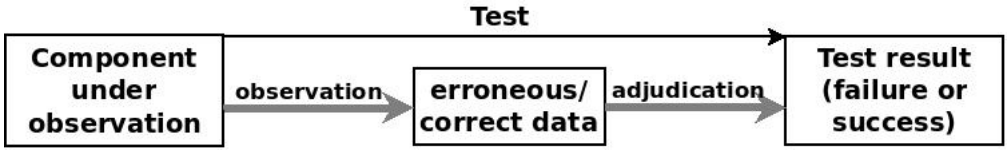

Figure 2.9: Phases of method for transient-permanent fault distinction.

The determination of if the fault is permanent is done according to the number of consecutive faults observed. In an infinite succession of observations, the probability of a permanent fault P(perm) is L, with L being higher than 0, because even if a fault has not been diagnosed, it is possible that a fault will occur in the future or that the module became faulty recently and has thus not been detected yet. If the infinite succession is of failures, then the fault is diagnosed as a permanent fault. Mixed series of failures and successes will result in oscillation of the value of the probability of having a permanent fault. In this case, a ration of successes/failures is needed to establish if the fault was permanent or not.

3 Circuit Diagnosis with Bayesian Belief Networks

Our goal is to develop a diagnosis method for self-checking circuits. It is an effect-cause strategy that combines Bayesian belief networks with thresholds, and should be able to locate the fault and to identify the type of fault it is, as this information is crucial for determining what to do next with the faulty component. In this section the complete functioning of the diagnosis is explained.

The starting point of the diagnosis is a log of the errors occurred in the circuit. We assume an architecture such as the one shown in figure 3.1, consisting of a self-checking circuit connected to a fail memory. The circuit's output is compared to that of the parity checker, and marked as an error if it does not match. In case of an error, the input provided to the circuit and the output it returned are provided to the fail memory as data. With the first logged fault, a counter will start increasing with every pattern provided to the circuit, to generate a sort of time stamp the diagnosis tool will later use for fault type discrimination.

3.1 Identification of the fault

The first thing to do in the diagnosis is to determine which fault is causing the incorrect behavior. This is what the Bayesian network is used for: it is used in this thesis to diagnose circuits from a fault log. The starting point of the diagnosis is a file with the record of faults occurred during execution, a set of the possible faults in the circuit, and a model of the circuit for simulation. An explanation of the structure of the net and its automatic construction, as well as the calculation of the probabilities to annotate the graph follows.

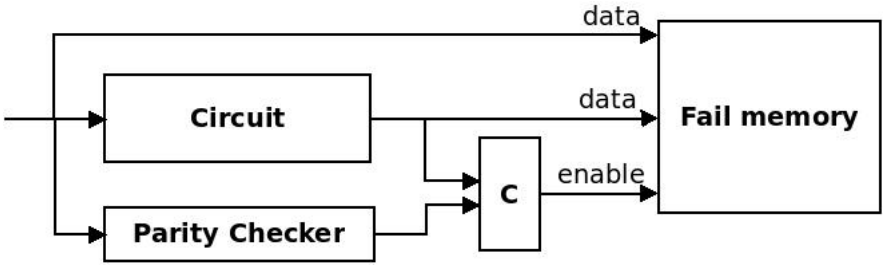

Figure 3.1: Architecture capable of error logging in fail memory.

3.1.1 Construction of the network

As explained in section 2, the structure of a Bayesian network is that of an acyclic directed graph. In our problem, we consider two kinds of nodes: fault nodes, also referred to as first-level nodes, and failed output nodes, or second-level nodes. The first group represent the possible faults in the circuit. They are the origin of the arcs because they are the cause of the faulty outputs (some fault causes the circuit to give back a wrong output). The nodes in the second group, on the other hand, represent every pair of input and faulty output registered in the log, and are the destination of the arcs. Generally the graph will be bigger and more complex, but a very simple net is shown in figure 3.2. It represents the net used to diagnose a small circuit which provided five faulty outputs (blue nodes O1 to O5), which may be caused by any, or any combination of faults of a set of three faults (yellow nodes F1 to F3).

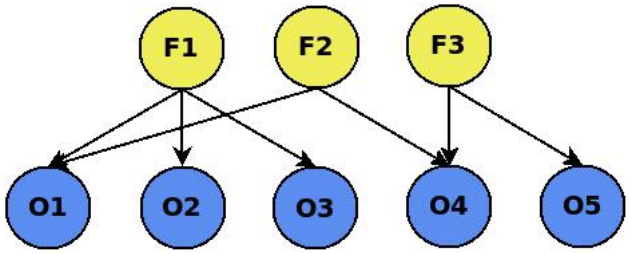

Figure 3.2: Example of a simple diagnosis net

The necessary information to build the net is obtained by reading the three files provided to the program. The children are created by reading the log file. Every line, this is, every faulty input/output pair creates a node in the second level. The possible causes for this fault are detected by simulation: the fault is injected as a permanent fault in a model of the circuit and then the input is provided. If the output returned by the circuit is the same as the one in the failure log, then the node corresponding to the injected fault is included if it did not exist in the net yet. To make this clear, we will now show the step by step construction of the net in figure 3.2

Initially, there is no information about the net. The program reads the first input and the wrong output provided by the circuit under diagnosis. Let us imagine that there are seven possible faults in such a circuit. The first of them is injected in the model, the input is provided and then the output is compared to the one the circuit returned. If it matches, the fault node is looked for, and if it is not there, it is added and the algorithm will jump to the next fault. Note that if when the fault is injected the output is wrong but does not match the one provided by the circuit, such fault will not be considered as a candidate by the diagnosis program. In our example, only faults F1 and F2 may have caused exactly this wrong output, so they are both created and added as parents of O1, which represents the first failure logged, as shown in the figure 3.3.

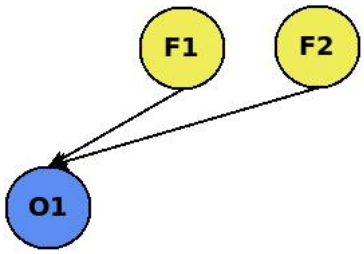

Figure 3.3: First iteration of inference: first detected fault is created with its possible causes as predecessors.

For the second faulty output, however, only F1 is identified as a possible fault. As it is also created, only the node O2 and the arc (F1,O2) are added to the net, as illustrated by figure 3.4.

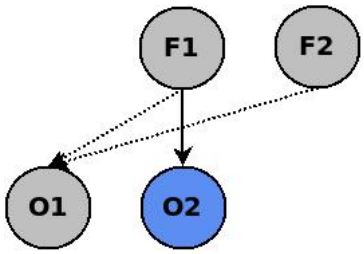

Figure 3.4: Second step of inference: second detected fault is added as child of its cause.

The case is the same in the third iteration: only F1 is a possible cause for the faulty output O3, so the colored part of figure 3.5 is added.

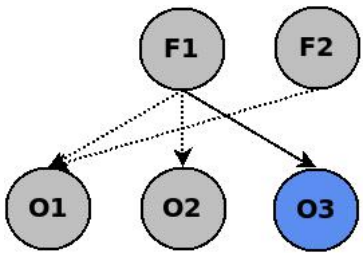

Figure 3.5: Third step of inference: third fault is added as child.

When the fourth input/faulty output is read, however, two possible candidates are found. Since the first one, F2, is already created, nothing is done for it. For the second one, on the other hand, the node must be created because it is the first time this fault is identified as a possible cause, so the node for the fault F2, the node for the faulty output O4 and the arc connecting both are added, as indicated by the colors in figure 3.6.

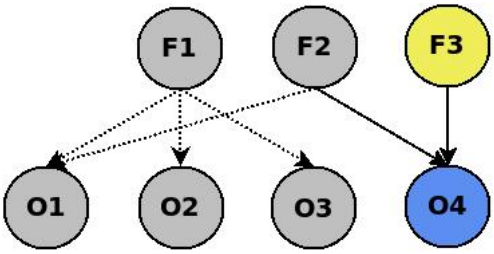

Figure 3.6: Fourth iteration: a fault is added as child. Its second cause, which did not belong to the net yet, is created.

Finally, for the fifth iteration, only the second level node is added O5, for its cause, fault F3, is already created on the net 3.7. The resulting net is exactly the one presented at the beginning of this section.

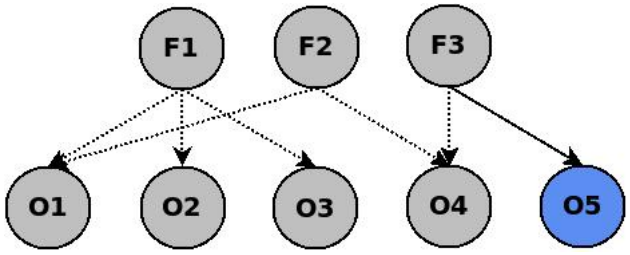

Figure 3.7: Final iteration: fifth output is added, making the net complete and ready for inference.

Although the structure of the net is now complete, the probability tables still have to be calculated. So that no expert is needed to set these probabilities, simulation is used to estimate them. For the first level nodes, we know that at least one of them is happening, but before the inference all of them are equally probable, so we first set the probabilities of the first level or parent nodes. As there is no condition because they are root nodes, their probabilities are vectors with two values: one for "Yes" (this is, the fault is in the circuit) and one for "No" (which means the fault is not really present in the circuit). Initially, for every root node we set the probability for "Yes" to 1/k, where k is the number of first level nodes, to make them equally probable. As explained in the previous section, the addition of the probabilities of all possible values of the variable, if they are mutually exclusive, has to be 1. For this reason, the second probability in the vector, which corresponds to the value "No", is set to 1 - P(yes). In the previous example, since we have three possible faults, we initialize the probabilities of occurring to 1/3 for all of them, and the probabilities of not occurring to 2/3 (figure 3.8).

The estimation of the probabilities of the second level nodes, which are conditioned to the first, changes slightly. The probabilities to be calculated now are P(O=Yes | F=Yes), which

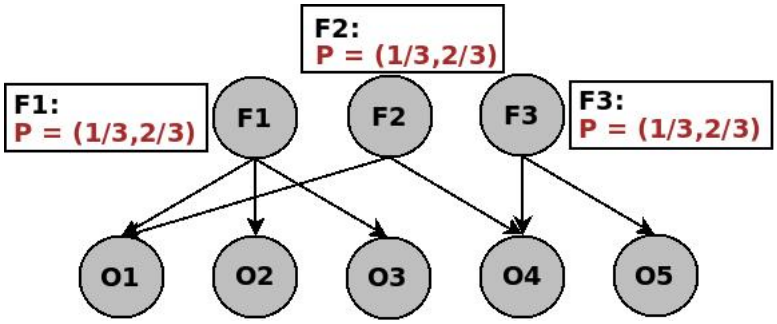

Figure 3.8: Initial probabilities for the first level nodes.

means the probability of observing faulty output O if fault F is in the circuit. To set this probability, we will take into account how detectable the fault is, this is, the probability that having detected fault F we observe output O is 1/n, where n is the total number of patterns that detect fault F. Again, simulation is required to set this number. For every possible fault in the net, every pattern or at least a reasonably large set is provided and the total number of failing patterns is stored in the fault node. The probability of the output occurring conditioned to the fault occurring too is easily calculated now:

$$P(O = Yes|F = Yes) = 1/F.patterns$$

where F.patterns is the number of patterns that will cause the circuit to return a faulty output. The probability of the output not being faulty if the fault occurs is again easy to calculate because it is the complementary of the formula above and thus must sum up to unity.

$$P(O = No|F = Yes) = 1 - P(O = Yes|F = Yes)$$

The probabilities of the output being faulty given that fault F does not occur depend on the other parents of the faulty output node. It is calculated as the addition of the probabilities of the circuit returning the faulty output given that the other parents occur:

$$P(O = Yes|F = No) = \sum_{p \in parents} P(O = Yes|p = Yes)$$

The complementary probability, this is, P(O=Yes | F=No), is calculated again as the difference with 1.

$$P(O = No|F = No) = 1 - P(O = Yes|F = No)$$

Because the example is not a real case, the number of detecting patterns of every fault has been set again "randomly". However, as explained, in the experiments the number are set with simulations, and thus correspond exactly to reality. The probability tables of the second level nodes are updated according to the number of patterns that detect the ancestors of a node.

We will develop the calculations of probabilities of node O1 to clarify it. P(O1 = Yes | F1 = Yes) = 1/12, since 12 is the number of patterns that detect F1. For the complementary value O1=No, P(O1 = No | F1 = Yes) = 1 - 1/12 = 11/12. The same happens for the probabilities conditioned to F2 occurring: P(O1 = Yes | F2 = Yes) = 1/20, while P(O1 = No | F2 = Yes) = 1 - 1/20 = 19/20. The probabilities conditioned to the parent nodes not occurring, on the other hand, are computed as follows: P(O1 = Yes | F1 = No) is the probability of the circuit returning O1 given that F1 is not present in the circuit. This means that some other cause of O1 must be present and O1 occurs. Since O1 has one ancestor other than F1, the probability of O1 occurring without F1 being present is the same as the probability of F2 being present and O1 happening, which in probability theory is expressed like this:

$$P(O1 = Yes|F2 = Yes) \cap P(F2) = (19/20) * (1/3) = 19/60$$

The same applies for P(O1 = Yes | F2 = No), which is computed as

$$P(O1 = Yes|F2 = No) = P(O1 = Yes|F1 = Yes) \cap P(F1) = (11/12) * (1/3) = 11/60$$

as observed in the tables in figure 3.9

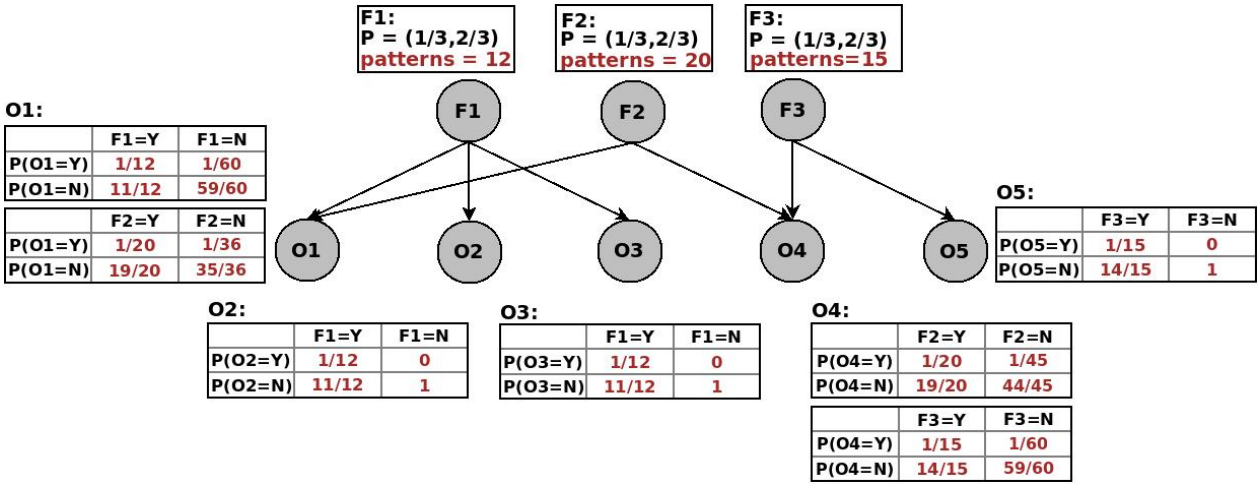

Figure 3.9: Tables of detection probabilities.

Once the network is built and annotated with the conditioned probabilities it is ready to start the inference.

3.1.2 Inference

It is easy to see that in the networks built by the diagnosis program, every second level node is an evidence node, for we only create one if the faulty output was returned by the

circuit. As the reader may remember from section 2.4.3, the sending of pi and lambda messages is stopped in every evidence node. Also, the order in which the evidences are observed does not change the final results of the net. For this reason and to make the diagnosis faster, all evidences are set at the beginning. This means probabilities for every second level node are set to (1.0, 0.0), since we know that the first value, "Yes", happened, so it is now impossible that the second, "No", will happen. After this, every first level node will request the lambda messages of all its successors. Since they are root nodes, no pi messages need to be requested. Pi values of root nodes are the same as their probabilities, which we already have. With all of them, the new probabilities for the first level nodes are computed according to the explanation given in section 2.4.3. Again, to make it clearer, we will now calculate the updated belief of node F1 in our previous example.

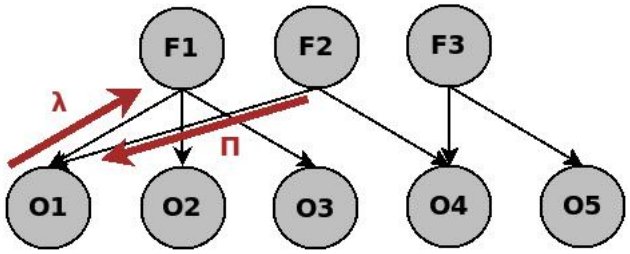

Figure 3.10: Pi and lambda message passing to infer belief in node F1.

We first start with F1 (figure 3.10). Since F1 has three successors, we calculate the lambda messages of all of them. F1 has two possible values, so the lambda messages sent by the successors will also have two components. For the first one, the first component of the message is calculated as follows:

$$\begin{aligned}\lambda_{O1}(F1 = Y) &= \lambda(O1 = Y) * P(O1 = Y|F1 = Y) + \lambda(O1 = N) * P(O1 = N|F1 = Y) = \\ &= 1.0 * 1/12 + 0.0 * 11/12 = 1/12\end{aligned}$$

For the second, the other parents of the node need to be taken into account. The probability of O1 happening if F1 does not occur is calculated as follows:

$$P(O1 = Yes|F1 = No) = P(O1 = Yes|F2 = Yes) * P(F2 = Yes) = 1/20 * 1/3 = 1/60$$

$$P(O1 = No|F1 = No) = 1 - P(O1 = Yes|F1 = No) = 59/60$$

These values are used to calculate the second component of the lambda message:

$$\begin{aligned}\lambda_{O1}(F1 = No) &= \lambda(O1 = Yes) * p(O1 = Yes|F1 = No) + \lambda(O1 = No) * p(O1 = No|F1 = No) \\ &= 1.0 * 1/60 + 0.0 * 59/60 = 1/60\end{aligned}$$

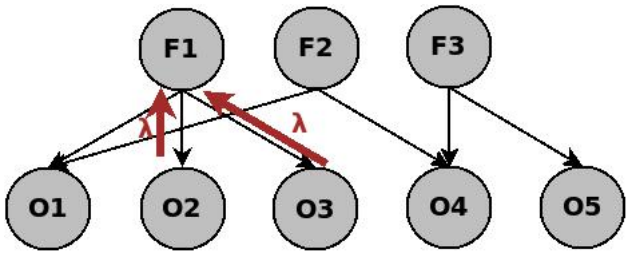

Figure 3.11: Lambda messages from O2 and O3 to F1.

The messages of the other two children (figure 3.11) are easier to calculate, since they have no other parents and the probability of getting those faulty outputs if F1 does not occur is null, so for O2:

$$\begin{aligned}\lambda_{O2}(F1 = Yes) &= \lambda(O2 = Yes) * P(O2 = Yes|F1 = Yes) + \lambda(O2 = No) * P(O2 = No|F1 = Yes) \\ &= 1.0 * 1/12 + 0.0 * 11/12 = 1/12\end{aligned}$$

$$\begin{aligned}\lambda_{O2}(F1 = No) &= \lambda(O2 = Yes) * p(O2 = Yes|F1 = No) + \lambda(O2 = No) * p(O2 = No|F1 = No) \\ &= 1.0 * 0.0 + 0.0 * 1.0 = 0\end{aligned}$$

and O3:

$$\begin{aligned}\lambda_{O3}(F1 = Yes) &= \lambda(O3 = Yes) * P(O3 = Yes|F1 = Yes) + \lambda(O3 = No) * P(O3 = No|F1 = Yes) \\ &= 1.0 * 1/12 + 0.0 * 11/12 = 1/12\end{aligned}$$

$$\begin{aligned}\lambda_{O3}(F1 = No) &= \lambda(O2 = Yes) * p(O3 = Yes|F1 = No) + \lambda(O2 = No) * p(O3 = No|F1 = No) \\ &= 1.0 * 0.0 + 0.0 * 1.0 = 0\end{aligned}$$

The lambda component of the node F1 is the multiplication of all the lambda messages received from its successors, thus:

$$\lambda(F1 = Yes) = \lambda_{O1}(F1 = Yes) * \lambda_{O2}(F1 = Yes) * \lambda_{O3}(F1 = Yes) = 1/12 * 1/12 * 1/12 = 1/1728$$

$$\lambda(F1 = No) = \lambda_{O1}(F1 = No) * \lambda_{O2}(F1 = No) * \lambda_{O3}(F1 = No) = 1/60 * 0 * 0$$

The pi values of node F1 are the same as its probability because it is a root node. $\pi(F1) = (1/3, 2/3)$ so the updated belief is calculated as

$$\begin{aligned}P(F1) &= \alpha * (\lambda(F1 = Yes) * \pi(F1 = Yes), \lambda(F1 = No) * \pi(F1 = No)) \\ &= \alpha * ((1/1728) * (1/3), 0 * (2/3)) = \alpha(1/5184, 0)\end{aligned}$$

The α is used to normalize, since the addition of the two possible values of the range has to be one. In this case it is very simple, since the second component is 0, the updated belief of F1 is (1.0,0.0). The same process is repeated for all the nodes in the first level, so in the end the result of the Bayesian inference on this example network are three pairs of values, each pair representing the probabilities of a node of occurring or not occurring. As the Bayesian network is used to identify the cause, assuming that there is only one cause, some interpretation of the results is needed in order to correctly understand the results provided by the network.

3.1.3 Interpretation

Bayesian belief networks do not return an answer, but rather a pair of probabilities for every node. For this reason, results of Bayesian inference have to be interpreted. The solution offered by the program is a set of faults with their final probability. Until every second level node has a cause included in the solution, the program picks the highest probability of the remaining set and adds it to the solution. In our example, the probabilities of occurrence of both F1 and F3 are 1.0, while F2 has a lower probability. The program will then return F1 -> 1.0 , F3 -> 1.0. This means that either F1 or F3 (sometimes both) happened. This information is not enough: we need to know if it was a permanent or an intermittent fault. To distinguish both, we use a threshold and compare the time stamp to it. The time stamp used is the number of inputs the circuit received since it registered the first fault to the moment the fault memory was full. This information is important because, as explained in the theory, intermittent faults are likely to need more time to produce faulty outputs, while permanent faults will fill the memory faster. However, we cannot establish a random number of patterns, because not every fault is equally detectable, so the number of patterns will be compared to a specific threshold for every pattern. Assuming that every pattern has equal probability of being provided to the circuit, it seems reasonable that the number of patterns to fill the memory can be calculated as:

$$memory\_size = \frac{dp}{tip} tnpp$$

where the variables are:

1. memory_size:
2. dp is the number of patterns that detect the fault
3. tip is the total number of possible input patterns
4. tnpp stands for total number of provided patterns.

So, for example, for a memory size of ten faults, a circuit with 64 possible input patterns and an identified fault detected by 32 patterns, the total number of patterns to provide is calculated:

$$10 = \frac{32}{64}\, tnpp$$

so

$$tnpp = \frac{64}{32}10 = 20$$

which means that, to fill the memory, 20 patterns have to be provided. Because this threshold is too strict, we give some error margin and make it a bit higher. If a mistake was made because of this, it means that an intermittent fault which happens very often is mistaken for a permanent one, and although this is not entirely correct, it is also not a severe mistake as intermittent faults already in that stage tend to turn into permanent faults. So, if a fault detectable by 32 patterns of a total set of 64 needs 24 patterns to fill in a memory of size 10, it is still diagnosed as a permanent fault. Otherwise, it is interpreted as an intermittent fault. This makes sense since the more detectable a fault is, the more probable it is that a provided pattern to the circuit belongs to the set of detecting patterns, and thus the faster the memory will be filled.

This information is still not enough. Because the consequences are very different if a component has a permanent, an intermittent or a transient fault, the distinction between them is very important. If one permanent fault occurs, the probabilities obtained will be very high, and generally the fault is detected as permanent. Let us represent the total space of input patterns by the space S in figure 3.12. The subset P is the subset of patterns that detect fault F1, Q detects a certain fault F2 and T detects another fault, F3. The red crosses represent the faulty outputs registered. Because a big part of the subsets of three different faults are covered, the three are considered possible solutions, obtaining high probabilities in the net.

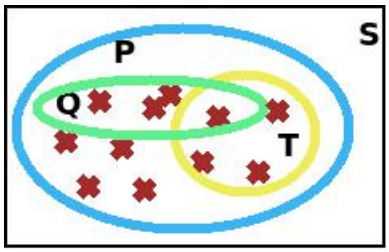

Figure 3.12: Intersection of the identifying pattern sets of different faults.

For intermittent faults, the probabilities are equally high. Since intermittent faults have a cause that does not change with time, the fault location is just the same as with permanent

faults. Only the temporal irregularities make the diagnosis different, but it does not affect the fault location algorithm. To distinguish the fault from a permanent one, thresholds are taken into account, and this is why the fault is detected as intermittent.

However, transients are a special case. They do not have defined characteristics. There is therefore no subset of patterns that detect the transient fault as there is for permanents or intermittents, so the diagnosis has to be a bit different. Transients have to be diagnosed from information about the probabilities of defined faults in the set. When a transient error occurs, the evidence for the defined set of faults is generally low, so the probabilities returned by the algorithm are generally low, since faults located in different areas of the circuit happened, affecting very different sets of input patterns. Although it might not always be the case, the general behavior is that patterns belonging to different sets of "detecting patterns" will give faulty outputs. Thus, faults with very low probabilities will be detected because there is no more evidence to be able to assure that any of the faults is the real cause of the behavior of the circuit. Figure 3.13 represents this case, where the faulty outputs (red crosses) are spread all over the space instead of belonging to one set of detecting patterns. Because the faulty outputs belong to lots of different detecting pattern sets, it is impossible for the system to decide which fault is occurring, because the evidence is scarce for all of them. In case of a large set of faults detected with at least a part of them having low probabilities, we are possibly talking about a transient fault.

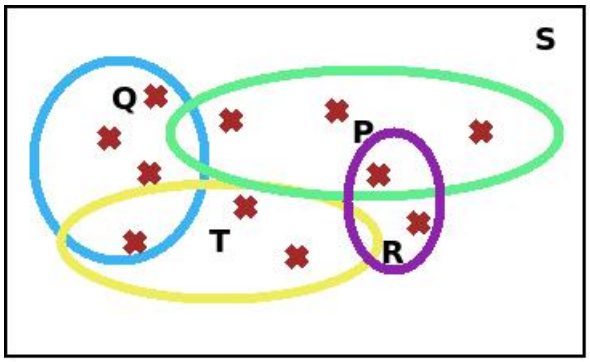

Figure 3.13: Occurring faults distributed over the patterns space.

4 Experiments and Results

For the experiments presented in this section, a simulation framework with fault injection has been used. All the experiments consisted of a simulation of random input patterns in the circuit after injecting faults. The data produced was logged in a fail memory and was later read by the diagnosis tool. The tool has been tested with permanent, intermittent and transient faults. Also, the impact of the fail memory size on the diagnostic accuracy has been studied. Results are presented in the following subsections.

4.1 Output of the program

To fully understand the experiments and results, let us first have a look at the outcome of the diagnosis tool. The following excerpt corresponds to the output of a diagnosis:

```
StuckAt { U1_U14_NOR_2_1/1 0 } 0.9962527664116074 permanent fault
StuckAt { U1_U12_NOR_2_0/2 0 } 0.9961096641087077 permanent fault
```

The result shown means the diagnosis tool gives those two faults as the possible cause of the errors, and shows the probability of each of them occurring. In this case, the fault StuckAt { U1_U12_NOR_2_o/2 0 } was injected as a permanent fault, so the diagnosis is considered correct. In case the fault did not appear, was interpreted as from another type (permanent when it is intermittent and vice-versa) or had a low probability (under 80%), it would be considered a wrong diagnosis, as in the following case:

```
StuckAt { U1_U15/0 0 } 0.9990051871673435 permanent fault
StuckAt { U1_U14_NOR_2_1/1 0 } 0.9950143167762244 permanent fault
StuckAt { U1_U14_OR_2_2/0 1 } 0.9917199287579673 intermittent fault
StuckAt { U1_U12_NOR_2_1/2 0 } 0.9360004222400667 intermittent fault
StuckAt { U1_U10/1 1 } 0.7155165685668402 intermittent fault
StuckAt { U1_U9/2 0 } 0.49755619003374313 intermittent fault
```

Here, StuckAt { U1_U9/20} was injected as an intermittent fault, and while correctly diagnosed and contained in the solution, it is shown with a very low probability. In case of a very large number of faults and a very wide range of probabilities, a transient fault is assumed.

4.2 Experiments

This sections shows the results of experiments of the diagnosis of permanent, intermittent and transient faults. The solution of the diagnosis tool as well as the interpretation are presented here. The circuits used in the experiments to try the diagnosis tool belong to the benchmark set ISCAS'85.

4.2.1 Permanent faults

Experiments for this section have been performed on c17, a very small circuit with a set of 54 possible occurring stuck-at faults. Each of them was injected as permanent, and the fail memory size was set to ten. Then, the circuit's behavior was simulated with random patterns until the memory was full. Because the circuit only has 9 inputs, to calculate the detection probabilities every possible input patterns has been simulated.

Results for these experiments are summarized in table 4.1. In 37 of the 54 experiments, the fault was contained in the solution with a very high probability, and correctly identified as a permanent fault and presented a high probability. Moreover, in 33 of them the injected fault was presented as the only fault contained in the solution.

Permanent faults experiments		
Correctly diagnosed	Unique solution	33
	Multiple solution	4
Incorrectly diagnosed	Incorrectly located	14
	Incorrectly classified	2
	Low probability	2

Table 4.1: Results of diagnosis of permanent faults

From the incorrectly diagnosed ones, in two cases, the fault was correctly located, but incorrectly classified as intermittent, and in other two experiments, the fault is correctly located and classified but presents a slightly lower probability (79.5278% and 76.4255% respectively). In the remaining 14 experiments the fault was not correctly located.

It is not surprising that the faults included in the solutions of the incorrectly located faults and of every fault presenting a multiple solution are always the same. The sets of patterns that detect these faults are large and thus have large intersections with the sets of patterns detecting the injected faults, which is why they repeatedly get high probabilities and are included in the solution. The incorrect classification of some faults as intermittent when they are actually permanent is due to the threshold. The accuracy of the diagnosis tool depends also strongly on these two factors.

Although it may seem irrelevant now, let us also note that no solution presented more than 5 faults. Most of them presented one or two faults included in the solution, very exceptionally including 4 or 5. In no case were there more solutions presented, and the average probability of the solutions was around 0.90. In the scenario at hand we may further note that the correct classification of a fault is more important than its identification. Thus, 33 + 4 + 14 faults were successfully classified.

4.2.2 Intermittent faults

The experiments for this section have been done with the same settings as for the permanent faults. The size of the fail memory is set to 10, and the experiments have been done using c17. As already explained, the low number of inputs of the circuit makes it possible to set the detection probabilities by simulating every possible input pattern. Again, 54 experiments have been done, this time injecting the fault as intermittent. The event rates used, this is, the frequency with which the faults were injected was initially 0.5. The results are shown in table 4.2.

Intermittent faults experiments		
Correctly diagnosed	Unique solution	9
	Multiple solution	27
Incorrectly diagnosed	Incorrectly located	16
	Incorrectly classified	1
	Low probability	1

Table 4.2: Results of diagnosis of intermittent faults

Of all the experiments presented here, 36 were correctly diagnosed. Of the incorrectly diagnosed ones, one presented the right fault and the right type, but had a very low probability, and another one showed the fault as permanent instead of intermittent.

We will now focus on another important characteristic of the solutions presented until now: generally, the solutions include around 2 or 3 faults, going up to 4 or 5 in very rare cases (6 of 54 in this set of experiments). The average probability of the solutions is around 0.85. The importance of this information will be explained in following sections.

As explained before, it is more important in this case to classify the fault correctly than to locate it. Out of the 54 faults, 43 of the solutions only included intermittent faults. Some of the solutions, although they included the correct solution with a high probability, sometimes included permanent faults. The accuracy classifying the type of fault was thus 79.629%.

4.2.3 Transient faults

Because of the nature of transient faults, the circuit has been changed. C17 is very small for the transient experiments, and any fault has an impact on many outputs. Sets of identifying patterns of one fault are often intersected with sets of identifying patterns of other faults, making it very difficult to assess the accuracy of the tool for transient faults on this circuit (figure 4.1). Thus, a bigger circuit has been used. C432 is larger than c17, giving the chance to evaluate transient faults. A larger circuit means also a longer simulation time, reaching in this case unaffordable amounts of time for an exact assessment of detection probabilities. For this reason, in c432 the calculation of the detection probabilities of a fault is done with a large set of random patterns, instead of generating every possible pattern and providing the circuit with it.

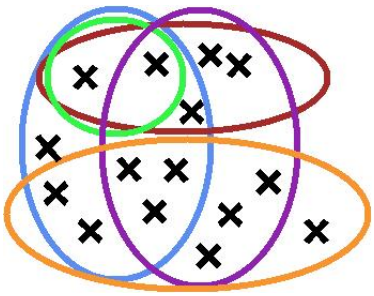

Figure 4.1: Common detecting patterns for many faults in c17.

Results of transient faults differ greatly from those of permanent and intermittent. The tool returns a very large list of faults, with probabilities varying from 1.0 down to 0.20, and sometimes even lower. This is no error: since transients are random, and respond to no predictable behavior, it makes sense that the tool will find it impossible to decide which fault is occurring. This result will be interpreted as a transient, and will cause the program to show all the faults included in the solution:

```
StuckAt { U0_U670/1 1 } 0.9829401572951355 intermittent fault
StuckAt { U0_U1689_AND_2_1/1 1 } 0.9607423545263335 intermittent fault
StuckAt { U0_U1231/1 1 } 0.9218028209712994 intermittent fault
StuckAt { U0_U1689_AND_2_2/1 1 } 0.8175986373760716 intermittent fault
StuckAt { U0_U1685_NOT_1_0/0 0 } 0.785308000441254 intermittent fault
StuckAt { U0_U1001/1 0 } 0.6294408417435818 intermittent fault
StuckAt { U0_U1133/2 1 } 0.23166209148194475 intermittent fault
StuckAt { U0_U1133/1 0 } 0.21166494989026322 intermittent fault
StuckAt { U0_U1679_AND_2_3/1 1 } 0.21147204166785075 intermittent fault
StuckAt { U0_U1182/1 0 } 0.20181989507528228 intermittent fault
StuckAt { U0_U1515/1 1 } 0.17092557781505965 intermittent fault
```

There are a lot of important distinctions between the result of this diagnosis and the previously shown ones. First, the number of faults shown is very large, which did not

happen in the diagnosis of permanents and intermittents in c17, and does not happen in permanents and intermittents in c432 either. Second, the probabilities of the last faults included in the solution are very low. This suggests that there are a lot of isolated errors, each of which needs a fault to be explained. Finally, note that every fault is detected as intermittent. These three facts are used to identify transient faults. Thirty experiments were performed with different event rate values (0.2, 0.5 and 0.7). All the experiments returned solutions like the previously presented.

4.2.4 Impact of faults memory size on diagnosis accuracy

The size of the fail memory is an important data to determine. If it is too small, the diagnosis tool will not have enough information to locate the fault. If it is too big, the memory is wasted. Although the following results are presented in absolute number, it is important to notice that this number cannot be set based on c17, but on the characteristics of the circuit to be diagnosed. C17 has 9 primary inputs and thus 29 possible input patterns. Each fault in c17 is detectable by 11 to 210 of these patterns. This means that there are some faults detected by 11 patterns, some by 210 and some by a number in between. As table 4.4 shows, the depth of the fail memory only starts making sense for this circuit from 10 on. For a fail memory size of 5 the diagnosis is less accurate, and solutions are associated to very low probabilities. The classification is acceptable, since it classifies correctly 49 out of the 54 cases. While the diagnosis accuracy increases until fail memory size 15 and then gets stuck, the classification not only does not improve from 15 on, but it actually decreases slightly.

Memory Size	5	10	15	25
Diagnosed	25	37	43	43
Classified	49	52	54	52

Table 4.3: Diagnosis accuracy of permanent faults depending on fail memory size.

Diagnosis of intermittent faults has a similar evolution, although with slightly lower levels. While fail memory 5 returns very low probabilities and not conclusive solutions, memory size 10 makes it able to correctly classify 44 out of 54 faults. It increases slightly until memory size 25, and then maintains its level at around 48 correctly classified faults.

Memory Size	10	25
Diagnosed	35	37
Classified	44	48

Table 4.4: Diagnosis accuracy of intermittent faults depending on fail memory size.

As for transient faults, the memory size does not have a noticeable impact on the solution, except for size memory 5, because only 5 patterns results in lack of information to diagnose a transient fault.

5 Conclusion and Further Work

The location and classification of faults in a circuit is necessary to decide what to do next with the faulty component. The results of a diagnosis are crucial to decide between repairing or replacing the circuit or ignoring the faults.

In this thesis, a fault analysis method was presented that can support the decision making in case of detected faults in the field. It is designed for analyzing errors detected by self-checking circuits and is accompanied by a simulation framework which helps to assess necessary parameters for the implementation of a self-checking circuit.

The tool is based on Bayesian networks. The use of Bayesian networks for fault location is very effective, achieving high rates of correct diagnosis in the experiments, while the combination of these results with thresholds allows to distinguish among the types of fault. The construction of the network and the setting of the probability tables have been automatized, making it unnecessary to have an expert dedicated to it. Although the accuracy of the tool greatly depends on the characteristics of the circuit, in general the effectiveness has been demonstrated to be high.

In the future, it may be interesting to establish more general heuristics for the fail memory depth, allowing any user to comfortably adjust the fail memory size by applying a given formula. A study about the relationship between the detectability of the possible faults and the fail memory size would help greatly.

Finally, it is not always easy to obtain the input that caused the faulty output. Thus, it is also worth studying codes for the self-checking circuit that would not require the inputs to be stored.

Bibliography

[BA00] M. L. Bushnell, V. D. Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Kluwer Academic Publishers, Boston, 2000. (Cited on page 9)

[BCGG97] A. Bondavalli, S. Chiaradonna, F. D. Gi, F. Gr. Discriminating Fault Rate and Persistency to Improve Fault Treatment. In *in Proc. 27th IEEE FTCS - International Symposium on Fault-Tolerant Computing*, pp. 354–362. IEEE, 1997. (Cited on page 10)

[BGH+02] R. Barco, R. Guerrero, G. Hylander, L. Nielsen, M. Partanen, S. Patel. Automated Troubleshooting of Mobile Networks using Bayesian Networks. In *Proceedings of the IASTED International Conference Communication Systems and Networks*. Malaga, Spain, 2002. (Cited on page 21)

[BGJ+09] B. Bertsche, P. Göhner, U. Jensen, W. Schinköthe, H.-J. Wunderlich. Zuverlässigkeit mechatronischer Systeme, 2009. (Cited on pages 9 and 10)

[Con07] C. Constantinescu. Intermittent Faults in VLSI Circuits. In *IEEE Workshop on Silicon Errors in Logic - System Effects*. 2007. (Cited on page 10)

[Els10] A. M. Elsayad. Predicting the Severity of Breast Masses with Ensemble of Bayesian Classifiers. *Journal of Computer Science*, 6:576–584, 2010. (Cited on page 21)

[FMM01] W. Fenton, T. McGinnity, L. Maguire. Fault diagnosis of electronic systems using intelligent techniques: a review. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 31(3):269 –281, 2001. doi:10.1109/5326.971655. (Cited on page 21)

[FN07] N. E. Fenton, M. Neil. Managing Risk in the Modern World: Bayesian Networks and the Applications. Technical report, London Mathematical Society, Knowledge Transfer Report, 2007. (Cited on page 17)

[GM05] S. van Gosliga, M. Maris. Communication cost in Distributed Bayesian Belief Networks. In *16th Belgium-Dutch Conference on Artificial Intelligence (BNAIC-2005)*. Brussels, Belgium, 2005. (Cited on page 18)

[HMW95] D. Heckerman, A. Mamdani, M. P. Wellman. Real-world applications of Bayesian networks. *Commun. ACM*, 38:24–26, 1995. doi:http://doi.acm.org/10.1145/203330.203334. URL http://doi.acm.org/10.1145/203330.203334.

[KL04] S.-B. Ko, J.-C. Lo. Efficient Realization of Parity Prediction Functions in FPGAs. *J. Electron. Test.*, 20:489–499, 2004. (Cited on page 13)

[KP83] J. H. Kim, J. Pearl. A computational model for causal and diagnostic reasoning in inference systems. In *Proceedings of the Eighth international joint conference on Artificial intelligence - Volume 1*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1983. (Cited on page 18)

[Lal01] P. K. Lala, editor. *Self-checking and fault-tolerant digital design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001. (Cited on page 13)

[Lug97] G. F. Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1997. (Cited on page 15)

[Pea09] J. Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, New York, NY, USA, 2009. (Cited on page 14)

[PSBDG98] M. Pizza, L. Strigini, A. Bondavalli, F. Di Giandomenico. Optimal Discrimination between Transient and Permanent Faults. In *3rd IEEE High Assurance System Engineering Symposium*, pp. 214–223. Bethesda, MD, USA, 1998. (Cited on pages 11 and 21)

[SGM+08] A. C. Salvador, E. García, F. Morant, R. B. Giménez, E. Q. Cucarella. Diagnosis of intermittent fault dynamics. In *ETFA*, pp. 559–566. 2008. (Cited on page 9)

[SMM02] L. Schiano, C. Metra, D. Marino. Design and Implementation of a Self-Checking Scheme for Railway Trackside Systems. *Memory Technology, Design and Testin, IEEE International Workshop on*, 0:49, 2002. doi:http://doi.ieeecomputersociety.org/10.1109/MTDT.2002.1029763. (Cited on page 13)

[WWW06] L.-T. Wang, C.-W. Wu, X. Wen. *VLSI Test Principles and Architectures: Design for Testability (Systems on Silicon)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006. (Cited on pages 10 and 11)

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Laura Rodríguez Gómez)