Méthode quantitative d'évaluation de la fiabilité des systèmes critiques programmés

Hicham Belhadaoui, Grégory Buchheit, Olaf Malassé, Jean-François Aubry, Hicham Medromi

METHODE QUANTITATIVE D'EVALUATION DE LA FIABILITE DES SYSTEMES CRITIQUES PROGRAMMES

QUANTITATIVE METHOD FOR ASSESSING THE PROGRAMMABLE CRITICAL SYSTEMS RELIABILITY

H. BELHADAOUI[1, 2,3], G. BUCHHEIT[2], O. MALASSE[2], J.F. AUBRY[1], H. MEDROMI[3]

[1]CRAN CNRS UMR 7039
Nancy – Université, INPL – ENSEM
2, avenue de la forêt de Haye
54516 Vandœuvre-lès-Nancy, France
Tel : 33(0)383595578
Hicham.BELHADAOUI-4@etudiants.ensam.fr
gregory.buchheit@metz.ensam.fr

[2]A3SI, Arts et Métiers ParisTech,
4 rue Augustin Fresnel, 57078 Metz France
Tel : 33(0)387375468
[3]EAS, ENSEM Casablanca, Oasis BP. 8118
route El Jadida, Casablanca MAROC
Tel : 2125(0) 61616467
jean-francois.aubry@isi.u-nancy.fr
Olaf.Malasse@metz.ensam.fr

Résumé

Cet article propose une méthode d'évaluation de la fiabilité et de la disponibilité des systèmes critiques programmés, dès leur phase de conception. L'objectif principal est d'évaluer la probabilité qu'une architecture matérielle d'un processeur exécute correctement les instructions et les routines d'une application logicielle. Cette probabilité est calculée à partir des taux de défaillance d'exécution des instructions logicielles sur une architecture matérielle d'un processeur à pile, ce qui permet de traiter la problématique d'interaction matériel/logiciel dans les systèmes critiques programmés. Après définition des modes de dysfonctionnement on décrie une méthode d'évaluation de leur sureté de fonctionnement basée sur une approche originale utilisant comme support le flux informationnel.

La fiabilité du logiciel est définit selon la norme ISO/IEC 9126 (2001) [1] comme étant «*l'aptitude du logiciel à maintenir un niveau de performance requis lorsqu'il est utilisé dans les conditions spécifiées* ». Pour remédier au problème de la confusion entre la fiabilité du matériel et la fiabilité du logiciel, on se met sous l'hypothèse d'évaluation d'une architecture matérielle en présence d'une application logicielle qui est parfaitement déterministe, sans tenir en compte de la fiabilité en mode aléatoire de cette dernière [2].

Summary

This article provides the method for evaluating the reliability and availability of programmable critical systems, in their design phase. The main objective is to assess the probability of a hardware architecture of a processor executes instructions correctly and routines of a software application. This probability is calculated from the failure rate of instruction execution software on hardware architecture of a processor cell, which can treat the problem of interaction hardware/software in critical systems programmed. After the definition of different dysfunctional modes we introduce the original method of evaluating of their safety based on an approach using as support the propagation of information.

The reliability of software is defined according to ISO/IEC 9126 (2001) [1] as "the ability of software to maintain a level of performance required when used under the conditions specified." To address the problem of confusion between the trustworthiness of hardware and software reliability, it turns on the assumption of evaluating hardware architecture in the presence of a software application that is perfectly deterministic, without taking into account reliability mode random latter [2].

MOTS-CLES : système critique sûr, conception et évaluation, flux informationnel, processus Markovien.

Introduction

Les systèmes complexes nécessitent des entités fiables. L'intégration de ces entités est un problème complexe qui demande un très haut niveau de vérification des propriétés structurelles, fonctionnelles, non-fonctionnelles ainsi que l'interactivité de ses composants et entités, ce qui est important quand on cherche notamment la fiabilité de ce système complexe.

Parmi les problèmes dans les systèmes programmables complexes -spécialement dans les réseaux- et dans son reconfiguration de redondance structurelle c'est le software qui définie la connexion entre les composants matériels durant le temps. La validation de design et d'exécution de ces systèmes complexes, souvent utilisé dans le contrôle des applications critiques, est très compliquée.

Notre article est organisé de manière suivante : après une brève introduction sur les aspects conceptuels et les hypothèses fondamentales sur la stratégie globale d'évaluation de la disponibilité (dans la section I), la section II présente la méthode utilisé pour évaluer le taux de défaillance pour un programme de Tri de variables. Cette étude est commencée par l'explication de la méthode flux informationnel et l'évaluation d'une instruction assembleur simple, suivie par l'exploitation de ses résultats dans le cas de programme entier sans prise en compte de la stratégie de tolérance. La section III est le résultat d'évaluation de la même application (programme de Tri) mais avec l'ajout de boucles de recouvrement (sauvegarde et restauration). La Section IV sera une conclusion de ce travail et perspective des autres travaux.

1. Stratégie Globale d'évaluation de la disponibilité

L'objectif principal de notre article est de valider la conception intégré, de la partie intelligente d'un capteur mécatronique, sûr de fonctionnement, par une évaluation intégré en processus de conception via une méthode de modélisation dynamique. Le modèle dynamique VHDL-RTL [3] simplifie le passage automatique au modèle de haut niveau destiné à l'étude fiabiliste. La modélisation de ressources matérielles utilisées pendant l'exécution des instructions assembleur, permet une évaluation de l'architecture matérielle vis-à-vis du code de programme en termes de fiabilité [4], disponibilité et crédibilité de l'information.

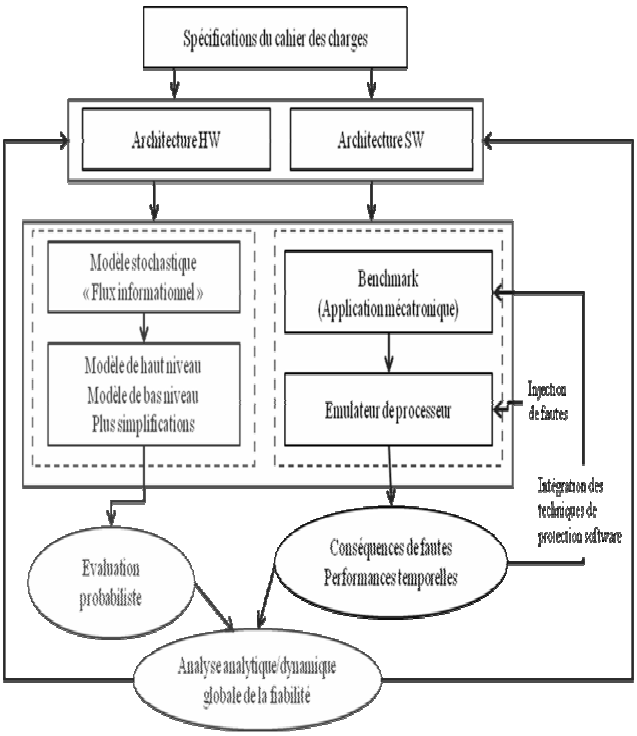

Figure 1. Co-design utilise une stratégie globale d'évaluation fiabiliste.

Les techniques de protection consistent un partage entre les techniques matérielles et les techniques de logiciels (figure 2). Ici, nous nous intéressons à des erreurs qui peuvent être produits dans le processeur (architecture matérielle) sous l'hypothèse d'une application parfaite. Si nous examinons les conséquences de ces techniques, nous remarquons que, d'une part, la mise en œuvre de techniques de protection matérielle, a pour conséquence une zone supplémentaire de silicium et peut rajouter des chemins critiques. D'autre part, la mise en œuvre de techniques de protection logicielle, a pour conséquence une perte de temps, puisque l'ajout des routines augmente la durée de l'ensemble du programme.

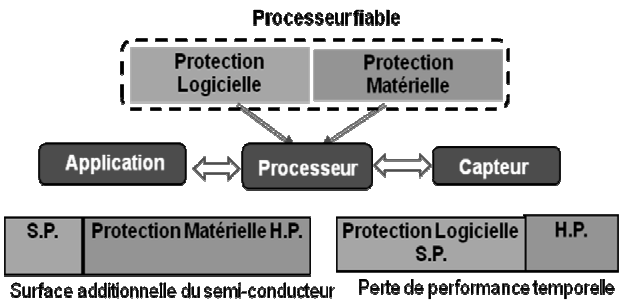

Figure 2. Le compromis entre la protection Matérielle/Logicielle.

2. Approche Flux informationnelle appliquée à un programme de Tri (Sans tolérance aux fautes)

2.1 Cas d'étude

L'ensemble de l'architecture considérée continue de rendre le service attendu en présence de types de fautes prévues, grâce à des stratégies de recouvrement spécifiques. Un exemple concret est le cas de l'exécution d'un jeu d'instructions respectant l'intégrité des données en mémoire et le temps de traitement conforme aux spécifications [5].

Le Hardware/Software Co-conception [6] et [7] est un domaine de recherche récent et très actif, générant des méthodologies de conception efficaces. Les premières tentatives datent du début des années 1990. Les méthodes de conception conventionnelles faisaient la distinction entre la conception de la partie matérielle (HW) et de la partie logicielle (SW). Le HW est la partie physique (UAL, transistors et connecteur…), tandis que le SW est la partie abstraite (sous forme de séquences d'instructions).

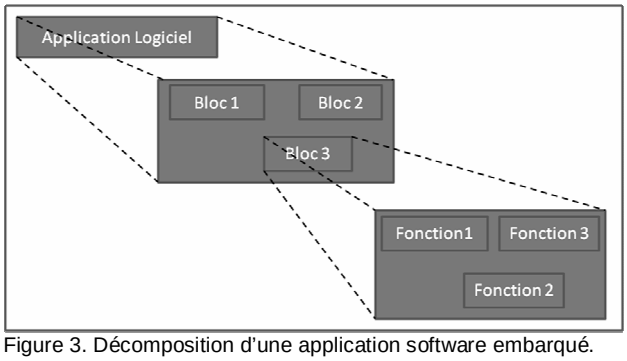

Figure 3. Décomposition d'une application software embarqué.

L'étude de cas est un programme de Tri des variables. Dans un tableau contenant des variables, le programme consiste en la lecture des données, les déplacées dans le cas de désordre. Tout d'abord, nous commençons par l'initialisation des variables dans la mémoire. Nous faisons une remise à zéro du drapeau de permutation pour toutes les données comparées. Si une permutation est faite, le drapeau est remis à 1. Nous faisons la même procédure jusqu'à la fin de variables et jusqu'à ce que le drapeau soit nul.

Pour évaluer la probabilité de la bonne exécution de ce programme après le tri des variables, nous sommes placés dans le pire des cas où ces variables sont totalement désordonnées. La figure 3 montre que, en réalité, l'application logicielle est composée de blocs. Les fonctions et routines qui composent ces blocs sont construites par plusieurs instructions assembleur. Le nombre de ces instructions varie en fonction de l'application.

Nous nous basons sur les résultats de la probabilité d'exécuter correctement les différentes instructions, ce travail montre la faisabilité d'évaluer un programme formé par plusieurs instructions. Le résultat de cette démarche est utile lors de la validation de la stratégie de tolérance proposée, et par la suite concrétiser la modélisation des interactions matériel-logiciel. Dans cette étape, il est nécessaire de montrer comment nous trouvons les différentes valeurs de probabilité pour des simples instructions.

2.2 Evaluation d'une instruction simple

Dans cette partie, un modèle préliminaire VHDL-RTL est fait pour toutes les instructions assembleur afin de faciliter la modélisation selon l'approche flux informationnel [8]. Nous donnons un exemple : l'instruction DUP. Avant d'expliquer ces démarches, il convient de mentionner que le processeur a deux piles. Une pile est utilisée pour le traitement des données appelée pile de données (DS). Le sommet de la pile (TOS) et la prochaine valeur (NOS) correspondent respectivement au premier et au deuxième élément de la pile. La deuxième pile est utilisée pour la sub-routine de retour d'adresse, des adresses d'interruption et des

copies de données temporaires, elle s'appelle pile de retour (RS). Le sommet de la pile de retour (TORS) correspond au premier élément de cette pile. La pile tampons est gérée par une mémoire externe au processeur afin d'éviter une restriction sur la profondeur de la pile. Ils sont adressées par des pointeurs (pointeur de pile de données DSP et pointeur retour de pile RSP).

A partir de ces explications, nous détaillons par la suite l'instruction DUP qui permet la duplication du sommet de la pile (TOS).

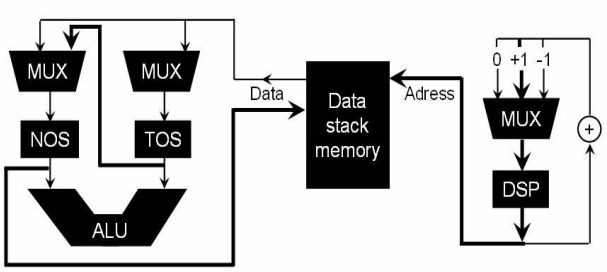

Figure 4. Modèle RTL-VHDL de l'exécution de l'instruction DUP.

Pendant l'exécution de l'instruction DUP, nous remarquons les changements suivants à l'intérieur au processeur :

- La pile de mémoire de données est en mode écriture (Data Stack Memory);
- Les signaux de contrôle de MUX_DSP en position d'écriture, l'entré (+1) est sélectionnée;
- Les signaux de contrôle de MUX_NOS en position d'écriture, la connexion de TOS est sélectionnée.

Au front montant de l'horloge, toutes les opérations sont exécutées et l'instruction DUP est complètement réalisée.

L'idée de base de l'approche flux informationnelle [8] s'articule sur les points suivants :

- Existence de flux d'information.
- Existence de flux d'événement de contrôle.
- Existence des procédures de contrôle.
- Existence des entités de stockage de l'information.

Dans cette optique, on construit le model de haut niveau [8] qui permet une subdivision efficace de l'architecture fonctionnelle, la perturbation d'information par une entité sous-fonctionnelle induit l'utilisation de l'information perturbé en aval du point de perturbation, cette information pouvant éventuellement être partagée par deux entités sous-fonctionnelles ou plus distinctes, on observera alors le résultat de perturbation le long de flux de cette information. Parmi les blocs des entités sous-fonctionnelles, nous présentons :

- TF : entité de transformation de signal, transformation d'une information d'entrée unique en une information de sortie unique par changement de la nature de ce signal.
- SB : entité de stockage du signal, elle permet la mémorisation sur un support d'un signal d'entré et sa restitution ultérieure, ces entités peuvent permettre de représenter les procédés de mémorisation nécessaire à la synchronisation des entités d'un processus de décision.
- IP : entité de décision, elles permettent de délivrer une information de sortie au regard de deux information d'entrées.
- CT : entité de contrôle de flux, correspond aux opérations de vérification de la réception d'une information en un temps raisonnable sans dépassement de délais, elle permet aussi de prendre en compte des procédés d'actualisation de données dans notre architecture (Watch Dog, bascule d'actualisation….).
- ST : entité de test en ligne, correspond aux opérations de test en ligne.

Le résultat d'application du modèle de haut niveau sur l'instruction DUP est illustré par la figure 5:

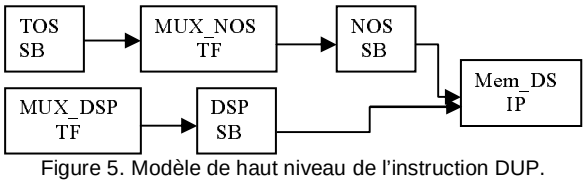

Figure 5. Modèle de haut niveau de l'instruction DUP.

L'instruction DUP copie le sommet de la pile (TOS) dans le prochain élément de registre (NOS) et empiler la NOS dans le troisième élément de la pile mémoire de données (Mem_DS). Cette instruction doit incrémente le pointeur de pile de données (DSP) afin de créer une nouvelle cellule dans Mem_DS.

Concernant le modèle de bas niveau [8], il s'agit modéliser le comportement fonctionnelle et dysfonctionnel de chaque sous-entité fonctionnelle du modèle de haut niveau en lui associant un automate d'états finis [9]. Nous classons les événements de transitions des ces automates, en associant à chacune des transitions internes entre états certains attribues, nous avons modélise principalement les défaillances transitoire (bit-flip) responsable à la qualité de l'information, à ce niveau nous pouvons parler de la crédibilité de l'information, il s'agit d'un mode de défaillance lié à la disponibilité du système par sa nature transitoire. Nous avons aussi modélisé les défaillances matérielles (dM) lié à la fiabilité du système puisque il s'agit d'un mode de défaillance permanant. Il était possible de modéliser les modes suivants :
Traitement des cas non exécutables ; Les cas puits, Dead-Lock situation; Les boucles infinies, Live-Lock situation; Les types interdits de communication entre les états.

Le résultat de l'application de modèle de bas-niveau sur l'instruction DUP est illustré sur la figure 7. Dans le modèle de bas niveau la transition entre deux états de l'automate représente le flux d'informations erronées [16, 17, 18].

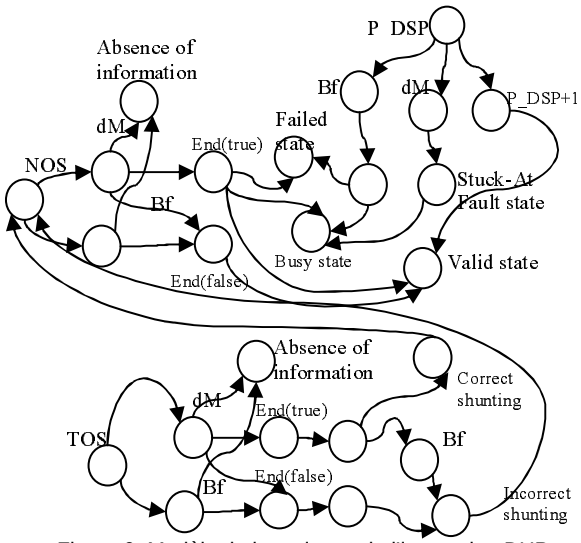

Figure 6. Modèle de bas niveau de l'instruction DUP.

Ldef=
{TOS.Fin(true).Bf.Aig_corr.NOS.Fin(true).P_DSP.dM.Val.collé.Etat_occup }

Cette liste est constituée des mots qui sont la base de langage de défaillance, TOS.Fin(true) signifie que l'information issue de TOS est incorrect, cette information erroné sera propager le long de MUX (élément aval dans le model de haut niveau), aussi le mot

Bf.Aig_erro traduit l'erreur de l'aiguillage provoqué par un Bf (Bit-flip), la même chose au niveau de NOS qui est une zone comme le TOS. Finalement le mot P_DSP.dM.Val_collé.Etat_occup signifie qu'un pro-blème matériel peut provoquer une valeur collé et par la suite un état mémoire occupé.

Après une étude qualitative préliminaire de risque (AMDEC) [10] que nous avons fait sur l'architecture de processeur à pile, nous avons pu comprendre l'aspect fonctionnel et dysfonctionnel du système, et par la suite de définir six modes de fonctionnement dans laquelle nous parlons de niveau de crédibilité de l'information [13].

- Mode 1: Tout va bien, mesure correct et pas de défaillance détectée.
- Mode 2: Recherche de la disponibilité, mesure incorrecte, défaillance détectée mais tolérée.
- Mode 3: Recherche de la fiabilité, mesure incorrecte, défaillance détectée mais non tolérée.
- Mode 4 : non sécurité, mesure incorrecte, défaillance non détectée.
- Mode 5: arrêt intempestif, mesure correcte et défaillance détectée.
- Mode 6: arrêt du système, absence de mesure.

Les séquences de propagation d'erreurs générées permettent, à l'aide d'un modèle markovien non homogène, de quantifier et d'obtenir de manière analytique les paramètres de sûreté de fonctionnement du système telle que la fiabilité, la disponibilité, ou la sécurité. Elles permettent de caractériser l'état de fonctionnement du système selon les six modes définis précédemment.

Nous considérons que le taux de défaillance d'un composant (taux de panne) résulte à la fois d'un niveau d'usure du composant et de l'exposition aux stress environnementaux. Nous proposons l'équation suivante portant sur le taux d'apparition de panne d'un composant, noté $\lambda i(t)$, comme étant égale, sous l'hypothèse d'une sollicitation effective au composant sur un cycle d'horloge T (un pas de calcul), la probabilité d'apparition de panne sur cette intervalle [t, t+T].

Nous supposons sur un cycle d'horloge, l'existence d'une fonction Ωc portant sur les stress environnementaux (température, pression, niveau de rayonnement, susceptibilité électromagnétique, présence de neutrons et de particules alpha) auxquelles le composant est soumis. Cette fonction fait partie de la relation générale suivante :

$$\lambda i\,(t) = \alpha.\lambda i(t+T) + \Omega c\,(stress, T) \qquad [1]$$

La fonction α est liée au mode de sollicitation de la ressource. Elle permet la prise en compte d'une usure progressive du composant. Certaines bases de données de taux de défaillances des composants comme EADS/Thales, Siemens SN29500, MIL-HDBK-217F donnent ces fonctions sous forme d'un produit de facteurs correctifs liés aux différentes contraintes environnementales :

$$\Omega c(stress, T) = F(T).F(P).F(R)\ldots\ldots \lambda ref \qquad [2]$$

T : Température ;
P : Pression ;
R ; Radiation.

Après la génération de toutes les listes caractéristique (scénarios) à partir du modèle de bas niveau, nous les classons en modes de défaillance selon les six modes définis précédemment.

L'analyse basée sur les arbres de défaillance peut être qualitative ou quantitative. Les arbres de défaillance couplé aux diagrammes de discision binaires (BDD), est une technique peut être utilisée pour évaluer la fiabilité des systèmes électroniques programmables. Toutefois, des travaux de recherche montrent que les algorithmes utilisés dans ces arbres de défaillance ne sont pas suffisants dans le cas des systèmes complexes programmables pour plusieurs raisons, tels que le problème d'explosion combinatoire des Etats [11]. L'approche flux informationnel est proposée comme solution. L'analyse qualitative montre quelles sont les combinaisons qui peuvent se produire pour provoquer une défaillance du système. Concernant l'analyse quantitative, quand nous voulons quantifier la probabilité de ces événements, les listes caractéristiques générées seront transformée à des arbres de défaillance. La probabilité de l'élément sommet est calculée à partir des probabilités des événements de base e l'arbre. Des outils tels que Aralia1, dans de nombreux cas, donnent des résultats plus précis que les outils conventionnels grâce à sa vitesse de simulation (1000 fois plus rapide) [12].

Après la transformation des sous-entités fonctionnelles du modèle de haut niveau à des automates d'états finis, il est nécessaire de calculer le taux d'échec de transition pour atteindre chaque état final. Nous utilisons des outils comme le processus Markovien. L'approche d'évaluation basée sur le flux d'informationnel représente à la fois la défaillance de cause commune, les défaillances multiples... Les transitions entre états de modèle de bas niveau (niveau automate) représentent le taux de défaillance de l'information. Ces valeurs représentent également la matrice de transition du processus Markovien. Les valeurs calculées par le processus Markovien remplacent les différentes probabilités des éléments de base d'un arbre de défaillance.

Le tableau 1 donne les résultats de probabilité selon différents modes de fonctionnement de l'instruction DUP :

Probabilité d'existence dans :	**Mode 2**	**Mode 3**	**Mode 4**	**Mode 5**
DUP	$72\ 10^{-3}$	$9\ 10^{-5}$	$8.99\ 10^{-10}$	$4.9\ 10^{-22}$

Tableau 1. Probabilité de l'instruction DUP.

Les exigences de sécurité et de qualité de service souvent requises en systèmes critiques programmable nécessitent la mise en œuvre de techniques de détection et de recouvrement de fautes. L'implémentation (matérielle ou logicielle) de ces mécanismes nécessite une analyse de l'impact des fautes sur le comportement de l'architecture.

2.3 Evaluation de programme de Tri sans tolérance aux fautes

L'objectif principal est de savoir à chaque instant les valeurs des probabilités des six modes de fonctionnement du programme exécuté. Grace à l'exploitation des résultats des instructions simples, ainsi que les probabilités d'exécution des routines.

La probabilité de l'élément sommet de la Figure 7 est facilement calculée par l'évaluation des probabilités des éléments de base (les probabilités des éléments sommet des sous-arbres). La subdivision de l'arbre global à des sous-arbres est un moyen pratique et efficace pour calculer la probabilité globale. Le calcul de probabilité de chaque mode de dysfonctionnement du programme nécessite le remplacement des probabilités des éléments de base qui sont tout simplement les probabilités des sous-arbres (figure 8) correspondant au même mode de dysfonctionnement.

L'arbre du programme global est le suivant :

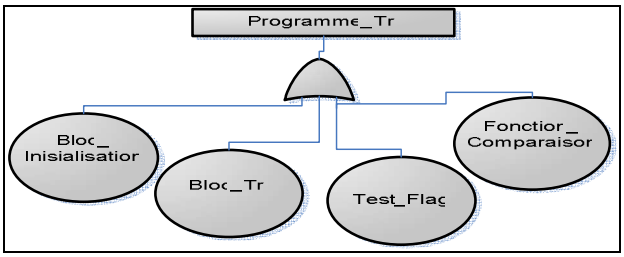

Figure 7. Arbre de défaillance du programme de Tri sans tolérance.

Par exemple, la probabilité de l'élément de base «Flag_Test » est donnée par le sous-arbre comme le montre la Figure 8.

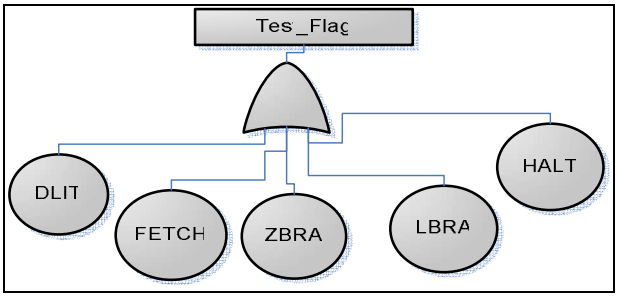

Figure 8. Arbre de défaillance du bloc 'Test_Flag'.

Dans cette configuration, nous devons trouver la probabilité de différentes instructions assembleur qui compose cet arbre avec la même manière que l'instruction DUP détaillée précédemment. Les résultats de cette modélisation sont illustrés par le tableau 2. Dans ce tableau, on peut constater que la probabilité d'être en mode 3 est importante. Cela signifie que notre architecture n'est pas en mesure de tolérer certaines fautes.

La probabilité de mode 2 est nulle parce que nous n'avons pas encore mis en œuvre la technique de recouvrement. Dans la prochaine étape de ce travail, nous montrons comment nous pouvons changer ces résultats par l'ajout des techniques de tolérance aux fautes.

	Mode 1	**Mode 2**	**Mode 3**
Initialisation_Bloc	$4.894\ 10^{-8}$	0.0	$3.091\ 10^{-3}$
Sorting_Bloc	$1.27\ 10^{-11}$	0.0	$10.5\ 10^{-3}$
Flag_Test	$9.17\ 10^{-6}$	0.0	$4.59\ 10^{-3}$
Comparison_Function	$58.49\ 10^{-8}$	0.0	$7.9\ 10^{-4}$
Sorting_program	$\mathbf{7.49\ 10^{-6}}$	0.0	$\mathbf{6.67\ 10^{-2}}$

Tableau 2. Probabilités de programme de Tri sans tolérance aux fautes.

3. Evaluation de programme de Tri avec tolérance aux fautes

Le développement d'un système programmé est souvent entaché de fautes, lors de sa phase de conception et/ou de son implémentation, sans qu'elles soient corrigées lors de la phase de validation. Ces erreurs systématiques restent latentes lors de la phase d'exploitation jusqu'à ce qu'elles soient activées [13]. L'origine d'une erreur est une faute, qui peut être dans un certain nombre de cas d'ordre conceptuelle ou physique (défaillance matérielle). L'origine d'une erreur est, en général, une faute humaine au niveau de la conception, ou une faute physique d'un composant matériel de ce système. L'activation d'une erreur d'un composant dans le système, peut affecter le service que rend le système. Le composant est dans ce cas déclaré défaillant. Par ailleurs, la défaillance d'un composant représente une faute auprès des autres composants avec lesquels il interagit. Les défaillances, les erreurs et les fautes, en présence de mécanismes de tolérance aux fautes, ne doivent plus entraîner la défaillance du système.

La tolérance aux fautes représente la capacité d'un système à remplir sa fonction en dépit des fautes [14]. Plusieurs moyens de la sûreté de fonctionnement peuvent être envisagés pour bâtir des systèmes sûrs de fonctionnement (i.e. l'élimination, la prévention ou encore la prévision des fautes).

La prise en compte des défaillances envisageables permet, vis-à-vis des contraintes de la mission, de prévoir les actions de recouvrement jugées pertinentes. En général, le développement (tenant compte des contraintes environnementales de la mission) devra fournir des moyens permettant d'intégrer les réactions aux défaillances (mécanismes de tolérance et de recouvrement).

Nous considérons l'exemple de programme de Tri décrie précédemment, la stratégie de recouvrement se traduit par l'ajout des fonctions de sauvegarde et de récupération que nous modélisons par la suite.

Comme on peut le voir dans l'organigramme de l'algorithme de Tri (figure. 9), l'implémentation de la stratégie de recouvrement est faite avant chaque itération de tri afin de garder en mémoire les données de pointeurs jugées correctes avant défaillance.

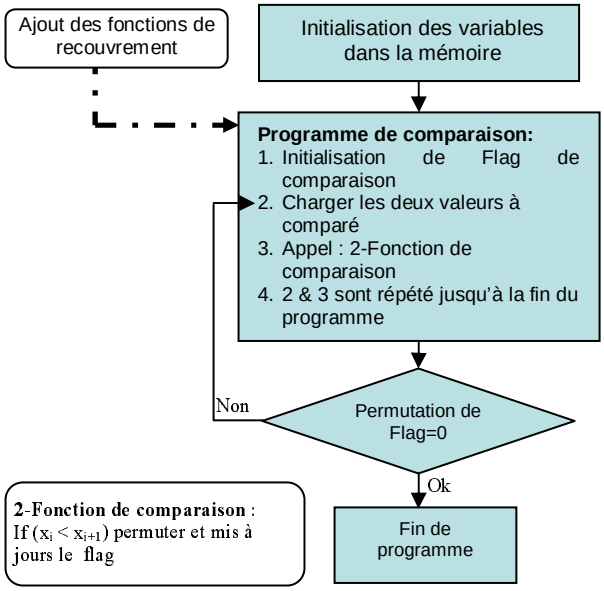

Figure 9. Organigramme de l'algorithme de Tri avec prise en compte de recouvrement.

La différence par rapport au programme précédent, c'est l'ajout de la stratégie de recouvrement. Cette stratégie est basée sur la sauvegarde des pointeurs avant chaque début de cycles de comparaison. En cas de problème pendant le déroulement du programme, la stratégie de recouvrement consiste à restaurer le dernier enregistrement des ces pointeurs. Après cette restauration le programme continue à s'exécuter avec des données sûres.

La démarche pour calculer la probabilité de défaillance d'exécution de ce programme (avec recouvrement), est la même qu'au paragraphe précédent. L'ajout de la partie sauvegarde du contexte, après chaque aller-retour sur le tableau de dix valeurs, nécessite la prise en compte des valeurs de probabilité afin de pouvoir récupérer à tout moment les valeurs des pointeurs (TOS, NOS, DSP, TORS…). La probabilité de défaillance du programme sera influencée par la probabilité de défaillance des fonctions de sauvegarde et de récupération [15] comme la montre l'arbre de défaillance figure 12.

À titre d'exemple, nous considérons la sauvegarde et la restauration de pointeur de pile données (DSP), réalisé par les fonctions suivantes :

```
---------- Save_DSP ----------
Push_DSP
DLIT @_Sauv_DSP
STORE

---------- Restore_DSP --------
DLIT @_Sauv_DSP
FETCH
Pop_DSP
```

Figure 10. Fonction de sauvegarde et de restauration de DSP.

Pour évaluer par exemple la probabilité de la fonction "Save_DSP, il faut avoir les probabilités de toutes les instructions assembleur (Push_DSP, DLIT et STORE).

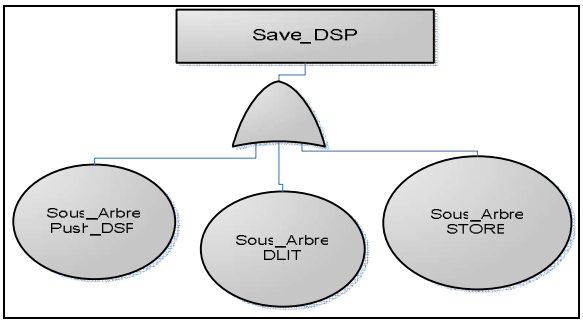

Figure 11. Arbre de défaillance de la fonction sauvegarde DSP.

L'ajout du contexte de stockage après chaque opération de tri, implique la prise en compte des probabilités des fonctions de sauvegarde et restauration à chaque instant pour les éléments (TOS, NSA, TDR, DSP, RER et PC). L'arbre global aura la forme suivante :

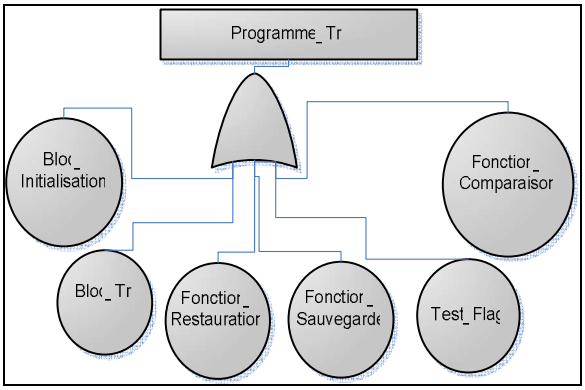

Figure 12. Arbre de défaillance du programme de Tri avec prise en compte de tolérance.

Le tableau suivant résume les valeurs de probabilité pour chaque mode de fonctionnement selon les six modes de fonctionnement que nous avons définis.

	Mode1	Mode2	Mode3
Initialisation_Bloc	$4.894\ 10^{-8}$	$1.430\ 10^{-2}$	$3.091\ 10^{-3}$
Sorting_Bloc	$1.27\ 10^{-11}$	$1.57\ 10^{-2}$	$10.5\ 10^{-3}$
Flag_Test	$9.17\ 10^{-6}$	$3.55\ 10^{-3}$	$4.59\ 10^{-3}$
Comparison_Function	$58.49\ 10^{-8}$	$3.70\ 10^{-2}$	$7.9\ 10^{-4}$
Save_Function	$7.44\ 10^{-5}$	$1.55\ 10^{-2}$	$2.2\ 10^{-4}$
Restore_Function	$11\ 10^{-5}$	$0.33\ 10^{-2}$	$1.9\ 10^{-5}$
Sorting_Programme	$\mathbf{6.32\ 10^{-5}}$	$\mathbf{1.57\ 10^{-2}}$	$\mathbf{2.39\ 10^{-4}}$

Tableau 3. Probabilités de programme de Tri avec tolérance aux fautes.

Dans ce tableau, nous constatons que la probabilité d'être en mode 2 (défaillance non détecté mais toléré) devient importante et la probabilité du mode 3 (défaillance non détecté mais non tolérée) diminue en raison de l'ajout des fonctions de recouvrement des éléments sensibles dans l'architecture du processeur à pile. Cela montre l'avantage et l'efficacité de la technique logicielle de tolérance aux fautes.

Ces résultats peuvent être analysés et comparés avec les résultats obtenus en appliquant la même approche sur le programme de Tri sans tolérance. Cette comparaison est illustrée dans le tableau 4.

	Mode 1	Mode 2	Mode 3
Sorting_Program without tolerance	$7.49\ 10^{-6}$	0.0	$6.67\ 10^{-2}$
Sorting_Program with tolerance	$6.32\ 10^{-5}$	$1.57\ 10^{-2}$	$2.39\ 10^{-4}$

Tableau 4. Comparaison des probabilités de programme de Tri avec/sans tolérance aux fautes.

Par rapport au programme de Tri sans tolérance, nous remarquons que, la probabilité d'être en mode 1 (résultat correct et aucune défaillance détectée) et la probabilité d'être en mode 2 (défaillance détectée mais tolérée) ont augmenté. C'est un résultat logique dû à l'effet de la technique de tolérance. Contrairement à la probabilité d'être en mode 3 (défaillance détectée mais pas tolérée) a diminué. Cela signifie que la non-tolérance de défaillance est moins probable. Ainsi, il est clair dans ce tableau que la mise en œuvre des fonctions de récupération a un effet sur les résultats global de probabilité, ainsi que l'avantage de cette technique et son efficacité à tolérer les fautes transitoires [14].

4. Conclusions et perspectives

Dans cet article, nous proposons une approche de vérification de la conception, complémentairement à la stratégie de test/validation, et permettant d'évaluer les probabilités de défaillance, comme définies dans la norme IEC 61508 [19]. L'intérêt est de mettre en œuvre, après évaluation, des mécanismes de la tolérance aux fautes, et de positionner notre système dans le mode où la tolérance est efficace. Ceci est traduit par la réduction de la probabilité du mode 3 (défaillance détecté mais non tolérée), ce qui est en étroite relation avec l'augmentation de la crédibilité.

Nous avons tout d'abord détaillé la fiabilité de notre stratégie globale. Parce que notre priorité est la conception de systèmes fiables, une technique de protection logicielle mises en œuvre est évaluée en utilisant le modèle RTL-VHDL et la modélisation du jeu d'instructions selon l'approche flux informationnelle. Cette approche est composée d'un modèle de haut niveau et un modèle de bas niveau modèle. Le premier facilite la modélisation des ressources matérielles de l'architecture par des sous-entités fonctionnelles. La seconde consiste à trouver un automate d'états finis comportemental pour chaque entité de haut niveau.

Afin d'évaluer une architecture matérielle en présence d'une application logicielle (la technique logicielle de tolérance compris) selon l'approche flux informationnel, nous avons tout d'abord appliqué cette approche sur des instructions assembleur simple. Les résultats obtenus sont exploités pour l'évaluation d'un programme de tri (ensemble d'instructions exécutées séquentiellement). Ensuite, la même application est également évaluée avec prise en compte de la stratégie de tolérance aux fautes. Leurs résultats sont analysés et comparés.

Les conclusions tirées de ces résultats numériques sont variées. Avant l'ajout de la stratégie de recouvrement, les parties de programme 'Bloc_initialisation', 'Bloc_Tri', Bloc_Test_Flag' et 'Bloc_Fonct_Comparaison' ont des probabilités nulles d'être dans le mode 2, et une probabilité non négligeable pour que ces erreurs soient non tolérées, référant au mode 3. Après l'ajout de fonctions de recouvrement, la probabilité du mode 2 de ces blocs de programme devient non nulle et plus importante devant la probabilité du mode 3. Ceci dit que l'effet de ces fonctions est incontournable, et les instructions qui constituent ces fonctions sont à l'origine de l'augmentation de la probabilité du mode 2 de ces fonctions et par la suite de tout le programme de Tri.

Références

[1] Norme CEI 9126. – Génie du logiciel – Qualité des produits – Partie 1 : Modèle de qualité, Genève 2001.

[2] F. Vallée et D. Vernos, "Le test et la fiabilité du logiciel sont ils antinomiques?" *12ème Colloque national de Fiabilité et Maintenabilité*, Montpellier, 2000.

[3] M. Jallouli, C. Diou, F. Monteiro and A. Dandache "Stack processor architecture and development methods suitable for dependable applications", in Proc. *3rd International Workshop on Reconfigurable Communication Centric System-On-Chips* (ReCoSoC'07), June 2007.

[4] Z.Lei, H.Yinhe, L.Huawei, L.Xiaowei, "Fault Tolerance Mechanism in Chip Many-Core Processors", *Tsinghua Science and Technology*, ISSN 1007-0214 30/49, vol. 12, No. S1, July 2007 pp.169-174.

[5] Xiao Qin and Hong Jiang, 'Estimation of software reliability with execution time model using the pattern mapping technique of artificial neural network'. Computers & Operations Research, Volume 32, Issue 1, January 2005, Pages 187-199 Nidhi Gupta and Manu Pratap Singh.

[6] Janusz Sosnowski, 'Software-based self-testing of microprocessors'. Journal of Systems Architecture, Volume 52, Issue 5, May 2006, Pages 257-271.

[7] Wayne Wolf, 'Hardware and Software Co-design'. High-Performance Embedded Computing, 2007, Pages 383-432.

[8] K. Hamidi, O.Malassé, J.F. Aubry "Coupling of information-flow aggregation method and dynamical model for a more accurate evaluation of reliability", *European Safety and Reliability Conference*, ESREL, June 2005.

[9] C.Bolchini, R.Montandon, F.Salice and D.Sciuto, "Finite State Machine and Data-Path Description", *IEEE Transactions on very large scale integration (VLSI) systems*, vol.8, No.1, February 2000 pp. 98-103.

[10] H. Belhadaoui, O.Malassé, J.F. Aubry, H.Medromi, 'Outil d'aide à la conception des systèmes mécatroniques'. Qualita-Tanger Maroc, Mars 2007.

[11] J D.Esary, H.Ziehms, "Reliability analysis of phased missions" In *Proceedings of Reliability and Fault Tree Analysis*, Philadelphia, USA, 1975 pp. 213-236.

[12] Group Aralia, "Computation of prime implicants of a fault tree within Aralia" In *Proceedings of the European Safety and Reliability Association Conference*, ESREL'95, Bournemouth, UK, 1995: 190-202.

[13] ROBERT P. COLWELL and RICHARD A LETHIN, 'Latent design faults in the development of the multiflow TRACE' . IEEE Transactions on Reliability, 34(4), 557 (December 1994). Microelectronics and Reliability, Volume 36, Issue 4, April 1996, Page 537.

[14] Aiguo Li and Bingrong Hong, 'Software implemented transient fault detection in space computer'. Aerospace Science and Technology, Volume 11, Issues 2-3, March-April 2007, Pages 245-252.

[15] Kishor S. Trivedi, Jogesh K. Muppala, Steven P. Woolet and Boudewijn R. Haverkort, "Composite performance and dependability analysis", *Performance Evaluation*, vol. 14, Issues 3-4, February 1992 pp. 197-215.

[16] Jing-An Li, Yue Wu, King Keung, Ke Liu, "Reliability estimation and prediction of multi-state components and coherent systems", *Reliability Engineering and System Safety 88*, 2005 pp. 93-98.

[17] Mile K. Stoj cev, Goran Lj. Djordjevi c, Tatjana R. Stankovij "Implementation of self-checking two-level combinational logic on FPGA and CPLD circuits", *Microelectronics Reliability 44*, 2004 pp. 173–178.

[18] A. Rauzy, "Mode automaton and their compilation into fault trees", *Reliability Engineering and System Safety,* 2002.

[19] IEC 61508, 1999 Functional Safety of Electrical/ Electronic/Programmable Electronic Safety-Related Systems, Part 1-7. International Electrotechnical Committee.