Exploitation of Built in test for diagnosis by using Dynamic Fault Trees: Implementation in Matlab Simulink

E. Gascard, Zineb Simeu-Abazi

Exploitation of Built in test for diagnosis by using Dynamic Fault Trees: Implementation in Matlab Simulink

Eric Gascard
TIMA laboratory (CNRS - Grenoble INP - UJF), Grenoble, FRANCE

Zineb Simeu-Abazi
G-SCOP laboratory (CNRS - Grenoble INP - UJF), Grenoble, FRANCE

Joseph Younes
Polytech'Grenoble, Université Joseph Fourier, Grenoble, FRANCE

ABSTRACT: This paper presents the purpose of Dynamic Fault Tree (DFT) in Matlab Simulink in order to diagnose discrete event systems. The aim is to filter false alarms in automated systems that feature dependencies. Traditional Fault Tree is a tool used for system diagnostics, but it is limited because it uses traditional logic gates (OR, AND) which do not take into consideration the time and the dependencies of automated systems. As a result, new gates have been created to make it possible to consider dynamic aspects of automata. They are called temporal and dynamic logic gates. Consequently a research work has been done upstream to describe all the functioning cases of each gate and to represent them into digital timing diagram. Then, in order to provide an easy way to built DFT and analyze automated system faults, traditional, temporal and dynamic gates have been programmed using the StateFlow library of Matlab Simulink and added to a new toolbox created especially in Simulink for the construction and the simulation of DFT.

1 INTRODUCTION

Moderns systems or equipments are composed of hundreds of circuit boards, most of them integrate Build-In Test (BIT) facilities (Gao & Suryavanshi 2002). BIT technology allows the system functionality to be verified automatically and simplify the test equipment, improve efficiency, reduce costs of preventative maintenance. It is used widely in areas like aerospace for maintenance and flight operations. BIT indications, named alarms, are recorded during the mission and presented to maintenance personnel when the aircraft is on the ground for detection and isolation of failures. The diagnosis of these alarms is costly: requiring replacement of the failed components, re-qualification, maintenance man-hours, and aircraft downtime. Moreover, some of these alarms can be false alarms (Rosenthal & Wadell 1990, Westervelt 2004) and the localization of the system failures can be ambiguous which gives an additional delay of the maintenance actions and leads to useless maintenance actions such as false disassembling, or needs a long procedure of test to isolate the failure.

To establish a diagnostic, the maintenance teams exploit some correlations between different alarms expressed as specific rules. These rules aim at solving the problems above: the filtering of false alarms and the reduction in the size of the failure ambiguity group to more precisely locate the faulty equipment. However, these rules are static and do not integrate the dynamics of the failure and its implications on the time relationship between recorded alarms. So, there is a need for a modelization of dynamic rules and their simulation. Simeu-Abazi et al. tackle this problem in Simeu-Abazi et al. (2011) and Lefebvre (2009), they propose to model dynamic rules as temporal and dynamic fault trees.

The fault tree (FT) method is a technique used to analyze system failures at the level of design. The method consists of a top-down analysis which starts from prospected failures (top events) and looks for its possible causes (basic events). From the constructed fault tree, both quantitative and qualitative safety and reliability analysis can be performed. One of the main assumption in fault tree analysis is that basic events must be assumed to be statistically independent and are considered as non-repairable. FT describes the interaction of basic events by means of Boolean gates, so that the combination of events is relevant, not their sequence. To take into account

sequential relationships among events and statistical dependencies, Dugan et al. propose in (Dugan et al. 1990, Dugan et al. 1992) the dynamic fault tree model based on the traditional static fault tree modeling framework (Vesely, Goldberg, Roberts, & Haasl 1981). The DFT methodology is presented in (Vesely, Stamatelatos, Dugan, Fragola, Minarick, & Railsback 2002). Dynamic fault trees use four additional kinds of gates as modeling elements: Sequence Enforcing (SEQ), Functional Dependency (FDEP), Priority-AND (PAND) and Cold, Warm and Hot Spare (CSP,WSP and HSP).

There exists some extension of the DFT by adding simple temporal gates to capture temporal dependence between events and faults, this extension is called Temporal Fault Tree (TFT) (Palshikar 2002).

The work of Simeu-Abazi et al. consists to apply the formalism of dynamic fault tree for filtering the faults, and do not deal with the qualitative and quantitative analysis of DFT and TFT. Indeed, different diagnosis rules are defined for each functioning mode and these rules are translated using DFT for the alarm filtering. Their aim is to propose a new simulation scheme of DFT and TFT by giving a High Level Petri Net representation of existing dynamic gates. The authors propose a new class of temporal gates called PANDW, DUR and COUNT. These new gates are as well modeled using High Level Petri Nets. To take into account false alarms, the assumption of non-repairable events is giving up.

Several authors propose different tools to model behavior of dynamic gates, for example the specification language Z (Coppit, Sullivan, & Dugan 2000), timed automata (Santiago Barragan, Roth, & Faure 2006), Stochastic Petri nets (Bobbio & Raiteri 2004) or Bayesian networks (Boudali & Dugan 2005) with the objective of quantitative and qualitative analysis. Our objective is not to analyze the performance model, but to implement diagnosis rules by using dynamic gates. This paper is a continuation of Simeu-Abazi et al. work by proposing a toolbox in Matlab/Simulink/Stateflow which allow to build and simulate temporal and dynamic fault trees in Matlab Simulink. The paper is organized into three main parts. The first one presents our formalization of the temporal and dynamic fault tree gates, then the second one explain the principle used to program theses gates and their inclusion into a new toolbox, and the last one shows an example of application on alarm filtering in Matlab Simulink.

2 FORMALIZATION OF TEMPORAL AND DYNAMIC FAULT TREE GATES WITH REPAIRABLE EVENTS

An important problem in understanding temporal and dynamic fault tree is the informal nature of their notation. There is a need for formalization of the semantics of the DFT and TFT notation. In this section, we first define their semantics in term of traces that satisfy their behavior. Secondly, we build for each considered gate a Moore automaton that fulfills its semantics. These Moore automata provide an easy way to implement these gates by using the Stateflow library of Matlab Simulink.

Consider a finite set of failure events $\mathcal{F} = \{x_1, \ldots, x_n\}$ having values in the Boolean domain $\mathbb{B}$. The occurrence of failure is represented as 1 and its nonoccurence as 0. A trace ω over $\mathcal{F}$ is defined as a sequence of valuation of all the failure events of $\mathcal{F}$. We assume that failure events are repairable (take into account of false alarms) and time is discrete. Time consists of a sequence of time tick numbered starting from 0. A consequence of this interpretation is that, a trace over n failure events is a function from $\mathbb{N}$ to $\mathbb{B}^n$. The occurrence of a failure event x_i in a trace ω at time t is denoted by $\omega^t \vdash x_i$.

The rest of this section is organized as follows: firstly we define the semantics of the classical DFT gates: PAND, SEQ and FDEP. We do not take into account the Spares gates: we are interested in alarm filtering, this task is made off-line, so the SPARE gates do not have utilities. Secondly, we define the semantics of temporal gates which integrate time relationship between events.

2.1 *Dynamic gates*

2.1.1 *Definition and representation of PAND gate*

The priority-AND gate was introduced by Fussell et al. (1976). From (Dugan, Bavuso, & Boyd 1990), the PAND gate is equivalent to an AND gate, with the added condition that the failure events must occur in a specific order. The output is true if both input events occur and the left input occurs before the right input. In other words, if any failure events has not occurred or if the right input occurs before the left one, the output does not occur.

Let a and b denote the left and right input events of the PAND gate. The output of the PAND gate for a trace ω involving a and b at time t is defined as follows:

$$\omega^t \vdash PAND(a,b) \iff \exists k_1 < k_2 < t \cdot \left(\begin{array}{ll} & \omega^{k_1} \nvdash a \wedge \omega^{k_1} \nvdash b \\ \wedge & \forall j \in]k_1 \ldots k_2] \cdot (\omega^j \vdash a \wedge \omega^j \nvdash b) \\ \wedge & \forall j \in]k_2 \ldots t] \cdot (\omega^j \vdash a \wedge \omega^j \vdash b) \end{array} \right)$$

Figure 1 below shows the behavior of the PAND gate.

A Moore automaton can represent this behavior of PAND gate as shown in Figure 2. The output of the gate is associated with the states of the automaton. This output is "emitted" by the automaton when it is in the state. Transitions are labeled with the value of the inputs (concatenation of inputs a and b). If the

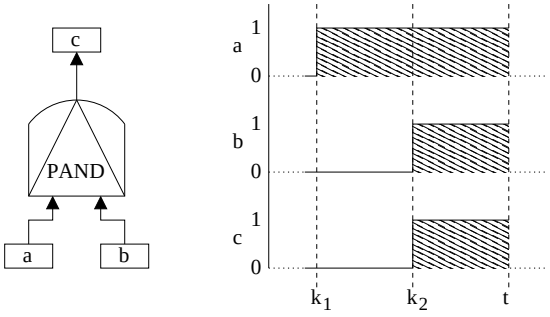


Figure 1: Chronogram of the PAND gate

event b appears before or in the same time than the event a, state s_3 becomes active and the output c of the gate is 0. If the event a appears before the event b, state s_1 becomes active, then if the event b appears and the event a is still present, state s_2 becomes active and the output c of the gate is 1. When the input events a and b are not present in the same time, the automaton is "reset" in order to detect the new a and b events and so on until the end of the simulation. Reset appears in the presence of false alarms.

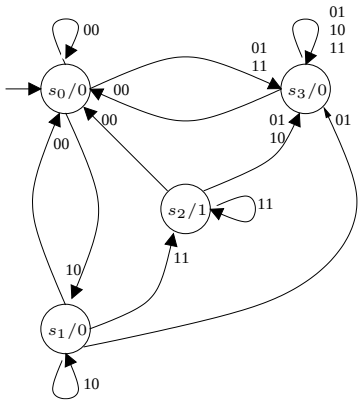


Figure 2: Moore automaton of the PAND gate

The PAND gate can be used in a Dynamic Fault Tree that implements a rule for filtering false alarms or a rule for improving the diagnostic of a system.

2.1.2 *Definition and representation of SEQ gate*

As defined in (Dugan, Bavuso, & Boyd 1992, Coppit, Sullivan, & Dugan 2000), the sequence enforcing gate asserts that inputs events can occur only in a given order. The input events are constrained to occur in the left-to-right order in which they appear under the gate. This means that the left-most event must occur before the event on its immediate right, which must occur before the event on its immediate right and so forth. The output of the gate is true as long as the sequence of input events is respected. We consider only SEQ gate with two inputs, SEQ gate with more inputs can be defined as a composition of SEQ gates with two inputs: $SEQ(x_1, x_2, x_3) \equiv SEQ(SEQ(x_1, x_2), SEQ(x_2, x_3))$. The SEQ gate will be used only in a Dynamic Fault Tree that implements a rule for filtering false alarms.

The output of the SEQ gate for a trace ω involving a and b at time t could be defined by six different scenarios as follows:

$$\omega^t \vdash SEQ(a,b) \overset{①}{\Longleftrightarrow}$$
$$\exists k_1 < t \cdot \left(\begin{array}{ll} & \omega^{k_1} \nvdash a \wedge \omega^{k_1} \nvdash b \\ \wedge & \forall j \in]k_1 \dots t] \cdot (\omega^j \vdash a \wedge \omega^j \nvdash b) \end{array} \right)$$

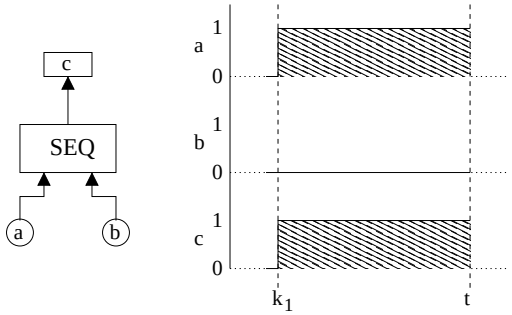


Figure 3: Chronogram of the SEQ gate - case ①

$$\omega^t \vdash SEQ(a,b) \overset{②}{\Longleftrightarrow} \exists k_1 < k_2 < t \cdot$$
$$\left(\begin{array}{ll} & \omega^{k_1} \nvdash a \wedge \omega^{k_1} \nvdash b \\ \wedge & \forall j \in]k_1 \dots k_2[\cdot (\omega^j \vdash a \wedge \omega^j \nvdash b) \\ \wedge & \forall j \in]k_2 \dots t] \cdot (\omega^j \vdash a \wedge \omega^j \vdash b) \end{array} \right)$$

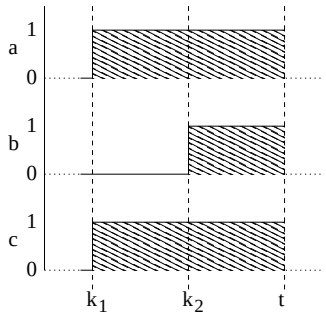


Figure 4: Chronogram of the SEQ gate - case ②

$$\omega^t \vdash SEQ(a,b) \overset{③}{\Longleftrightarrow} \exists k_1 < k_2 < k_3 < t \cdot$$
$$\left(\begin{array}{ll} & \omega^{k_1} \nvdash a \wedge \omega^{k_1} \nvdash b \\ \wedge & \forall j \in]k_1 \dots k_2[\cdot (\omega^j \vdash a \wedge \omega^j \nvdash b) \\ \wedge & \forall j \in]k_2 \dots k_3[\cdot (\omega^j \vdash a \wedge \omega^j \vdash b) \\ \wedge & \forall j \in]k_3 \dots t] \cdot (\omega^j \vdash a) \\ \wedge & \omega^{k_3} \nvdash b \end{array} \right)$$

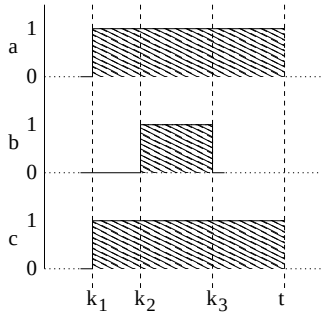


Figure 5: Chronogram of the SEQ gate - case ③

$$\omega^t \vdash SEQ(a,b) \overset{④}{\Longleftrightarrow} \exists k_1 < k_2 < k_3 < t \cdot \left(\begin{array}{ll} & \omega^{k_1} \nvdash a \wedge \omega^{k_1} \nvdash b \\ \wedge & \forall j \in]k_1 \ldots k_2[\cdot (\omega^j \vdash a \wedge \omega^j \nvdash b) \\ \wedge & \forall j \in]k_2 \ldots k_3[\cdot (\omega^j \vdash a \wedge \omega^j \vdash b) \\ \wedge & \forall j \in]k_3 \ldots t] \cdot (\omega^j \vdash b) \\ \wedge & \omega^{k_3} \nvdash a \end{array} \right)$$

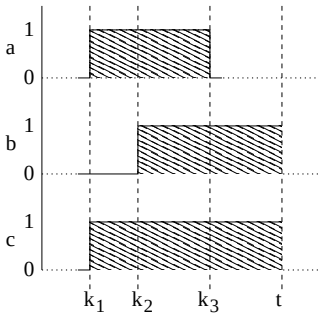


Figure 6: Chronogram of the SEQ gate - case ④

$$\omega^t \vdash SEQ(a,b) \overset{⑤}{\Longleftrightarrow} \exists k_1 < k_2 < t \cdot \left(\begin{array}{ll} & \omega^{k_1} \nvdash a \wedge \omega^{k_1} \nvdash b \\ \wedge & \forall j \in]k_1 \ldots k_2[\cdot (\omega^j \vdash a \wedge \omega^j \nvdash b) \\ \wedge & \forall j \in]k_2 \ldots t] \cdot (\omega^j \vdash b) \\ \wedge & \omega^{k_2} \nvdash a \end{array} \right)$$

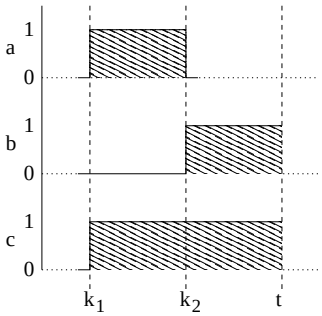


Figure 7: Chronogram of the SEQ gate - case ⑤

$$\omega^t \vdash SEQ(a,b) \overset{⑥}{\Longleftrightarrow} \exists k_1 < k_2 < k_3 < t \cdot \left(\begin{array}{ll} & \omega^{k_1} \nvdash a \wedge \omega^{k_1} \nvdash b \\ \wedge & \forall j \in]k_1 \ldots k_2[\cdot (\omega^j \vdash a \wedge \omega^j \nvdash b) \\ \wedge & \forall j \in]k_2 \ldots k_3[\cdot (\omega^j \nvdash a \wedge \omega^j \nvdash b) \\ \wedge & \forall j \in]k_3 \ldots t] \cdot (\omega^j \vdash b) \end{array} \right)$$

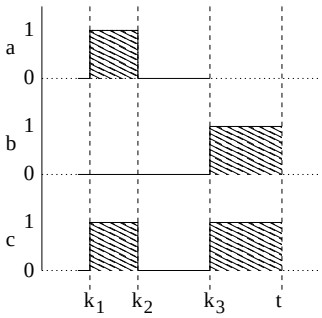


Figure 8: Chronogram of the SEQ gate - case ⑥

A Moore automaton can represent the behavior of all these cases as shown in Figure 9.

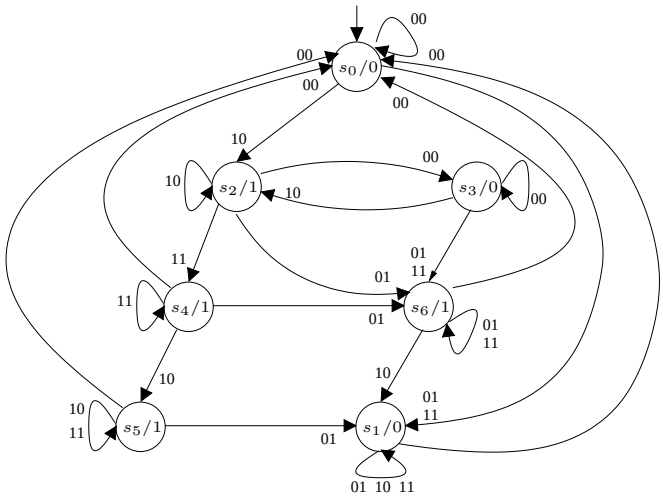


Figure 9: Moore automaton of the SEQ gate

2.1.3 *Definition and representation of FDEP gate*

Vesely et al. (2002) defines the functional dependency gate as follows: Suppose that a system is configured such that the occurrence of some event (trigger event) causes other dependent components to become inaccessible or unusable. A FDEP gate can be used to explicitly reflect a such dependence. An FDEP gate has a single trigger input and one or more dependent basic events. The dependent basic events are functionally dependent on the trigger event. When the trigger event occurs, the dependent basic events are forced to occur (fail). We consider only FDEP gate with one dependent basic event (see Figure 10), FDEP gate with more dependent basic events can be define as several FDEP gates: all these gates will share the same trigger event, and each gate will take as basic event one of the dependent basic events. The output of the FDEP gate reflects the status of the trigger event.

The output of the FDEP gate for a trace ω involving the trigger event e and the dependent event a at time t is defined as follows: $\omega^t \vdash FDEP(e,a) \Longleftrightarrow \omega^t \vdash e$

This gate will be used only in a Dynamic Fault Tree that implements a rule for filtering false alarms. Suppose we want to model a filtering rule involving some basic event a and a is an operand of an FDEP gate with a trigger event e. The expression of the filtering rule will not involve a as operand but the Boolean expression $OR(FDEP(e,a),a)$ which expresses the real status of a due to its functional dependency. A Moore automaton can represent this behavior as shown in Figure 10.

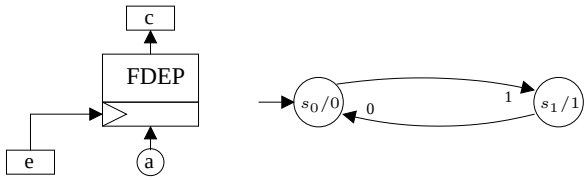


Figure 10: Moore automaton of the FDEP gate. Transitions are labeled with trigger event.

2.2 *Temporal gates*

In our application the event corresponds to the occurrence of a failure message. These failure messages may be steady or intermittent. It is then necessary to define a temporal window to take into account the time of occurrence of each event, the delay between the occurrence of each event, the event duration or the number of times that the same event is detected. Thus, new gates for temporal fault tree called "PANDW", "DUR" and "COUNT" are proposed in (Lefebvre, Simeu-Abazi, Derain, & Glade 2007, Lefebvre 2009). In this section, we do not treat the COUNT gate due to a lack of place.

2.2.1 *Definition and representation of PANDW gate*

This gate has the same functioning principle than a normal PAND gate but introduce a time dependency between failures in order to do the diagnostic. Two cases can be represented:

- The $PANDW_{>}$ gate is specified to "fail" if its inputs fail in left to right order and if the time between the failure of a and the failure of b is greater than T units of time.

- The $PANDW_{<}$ gate is specified to "fail" if its inputs fail in left to right order and if the time between the failure of a and the failure of b is less than T units of time.

The output of the $PANDW_{>}$ gate with time dependency T for a trace ω involving a and b at time t is defined as follows:

$$\omega^t \vdash PANDW_{>}(T,a,b) \Longleftrightarrow \exists k_1 < k_2 < t \cdot \left(\begin{array}{ll} & \omega^{k_1} \nvdash a \wedge \omega^{k_1} \nvdash b \\ \wedge & \forall j \in]k_1 \ldots k_2] \cdot (\omega^j \vdash a \wedge \omega^j \nvdash b) \\ \wedge & k_2 - k_1 \geq T \\ \wedge & \forall j \in]k_2 \ldots t] \cdot (\omega^j \vdash a \wedge \omega^j \vdash b) \end{array} \right)$$

A Moore automaton can represent this behavior as shown in Figure 11. Transitions are labeled with the value of the inputs. Some transitions use temporal conditions ("after" and "before") to determine the activation of transitions and duration of state activation.

The output of the $PANDW_{<}$ gate with time dependency T for a trace ω involving a and b at time t is defined as follows:

$$\omega^t \vdash PANDW_{<}(T,a,b) \Longleftrightarrow \exists k_1 < k_2 < t \cdot \left(\begin{array}{ll} & \omega^{k_1} \nvdash a \wedge \omega^{k_1} \nvdash b \\ \wedge & \forall j \in]k_1 \ldots k_2] \cdot (\omega^j \vdash a \wedge \omega^j \nvdash b) \\ \wedge & k_2 - k_1 \leq T \\ \wedge & \forall j \in]k_2 \ldots t] \cdot (\omega^j \vdash a \wedge \omega^j \vdash b) \end{array} \right)$$

A Moore automaton can represent this behavior as shown in Figure 12.

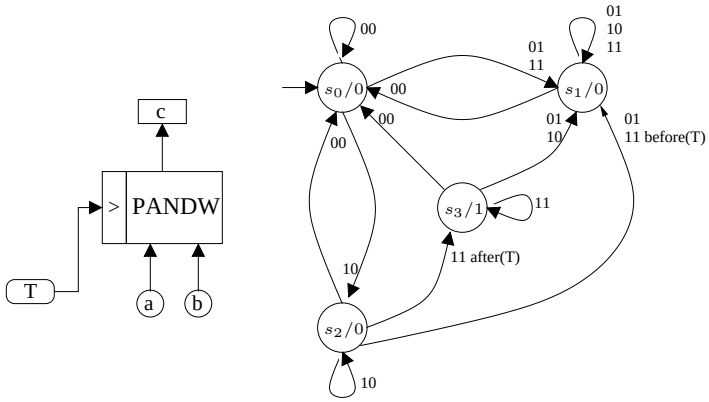


Figure 11: Moore automaton of the $PANDW_{>}$ gate

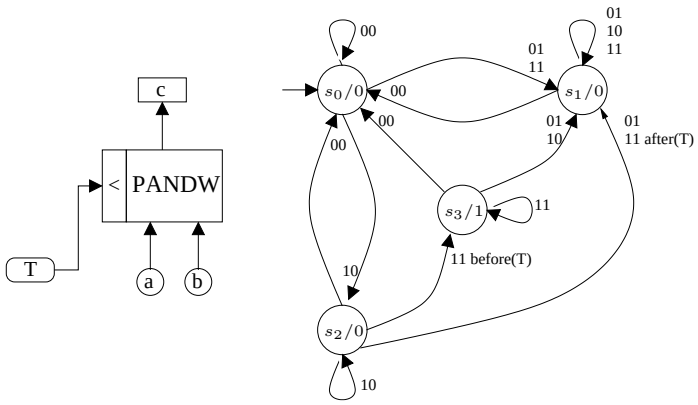


Figure 12: Moore automaton of the $PANDW_{<}$ gate

2.2.2 *Definition and representation of DUR gate*

The events are not necessarily persistent ones; it was thus necessary to define additional gates in order to take into account the transient or intermittent failure events. The duration gate compares the duration time of the failure to a threshold T, if it is greater or lower than the specified time T, then the transition is valid. Two cases can be represented:

- The $DUR_{>}$ gate is specified to "fail" if its input is considered as failed during a time greater than T units of time.

- The $DUR_{<}$ gate is specified to "fail" if its input is considered as failed during a time less than T units of time.

The output of the $DUR_{>}$ gate with time dependency T for a trace ω involving a at time t could be defined by two different scenarios as follows:

$$\omega^t \vdash DUR_{>}(T,a) \overset{①}{\Longleftrightarrow}$$

$$\exists k_1 < k_2 < t \cdot \left(\begin{array}{ll} & \omega^{k_1} \nvdash a \\ \wedge & \forall j \in]k_1 \ldots k_2] \cdot (\omega^j \vdash a) \\ \wedge & k_2 - k_1 = T \\ \wedge & \forall j \in]k_2 \ldots t] \cdot (\omega^j \vdash a) \end{array} \right)$$

$$\omega^t \vdash DUR_{>}(T,a) \overset{②}{\Longleftrightarrow}$$

$$\exists k_1 < k_2 < t \cdot \left(\begin{array}{ll} & \omega^{k_1} \nvdash a \\ \wedge & \forall j \in]k_1 \ldots k_2] \cdot (\omega^j \vdash a) \\ \wedge & k_2 - k_1 \geq T \\ \wedge & \forall j \in]k_2 \ldots t] \cdot (\omega^j \nvdash a) \end{array} \right)$$

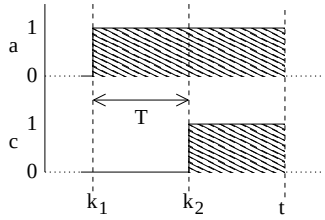


Figure 13: Chronogram of the $DUR_{>}$ gate - case ①

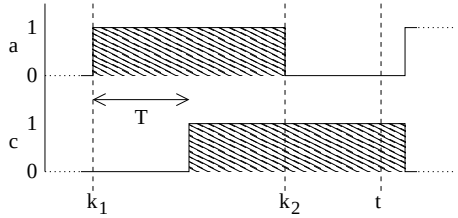


Figure 14: Chronogram of the $DUR_{>}$ gate - case ②

A Moore automaton can represent this behavior as shown in Figure 15.

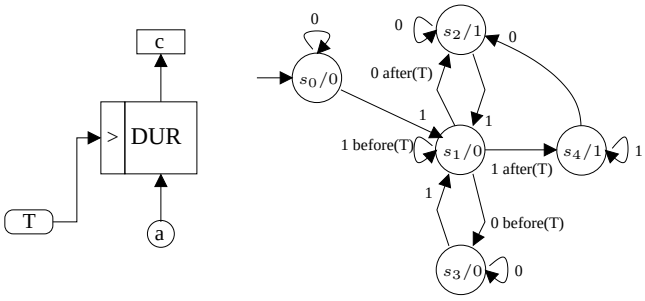


Figure 15: Moore automaton of the $DUR_{>}$ gate

The output of the $DUR_{<}$ gate with time dependency T for a trace ω involving a at time t could be defined as follows:

$$\omega^t \vdash DUR_{<}(T,a) \iff$$

$$\exists k_1 < k_2 < t \cdot \left(\begin{array}{ll} & \omega^{k_1} \nvdash a \\ \wedge & \forall j \in]k_1 \ldots k_2] \cdot (\omega^j \vdash a) \\ \wedge & k_2 - k_1 \leq T \\ \wedge & \forall j \in]k_2 \ldots t] \cdot (\omega^j \nvdash a) \end{array} \right)$$

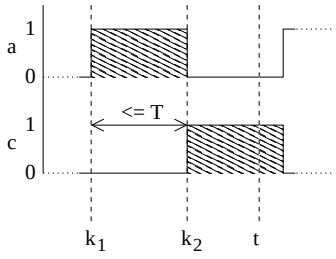


Figure 16: Chronogram of the $DUR_{<}$ gate

A Moore automaton can represent this behavior as shown in Figure 17.

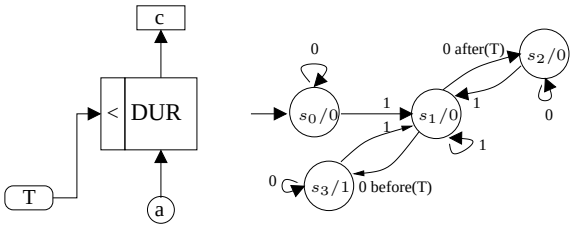


Figure 17: Moore automaton of the $DUR_{<}$ gate

3 CREATION OF A DFT AND TFT LIBRARY IN THE SIMULINK BROWSER

Each logic gate correspond to a Subsystem block which contains some blocks in order to treat the inputs signals that must be square waves in order to represent as real as possible the functioning of a discrete event system, some blocks to make it possible to define the parameters of the gate and the Chart block of the gate automaton. These Subsystem blocks had been masked with different icons in order to differentiate the logic gates and to assign to each one of them its standard symbol. Furthermore, specific parameters had been configured for some of the gates such as the DURATION gate, the PANDW gate and the COUNTER gate. The Figure 18 below shows the masked subsystem of the DURATION gate.

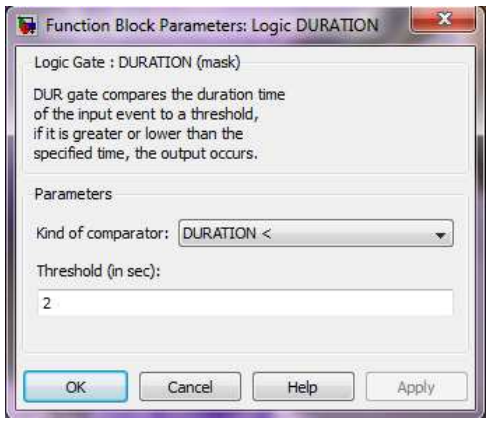


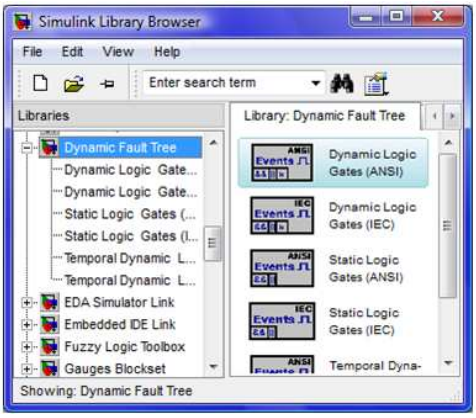
Figure 18: Masked subsystem of the DURATION gate

Then, in order to make it possible to use these gates anytime in Matlab, they have been added in a new Simulink library especially created for the construction of DFT. The Figure 19 shows what the DFT library looks like in Simulink.

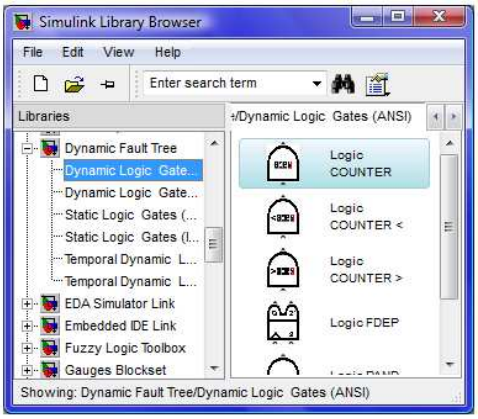
4 APPLICATION OF THE DFT AND TFT TOOLBOX ON ALARM FILTERING

To illustrate the use of the new Matlab Simulink DFT and TFT toolbox, the application example proposed by Lefebvre (2009) to diagnose a system by filtering alarms is taken as model. It is composed of inputs events A, B, C, D, E, F and will be faulty if one of the three following cases occurs:

- The events B and C occur, but the event C has to occur at least 5 seconds after the event B, and



(a) The DFT parent library



(b) The ANSI DFT sub library

Figure 19: DFT library and sub libraries in the Simulink browser

the event A does not have to appear before the occurrence of the events B and C.

- The event D occurs during at least 4 seconds.
- The events E and F are simultaneously present but the event E occurs at least 2 times.

The top event of the dynamic and temporal fault tree represents the output of an OR logic gate, then, the inputs of this OR gate correspond to the three failure cases possible:

- The first failure case can be translated by a $PANDW_{>}$ gate that connects the events B and C and by a SEQ gate that connects the output of the $PANDW_{>}$ gate and the event A.
- The second failure case can be translated by a $DUR_{>}$ gate for the event D.
- The third failure case can be translated by a $COUNTER_{>}$ gate for the event E and an AND gate that connects the output of the $COUNTER_{>}$ gate and the event F.

The dynamic and temporal fault tree that makes it possible to filter the alarm corresponding to the events A, B, C, D, E and F is visible in the Figure 21.

In order to evaluate the good functioning of the system, a test event vector is taken as example, see Figure 20. All the alarm signals are generated in a Signal Builder block connected to the inputs of the DFT and TFT gates.

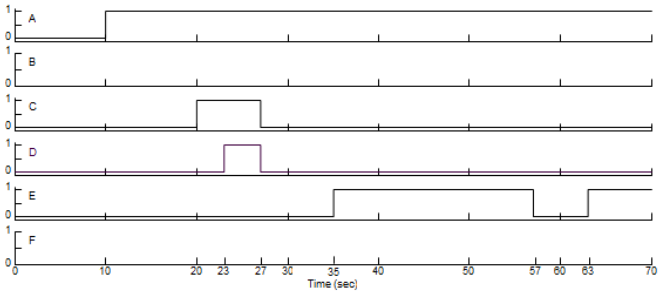


Figure 20: Digital Timing Diagram of the events

After the simulation of the Simulink model, the Scope block shows that:

- The output of the $PANDW_{>}$ gate which is the first input of the SEQ gate is not generated. As a result, the output of the SEQ gate (Failure Case 1 signal) is never generated which means that all the alarms of the first failure case have been filtered.
- The output of the $DUR_{>}$ gate (Failure Case 2 signal) is generated at T = 27 seconds because the event D occurred during 4 seconds. The alarm of the second failure case is not filtered.
- The output of the AND gate (Failure Case 3 signal) is not generated because the event F is never present. Regarding the COUNTER $>$ gate, its output is generated because the event E occurred exactly two times. All the alarms of the third failure case have been filtered.
- As a result the output of the tree (Alarm Filtering signal) is generated at the moment of the second failure case, consequently, the system is faulty.

5 CONCLUSION

Programming the logic gates that take into consideration the dynamic aspects of automata has been very interesting because it makes it possible to build and simulate temporal and dynamic fault trees in Matlab Simulink in order to improve the diagnostics of discrete event systems. Moreover, the results obtained are very encouraging because they confirm the fact that DFT and TFT makes it possible to improve a diagnostic by filtering false alarms which cannot always be filtered by static fault trees.

Consequently, this method using DFT and TFT in

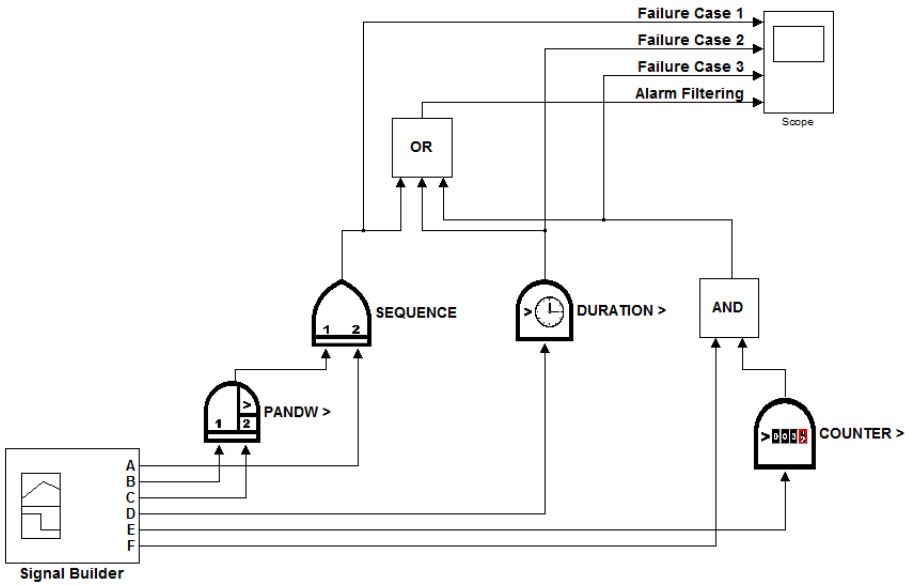


Figure 21: Dynamic and Temporal Fault Tree of the system

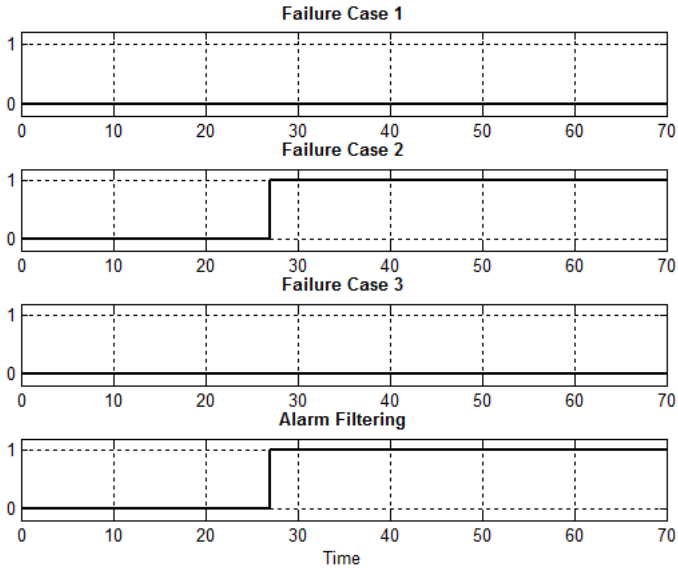


Figure 22: Results of the DFT simulation

Malab Simulink can be used for example in the aeronautic field to filter false alarms recorded by an aircraft during a flight. The first application of this approach lead to reduce at least 50 percent the number of false and then the cost of maintenance. Furthermore, some improvements to make the user task easier are possible. The first one is the conception of a program that will automatically format alarm data in the excel data file Data.xls to make it possible to easily generate the logic gate inputs. Another one would be to create a program that will automatically generate a Matlab Simulink DFT from a text file according to certain syntax rules.

REFERENCES

Bobbio, A. & D. Raiteri (2004). Parametric fault trees with dynamic gates and repair boxes. In *Proceedings of the Annual Reliability and Maintainability Symposium, RAMS'04*.

Boudali, H. & J. Dugan (2005). A discrete-time bayesian network reliability modeling and analysis framework. *Reliability Engineering & System Safety 87*(3), 337–49.

Coppit, D., K. Sullivan, & J. Dugan (2000). Formal semantics of models for computational engineering: a case study on dynamic fault trees. In *Proceedings of the 11th International Symposium on Software Reliability Engineering, ISSRE '00*.

Dugan, J., S. Bavuso, & M. Boyd (1990). Fault trees and sequence dependencies. In *Proceedings of the Annual Reliability and Maintainability Symposium, RAMS'90*.

Dugan, J. B., S. J. Bavuso, & M. A. Boyd (1992). Dynamic fault tree models for fault tolerant computer systems. *IEEE Trans. Reliability 41*(3), 363–377.

Fussell, J., E. Aber, & R. Rahl (1976). On the quantitative analysis of priority-and failure logic. *IEEE Transactions on Reliability 25*(5), 324–326.

Gao, R. & A. Suryavanshi (2002). Diagnosis from within the system [built-in test]. *Instrumentation Measurement Magazine, IEEE 5*(3), 43–47.

Lefebvre, A. (2009). *Contribution à l'amélioration de la testabilité et du diagnostic de systèmes complexes : Application aux systèmes avioniques*. Ph. D. thesis, University Joseph Fourier - Grenoble (FRANCE).

Lefebvre, A., Z. Simeu-Abazi, J.-P. Derain, & M. Glade (2007). Diagnostic of the avionic equipment based on dynamic fault tree. In *International Conference on Cost Effective Automation in Networked Product Development and Manufacturing*.

Palshikar, G. K. (2002). Temporal fault trees. *Information & Software Technology 44*(3), 137–150.

Rosenthal, D. & B. Wadell (1990). Predicting and eliminating built-in test false alarms. *Reliability, IEEE Transactions on 39*(4), 500–505.

Santiago Barragan, I., M. Roth, & J.-M. Faure (2006). Obtaining temporal and timed properties of logic controllers from fault tree analysis. In *Proceedings of the 12th IFAC Symposium on Information Control Problems in Manufacturing, INCOM 2006*.

Simeu-Abazi, Z., A. Lefebvre, & J.-P. Derain (2011). A methodology of alarm filtering using dynamic fault tree. *Reliability Engineering and System Safety 96*(2), 257–266.

Vesely, W., F. Goldberg, N. Roberts, & D. Haasl (1981). *Fault Tree Handbook*. U.S. Nuclear Regulatory Commission.

Vesely, W. E., M. Stamatelatos, J. Dugan, J. Fragola, J. Minarick, & J. Railsback (2002). *Fault Tree Handbook with Aerospace Applications*. NASA Office of Safety and Mission Assurance.

Westervelt, K. (2004). Root cause analysis of bit false alarms. In *IEEE Aerospace Conference*, Volume 6, pp. 3782–3790.