Evaluation de la disponibilité d'un processeur tolérant aux fautes.

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

Evaluation de la disponibilité d'un processeur tolérant aux fautes.

Hicham BELHADAOUI[1][2], Olaf MALASSE[1], Gregory BUCHHEIT [1], Jean-François AUBRY[2]

[1] : *A3SI, ENSAM, A&M ParisTech, 4 Rue Augustin Fresnel 57078 Metz Cedex 3, France*

[2] : *CRAN, Nancy Université, CNRS, 2, Avenue de la forêt de Haye 54 516 Vandœuvre-lès-Nancy Cedex, France*

Résumé :
Actuellement, les efforts se concentrent sur l'intégration d'outils pour la conception de systèmes complexes, en omettant les aspects sûreté. La sûreté de fonctionnement relevant du niveau système, chaque métier se doit de l'intégrer dans sa doctrine, mais la validation des avancées de chacun ne peut qu'être réalisée globalement.

Le travail mené par l'équipe A3SI-CRAN a pour objectif de définir une méthodologie de conception d'un système complexe programmable dédié à une application mécatronique [1], intégrant dès les premières phases du cycle de développement [2], les aspects de la sûreté de fonctionnement. L'apport d'une telle méthodologie doit permettre de faire face à un certain nombre de contraintes propres au domaine des capteurs intelligents (les exigences du cahier des charges, le respect des normes législatives en vigueur).

Abstract:
Currently, the efforts are focused on the integration of designing tools for complex systems, without taking the security aspects. The security operation under the system level, each field must integrate it into its compte, but the validation of the progress of each can only be done globally.

The work given by the team A3SI-CRAN aims to define a methodology for designing of complex programmable system dedicated to an application mechatronics [1], integrating in the start phases of development [2], the dependability aspects. Providing such methodology would be able to support a number of constraints in the intelligent sensors field (the requirements of the specifications, standards legislation).

Mots clés : Système embarqué, outil de modélisation, outil de fiabilité, modélisation RTL, tolérance aux fautes, IEC61508, processeur à pile.

Keywords : Embedded systems, device modeling, reliability issues, RTL modeling, fault-tolerant, IEC61508, stack processor.

I-Introduction :

La criticité des systèmes complexes programmables nécessite de garantir un niveau de fiabilité et de sécurité convenable. Des études de sûreté de fonctionnement doivent être menées tout au long du cycle de développement du système. Ces études permettent une meilleure maîtrise des risques et de la fiabilité. Les points faibles sont mis en évidence et permettent aux concepteurs de spécifier des stratégies de reconfiguration avant la phase de prototype réel et les tests réels. Les études de sûreté de fonctionnement doivent être menées au plus tôt dans la phase de conception, afin de réduire les coûts et le nombre de prototypes nécessaires à la validation du système. Le processus macroscopique de développement reste donc celui d'une ingénierie concourante (séquentiel récursif), comme l'illustre le schéma ci-dessous :

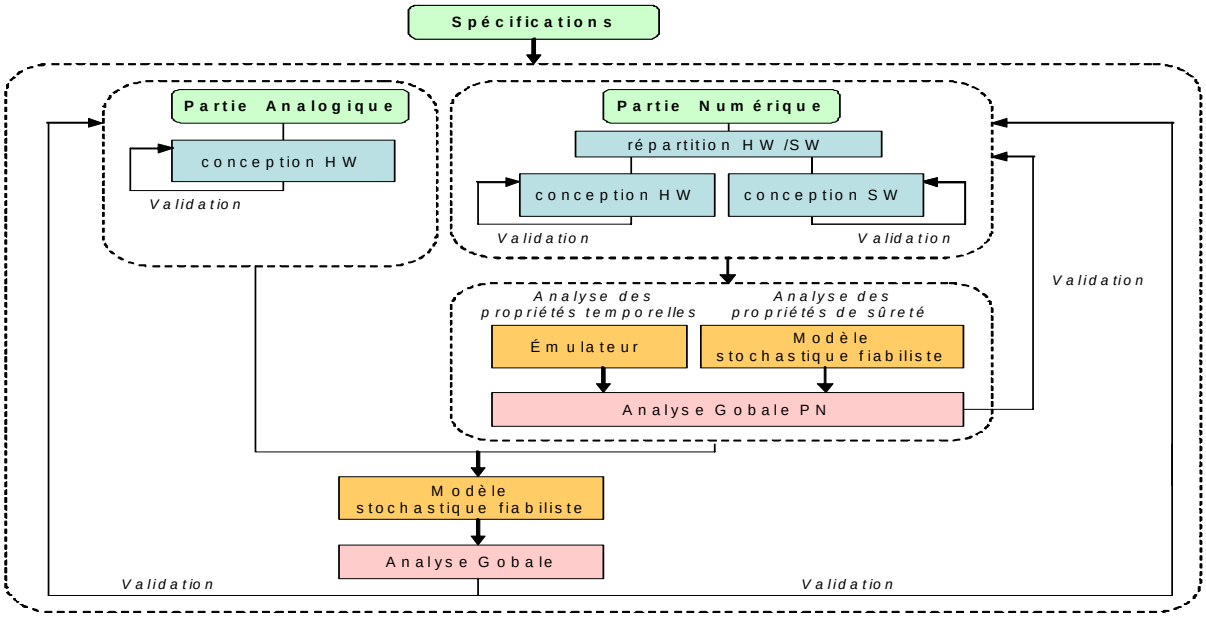

Figure 1 : Processus macroscopique de développement des systèmes mécatroniques.

Dans ce travail nous interprétons la défaillance de l'information comme étant le résultat de l'initiation et de la propagation d'informations erronées à travers l'architecture d'un capteur intelligent dédié à une application mécatronique. Cette propagation s'est accompagnée de contraintes (partage de ressources matérielles et informationnelles, modes dégradés d'information…) qui tendent à influencer fortement la crédibilité de cette information.

Apres l'introduction, nous détaillons dans la Section II l'évaluation de la probabilité qu'une architecture matérielle d'un processeur d'exécuter correctement une couche logicielle de plusieurs routines d'une application embarquée. Cette probabilité est calculée à partir des probabilités de défaillance d'exécution des instructions simples du programme. La section III présente l'évaluer de taux de défaillance pour un programme de Tri de variables. Cette étude est achevée grâce à l'exploitation de la méthode flux informationnel pour 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 IV 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 V sera evidement une conclusion de ce travail et perspective des autres travaux.

II- Evaluation d'une instruction Simple :

Dans ce travail, nous proposons une extension du travail réaliser dans le cadre d'une thèse au sien de notre équipe [5]. Nous allons traiter un cas plus complexe (l'interaction matériel-logiciel). Une étude bibliographique a démonté les lacunes des méthodes conventionnelles actuelles d'évaluation quantitative (RBD [6]; Réseaux de fiabilité [7]; FTA [8]…), et qui semblent aujourd'hui insuffisantes pour une prise en compte correcte de certain modes de défaillances (défaillance de cause commune [9]; modes latents de défaillance [10]; problème d'autotest, arrêt intempestif…). Nous avons construis une méthode d'évaluation analytique, contrairement aux approches numériques basés sur la simulation de type (réseau de Pétri, Monte-Carlo…).
Apres avoir détaillé les spécifications du cahier des charges, la modélisation selon l'approche flux informationnel consiste à décrire par une modèle de haut niveau constitué d'entités sous-fonctionnelles toutes les ressources matérielles requièrent pour l'exécution du programme.

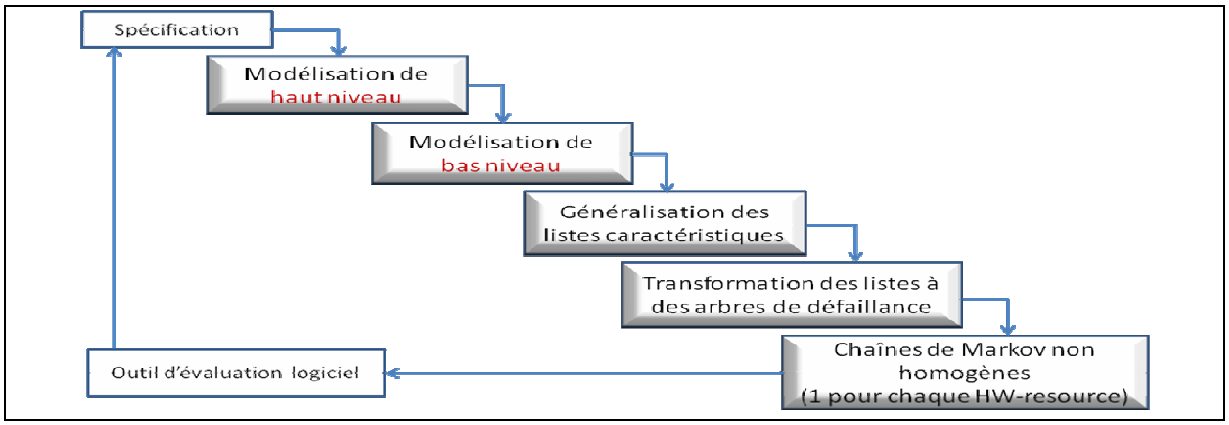

Figure 2 : Démarche de l'approche Flux Informationnel.

Le modèle de haut niveau caractérise les échanges d'informations (flux de données, flux de contrôle, stockage de l'information) entre les entités sous-fonctionnelles. La deuxième phase de modélisation consiste à traduire chaque entité sous-fonctionnelle de haut niveau en un automate d'état fini approprié qui décrit son comportement, permettant de tracer le chemin de propagation des erreurs au sein de l'architecture du système et de concrétiser leurs effets sur le mode de fonctionnement du système, le résultat de cette modélisation sera un ensemble de listes caractéristiques. Ces listes sont constituées des mots de langage d'automates d'états fini. Ces mots représentent l'ensemble des transitions entre les états des automates, et ces transitions n'est autre que la représentation concrète de la perturbation de l'information, inhibition de l'information erronée ainsi que l'absence totale de l'information. Dans cet esprit, les listes ainsi générées représentent d'une manière exhaustive tous les modes de défaillance possibles, à partir d'une sélection d'erreur des cas types les plus souvent rencontrés, avec la prise en compte des stress environnementales. Les six états fonctionnels définis pour cette étude sont :

- **Mode1** : Fonctionnement nominal, mesure correcte et pas de défaillance détectée.
- **Mode2 :** Défaillance sûr disponible, signifie qu'il y a une mesure incorrecte, défaillance détectée et tolérée.
- **Mode3 :** Défaillance sûr indisponible, signifie qu'il y a une mesure incorrecte, défaillance détectée et non tolérée.
- **Mode4 :** Défaillance dangereux, cela traduit une mesure incorrecte, défaillance non détectée.
- **Mode 5 :** Arrêt intempestif, mesure correcte et défaillance détectée (indisponibilité occasionnelle).
- **Mode 6 :** Arrêt du système, absence de mesure (indisponibilité totale).

I-1 Exemple d'évaluation d'instruction simple :

Le schéma de la figure 3 est une partie du processeur à pile conçu par l'équipe LICM de Metz. Les schémas RTL (Register Transfer Level) donnent une idée de l'architecture matérielle nécessaire (ensemble de composants, mémoire, pointeur, bus…) à l'exécution d'une instruction assembleur. Il est faisable de passer de ce schéma RTL au modèle de haut niveau de l'approche flux informationnel, nous signalons à titre indicatif que cette phase pouvant être automatisée. La réalisation de l'instruction DUP nécessite l'exécution de microcodes selon le séquencement suivant :

- ✓ DSP ← DSP +1 « incrémentation de DSP pour ajouter un nouvel élément »
- ✓ 3ème élément DS (Valeur mémoire dont l'adresse est le nouveau DSP) ← NOS « NOS devient le 3ème élément de DS»

- ✓ NOS ← TOS « duplication de la valeur de TOS»

La figure 3 représente le schéma RTL de l'instruction DUP sur une architecture de processeur à pile. Il s'agit d'une représentation interne (composants et cheminements de données entre composants) du processeur à pile.

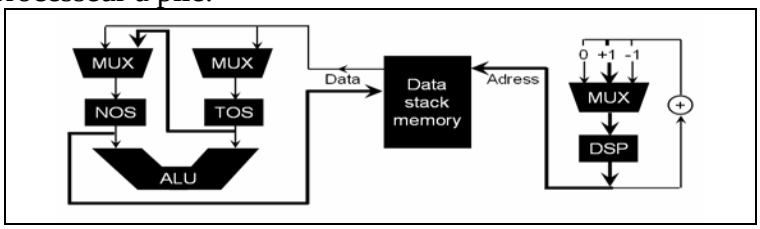

Figure 3 : Modèle RTL de l'instruction DUP.

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.

La connaissance des ressources matérielles utilisées pendant l'exécution de l'instruction DUP, nous permettons d'établir le modèle de haut niveau suivant (figure 4). Ce modèle de haut niveau est constitué par des entités sous-fonctionnelles, dont chacune représente un composant matériel de l'architecture du processeur à pile.

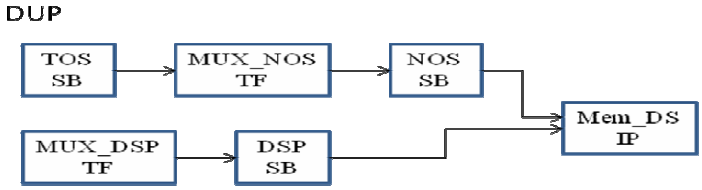

Figure 4 : Modèle de haut niveau de l'instruction DUP

Nous donnons la signification de chaque entité sous-fonctionnelle :

- ✓ 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'entrée et sa restitution ultérieure, ces entités permettent 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 informations 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.

La figure 5 représente le modèle de bas niveau de l'instruction DUP. A partir de ce graphe d'automate d'états finis (ensemble d'états et des transitions), on génère un ensemble de langages sous forme de listes. Les listes caractéristiques de ce langage décrivent tous les chemins possibles menant à un état défaillant. Les listes générées vont permettre de quantifier

la probabilité de défaillance des différents modes dysfonctionnels résultant d'un événement externe (exemple : bit-flip), ou interne (exemple : erreur matérielle, erreur de cheminement…). A partir de ce modèle d'automate d'état fini (figure 5), nous extrairons un exemple d'une information erroné résultat d'une perturbation ou une défaillance matérielle au niveau des ressources matérielle (TOS, NOS, DSP, MUX) du schéma RTL (figure 3), l'ensemble des ces ressources participent à l'exécution de l'instruction DUP.

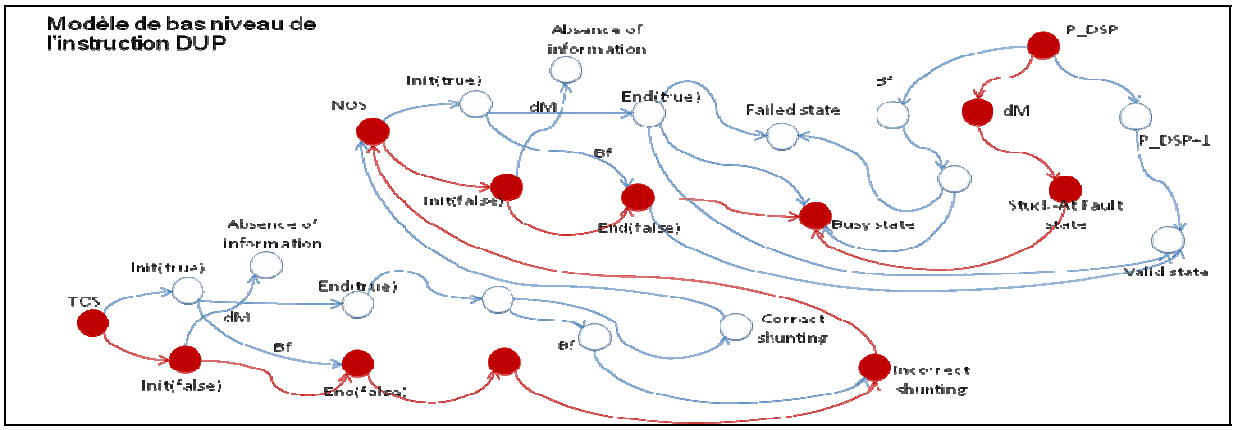

Figure 5 : Modèle de bas niveau de l'instruction DUP.

$$L_{def}=\{TOS.Init(false).End(false).Incorrect\_shunting,\ NOS.Init(false).End(false),\ P\_DSP.dM.Stuck\_At\_Fault\_state.Busy\_State\}$$

Pour quantifier la probabilité d'avoir la combinaison d'erreur donnée par cette liste, nous traduisons la propagation de l'information par un arbre de défaillance schématisé par la figure 6. La probabilité de l'élément sommet (information erroné = exécution incorrecte de l'instruction DUP, dans la liste au dessus Busy_State) est liée aux probabilités de différents états de défaillance des éléments de bases (défaillance de TOS, défaillance de NOS…).

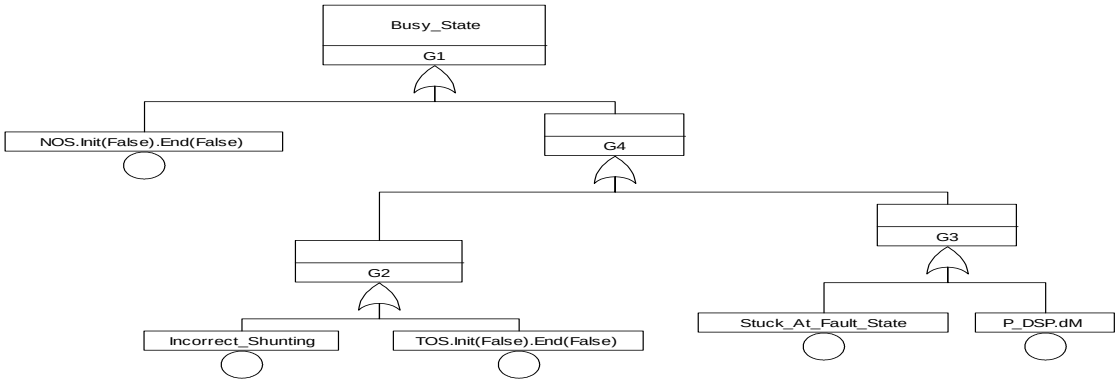

Figure 6 : Arbre de défaillance d'une combinaison d'erreur d'exécution de l'instruction DUP.

Les probabilités des éléments de base représentent les coefficients des matrices de transition sachant que le processus d'évolution de ces entités matérielles est un processus markovien non homogène.

III- Etude de programme de Tri :

- **Cas d'étude:**

La Co-conception matérielle/logicielle [11, 12] est un domaine de recherche récent et très actif, générant des méthodologies de conception efficaces. 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). L'application de notre approche sur un programme de Tri (figure 7), sur l'architecture d'un processeur à pile, a permis d'obtenir des résultats numériques, et de tirer quelques réflexions prouvant l'intérêt de l'approche dite flux informationnel.

Nous pouvons imaginer la structure globale du programme embarqué sous une forme modulaire. Chaque module (bloc) du programme est constitué par plusieurs fonctions (figure 8). Les fonctions sont construites aussi par des instructions assembleurs. L'étude de la fiabilité de la bonne exécution de programme sur une architecture matérielle du processeur commence d'abord par l'étude des instructions simple, le résultat sera exploiter premièrement dans l'évaluation des fonctions, puis après dans l'évaluation des blocs et finalement dans l'évaluation de toute l'application.

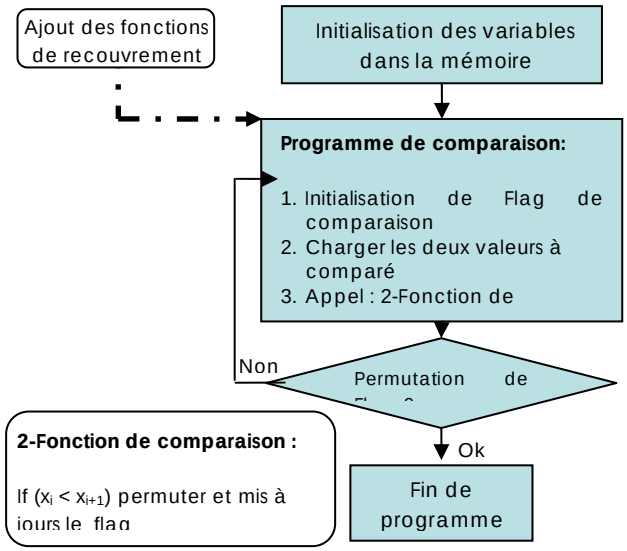

Figure 7 : Organigramme de l'application de Tri.

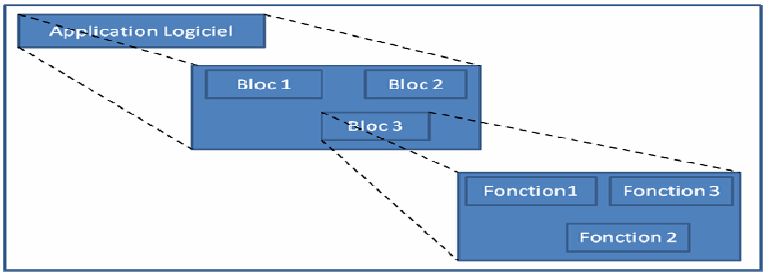

Figure 8 : Vue globale de la hiérarchie d'une application logicielle.

Pour calculer la probabilité totale du programme, après le tri de tous les variables, on se met dans le pire cas, c'est le cas le plus défavorable dans lequel tous les variables sont totalement désordonnés. Nous décomposons le programme global en blocs (sous-programme) afin de simplifier le calcul des probabilités. Par exemple le bloc d'initialisation traduit par le code assembleur suivant est présenté par l'arbre figure 9. Le dysfonctionnement de ce bloc de programme est principalement dû au dysfonctionnement d'initialisation des pointeurs, ou bien le dysfonctionnement d'initialisation des variables :

/*initialisation des pointeurs*/ DLIT 0x0090 D2R DLIT 0x00B1 DLIT 0x00A1	/*initialisation des mesures à l'entrée*/ DLIT 0x0001 --donnée i DLIT 0xe000 --adresse i STORE

La défaillance de ce code est dû à l'échec de l'exécution d'une des ses instructions. L'arbre de défaillance de ce bloc de programme est un ensemble d'éléments de base (tous les instructions). Chaque élément de base est aussi le sommet d'un arbre correspond à une instruction, sur l'arbre d'une instruction nous pouvons présenter les ressources matérielles permettant l'exécution de cette instruction et leurs combinaisons de défaillance (résultat de modèle de haut niveau et de bas niveau de l'instruction).

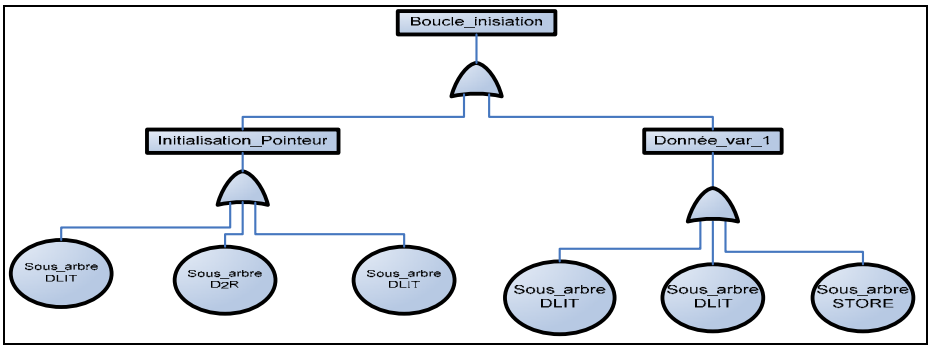

Figure 9 : Arbre de défaillance du bloc initialisation.

Le programme global de Tri est constitué de plusieurs sous-programmes (bloc d'instruction) concaténés. Nous supposons l'hypothèse qui consiste en l'exécution d'instructions d'une manière séquentielle, les unes après les autres. Nous rassemblons les sous-arbres qui représentent chaque bloc de programme. L'arbre de défaillance du programme global sera l'ensemble de tous ces arbres, il va prendre la forme :

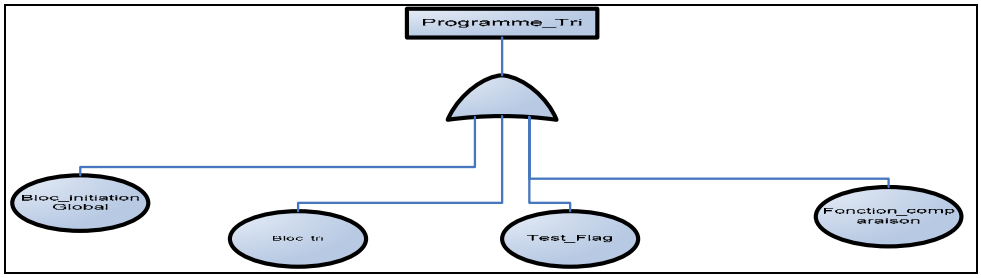

Figure 10 : Arbre global de défaillance du programme de tri.

Le sous-arbre 'Bloc_initiation_Global' est constitué de dix sous arbres, puisqu'il s'agit de tri de dix variables, pour chaque variable nous utilisons le bloc de code d'initiation au dessus qui est présenté par l'arbre de la figure 9. La défaillance du programme de tri est dû à la défaillance de l'un de ses bloc de programme, sachant que la défaillance d'un bloc de code est lié à la défaillance au moins d'une de ses instructions. Au début nous avons expliqué que la défaillance d'une instruction est une conséquence direct de la défaillance d'un compostant de l'architecture qui l'exécute ou bien d'un incident externe (exemple bit-flip).

Pour récapituler, la défaillance d'une ressource matérielle (défaillance matérielle dM, bit-flip, champs magnétique externe…) à pour conséquence la défaillance d'une instruction, toutes les combinaisons de défaillance se résume sur l'arbre de défaillance de l'instruction traité. Le sommet de cet arbre sera un élément de base de l'arbre de défaillance de bloc de programme qui contient cette instruction. La défaillance de cette instruction est une condition nécessaire et suffisante pour la défaillance du bloc de programme. Un bloc de programme est un ensemble d'instruction assembleur, la défaillance au moins d'une de ses instructions va déclencher la défaillance de tout le bloc. Les combinaisons possibles de la défaillance d'un bloc de code se résument dans son arbre de défaillance. Le sommet de l'arbre d'un bloc sera par la suite un élément de base pour l'arbre de programme entier. Sachant que le programme entier se compose de plusieurs blocs de sous-programme, la défaillance d'un seul bloc est une condition nécessaire est suffisante pour la défaillance du programme globale. Nous trouvons les résultats suivants :

- **Résultats :**

	Mode1	**Mode2**	**Mode3**	**Mode4**	**Mode5**	**Mode6**
Bloc_initiation	$4.894\ 10^{-1}$	0.0	$3.091\ 10^{-3}$	$77.3\ 10^{-6}$	$95\ 10^{-9}$	$20\ 10^{-5}$
Bloc_tri	$1.27\ 10^{-3}$	0.0	$10.5\ 10^{-3}$	$145\ 10^{-6}$	$591.4\ 10^{-9}$	$17.32\ 10^{-4}$
Test_Flag	$9.17\ 10^{-2}$	0.0	$4.59\ 10^{-3}$	$12.1\ 10^{-5}$	$55\ 10^{-10}$	$13.5\ 10^{-4}$
Fonc_Comparaison	$58.49\ 10^{-2}$	0.0	$7.9\ 10^{-4}$	$123\ 10^{-5}$	$280.1\ 10^{-11}$	$14.22\ 10^{-4}$
Programme_tri	**$7.49\ 10^{-2}$**	**0.0**	**$6.67\ 10^{-2}$**	**$128.45\ 10^{-6}$**	**$17.8\ 10^{-10}$**	**$63.04\ 10^{-4}$**

Tableau 2 : Valeurs de probabilités du programme de Tri sans prise en compte de fonctions de recouvrement.

IV- Programme de Tri avec prise en compte de tolérance aux fautes :

Considérons maintenant l'exemple du programme précédent de Tri des variables dans lequel nous injectons des fonctions de recouvrement. La stratégie de recouvrement se traduit par l'ajout des fonctions de sauvegarde et de restauration que nous allons modéliser par la suite. Ces fonctions ont pour objectif de minimiser l'erreur sur les données sensibles lors de l'exécution du programme sur l'architecture de processeur à pile. Parmi ces données critiques nous pouvons citer : pointeur pile de données (DSP), le sommet de la pile (TOS), l'élément au dessous de sommet de la pile (NOS)…
Comme le montre l'algorithme du programme (figure 7), l'appel du bloc de code de ces fonctions se fait avant chaque itération du programme de tri, afin de garder en mémoire les données jugées correctes de pointeurs. 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 d'avoir une défaillance d'exécution du 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 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 comme la montre l'arbre de défaillance de la figure 11.
Il est important de connaître à chaque instant les valeurs des probabilités des six modes fonctionnel/dysfonctionnel pour l'ensemble du programme [13]. Ceci peut se réaliser par l'exploitation des résultats relatifs aux instructions simples, ainsi que l'exploitation des

résultats relatifs aux blocs du programme. La probabilité de l'élément sommet (figure 11) est la probabilité globale du programme de tri, elle s'obtient par la connaissance des probabilités des éléments de base correspondent aux sommets des sous-arbres (Bloc_Tri, Bloc_initialisation_Global, Test_Flag…). La subdivision de l'arbre de défaillance global en sous-arbres est une manière pratique et efficace pour représenter les scenarii et les combinaisons possibles d'événement et le calcul de leurs probabilités. Le calcul de probabilité d'avoir un mode dysfonctionnel quelconque du programme, nécessite la connaissance de la probabilité de chaque élément de base (sous-arbre) de la figure 11. A condition que cette valeur de probabilité représente le même mode de dysfonctionnement. Ces valeurs sont déjà calculées de la même manière que la figure 9 et les résultats sont classés dans le tableau 3.

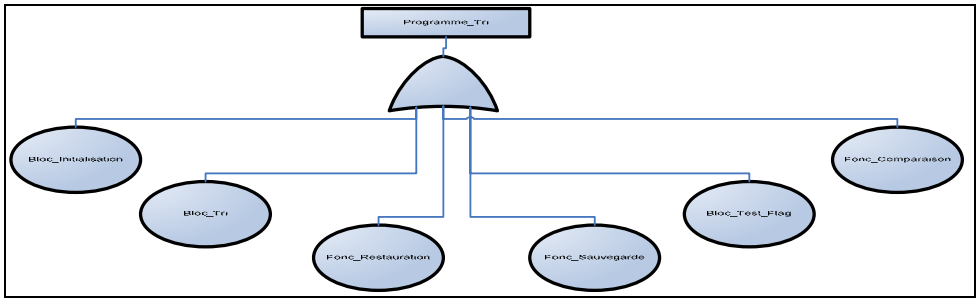

Figure 11 : Arbre de défaillance du programme de tri avec la stratégie de tolérance.

	Mode 1	**Mode 2**	**Mode 3**	**Mode 4**	**Mode 5**	**Mode 6**
Fonc_Sauvegarde	$7.44\ 10^{-2}$	$1.55\ 10^{-2}$	$2.2\ 10^{-4}$	$3.23\ 10^{-6}$	$7.45\ 10^{-9}$	$4.87\ 10^{-4}$
Fonc_restauration	$11\ 10^{-2}$	$0.33\ 10^{-2}$	$1.9\ 10^{-5}$	$54.3\ 10^{-6}$	$89.1\ 10^{-9}$	$6.77\ 10^{-4}$
Programme_tri	**$6.32\ 10^{-2}$**	**$1.57\ 10^{-2}$**	**$2.39\ 10^{-4}$**	**$55.8\ 10^{-5}$**	**$73\ 10^{-9}$**	**$105.7\ 10^{-4}$**

Tableau 3 : Valeurs de probabilités relatives au programme de Tri avec prise en compte de fonctions de recouvrement.

Afin de constater l'effet de l'ajout des fonctions de recouvrement, nous allons comparer les résultats des tableaux 2 et 3.

	Mode 1	**Mode 2**	**Mode 3**	**Mode 4**	**Mode 5**	**Mode 6**
Programme_tri (Sans prise en compte de tolérance)	**$7.49\ 10^{-2}$**	**0.0**	**$6.67\ 10^{-2}$**	**$128.45\ 10^{-6}$**	**$17.8\ 10^{-10}$**	**$63.04\ 10^{-4}$**
Programme_tri (Avec prise en compte de tolérance)	**$6.32\ 10^{-2}$**	**$1.57\ 10^{-2}$**	**$2.39\ 10^{-4}$**	**$55.8\ 10^{-5}$**	**$73\ 10^{-9}$**	**$105.7\ 10^{-4}$**

Tableau 4 : Résultats de comparaison de programme de Tri dans le cas de tolérance et sans tolérance.

V- Discutions des résultats numériques :

L'application de cette approche sur un programme de Tri, en exécution sur l'architecture d'un processeur à pile, a permis d'une part d'obtenir des résultats numériques, et d'autre part de tirer quelques réflexions sur l'intérêt de l'approche décrite. D'une manière générale, l'appel périodique des fonctions de sauvegarde et de restauration nécessite plus de sollicitation des composants matériels. D'après le Tableau 4, nous constatons une baisse de probabilité d'avoir le Mode 1, dû à la sollicitation excessive de ressources matérielles par l'appel de recouvrement. Le Mode 2 (défaillance sûr disponible) et le Mode 3 (défaillance sûr non disponible) sont des modes non bloquant du système, le gain au niveau de leur probabilité (cent fois) est un résultat positif de la stratégie de tolérance en question. Le Mode 4

(défaillance dangereux) et le Mode 5 (arrêt intempestif) sont des modes bloquant du système, le gain au niveau de leur probabilité est un résultat négatif de la stratégie de recouvrement.
Les fonctions supplémentaires de recouvrement, même si leur pouvoir de corriger certains erreurs sur le système (Mode 2 et 3), elles ont la tendance d'augmenter la probabilité d'avoir plus de bit-flip, ce qui est expliquer par l'augmentation des probabilités (Mode 4 et 5). A ce stade, il est légitime de poser la question autour de l'intérêt d'utiliser la présente stratégie de recouvrement, et est ce que nous avons fait le bon choix….

Bibliographie

[1] H. Belhadaoui, M. Jallouli, B. Dubois, O. Malassé, K. Hamidi, V. Idasiak, J-B. Kammerer, L.Hébrard, F. Monteiro, C. Diou, M. Hehn, J-F. Aubry, H. Medromi, F. Braun, A. Dandache, S. Piestrak and B. Lepley "Instrumentation sûre de fonctionnement - Une synergie multidisciplinaire", 4ème Colloque Interdisciplinaire en Instrumentation (C2I 2007), Octobre 2007.

[2] 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.

[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] N.Wirk, Hardware/Software co-design then and now, Information Processing Letters 88 (2003) 83-87.

[5] Wayne Wolf, "Hardware and Software Co-design". High-Performance Embedded Computing, 2007, pp. 383-432.

[6] 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.

[7] A.Li, B.Hong, "Software implemented transient fault detection in space computer", Aerospace Science and Technology 11, 2007 pp. 245–252.

[8] 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.

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

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

[11] A. Shaout and T. Eldos, "On the classification of computer architecture", International Journal of Science and Technology, vol. 14, Summer 2003.

[12] Philip J. Koopman, Jr., "Stack Computers: The New Wave", California : Ed. Mountain View Press, 1989; http://www.ece.cmu.edu/~koopman/stack_computers/.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

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

[19] M.Yuchang, L.Hongwei, Y.Xiaozong, "Efficient Fault Tree Analysis of Complex Fault Tolerant Multiple-Phased Systems", Tsinghua Science and Technology, ISSN 1007-0214 22/49 vol. 12, No. S1, July 2007 pp.122-127

[20] M.Yuchang, L.Hongwei, Y.Xiaozong, "Efficient Fault Tree Analysis of Complex Fault Tolerant Multiple-Phased Systems", Tsinghua Science and Technology, ISSN 1007-0214 22/49 vol. 12, No. S1, July 2007 pp.122-127

[21] 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.