Thèse

présentée par

Hervé BERVILLER

pour l'obtention du titre de

Docteur de l'Université de Metz

en Micro-électronique

Amélioration de la sûreté de fonctionnement des systèmes à commandes unidirectionnelles basée sur la conception d'interfaces en ASIC

Soutenue publiquement le 21 septembre 1998 devant la commission d'examen :

Composition du Jury :

Président	Michel ROBERT	Professeur, Université de Montpellier II
Rapporteurs	Mihail NICOLAIDIS	Directeur de recherches CNRS (INPG, Grenoble)
	Stanislaw J. PIESTRAK	Professeur, Université de Wroclaw
Directeur	Bernard LEPLEY	Professeur, Université de Metz
Examinateurs	Francis BRAUN	Professeur, Université Louis Pasteur de Strasbourg
	Abbas DANDACHE	Maître de conférence, Université de Metz
	Fabrice MONTEIRO	Maître de conférence, Université de Metz

sein du

licro-électronique- CLOES

Thèse

présentée par

Hervé BERVILLER

pour l'obtention du titre de

Docteur de l'Université de Metz

en Micro-électronique

Amélioration de la sûreté de fonctionnement des systèmes à commandes unidirectionnelles basée sur la conception d'interfaces en ASIC

Soutenue publiquement le 21 septembre 1998 devant la commission d'examen :

Composition du Jury :

Président	Michel ROBERT	Professeur, Université de Montpellier II
Rapporteurs	Mihail NICOLAIDIS	Directeur de recherches CNRS (INPG, Grenoble)
	Stanislaw J. PIESTRAK	Professeur, Université de Wroclaw
Directeur	Bernard LEPLEY	Professeur, Université de Metz
Examinateurs	Francis BRAUN	Professeur, Université Louis Pasteur de Strasbourg
	Abbas DANDACHE	Maître de conférence, Université de Metz
	Fabrice MONTEIRO	Maître de conférence, Université de Metz

Thèse préparée au sein du

Laboratoire Interfaces Composants Micro-électronique- CLOES

Remerciements

Je tiens avant tout à remercier :

Monsieur Bernard LEPLEY, Directeur du Laboratoire LICM/CLOES pour m'avoir accueilli au sein de son laboratoire et dirigé mes travaux. Qu'il trouve dans ces quelques lignes ma profonde sympathie et toute ma reconnaissance.

Monsieur Michel ROBERT, Professeur à l'Université de Montpellier II, pour m'avoir fait l'honneur de présider le jury de cette thèse.

Monsieur Mihail NICOLAIDIS, Directeur de recherches au CNRS à l'INPG de Grenoble, et Monsieur Stanislaw J. PIESTRAK, Professeur à l'Université de Wroclaw, pour avoir accepté d'être rapporteur de ce travail, ainsi que pour leurs conseils avisés.

Monsieur Francis BRAUN, Professeur à l'Université Louis Pasteur de Strasbourg, pour avoir accepté d'être membre de ce jury.

Monsieur Abbas DANDACHE et Monsieur Fabrice MONTEIRO pour avoir encadré mes travaux ainsi que pour leur soutien moral et logistique.

Enfin, je n'oublierai pas toutes les personnes, membres du Laboratoire, membre de ma famille ou amis, qui par leurs actions, gestes, paroles ou écoutes m'ont soutenu tout au long de cette thèse. Je citerai tout particulièrement (et par ordre alphabétique !) : Angélique, Carole, Cathia, Gaëlle, Isabelle, Jean-François, Jean-Philippe, Laurent, Maman, Papa, Régis, Rémy, Serge, Stéphane, Thierry, Yves.

Special Thanks to Isabelle and Gaëlle

Je dédie ce travail à toutes ces personnes, mais également aux personnes de ma famille trop tôt disparues, notamment Lucien mon parrain, et enfin à une personne qui se reconnaîtra et qui a partagé les bons (sans compter ! ! !) et les mauvais moments (avec plus de modération ! ! !) de la dernière année.

Références Bibliographiques

Références

[1] S.J. PIESTRAK, « Design of self-testing checkers for unidirectional error detecting codes ».
Scientific Papers of the Institute if Technical Cybernetics of the Technical University of Wroclaw, n°92. Monograph n°24, 1995.

[2] D.P. SIEWIOREK and R.S. SWARZ, « Reliable Computers systems: Design and Evaluation »
2nd Ed., Digital Press, Bedford, March 1992.

[3] F.L. VARGAS « Amélioration de la sûreté de fonctionnement de systèmes spatiaux basée sur le contrôle de courant »
Thèse de doctorat, INP Grenoble, Mai 1995.

[4] G. KONEMANN, J. MUCHA, G. ZWIEHOFF, « Built-in logic bloc observation techniques » 1979 International Test Conference, October 1979.

[5] M.NICOLAIDIS, Y.ZORIAN, «On-line testing for VLSI - A compendium of approaches»
Journal of Electronic Testing, Theory and Applications (JETTA), Volume 2, N° 1 / 2, February / April 1998, pp 7-20.

[6] D.A. ANDERSON, « Design of self-checking digital networks using coding techniques »
Urbana, CSL Univ. Of Illinois, September 1971 (Report 527).

[7] S.M. THATTE, J.A. ABRAHAM, « A methodology for functional level testing of microprocessors »
8th Int. Symp. on Fault-Tolerant Comp., Toulouse, June 1978.

[8] B. COURTOIS, « Test et LSI »
Thèse d'état, USM Grenoble/INP Grenoble, Juin 1981.

[9] N. BENOWITZ et al. , « An advanced fault isolation system for digital logic »
IEEE Transactions on Computers, vol. C-24, n°5, May 1975.

[10] R. DAVID, P. THEVENOD-FOSSE, «Panorama des méthodes de test non déterministes des circuits logiques»
RAIRO, Vol 13, N°1, 1979.

[11] M. WILLIAMS, J. ANGEL, « Enhancing testability of LSI circuits via test points and additional logic »
IEEE Trans On Comp., Vol C-22, n°1, January 1973

[12] S. FUNATSU, N. WAKATSUKI, T. ARIMA, « Test generation in Japan »
14[2h] Design Automation Conference, June 1975.

[13] H. FUJIWARA, « Logic testing and design for testability »
MIT Press series in computers systems, Cambridge 1985.

[14] E.B. EICHELBERGER, T.W. WILLIAMS, « A logic design structure for LSI testability »
14[th] Design Automation Conference, June 1977.

[15] « Standard Test Access Port an Boundary-Scan Architecture »
Document P1149.1, June 20, 1989.

[16] J.C LAPRIE, « Sûreté de fonctionnement des systèmes informatiques (matérels et logiciels) »
AFCET, Dunod Informatique, Mars 1989.

[17] M. NICOLAIDIS, S. NORAZ and B. COURTOIS, « A generalised theory of fail safe systems »
19[th] Int. Symp. On Fault-Tolerant Comp., Chicago, June 1989.

[18] B. COURTOIS, « Failure mechanism, fault hypotheses and analytical testing of LSI-NMOS (HMOS) circuits »
VLSI 81, Univ. of Edinburgh, August 1981.

[19] J.M. BERGER, « A note on error detection codes for asymmetric channels »
Information and Control, New York, March 1961.

[20] G.G. LANGDON, C.K. TANG, « Concurrent error detection for group look-ahead Binary Adders »
IBM J. Res. Develop., pp.563-573, September 1970.

[21] W.W. PETERSON, E.J. WELDON, « Error Correcting Codes »
2[nd] Ed., The MIT press; Cambridge, Massachusetts, 1972.

[22] D.T. BROWN, « Error detecting and correcting binary codes for arithmetic operations »
IRE Trans. On Elec. Comp., September 1960, pp. 333-337.

[23] W.W. PETERSON, « On checking an adder »
IBM J. Res. And Dev, Vol.2, pp.166-168, April 1958.

[24] D.S. HENDERSON, « Residue class error checking codes »
Proc. 16[th] Nat. Meeting of the ACM, Los Angeles, Ca., September 1961.

[25] C.V. FREIMAN, « Optimal error detection codes for completely asymetric binary channels »
Inform. Contr., vol 5, pp. 64-71, March 1962.

[25] Y.H. CHUANG, S. DAS, « An approach to the design of highly reliable and Fail Safe digital systems »
National Computer Conference 1974.

[27] M. NICOLAIDIS, « Conception de circuits intégrés auto-contrôlables pour des hypothèses de pannes analytiques »
Thèse de Docteur-Ingénieur, INP Grenoble, Janvier 1984.

[28] J.E. SMITH, G. METZE, « Strongly fault secure logic networks »
IEEE Trans. On Comp., Vol C-27, June 1978.

[29] W.C. CARTER, P.R. SCHNEIDER, « Design of dynamically checked computers »
IFIPS Congress 1968, Edimbourg.

[30] M. NICOLAIDIS, « Fail Safe Interfaces for VLSI :Theoretical Foundations and Implementation »
IEEE Trans. On Comp., Vol.47 N°1, January 1998.

[31] H. MINE, and Y. KOGA, « Basic properties and a construction method for fail safe logical systems »
IEEE Trans. Elec. Comp., Vol. EC-16, June 1967.

[32] T. TAKAOKA, H. MINE, « N fail Safe logical systems »
IEEE Trans. On Comp. Vol C-20, May 1971.

[33] A. DANDACHE, H. BERVILLER, F. MONTEIRO, B. LEPLEY « Feasibility of Fail Safe compliant ASICs »
Proceeding from 2nd Archimedes Open Workshop on Synthesis of Testable Circuits (STC'95), Barcelona, February 1995

[34] A. DANDACHE, H. BERVILLER, F. MONTEIRO, B. LEPLEY, « Outil d'aide à la conception de circuits ASICs pour les systèmes à haute sécurité de fonctionnement »
Proceeding from MCEA 95, Grenoble, France, septembre 1995

[35] A. DANDACHE, H. BERVILLER, F. MONTEIRO, B. LEPLEY, « CAD of Fail Safe compliant ASICs »
Proceeding of the 1st IEEE International On-Line Testing Workshop, St-Nice, France, July 1995.

[36] H. BERVILLER, A. DANDACHE, F. MONTEIRO, B. LEPLEY, « Using FPGAs for the implementation of Fail Safe interfaces »
Proceeding of the 2nd IEEE International On-Line Testing Workshop, St-Jean de Luz, France, July 1996.

[37] F.MONTEIRO, H. BERVILLER, A. DANDACHE, B. LEPLEY, « Reliable applications using a new interface based on hybrid (static and dynamic) coding techniques »
Proceeding from the IEEE European Test Workshop, Cagliari, Italy, may 1997

[38] F.MONTEIRO, H.BERVILLER, A. DANDACHE, B. LEPLEY, « Secure applications using a new interface based on hybrid (static and dynamic) coding techniques »
Proceeding of the 3rd IEEE International On-Line Testing Workshop, Agia Pelaglia, Crete-Greece, July 1997.

[39] F.MONTEIRO, A. DANDACHE, H.BERVILLER, B.LEPLEY, « A secure interface based on BIST and Hybrid coding techniques »
Proceeding of the 4th IEEE International On-Line Testing Workshop, Capri, Italy, July 1998.

Introduction

Introduction

Les domaines dans lesquels l'homme a été remplacé par la machine n'ont cessé de s'étendre. Cette évolution, signe du progrès, a d'abord visé à soulager l'homme des tâches les plus ingrates, puis à améliorer son confort. Le domaine dans lequel le choix de l'automatisation a pu être délicat à accepter est celui mettant en jeu des vies humaines.

Jusqu'où doit-on, en effet, pousser l'automatisation, dans les domaines que sont, par exemple, le transport et les industries chimiques ou nucléaires, pour lesquels la moindre erreur d'appréciation peut engendrer une situation catastrophique. La machine n'est certes pas vulnérable à des problèmes d'ordre psychologique, ne cède pas à la panique, mais peut, tout comme l'être humain subir des défaillances aussi subites qu'imprévisibles. Lequel de l'homme ou de la machine est le plus fiable ? Le dilemme semble avoir été tranché puisque, depuis nombre d'années, des systèmes automatiques ont la lourde tâche d'assurer directement ou non la sécurité des personnes.

Cette solution n'étant viable que si la probabilité de défaillance de l'homme est plus importante que celle de la machine, ces systèmes ont été réalisés, sur le plan technique, avec des éléments dont la fiabilité ou la robustesse n'est plus à démontrer.

Dans le domaine de l'électronique, qui nous concerne plus particulièrement dans ce travail, les principaux systèmes exigeant une très forte sécurité de fonctionnement, utilisent des composants discrets ayant une probabilité de défaillance quasi nulle. Cette solution, qui a fait ses preuves bride cependant la complexité et l'évolution de ces systèmes.

La micro-électronique s'étant parallèlement développé de façon spectaculaire, son utilisation dans des systèmes critiques a été étudiée.

L'utilisation de circuits intégrés spécifiques permettrait non seulement de réduire considérablement la taille de ces systèmes, mais permettrait également de profiter de la formidable puissance de calcul pour réaliser des systèmes bien plus complexes et moins coûteux.

Les systèmes sûrs tels qu'ils ont été conçus initialement ne peuvent pas êtres intégrés en utilisant la même méthodologie.

Les systèmes utilisant des composants discrets spécifiques étaient en effet conçus en tenant compte du modèle de défaillance de chaque composant et de sa fiabilité. Cette méthode n'est pas applicable aux circuits intégrés VLSI car les modèles de pannes sont bien plus complexes et les états logiques de sortie provoqués par la manifestation d'une défaillance ne sont pas maîtrisable à la fabrication.

En outre, la complexité de certains circuits intégrés est devenue telle qu'il n'est aujourd'hui plus possible de tester complètement un circuit après fabrication.

Les progrès réalisés dans le domaine de la micro-électronique ont donc permis de disposer d'une puissance de calcul suffisante, mais cela s'est fait au détriment de la sécurité intrinsèque.

La seule possibilité d'exploiter les microprocesseurs dans les systèmes critiques est donc d'assurer que l'apparition d'une panne dangereuse ne survienne qu'avec une probabilité telle qu'une catastrophe puisse être raisonnablement jugée improbable.

La complexité des circuits rendant impossible leur test complet après fabrication, il a donc fallut prévoir le test dès leur conception.

La solution consiste donc à intégrer dans le circuit tous les mécanismes nécessaires à son propre test, c'est ce que l'on appelle l'auto-test intégré.

Les deux catégories de test intégré que l'on peut distinguer sont le test hors-ligne et le test en-ligne.

Comme l'évoque son nom, le test hors-ligne ne se fait pas pendant le déroulement de l'application, il exige donc une interruption du fonctionnement. Une de ces techniques, connue sous le nom de BIST (Built In Self Test), consiste à incorporer dans le circuit des générateurs de vecteur de test à l'entrée et des analyseurs de signature pour interpréter les réponses à la sortie.

Ces techniques simplifient énormément le test des circuits complexes. Le temps d'application des séquences est court, les circuits sont contrôlés à vitesse d'utilisation, la phase d'élaboration des vecteurs de test est évitée par l'emploi de séquence pseudo-aléatoires, et la lecture des registres n'est pas coûteuse en temps.

L'augmentation en surface, due à l'emploi de registres spécifiques, peut toutefois ne pas être négligeable.

La technique BIST concerne généralement le test hors ligne des circuits. Elle peut, cependant, également être utilisée pour faire un test en-ligne discontinu en activant

périodiquement les séquences de test nécessaires. La détection de la panne est alors réalisée avec un retard par rapport à son apparition qui est liée à la périodicité du test.

Ces solutions ne sont donc pas satisfaisantes pour les applications qui exigent une sûreté de fonctionnement importante. Il est en effet, dans ce cas, indispensable d'assurer immédiatement la détection des erreurs causées par une panne dans le système, afin d'intervenir rapidement, et d'éviter une évolution catastrophique de la situation.

C'est pour répondre à ce besoin qu'ont été conçus les circuits auto-contrôlables.

Le but de l'autotest en-ligne est de détecter immédiatement l'apparition d'une panne dans un circuit au cours de son fonctionnement.

Ce type de test en-ligne, concurrent avec le fonctionnement de l'application, peut toujours être assuré par codage logiciel. Cependant cette technique conduit généralement à une dégradation des performances et, surtout, couvre des modèles de défaillances à un niveau d'abstraction éloigné des défaillances réelles des circuits intégrés.

L'autre solution consiste donc à prendre en compte le test en-ligne dès la conception des circuits. Cette technique, basée sur la réalisation de circuits auto-contrôlables (*Self Checking*) présente l'avantage de couvrir des modes de défaillances réels qui sont bien connus actuellement. Cette approche ne nécessite aucun logiciel pour assurer le test en-ligne, la détection des erreurs se faisant par le matériel.

La conception d'un circuit auto-contrôlable consiste donc en un ajout de matériel pour que le circuit en fonctionnement soit capable de détecter toute apparition de panne. Cette conception est basée sur des techniques de codage spécifiques et sur les propriétés mathématiques des différents blocs qui constituent la fonction.

Toutefois, le test en-ligne d'un circuit auto-contrôlable ne peut être effectif que si certaines conditions restrictives sont satisfaites. Etant donné qu'il n'est pas toujours facile, en pratique, de vérifier ces conditions, une solution consiste à combiner le test hors-ligne et les mécanismes *Self Checking* pour le test en-ligne, afin d'obtenir une stratégie pour le test unifié de circuits. L'unification de ces deux types de test mène à la technique nommée UBIST (*Unified* BIST), qui peut être appliquée indistinctement aux tests de conception, de fabrication, de maintenance et au test accompli en cours de fonctionnement.

Bien que ces circuits auto-contrôlables soient en constante évolution, du fait de la complexité des fonctions intégrées sur une puce, ils ne peuvent malheureusement pas se substituer aux circuits *Fail Safe* conventionnels.

Les systèmes *Fail Safe* ont été conçus pour que toute sortie erronée résultant d'une panne interne ne donne pas lieu à un événement catastrophique. Ils sont utilisés pour le pilotage d'éléments électromécaniques qui se trouvent dans de nombreux systèmes de surveillance des industries chimiques, nucléaires ou des transports, où les normes de sécurités sont draconiennes. Les systèmes critiques sont en effet tous réalisés suivant le même schéma de principe qui se décompose en trois parties, une logique de traitement des données, un étage de sortie et les fonctions sécuritaires de commande (par exemple, l'utilisation de la technique de codage en fréquence).

Les gros problèmes liés à la sécurité se trouvant localisés au niveau de l'étage de sortie : c'est ici qu'interviennent les systèmes *Fail Safe*. Il n'y a en effet, à ce niveau, plus aucun retour d'information qui permette de vérifier que la commande désirée soit parvenue sans erreur à l'actionneur. Les systèmes *Fail Safe* ont donc été conçus de telle sorte qu'en présence d'une panne, la sortie du système prenne soit la valeur correcte, soit une valeur dite *sûre* qui ne peut jamais entraîner de situation dangereuse.

Des travaux récents ont abouti à la combinaison des circuits *Self Checking* pour le calcul et des interfaces *Fail Safe* responsable du contrôle des actionneurs de manière à obtenir que l'ensemble du système critique soit entièrement *Fail Safe*. C'est dans cette voie que nous nous sommes engagés pour cette étude.

Le premier chapitre présente une synthèse de toutes les techniques de test utilisées pour le contrôle des circuits intégrés. Après un bref rappel sur l'intérêt de la « testabilité », et donc de son rôle lors des différentes étapes de vérification d'un circuit intégré, les différentes approches du test des systèmes intégrés sont présentées. La deuxième partie du chapitre présente l'inventaire des techniques de test en-ligne dédiées aux applications de sécurité, qui concernent directement le sujet de cette étude. L'apport des codes détecteurs d'erreur pour ce type d'application est précisé, en insistant sur les codes les mieux adaptés. Cette partie se poursuit par la présentation des circuits auto-contrôlables qui constituent un des moyens de parvenir à la réalisation des circuits *Fail Safe* définis en fin de chapitre.

Le chapitre II présente les travaux ayant aboutis à la réalisation d'une interface *Fail Safe* en circuits VLSI. Le principe de l'interface étudiée est exposé après avoir repositionné les

définitions des systèmes *Fail Safe* dans le contexte de notre étude. Ce chapitre s'attache à analyser le fonctionnement de l'interface étudiée en mettant en lumière la rigueur de conception requise pour l'élaboration de ce type de circuit. Les contraintes d'implémentation liées à la réalisation physique de l'interface sont alors détaillées ainsi que les résultats de simulations. La dernière partie présente les caractéristiques des différentes bibliothèques développées, qui permettent la génération semi-automatique de l'interface entière.

Pour envisager la vulgarisation de ce type de circuits, leur implémentation en circuits programmables ne peut être ignorée. C'est dans le cadre de cette réflexion que s'inscrit le troisième chapitre, en précisant les limites de l'interface présentée au chapitre précédent. Les avantages de ce type de support sont rappelés, ainsi que les problèmes qu'ils posent pour une application *Fail Safe*. Les modifications à apporter à l'interface ainsi que les contraintes à respecter sont ainsi analysées.

La dépendance à un type de support, un type d'outil, ou à une technologie allant à contre courant de la tendance actuelle, le chapitre IV étudie la conception d'interfaces fortement sûres, indépendantes d'une technologie. Ces interfaces, définies à un niveau d'abstraction élevé peuvent être simulées ou réalisées à l'aide d'outils logiciels travaillant à partir d'une description VHDL. Après un rappel des avantages d'une telle démarche, le principe utilisé pour la réalisation de ces interfaces est présenté. Le fonctionnement d'une première interface est alors exposé, ainsi que le calcul de son taux de sûreté. Cette première partie se termine par l'analyse des possibilités d'évolution vers une interface *Fail Safe*. La seconde partie du chapitre présente une seconde interface permettant d'améliorer sensiblement les performances de l'interface précédente. Les possibilités d'évolution vers la propriété *Fail Safe* sont également étudiées. Le concept de cette interface est enfin validé par la simulation de son implémentation sur un circuit FPGA.

Chapitre I :

Généralités sur le test

Chapitre I : Généralités sur le test

1. Introduction :

La formidable évolution de la micro-électronique ces quinze dernières années, notamment en ce qui concerne la densité d'intégration, a conduit à la commercialisation de circuits regroupant plusieurs millions de transistors. Ces progrès ont également permis d'améliorer la fiabilité de ces composants élémentaires.

Cependant, au regard de la complexité des circuits actuels, et donc du nombre de composants internes, la fiabilité du circuit n'a pu augmenter dans les mêmes proportions. C'est pourquoi la probabilité de défaillance d'un circuit complexe ne peut être négligée.

Cet aspect a tout d'abord été pris en compte dans un souci économique. Il est en effet important de pouvoir détecter le plus rapidement possible un composant défectueux. Plus cette détection est tardive, plus son coût sera important, aussi bien financièrement qu'en terme d'image et de savoir faire.

La prise en compte de ces facteurs est à l'origine du développement de la testabilité. La prise en compte de la testabilité se fait à toutes les étapes de la fabrication d'un produit. Il est donc important de la prendre en compte dès la conception, c'est la «conception en vue du test». Cette conception s'appuie sur un grand nombre de techniques qui répondent chacune pl

us particulièrement à un besoin particulier.

Au fil des années, étant donné le nombre et la diversité des applications faisant appel aux circuits VLSI, le principe de la testabilité a été étendu à la conception de systèmes nécessitant une grande sécurité de fonctionnement. Quelques exemples de dysfonctionnement de systèmes automatisés faisant appel à ce genre de circuits ont été recensés dans **[1]**.

- 15/01/90 : Panne du réseau longue distance de AT&T aux USA pendant 9 heures.
- 02/90 : Crash d'un Airbus A320 « Fly by Wire »en Inde.
- 17/09/91 : Panne du réseau de télécommunication de New York pendant 7h30
- 26/06/93 : Panne pendant 30 heures du réseau de carte bancaires en France affectant 40% des 21 millions de possesseurs de carte bancaires pendant tout le week-end.

Lors de la conception de tels systèmes, il est donc crucial de prévenir la situation paralysante en détectant immédiatement l'apparition d'une panne et en déclenchant les mécanismes de substitution ou de maintenance.

Etant donné la complexité de ces systèmes et la diversité des mécanismes de panne, seul le test intégré, et plus particulièrement le test en-ligne, permet de résoudre le problème.

Les circuits auto-contrôlables, généralement basés sur l'utilisation de codes détecteurs d'erreur, ont permis de répondre à ce besoin. Ces circuits sont particulièrement adaptés aux circuits modernes puisque 90% des pannes affectant un système pendant son fonctionnement sont temporaires **[2]**, et donc difficiles à détecter par l'application de tests périodiques.

Ces circuits sont donc indispensables à la réalisation de systèmes intégrés à défaillance sûre (*Fail Safe*), mais ne sont pas suffisants. Les systèmes *Fail Safe* garantissent qu'en cas de panne, le système délivre soit la valeur correcte, soit une valeur sûre qui n'engendrera pas de situation dangereuse.

Bien que n'étant pas mentionnée dans la suite, la technique de détection de panne par contrôle de courant (Iddq) pourrait contribuer à l'amélioration de la couverture de panne. Cette technique utilise des détecteurs de courant (internes ou externes) des alimentations. La maîtrise de cette technique permet la détection de modèles de pannes non décelables par les méthodes classiques. L'utilisation de cette technique, pour l'amélioration de la sûreté de fonctionnement de systèmes spatiaux, a été étudiée dans **[3]**. Notons toutefois que l'utilisation de cette technique ne peut se faire qu'en complément des techniques énoncées précédemment.

Ce chapitre s'attache à présenter l'historique du test en rappelant l'apport de la testabilité puis en introduisant la conception en vue du test.

La deuxième partie traite du test en-ligne dédié aux applications de sécurité, en présentant les principaux codes détecteurs d'erreurs, les circuits auto-contrôlables, et enfin les systèmes *Fail Safe*.

2. Historique du test

2.1. La testabilité

2.1.1. Rôle

L'utilisation des derniers progrès, en matière de puissance de traitement et de complexité des fonctions intégrées sur une puce, devient une nécessité pour répondre aux cahiers des charges des nouvelles applications. A cette complexité, il convient désormais d'ajouter une contrainte de sécurité, dont le degré peut différer selon le type d'application auquel le produit est destiné. Ces paramètres rendent le problème de la testabilité de plus en plus critique. Les produits conçus de nos jours sont donc toujours plus complexes mais également soumis à une concurrence de plus en plus sévère. Les fabricants cherchent donc à diminuer les coûts de fabrication tout en fournissant des produits à la pointe du progrès. Ils doivent cependant veiller à ce que cette baisse des coûts n'altère en rien la qualité et la fiabilité des produits.

Ce souci présent dans quasiment tous les domaines d'activité est encore plus présent dans le secteur de l'électronique intégré (fabrication des microprocesseurs pour ce qui concerne l'usage grand public). Un des moyens de parvenir à maîtriser la qualité, tout en améliorant la productivité mais sans altérer la confiance à porter au produit fabriqué, est d'améliorer la testabilité. La testabilité est donc l'aptitude d'une conception à favoriser la détection de défauts.

Compte tenu de l'explosion du marché des circuits intégrés et des perspectives technologiques à court terme, les fabricants de ces circuits mettent de plus en plus l'accent sur le test. Cette stratégie est motivée par les points suivants :

- Complexité croissante des circuits (jusqu'à 9 millions de transistors en 1996, 30 millions en l'an 2000),
- Augmentation des fréquences d'horloge (jusqu'à 300 Mhz en 1996, plus de 1Ghz en l'an 2000),
- Mixité logique/analogique de certains circuits (télécommunications, GSM ....).

2.1.2. Principe

La testabilité est un ensemble de techniques visant à surveiller et à améliorer le fonctionnement d'un système, en fonction du degré de sécurité exigé afin que ce système remplisse correctement la mission pour laquelle il a été conçu.

La manifestation d'une erreur sur le service délivré par le système conduit à une défaillance. Cette erreur peut être la conséquence d'une faute interne (défaillance physique, faute de conception) ou externe (faute humaine d'interaction ou environnement physique). La testabilité d'un système est donc basée sur la qualité de sa structure.

En fonction des objectifs économiques et techniques du produit à fabriquer, la testabilité peut être considérée comme la prise en compte, à la conception, des moyens qui vont permettre l'exécution et l'optimisation des phases de test et de diagnostic afin que celles-ci soient financièrement les plus rentables en délai, coût et qualité.

Sa mise en œuvre concerne aussi bien les concepteurs que les gens du test ou les utilisateurs du produit. Cette concertation entre tous les protagonistes est d'autant plus nécessaire que la testabilité ne doit pas être une préoccupation à un moment donné, mais elle doit être un souci permanent dans le cycle de vie du produit, depuis sa conception jusqu'à sa maintenance.

2.1.3. Intérêt

Chaque étape de la réalisation d'un système peut introduire des défauts. Alors au fur et à mesure de la progression dans le processus de fabrication (composant, carte, système, sur site), le coût de détection et de localisation d'une panne croit en amplitude d'un facteur 10 (figure I.1). Le moment le plus opportun pour influencer les coûts du test se situe donc au plus bas niveau de la fabrication, c'est à dire au niveau du composant. Même à ce niveau, compte tenu des augmentations de densité d'intégration et de la complexité croissante des composants VLSI, il est important de disposer de très bons moyens de test.

La prise en compte de la testabilité dès la conception du produit permettra :

- D'abaisser les coûts de production en détectant plus rapidement les produits défectueux, mais également les coûts de maintenance en prévoyant cette phase dès la conception,
- D'augmenter la qualité du produit en minimisant les risques de défaillance,

•D'améliorer la compétitivité du produit en diminuant les délais de production et les coûts.

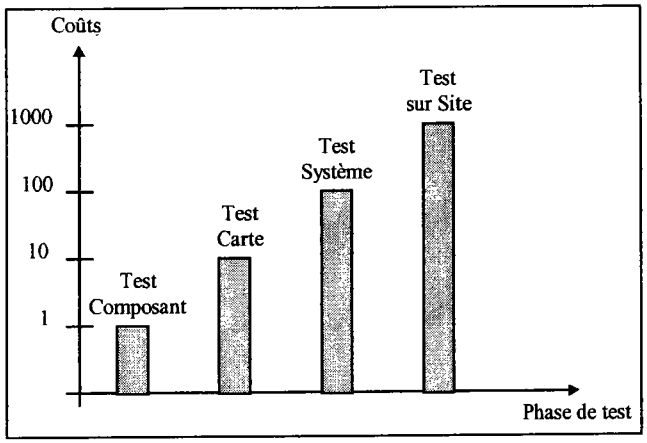


Figure I.1 : Economie de la testabilité

2.1.4. Stade d'application

Pour le choix d'un dispositif de test, les paramètres importants à prendre en compte sont la qualité de détection désirée, le coût du test ainsi que sa rapidité d'exécution. La qualité de détection dépend de la connaissance des pannes possibles. Le coût (temps de conception, prix d'un testeur ou surface de test supplémentaire) et la rapidité sont liés à l'approche de conception choisie. La testabilité peut être de deux types :

2.1.4.1. La testabilité extrinsèque :

Elle concerne l'environnement de test du produit et à ce titre elle est fonction des quantités.

2.1.4.2. La testabilité intrinsèque :

Elle se rapporte à tout ce qui est implanté dans le produit. C'est sur ce type de testabilité que se porte notre intérêt et plus spécifiquement sur les moyens à mettre en oeuvre au niveau composant puisque c'est à ce niveau que le coût est le moins important.

La prise en compte de la testabilité d'un composant fait appel aux notions de contrôlabilité et d'observabilité. La prise en compte de ces deux facteurs dès la conception, permet donc d'optimiser la testabilité. Enfin, la testabilité peut se concevoir à différents niveaux d'intégration :

- Passive : il y a un apport de composants simples hors du chemin fonctionnel (points de test),

- Active : il y a un apport de circuits actifs dans le chemin fonctionnel,

- Structurée : elle concerne les architectures élaborées en vue du test (bus de testabilité, structures logiques à scrutation (« Scan », BILBO), test intégré).

2.2. Les différentes étapes de vérification d'un circuit intégré.

Depuis sa conception, jusqu'à son utilisation, un circuit intégré subit un certain nombre de tests permettant de vérifier son bon fonctionnement. Les différents stades ou apparaissent ces tests sont les suivants :

Dès la sortie des premiers prototypes, la conception et la fonctionnalité du circuit sont validées par le test de validation en fin de conception.

Ensuite pendant la phase de production, les circuits défectueux sont éliminés par le test de fin de fabrication.

Une fois les circuits assemblés sur une carte, il faut vérifier que cette dernière est fonctionnellement bonne et ainsi de suite jusqu'au système complet.

Enfin, durant la vie du système (fonctionnement) il faut appliquer périodiquement un test de maintenance pour détecter l'éventuelle défaillance d'un circuit.

Pour éviter qu'une panne n'affecte trop longtemps un circuit sans être détectée, des mécanismes ou des logiciels peuvent être utilisés pour vérifier en permanence, la validité des opérations effectuées.

A ces différentes phases de vérification correspondent trois types de test :

2.2.1. Le test de mise au point :

C'est le test de fin de conception ou de validation, il est destiné à révéler et à diagnostiquer les erreurs de conception. Deux types de vérifications peuvent être envisagés selon les processus mis en œuvre pendant la conception. Ces processus peuvent être totalement manuels ou automatiques (compilateur de silicium). Le test de mise au point consiste alors à vérifier le résultat obtenu et/ou à vérifier préalablement les outils de conception automatique.

2.2.2. Le test hors-ligne :

C'est le test utilisé en fin de fabrication qui est destiné à déterminer quels sont les circuits bons à l'issue de la fabrication. Il peut également être utilisé pour les tests de maintenance hors fonctionnement du système.

2.2.3. Le test en-ligne :

C'est le test visant à améliorer la sécurité de fonctionnement, dont le but est de détecter l'apparition d'une panne pendant le fonctionnement du système. La détection peut-être assurée par des mécanismes ajoutés **[4]** ou par la conception de systèmes auto-contrôlables en-ligne **[5]**, **[6]**.

2.3. Principe du test des circuits intégrés

L'accroissement de la complexité et de la densité d'intégration des circuits ne permet plus la conception de systèmes sans se préoccuper du test très tôt dans la phase de développement du produit. La stratégie de test et les moyens à mettre en œuvre pour obtenir une bonne testabilité doivent êtres définis dès le stade des spécifications.

Le test d'un circuit consiste à appliquer des séquences de vecteurs de test sur ses entrées et à analyser la conformité des réponses en sortie. Les différentes techniques de test visent à faciliter cette action en améliorant les deux aspects suivants :

- La contrôlabilité, c'est à dire la facilité avec laquelle un nœud interne d'un circuit peut être excité à partir de ses entrées.

- L'observabilité, c'est à dire la facilité avec laquelle l'état d'un nœud interne d'un circuit peut être propagé vers ses sorties primaires.

La génération de ces vecteurs de test peut être effectuée selon plusieurs approches.

2.3.1. Les différentes approches

Tester un circuit nécessite d'appliquer, à un instant donné, une séquence de test à ses différentes parties (controlabilité) et d'observer les réponses afin de pouvoir les comparer à des réponses correctes (observabilité).

La nature de la séquence à appliquer ainsi que la manière dont elle est générée correspondent à plusieurs approches

2.3.1.1. L'approche déterministe

Dans cette approche, les vecteurs de test sont déterminés dans le but de détecter un type de panne précis. Si la séquence de vecteurs est déterminée à partir d'une fonction à laquelle est attaché un modèle de panne, le test est alors qualifié de test fonctionnel **[7]**. Si la séquence de test est élaborée à partir de la structure du circuit pour la détection d'une ou plusieurs pannes particulières, le test est qualifié de test structurel **[8]**.

L'analyse des sorties peut se faire, soit par comparaison avec des valeurs prédéterminées, soit par analyse de signature (analyse par compactage) **[9]**.

2.3.1.2. L'approche non-déterministe

Dans cette approche **[10]**, les vecteurs de test générés n'ont pas été déterminés pour la détection d'une panne précise (ou un seul type de panne). Cette approche, appelée séquence aléatoire (ou pseudo-aléatoire si la séquence de vecteurs est reproductible), permet la détection de pannes qui ne sont pas modélisées. La longueur de la séquence de test est fonction de la panne la plus difficile à détecter.

L'analyse des sorties peut être effectuée, soit par comparaison avec les sorties d'un circuit de référence, soit par analyse de signature.

2.3.2. Les différents types de test

On distingue plusieurs types de test selon le moment ou le test sera effectué.

2.3.2.1. Test en-ligne / test hors-ligne

Le test en-ligne d'un circuit permet la détection des pannes internes d'un circuit, pendant le déroulement de l'application. Le test hors-ligne s'effectue en dehors du contexte d'exécution de cette application.

2.3.2.2. Test continu / test discontinu

Le critère de test continu ou discontinu sert toujours à qualifier un test en-ligne. Ce dernier est continu si la simultanéité du test et de l'application est parfaite, tandis qu'il est discontinu lorsqu'il y a alternance d'une phase de test et d'une phase d'application.

2.3.2.3. Test in-situ /test ex-situ

Le critère de test in-situ ou ex-situ ne peut être employé que pour qualifier un test hors-ligne. Pour un test hors-ligne in-situ le circuit est testé dans son environnement physique de travail et il n'existe pas de moyen d'observer à tout instant les valeurs des signaux. Dans le cas du test hors-ligne ex-situ, le circuit est sorti de son environnement de travail et il est placé dans un environnement spécifique au test. L'observation des valeurs qui transitent sur les lignes est alors possible.

2.4. La conception en vue du test

L'avènement des circuits à très grande échelle d'intégration (VLSI) a justifié l'utilisation de techniques de conception permettant de nouvelles stratégies de test particulièrement adaptées aux fonctions de plus en plus complexes portées sur une seule puce de silicium. Ces stratégies de test doivent être définies très tôt dans le cycle de développement du circuit. La testabilité fait partie intégrante du cycle d'étude, le test étant une fonction devant être exécutée pendant le cycle de conception et de production. Elle doit être traitée comme un élément des spécifications fonctionnelles.

2.4.1. Les circuits conçus pour faciliter le test

La conception en vue d'une meilleure testabilité (Design For Testability), regroupe toutes les méthodes permettant d'améliorer, et de diminuer le coût, du test des circuits intégrés. Ces techniques tendent toutes à améliorer l'application des séquences de test citées précédemment (contrôlablilité) et leur observabilité.

Les premières techniques appelées techniques ad hoc, consistaient principalement en un respect de règles de conception particulières. Les autres techniques, appelées techniques « Scan Path » consistent à ramener le problème du test d'un circuit séquentiel à celui de plusieurs blocs combinatoires déconnectés et de taille plus réduite.

2.4.1.1. Méthode ad hoc

Ces techniques consistent en une liste de techniques permettant de résoudre les problèmes rencontrés au niveau du test du circuit. Ces techniques se traduisent généralement par une augmentation du nombre d'entrées de contrôle et de sorties d'observation et le partitionnement du circuit. Ces techniques n'ont pas de portée générale et sont donc liées à la structure du circuit.

2.4.1.2. Méthode « Scan Path »

Dans la méthode « Scan Path »**[11]**, le circuit possède deux modes de fonctionnement. Le mode de fonctionnement normal et un mode de test où tous les nœuds internes de mémorisation (bascules) sont interconnectés en formant un registre à décalage**[12], [13]**. Le mode normal permet ensuite de transmettre ces vecteurs sur les blocs combinatoires, qui modifient alors le contenu du registre. Les résultats complets peuvent alors être extraits du circuit en mode série puis comparés à la réponse correcte.

2.4.1.3. Méthode « LSSD »

La méthode LSSD *(Level Sensitive Scan Design)* **[14], [13]** utilisée industriellement par la compagnie IBM, reprend le principe du « Scan Path » en utilisant notamment des bascules réagissant sur les niveaux. Cette méthode est moins sensible aux problèmes de délais internes du circuit.

2.4.1.4. Méthode « Boundary Scan »

La technique « Boundary Scan » est une extension des techniques Scan Path au niveau de la carte **[15]**. Cette technique permet ainsi de tester de manière relativement simple les interconnexions entre les circuits intégrés. Elle permet également d'échantillonner les données en mode de fonctionnement normal pour effectuer une analyse ultérieure.

Toutes ces approches permettent de simplifier le test des circuits séquentiels et améliorent sensiblement les problèmes d'accessibilité. Le coût de la génération des séquences de test est réduit et les taux de couverture sont meilleurs. La durée du test est toutefois augmentée du fait de la sérialisation des données, qui n'est pas compensée par la diminution de la longueur de la séquence. La fréquence de fonctionnement est en général diminuée du fait de l'introduction de dispositifs logiques supplémentaires. Enfin, le rajout d'éléments pour obtenir des registres à décalage supplémentaires entraîne une augmentation conséquente de la taille du circuit.

D'autres techniques ont été proposées pour diminuer la complexité du testeur en déplaçant la production et l'analyse des vecteurs de test au niveau du circuit. Ces techniques sont regroupées sous le terme d'auto-test intégré (BIST) « Built-In Self Test ».

2.4.2. Le test intégré

La solution consiste à intégrer dans le circuit tous les mécanismes nécessaires à son propre test afin de pouvoir assurer le bon fonctionnement du circuit en présence défauts.

2.4.2.1. Le test intégré hors-ligne

Le principe de cette technique, connue sous le nom de BIST (« *Built In Self Test* »), est d'incorporer dans le circuit la génération de vecteurs de test et l'observation des sorties. Pour ce faire, différents générateurs de vecteurs de test et des analyseurs de signature (observation des réponses par compactage) sont réalisés avec des registres à décalage à rebouclage linéaire (« *Linear Feedback Shift Registers* » (LFSRs)). Ces techniques simplifient énormément le test des circuits complexes : le temps d'application des séquences est court, les circuits sont contrôlés à la vitesse d'utilisation, la phase d'élaboration des vecteurs de test est évitée (génération de vecteurs « pseudo-aléatoire » le plus souvent), l'initialisation et la lecture des registres ne sont pas coûteuses en temps. Cependant, l'augmentation en surface due à l'emploi de registres spécifiques peut ne pas être négligeable.

Dans ce domaine, la méthode de conception BIST la plus représentative est la méthode BILBO (« *Built In Logic Bloc Observer* ») proposée en **[4]**.

Les différentes techniques présentées jusque là concernent le test hors-ligne des circuits. Toutefois elles peuvent être utilisées pour faire un test en-ligne discontinu en activant périodiquement les séquences de test nécessaires. La détection de la panne est alors réalisée avec un retard par rapport à son apparition qui est liée à la périodicité du test. Ces solutions ne sont pas satisfaisantes pour les applications qui exigent une sûreté de fonctionnement importante où la sécurité est le facteur essentiel **[16]**. En effet, dans ce cas il est indispensable d'assurer immédiatement la détection des erreurs causées par une panne dans le système afin d'intervenir rapidement et éviter une évolution catastrophique de la situation. Pour répondre à ce besoin, les circuits sont conçus pour être auto-contrôlables.

2.4.2.2. Le test intégré en-ligne

Le but du test en-ligne est de détecter le plus rapidement possible l'apparition d'une panne dans un circuit au cours de son fonctionnement et ce pendant le déroulement de l'application.

Nous venons de voir que les techniques utilisées pour le test hors-ligne pouvaient être utilisées pour réaliser un test en-ligne discontinu. Il suffit pour cela que la phase de test s'exécute à une fréquence beaucoup plus élevée que celle du système et/ou que les valeurs spécifiques au test n'apparaissent pas en sortie **[17]**.

Parmi les diverses techniques possibles pour le test continu (codage logiciel, redondance massive, ...), la prise en compte du test en-ligne dès la conception (circuits *Self Testing* et *Self Checking*) permet d'obtenir des circuits où la détection des pannes est assurée par le matériel (blocs fonctionnels et contrôleurs).

Les circuits auto-contrôlables en-ligne sont basés sur le principe des circuits auto-contrôlables auxquels certaines propriétés peuvent être ajoutées pour former d'autres classes de circuits (*Fault Secure* etc...).

Une autre voie, qui peut être complémentaire, consiste à utiliser les techniques de détection de panne par contrôle de courant (« IDDQ »).

2.5. Conclusion

Les différentes méthodes proposées précédemment sont autant de choix possibles pour réaliser des circuits intégrés afin de faciliter les tests de validation des prototypes, les tests de fin de fabrication et les tests de maintenance. Cependant la conception de circuits auto-contrôlables est certainement la plus adaptée pour la réalisation de systèmes exigeant une sûreté importante (automatismes industriels de sécurité, systèmes de transport, etc.).

D'autre part le test intégré unifié propose la conception de composants auto-contrôlables aussi bien vis à vis du test hors-ligne (utilisation de LFSRS) que du test en-ligne (utilisation de contrôleurs).

C'est la voie des circuits auto-contrôlables que nous allons explorer dans la suite en commençant par étudier les codes détecteurs d'erreur qui sont souvent utilisés pour le test en-ligne.

3. Le test en-ligne dédié aux applications de sécurité

3.1. Introduction

Nous avons vu précédemment que les fantastiques progrès de la micro-électronique ont permis de créer des calculateurs de plus en plus puissants. Cependant, et bien que la fiabilité des circuits intégrés ait également bénéficié d'importants progrès, le nombre très important de transistors intégrés sur une même puce ne permet pas de négliger la probabilité d'occurrence d'une panne.

Pour concevoir des systèmes dont l'application requière une certaine sécurité de fonctionnement, il est donc nécessaire de détecter l'apparition d'une panne dans le circuit et de la signaler. Au regard de la complexité atteinte par les systèmes actuels, une approche ad hoc est devenue quasiment irréalisable. Les techniques de test hors-ligne atteignant leurs limites, il a fallu prendre en considération le test en-ligne du circuit dès sa conception.

Cette approche, basée sur la redondance et/ou l'utilisation de codes détecteurs d'erreurs permet donc de détecter l'apparition d'une panne (permanente ou transitoire) pendant le déroulement de l'application. Elle présente donc l'avantage de couvrir les modes de défaillance réels et permet une détection des pannes en temps réel.

De tels systèmes seront donc réalisés à partir d'un assemblage de fonctions élémentaires munies de mécanismes d'auto-test.

C'est vers cette voie que se dirigent actuellement les méthodes de test en-ligne, et plus particulièrement vers la conception de circuits dits « auto-contrôlables ».

La conception d'un circuit « auto-contrôlable » consiste donc en un ajout de matériel, prévu au niveau fonctionnel, de manière à ce que le circuit soit capable de détecter toute apparition de panne. Cette conception est basée sur des techniques de codage spécifiques et sur les propriétés des différents blocs qui constituent la fonction. Ces circuits « auto-contrôlables » sont donc utilisés dans tous les systèmes où l'occurrence d'une panne doit être détectée le plus rapidement possible.

Ils ne peuvent cependant pas êtres utilisés directement pour la réalisation de système exigeant une sûreté de fonctionnement vitale. De tels systèmes, dits à « défaillance sans danger » (*Fail Safe*) ont été conçus pour que toute sortie erronée résultant d'une panne interne ne produise pas un événement catastrophique. Ils interviennent au niveau de l'étage de commande des systèmes critiques.

Les systèmes *Fail Safe* garantissent qu'en présence d'une panne le système prend soit la valeur correcte soit une valeur dite *sûre* qui ne peut jamais entraîner de situation dangereuse. La réalisation de tels systèmes passe donc par l'utilisation de circuits auto-contrôlables permettant de détecter l'apparition de la première panne conjuguée à des interfaces *Fail Safe* permettant de générer des sortie sûres pendant tout le fonctionnement du système.

Dans les sections suivantes, nous présentons dans un premier temps les différents types de codes détecteurs d'erreur utilisés pour la détection des pannes susceptibles d'affecter un circuit VLSI. Nous expliquons ensuite le principe de fonctionnement et les techniques de réalisation des systèmes « auto-contrôlables ». Enfin, nous énonçons les propriétés des systèmes à défaillance sans danger et leur principe.

3.2. Les codes détecteurs d'erreur

3.2.1. Historique

Pendant longtemps le modèle pris en compte pour le test des circuits intégrés a été celui du collage logique. Dans ce modèle on représente les circuits par des portes logiques et on considère qu'une entrée ou une sortie d'une porte prend constamment la valeur logique 0 ou 1. Ce modèle n'est pas représentatif de toutes les pannes réelles des circuits intégrés. Par conséquent, de nouveaux modèles sont apparus dans lesquels les circuits sont représentés au niveau électrique par des réseaux de transistors MOS et on considère comme pannes possibles les collages des lignes à une valeur logique, des collages de MOS passant ou ouvert et des courts-circuits.

A ce niveau, on dispose de la classification des pannes établie dans **[18]**. Il y est montré que les mécanismes liés à la durée de vie des circuits ne peuvent pas produire des courts-circuits entre deux lignes de même niveau (ou entre deux lignes appartenant à des niveaux

différents). D'autre part, les mécanismes de défaillance liés à la durée de vie des circuits sont très lents. Il est donc réaliste de considérer que pour le test en-ligne, les pannes matérielles sont uniques (une seule panne présente). Toute la réussite de la méthode dépend donc de l'utilisation d'un modèle de pannes simples qui représente bien les défaillances réelles.

3.2.2. Les différents types d'erreurs

Au regard du grand nombre et de la diversité des défauts pouvant affecter un bloc fonctionnel, on utilise une étape intermédiaire de classification des défauts.

Cette classification se fait en fonction de la manifestation des pannes et permet de réduire l'étude aux différentes classes de défauts. La manifestation logique des défauts est appelée « erreur », et l'on distingue les trois types d'erreur suivants :

3.2.2.1. Les erreurs simples

Ces erreurs n'affectent qu'un seul bit dans un mot et correspondent au cas où une seule porte est affectée par une panne.

Par exemple : *1011011* au lieu de *1001011*.

3.2.2.2. Les erreurs unidirectionnelles

Ces erreurs affectent un nombre quelconque de bit mais uniquement dans un seul sens (les erreurs sont toutes de type 0→1 ou 1→0, mais pas les deux à la fois).

Par exemple : *0001001* au lieu de *1001011*.

3.2.2.3. Les erreurs multiples

Ces erreurs modifient un nombre quelconque de bits dans les deux sens.

Par exemple : *0101001* au lieu de *1001011*.

3.2.3. Les différents types de codes détecteurs d'erreurs

Principalement utilisés, à l'origine, pour la transmission de données sans erreur, les codes détecteurs d'erreurs sont désormais utilisés pour la détection de pannes dans les circuits intégrés. Les caractéristiques de ces derniers codes ne sont évidemment pas les mêmes que ceux pour la transmission de donnée. Leur théorie a été développée dans **[19]**, **[20]**, et **[21]**.

Le codage consiste en une redondance de données utilisant des bits de contrôle, appelés aussi bits de codage, avec chaque mot de donnée, procurant ainsi la possibilité de détecter les erreurs. La redondance d'un code est définie par le rapport du nombre de bits de codage au nombre de bits de l'information réelle. Le meilleur code sera celui qui réalisera la détection des types d'erreurs les plus probables, en nécessitant le minimum de bits supplémentaires. Le type d'erreur provoquée dépend du type de la panne et de la structure du bloc défectueux.

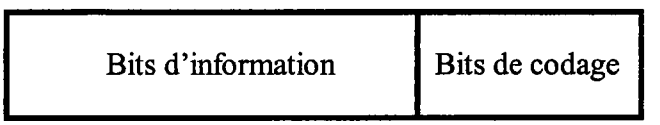
On distingue quatre grandes classes de codes détecteurs d'erreurs liés au test en-ligne. Chacune de ces classes regroupe un ensemble de codes différents. Il n'existe actuellement aucune méthode rigoureuse permettant de déterminer immédiatement le code optimal permettant de réduire le nombre de circuits de contrôle pour une application donnée.

Les codes les plus intéressants pour la réalisation des circuits auto-contrôlables sont présentés dans la suite. Les différents codes peuvent également être classés selon qu'ils appartiennent ou non à la catégorie des codes séparables et des codes ordonnés.

3.2.3.1. Les codes séparables

Un code séparable est un code qui est formé de deux groupes de bits séparables :

- Un groupe de bits d'information
- Un groupe de bits de codage



3.2.3.2. Les codes non ordonnés

Un code non ordonné est un code où il n'existe pas deux vecteurs tels que l'un recouvre l'autre. Un vecteur *A* couvre un vecteur *B* si tous les bits à 1 du vecteur *B* le sont également dans le vecteur *A* (figure I.2).

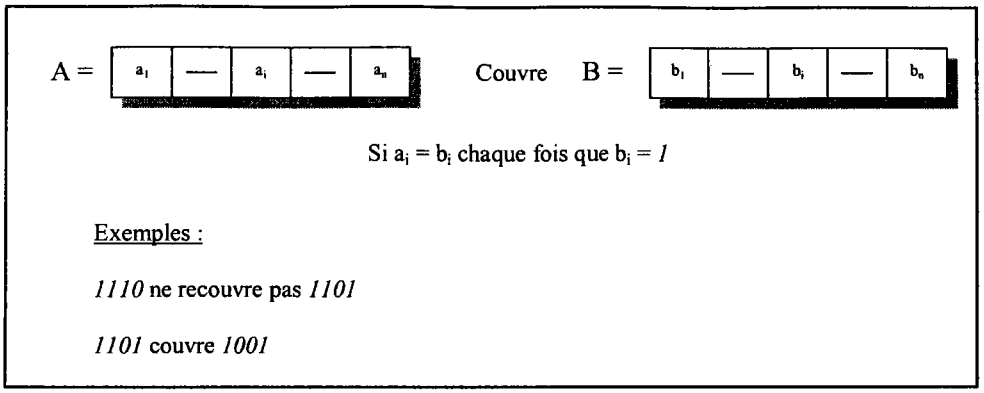


Figure I.2 : Recouvrement de deux vecteurs

Les codes non ordonnés sont utilisés pour la détection des erreurs unidirectionnelles.

Les quatre grandes classes de codes détecteurs d'erreurs énoncés précédemment sont les suivantes :

3.2.3.3. Les codes de parité

Le code de parité est un code séparable dont la mise en œuvre est facile et économique. C'est le code qui entraîne la redondance minimale (1 bit de contrôle).

Il permet la détection des erreurs simples.

Il existe une extension de ce code pour la détection d'erreurs multiples affectant *b* bits adjacents. On parle alors de code *b-adjacent.*

3.2.3.4. Les codes arithmétiques

Les codes arithmétiques peuvent être séparables ou non et présentent la propriété suivante :

Le résultat d'une opération arithmétique sur des mots appartenant à un code arithmétique diffère d'un mot de code au plus par une constante additive.

Ces codes sont utilisés dans des applications contenant des opérateurs arithmétiques . Le code le plus performant est le code résiduel qui est un code séparable pour lequel il existe des circuits de contrôle auto-contrôlables **[22]**, **[23]** et **[24]**

3.2.3.5. Les codes de duplication

Ce sont des codes séparables, faciles à mettre en œuvre mais impliquant une redondance maximale. Ils permettent la détection des erreurs multiples.

3.2.3.6. Les codes unidirectionnels

Ce sont obligatoirement des codes non ordonnés. Il existe un grand nombre de codes détecteurs d'erreurs unidirectionnels, qu'ils soient séparables ou non.

Comme leur nom le laisse supposer, ces codes détectent les erreurs unidirectionnelles.

La plus grande partie de ces codes sont dérivés des codes de ***BERGER*** et des codes m/n présentés dans la suite.

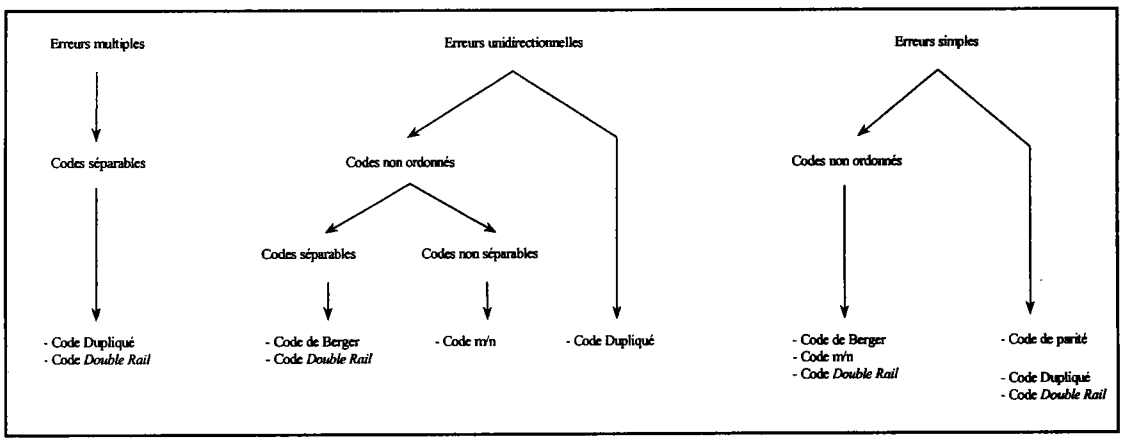


Figure I.3 : Schéma récapitulatif des différents codes classés selon le type d'erreur

Les paramètres à prendre en compte dans le choix d'un code détecteur d'erreur sont :

- Le type d'erreur détecté
- La redondance du code
- Le nombre de codes possibles
- La complexité et la rapidité des circuits de codage et de contrôles
- Le coût d'implémentation qui dépend :
  - du nombre d'entrées/sorties supplémentaires
  - de la surface supplémentaire
  - du temps de propagation

3.2.4. Présentation des principaux codes détecteurs d'erreur

3.2.4.1. Le code à parité

Un ensemble de configurations binaires de n variables e_i forme un code à parité si la relation suivante est vérifiée :

$e_1 \oplus e_2 \oplus \ldots\ldots \oplus e_n = a$

- Si $a = 0$ il s'agit d'un code à parité paire
- Si $a = 1$ il s'agit d'un code à parité impaire

Le code à parité est d'une mise en œuvre facile et économique en raison de la simplicité des circuits de codage et de contrôle. C'est un code séparable.

L'extension de ce code à la détection d'erreurs multiples est constituée par les codes b-adjacents. Ces codes sont obtenus en intercalant les variables de b codes à parité simple. La figure suivante décrit un code 2-adjacent.

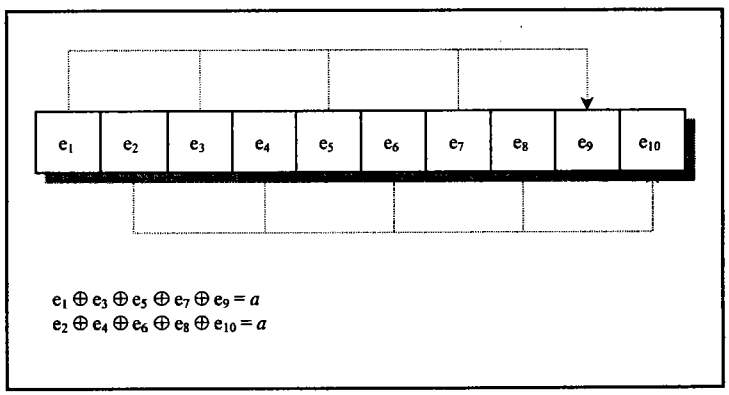


Figure I.4 : Exemple de code 2-adjacent

De tels codes permettent la détection de fautes multiples affectant b bits adjacents, donc de défauts affectant plusieurs variables adjacentes.

3.2.4.2. Le code de *BERGER*

Le code de Berger **[19]** est un code non ordonné séparable qui permet la détection des erreurs unidirectionnelles. Il est obtenu par la concaténation de *I* bits d'information et *K* bits de contrôle. Les *K* bits de contrôle correspondent au complément logique du nombre de *1* contenu dans les *I* bits d'information, comme l'illustre la figure suivante.

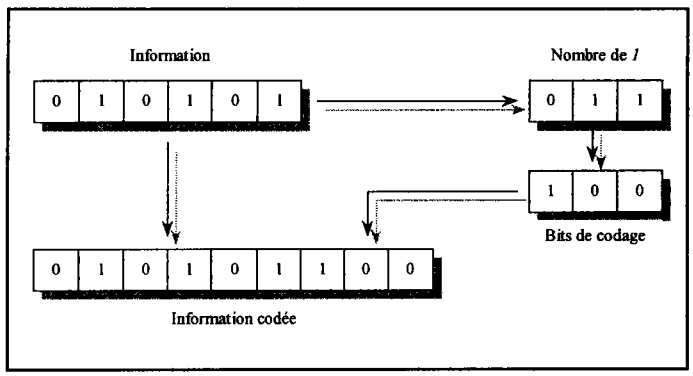


Figure I.5 : Principe du code de *BERGER*

Ce code est un code séparable optimal pour la détection des erreurs unidirectionnelles, en ce sens qu'il est celui qui nécessite le moins de bits de contrôle pour un nombre donné de bits d'information **[6]**.

Le nombre de bits de contrôle est donné par la borne supérieure de $[Log_2(I+1)]$.

3.2.4.3. Le code *m* parmi *n*

Un code *m* parmi *n* est un code non ordonné dont les mots ont exactement *m* bits à la valeur 1, et *n-m* bits à la valeur *0* **[25]**.

Ce sont des codes à pondération constante que l'on note également m/n.

Le nombre de mots d'un code *m* parmi *n* est égal à $C_n^m = \frac{n!}{m!(n-m)!}$.

Le codage d'une information de k bits nécessite l'emploi d'un code m/n tel que $C_n^m \geq 2^k$.

Le code *n* parmi *2n* est le code non ordonné optimal en ce sens qu'il n'existe pas d'autres codes non ordonnés de longeur *2n* qui possède plus de mots de code.

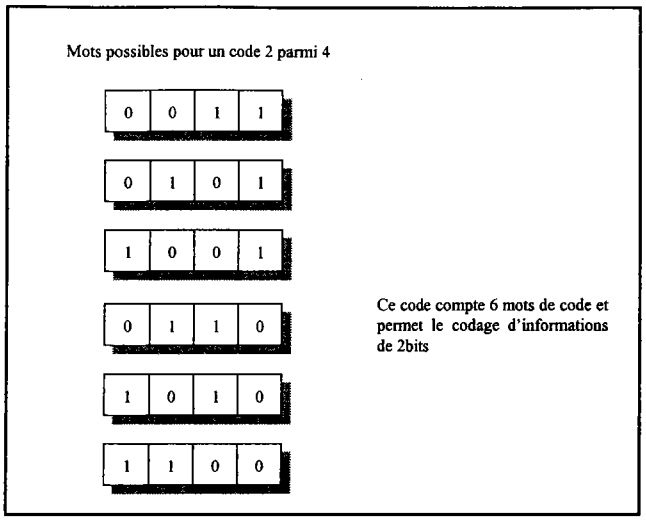


Figure I.6 : Exemple de code *m* parmi *n*

3.2.4.4. Le code à duplication

Le code à duplication **[25]** est le mieux adapté pour la détection d'erreurs multiples et peut être de deux types :

- Le code à duplication directe, composé de l'information utile et de son double.
- Le code à duplication et complémentation (code *Double Rail*), où le code ajouté est le complément logique de l'information utile.

Ce code est un code séparable dont la mise en œuvre est aisée mais c'est le code à redondance maximale.

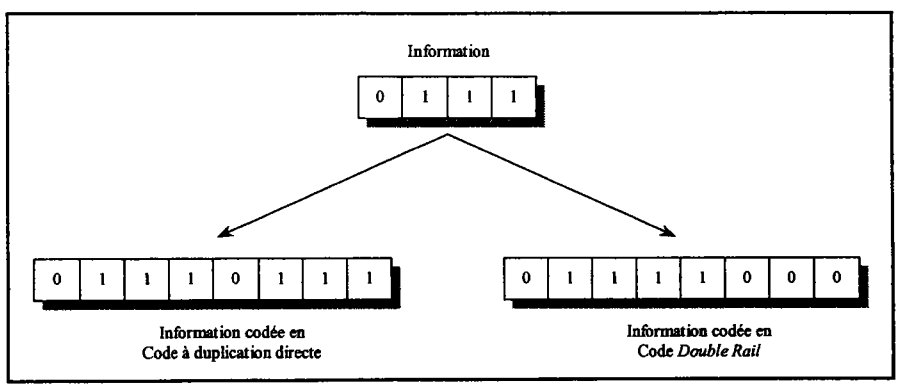


Figure I.7 : Exemple de code à duplication

D'un point de vue pratique, la duplication est obtenue en utilisant, soit un bloc fonctionnel identique qui génère en parallèle la même information utile (duplication directe), soit un bloc fonctionnel dual qui délivre en parallèle le complément de l'information.

3.3. Les circuits auto-contrôlables (Self Checking)

3.3.1. Introduction

L'évolution rapide de la micro-électronique, liée aux progrès réalisés dans le domaine de la technologie empêche désormais les techniques de prévention couramment utilisées jusqu'à aujourd'hui (test externe) de couvrir l'ensemble des pannes susceptibles de se produire dans les circuits intégrés logiques. Ce constat est d'autant plus problématique lorsque l'application est dite de sécurité, car la défaillance d'un système peut être à l'origine de situations critiques ou catastrophiques.

Il est donc nécessaire de signaler la présence de la première sortie erronée due à une panne. Cette mission n'est donc pas remplie par les techniques utilisées avant cette évolution.

Cet avènement des circuits à très grande échelle d'intégration (VLSI) a donc justifié l'utilisation de techniques de conception permettant de nouvelles stratégies de test particulièrement adaptées aux fonctions de plus en plus complexes portées sur une seule puce de silicium.

De nombreux travaux ont alors étudiés l'apport de la prise en compte du test en-ligne dès la conception pour ces techniques. Parmi les nombreuses techniques qui en découlent, la conception de circuits auto-contrôlables (*Self Checking*) est certainement la plus adaptée à la réalisation de systèmes exigeant une sûreté importante. Le but du test en-ligne est de détecter l'apparition d'une panne dans un circuit au cours de son fonctionnement (pendant le déroulement de l'application). Cette technique a l'avantage de couvrir des modes réels de défaillance qui sont actuellement bien étudiés, et elle ne nécessite aucun logiciel pour assurer le test en-ligne. La détection des erreurs est assurée par le matériel.

3.3.2. Pincipe d'un circuit *Self-Checking*

La structure générale d'un circuit auto contrôlable se compose d'un bloc fonctionnel et d'un contrôleur de la manière suivante **[6], [27], [28]**:

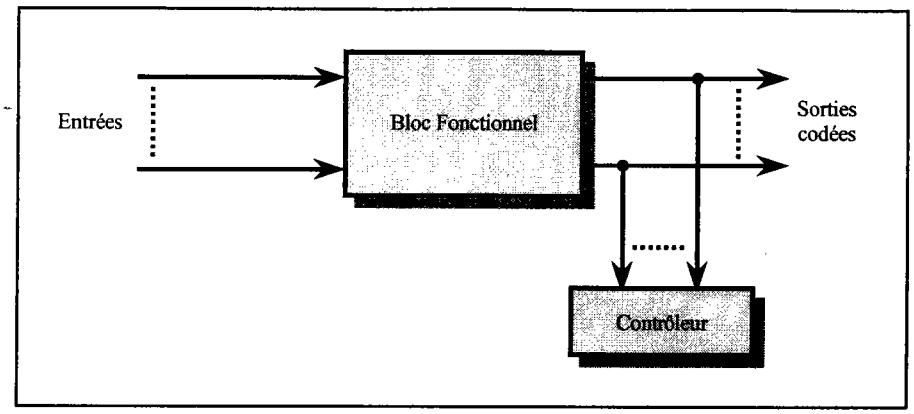


Figure I.8 : Structure d'un circuit auto-contrôlable.

Le bloc fonctionnel transforme les valeurs de ces entrées en valeurs codées sur ces sorties, de manière à ce que la première manifestation d'erreur (sur les sorties codées) due à une panne dans le circuit soit immédiatement signalée par le contrôleur. Ce dernier possède généralement deux sorties codées en *Double Rail*.

Des circuits complexes peuvent être conçus en réalisant les blocs de base dans la structure auto-contrôlable. Dans ce cas, on utilise un contrôleur *Double Rail* pour compacter les indications d'erreur des différents blocs en une indication d'erreur globale.

Pour que cet objectif appelé but d'auto-contrôlabilité totale (*Totally Self Checking Goal*) soit atteint, il faut que les différentes parties du système respectent les propriétés mathématiques que nous allons présenter dans la suite en adoptant les conventions suivantes.

Un bloc fonctionnel G qui a r entrées primaires et q sorties possède un ensemble X de vecteurs d'entrée formé de 2^r vecteurs binaires de longueur r. De même l'ensemble Y des vecteurs de sortie possède 2^q vecteurs binaires de longueur q.

En fonctionnement normal, le bloc fonctionnel G reçoit sur ces entrées un sous ensemble A de vecteurs de X correspondant au code d'entrée et il produit un ensemble B de vecteurs $G(a,\varnothing)$ de Y appelé code de sortie. En présence d'une panne f de l'ensemble des pannes F pris comme modèle, le bloc produit des valeurs $G(a, f)$ en sortie.

3.3.3. Définitions

Les définitions suivantes sont dues à ***ANDERSON*** **[6]**.

Pour détecter l'apparition d'une sortie erronée, due à une panne faisant partie d'un ensemble de pannes définies au départ, les circuits *Fault Secure* ont été définis de la manière suivante :

* **Définition** d'un circuit *Fault Secure* (*sûr* en présence de pannes).

Un circuit G est *Fault Secure* pour un ensemble de pannes F si :

$\forall f \in F, \forall a \in A,$ soit $G(a, f) = G(a, \varnothing)$, soit $G(a, f) \notin B$.

Autrement dit, en présence de pannes la sortie prendra soit une valeur correcte soit elle sera hors code.

Si $G(a, f) = G(a, \varnothing)$ la panne n'intervient pas sur le fonctionnement du circuit.

Si $G(a', f) \neq G(a', \varnothing)$ et $G(a', f) \in B$ la panne influe sur le fonctionnement et n'est pas détectée

Si $G(a'', f) \notin B$ il y a au moins un vecteur d'entrée qui détecte la panne.

Cependant, si une panne ne produit pas de sorties erronées pour un ensemble de valeurs d'entrée, cette panne reste indécelable, et si, plus tard, une deuxième panne survient, le système ne sera peut être plus *Fault Secure* pour la panne composée.

C'est pour cette raison que la définition des circuits *Self Testing* a été proposée.

* **Définition** d'un circuit *Self Testing* (auto-testable).

Un circuit G est *Self Testing* pour un ensemble de pannes F si :

$\forall f \in F : \exists a \in A$ tel que $G(a, f) \notin B$

Donc, pour un circuit *Self Testing* qui reçoit tous les vecteurs de A, la panne est détectée.

C'est à l'aide de ces deux propriétés que les circuits *Totally Self Checking* ont pu être définis.

***Définition** d'un circuit *Totally Self Checking* (Totalement auto contrôlable)

Un circuit *G Totally Self Checking* est un circuit *Self Testing* et *Fault Secure.*

Un circuit *G* accomplit le *Totally Self Checking Goal* si la première sortie erronée due à une panne *f* de l'ensemble des pannes *F* est en dehors du code de sortie. Ce but est atteint par les circuits *Totally Self Checking* sous l'hypothèse suivante :

*** Hypothèse 1** :

Entre l'occurrence de deux pannes quelconques de *F*, il s'écoule un temps assez long pour que tous les vecteurs du code d'entrée soient appliqués aux entrées du circuit *G*.

La plus grande classe de circuits fonctionnels permettant d'assurer le *Totally Self Checking Goal* en respectant cette hypothèse est la classe des circuits *Strongly Fault secure.*

*** Définition** d'un circuit *Strongly Fault secure* (fortement *sûr* en présence de panne) :

Un circuit G est *Strongly Fault secure* pour un ensemble de pannes *F* si pour toute panne $f \in F$:

a) soit le circuit est *Totally Self Checking*

b) soit le circuit est *Fault secure* et si une nouvelle panne *f* de *F* survient, pour la panne résultante on retombe dans le cas *a)* ou *b).*

Afin d'obtenir une grande sécurité de fonctionnement, il faut que le circuit soit capable de détecter un mot d'entrée qui ne figure pas dans l'ensemble prédéfini des mots du fonctionnement normal.

C'est dans ce but que la propriété à *Codes Disjoints* a été introduite.

*** Définition** d'un circuit à *Codes Disjoints*

Un circuit G est à *Codes Disjoints* si :

$\forall\, a \in A : G(a, \varnothing) \in B$, et $\forall\, x \in (X\text{-}A) : G(x, \varnothing) \notin B$.

Ainsi, le contrôleur d'un circuit auto-contrôlable doit disposer des propriétés suivantes:

3.3.4. Les contrôleurs *Self Checking*

* **Définition** d'un contrôleur *Totally Self Checking*

Un circuit *G* qui est *Totally Self Checking* et *Codes Disjoints* est un contrôleur *Totally Self Checking*.

3.4. Les circuits *Fail Safe*

3.4.1. Introduction

Les progrès réalisés dans le domaine des semi-conducteurs ont permis de disposer d'une puissance de calcul suffisante, mais au détriment de la sécurité intrinsèque. L'intégration d'un nombre de plus en plus important de fonctions sur une même puce augmente les risques de panne et diminue la possibilité de détection de toutes les pannes possibles.

De tels circuits ne sont donc pas compatibles avec des applications exigeant une sûreté de fonctionnement draconienne (transports ferroviaires et aéronotiques, industries chimiques et nucléaires, etc.).

Ces applications, dites *Fail Safe*, sont actuellement réalisées à l'aide de composants discrets répondant à des critères très stricts de sécurité. Les systèmes *Fail Safe* ont donc été conçus pour qu'aucune sortie erronée résultant d'une panne interne ne donne lieu à un événement catastrophique. En présence d'une panne la sortie doit prendre, soit la valeur correcte, soit une valeur dite *sûre* qui ne peut en aucun cas entraîner de situation dangereuse.

Cette propriété est nécessaire, par exemple, pour le pilotage d'éléments se trouvant en bout de chaîne de nombreux systèmes de surveillance utilisés dans l'industrie chimique ou les transports. C'est aussi le cas pour les systèmes de signalisation routière et ferroviaire (ex : il ne faut pas que tous les sémaphores d'un carrefour soient simultanément au vert).

Cependant, la réalisation en composants discrets est très coûteuse et limite la complexité des systèmes que l'on peut développer.

La seule possibilité d'exploiter les microprocesseurs dans des systèmes critiques est donc d'assurer que l'apparition d'une panne dangereuse ne survienne qu'avec une probabilité telle qu'une catastrophe puisse être raisonnablement jugée improbable.

Les circuits digitaux présentés dans la section précédente, permettent de détecter la première apparition d'une panne, mais ne délivrent pas de données conformes à la propriété *Fail Safe*. Ces circuits sont en effet susceptibles de fournir des réponses erronées lors de collages ou de courts-circuits sur les sorties par exemple.

De précédents travaux **[17]** ont introduit une nouvelle approche permettant de concilier les objectifs d'intégration et de sécurité *Fail Safe*. L'application présentée dans ces travaux présentait l'inconvénient d'utiliser un test en-ligne discontinu.

Ces travaux ont été finalisés dans **[30]** et ont abouti à la proposition d'une interface *Fail Safe* intégralement testé pendant le déroulement de l'application.

3.4.2. Principe

Le concept des circuits *Fail Safe* a été introduit par ***MINE*** et ***KOGA*** **[31]** pour des fonctions à une sortie primaire. Puis, ***TAKAOTA*** et ***AL*** **[32]** ont introduit le concept des circuits «*N Fail Safe*» dont la sortie primaire est dupliquée pour distinguer une valeur *sûre* correcte d'une valeur *sûre* qui est le résultat d'une panne dans le système.

Dans le paragraphe suivant nous énonçons les définitions fondamentales des circuits *Fail Safe* revues et généralisées par ***NICOLAIDIS*** **[17]**.

3.4.3. Définitions

3.4.3.1. Les Systèmes Fail Safe

Les systèmes *Fail Safe* sont définis de la manière suivante:

Définition D0' :

Un circuit G à une seule sortie primaire est «*0 Fail Safe*» (respectivement «*1 Fail Safe*») pour un ensemble de vecteurs d'entrée X et pour un ensemble de pannes F si:

$\forall x \in X, \forall f \in F$: $G(x, f) = G(x, \varnothing)$ ou $G(x, f) =$ «*0*» (respectivement « *1* »).

$G(x, \varnothing)$ correspond à la fonction correcte et $G(x, f)$ à la fonction défaillante.

Cette définition est similaire à celle donnée dans **[31]** excepté le fait qu'aucune restriction n'est prévue en ce qui concerne l'ensemble des pannes F.

Cette définition concerne les systèmes n'ayant qu'une seule sortie binaire. Il est cependant possible de réaliser des systèmes *Fail Safe* à sorties multiples composés de plusieurs systèmes *Fail Safe* à sortie unique. Toutefois, la notion d'état sûr en sortie d'un circuit peut être considérée vis à vis d'un groupe de sorties sur lesquels plusieurs vecteurs sont possibles.

L'espace de sortie Y peut donc être divisé en deux sous espaces On et Os. L'ensemble Os des vecteurs de sortie d'état *sûr* est composé de vecteurs qui ne peuvent pas entraîner de situation dangereuse même si leur apparition est involontaire. Les autres vecteurs de Y représentent l'ensemble On des vecteurs de sortie d'état *non sûr*. Nous avons donc $On = Y - Os$ et $On \cap Os = \varnothing$.

*** Définition *D2* :**

Un système G est *Fail Safe* de type *D2* pour un ensemble de pannes F, un ensemble de vecteurs d'entrée X et un ensemble Os de vecteurs de sortie d'état *sûr* si :

$\forall a \in X, \forall f \in F$: $G(x, f) = G(x, \varnothing)$,

ou $G(x, f) \in Os$.

Remarque importante

Il est intéressant de noter qu'un circuit *Fault Secure* conçu de façon à ce que l'occurrence de sorties erronées ne provoque pas de situation dangereuse est considéré comme un circuit *Fail Safe* de type *D2*. Soit Y l'ensemble des vecteurs de sortie d'un circuit G, et $B \subset Y$ l'ensemble des vecteurs de sortie qui appartiennent à un code donné. Cet ensemble B peut être partagé en un sous ensemble Bs de vecteurs de sortie codés et *sûrs* et en un sous ensemble Bn

de vecteurs de sortie codés et *non sûrs*. Si le concepteur réalise le système *Fault Secure* de manière à ce que tout vecteur qui appartienne à l'ensemble *Y-B* ne provoque pas d'action dangereuse, l'ensemble des vecteurs de sortie *sûr* sera $Os = Bs \cup (Y\text{-}B)$ et l'ensemble des vecteurs *non sûrs* sera $On = Bn$.

Comme *G* est *Fault Secure*, d'après la définition on a, pour un ensemble de vecteurs d'entrée *X* et un ensemble de pannes *F* :

$\forall\, a \in X, \forall\, f \notin F :\quad G(x, f) = G(x, \varnothing)$

ou $G(x, f) \in (Y\text{-}B)$ avec $(Y\text{-}B) \subseteq Os$.

Un tel circuit *Fault Secure* est donc *Fail Safe* selon la définition *D2*.

3.4.3.2. Le *Totally Fail Safe Goal*

Les systèmes *Fail Safe* définis dans **[31]** n'utilisaient pas de sorties redondantes ni de contrôleur pour vérifier le code de sortie. Ils ne pouvaient donc pas détecter leurs propres pannes.

Si on considère un système *Fail Safe* pour un ensemble de pannes *F*. Si une panne f_1 appartenant à *F* survient, de par sa définition le système ne générera jamais de sortie erronée *non sûre* et la sécurité est assurée. Cependant, si le système reste en fonctionnement longtemps après l'occurrence d'une panne, une seconde panne f_2 appartenant également à *F* peut se produire. La panne combinée $[f_1, f_2]$ n'appartiendra peut être pas à *F* et la propriété *Fail Safe* sera peut être perdue.

Il est donc très important qu'un système *Fail Safe* possède des capacités de détection de panne afin de pouvoir activer des mécanismes permettant d'éviter l'utilisation du circuit après cette panne.

La définition des circuits *Self Testing* ayant été donnée au paragraphe précédent nous pouvons énoncer la définition suivante:

*** Définition *D4* :**

Un circuit est *Totally Fail Safe* (totalement sûr en présence de défauts) si il est *Fail Safe* et *Self Testing*.

Si pendant un certain temps un circuit *Fail Safe* défaillant fournit plusieurs vecteurs de sortie erronés, cela ne présente aucun inconvénient dans la mesure où il s'agit de vecteurs d'état *sûr*. Cet objectif à atteindre est le *Totally Fail Safe Goal* par analogie au *Totally Self Checking Goal* des circuits *Self Checking*.

*** Définition *D5* :**

Un circuit atteint le *Totally Fail Safe Goal* si, jusqu'à la première détection d'une panne toutes les sorties erronées appartiennent à l'espace de sortie *sûr*.

Les circuits *Totally Fail Safe* accomplissent le *Totally Fail Safe Goal* sous l'hypothèse suivante :

*** Hypothèse *H1* :**

a) les pannes apparaissent une à une dans le circuit,

b) entre l'occurrence de deux pannes consécutives, il s'écoule un laps de temps assez long pour que, pendant ces modes de fonctionnement, le circuit soit «exercé» par un ensemble d'entrée qui détecte toutes les pannes.

3.4.3.3. Les systèmes Strongly Fail Safe

De même que pour les circuits *Strongly Fault Secure*, les circuits *Strongly Fail Safe* représentent la classe la plus large des circuits capables d'atteindre le *Totally Fail Safe Goal* sous l'hypothèse *H1*.

*** Définition D6 :**

Un circuit G est *Strongly Fail Safe* pour un ensemble de pannes F, si pour chaque panne $f_1 \in F$:

a) G est *Totally Fail Safe* ou,

b) G est *Fail Safe* et si une nouvelle panne f_2 de F apparaît, pour la panne composée (f_1, f_2) la clause *a)* ou *b)* est vérifiée.

Il est possible de concevoir des systèmes *Strongly Fail Safe* qui ne possèdent pas de moyen de détection de panne. Le théorème suivant donne la condition nécessaire et suffisante pour l'existence de tels systèmes.

*** Théorème 2 :**

Un système G qui ne possède pas ses propres moyens de détection d'erreur est *Strongly Fail Safe* pour un ensemble de pannes F si et seulement si, il est *Fail Safe* pour l'ensemble des pannes F^*, où F^* est constitué de pannes multiples qui sont formées par la combinaison des pannes simples de F.

Les circuits *Fail Safe* décrits dans la littérature sont généralement conçus en utilisant des techniques de redondance ou des techniques d'évitement de panne.

Pour les techniques de redondance, si, par exemple, la panne d'un composant implique un état *non sûr*, ce composant est dupliqué. Cette technique ne peut pas assurer la propriété *Strongly Fail Safe* car elle n'est pas *Fail Safe* pour les pannes multiples.

La technique d'évitement de panne consiste à utiliser des composants qui ont un très grand MTTF par rapport à la panne considérée. Cette technique présente l'inconvénient de ne pas être adaptée à une implémentation VLSI où des pannes à conséquences opposées (par exemple : Stuck-at 0 et Stuck-at 1, Stuck-on et Stuck-open) ont des probabilités d'occurrence non négligeables.

3.4.3.4. Ultimate Fail Safe Goal

Le *Totally Fail Safe Goal* garantie la sécurité du système jusqu'à la première détection d'une panne. Cependant, l'objectif ultime est que le système ne produise pas de sortie erronée *non sûre* même après l'occurrence d'une détection de panne, et quelles que soient les pannes ultérieures pouvant apparaître dans le circuit. Cette propriété est indispensable dans le contexte des interfaces *Fail Safe*. Ces interfaces représentent la dernière

couche du système et ne peuvent donc pas recourir à une autre partie du système pour agir lors de la détection d'une panne.

*** Définition** de l'*Ultimate Fail Safe Goal* («l'objectif final des circuits sécuritaires»)

Aussi longtemps qu'un circuit délivre des services, il ne doit pas produire de sorties erronées qui n'appartiennent pas à l'espace de sortie *sûr*.

Un circuit *Strongly Fail Safe* garantit le *Totally Fail Safe Goal*. Cependant, lorsque la première détection de panne apparaît, la situation du circuit correspond au cas *a)* de la définition *D6*. Dans cette situation, si plus tard une nouvelle panne survient, une sortie non sûre erronée peut être générée. Ainsi, afin d'atteindre l'*Ultimate Fail Safe Goal* il faut utiliser des mécanismes qui répondent à cette éventualité. Ce peut être par exemple l'isolation du circuit défaillant et l'utilisation d'autres ressources pour accomplir sa tâche, ou le verrouillage des sorties dans un état *sûr*. L'isolation du circuit en panne ou le verrouillage des sorties doit être irréversible quelles que soient les pannes qui peuvent se produire par la suite.

4. Conclusion

Après avoir énuméré les différentes méthodes facilitant les tests nécessaires à l'industrialisation et l'utilisation d'un circuit, nous avons présenté les principales techniques de test en-ligne. La forte demande de systèmes capables de détecter le plus rapidement possible toute anomalie de fonctionnement a entraîné de nombreuses recherches sur le test en-ligne et plus particulièrement dans le domaine des circuits auto-contrôlables. Ces circuits constituent l'élément de base indispensable à la réalisation de circuits complexes disposants de capacités de détection de pannes. Ces circuits ne sont malheureusement pas adaptés à la réalisation de systèmes destinés à des applications critiques.

Les systèmes critiques sont en effet tous réalisés suivant le même schéma de principe qui se décompose en trois parties, une logique de traitement des données, un étage de sortie et les fonctions sécuritaires de commande (par exemple, l'utilisation de la technique de codage en fréquence).

Les gros problèmes liés à la sécurité se trouvant localisés au niveau de l'étage de sortie : c'est ici qu'interviennent les systèmes *Fail Safe*. Il n'y a en effet, à ce niveau, plus aucun retour d'information qui permette de vérifier que la commande désirée soit parvenue sans erreur à l'actionneur.

Constitués à l'origine à partir de composants discrets spécifiques limitant leur évolution, leur intégration n'a été rendue possible que par de récents travaux.

Nous présentons dans le chapitre suivant, la réalisation matérielle d'une interface VLSI permettant de transformer un circuit auto-contrôlable en un système *Fail Safe*.

Chapitre II :

Conception d'interfaces *Fail Safe* en circuits VLSI.

Chapitre II : Conception d'interfaces *Fail Safe* en circuits VLSI.

1. Introduction

Pour être compétitifs, les produits d'aujourd'hui doivent être de plus en plus performants, de moins en moins chers et fabriqués de plus en plus vite.

Comme signalé au chapitre précédent, ces progrès n'ont été atteints qu'au détriment de la sécurité intrinsèque. L'intégration d'un nombre de plus en plus important de fonctions sur une même puce augmente les risques de pannes et diminue la possibilité de détecter toutes les pannes possibles.

De tels circuits ne sont donc pas compatibles avec des applications exigeant une très forte sûreté de fonctionnement.

Ces applications dites *Fail Safe* sont actuellement réalisées à l'aide de composants discrets répondant à des critères de sécurité très stricts. Leur réalisation est donc très coûteuse et limite leur complexité.

Les circuits digitaux classiques ne délivrent pas de données conformes à la propriété *Fail Safe* telle qu'elle avait été définie initialement **[31]**. Ces circuits sont en effet susceptibles de fournir des réponses erronées qui ne sont pas maîtrisables à la conception. Ces réponses peuvent êtres dues à des pannes de collages ou à des courts-circuits.

Des travaux récents **[17]** ont généralisé la théorie des systèmes *Fail Safe* en proposant une nouvelle approche permettant de concilier les objectifs d'intégration et de sécurité.

La réalisation de systèmes de traitement entièrement *Fail Safe* n'est cependant que très difficilement réalisable, voire impossible. Les paramètres à prendre en compte pour la réalisation de systèmes *Fail Safe* sont en effet très rigoureux et contraignants.

La seule solution réalisable actuellement consiste en l'utilisation d'une interface permettant d'assurer la sécurité des données générées par le système sur lequel elle est ajoutée. Une telle interface peut être utilisée soit avec un seul système de traitement disposant de capacités d'auto-contrôle rigoureuses, soit en dupliquant les systèmes de traitement. C'est cette solution **[33]**, illustrée à la figure suivante, qui est exploitée dans ce chapitre en s'appuyant sur l'étude faite dans **[30]**.

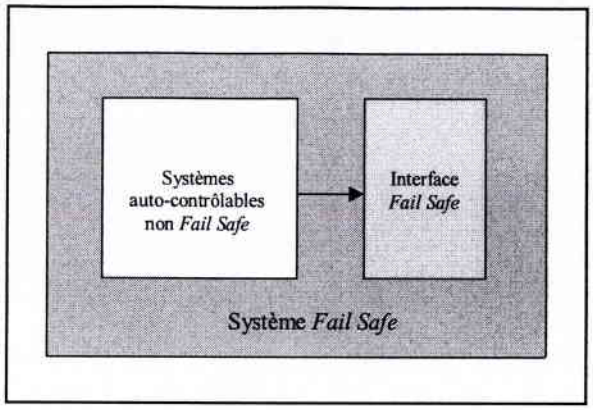


Figure II.1 :Principe du système *Fail Safe* étudié

L'objectif de cette étude est de valider le principe d'une interface *Fail Safe* intégrable en *ASIC* et d'intégrer un outil d'aide à la conception de systèmes *VLSI Fail Safe* dans une chaîne de conception de circuits *ASIC* **[34], [35]**.

Cet outil doit permettre de générer le plus facilement possible l'interface *Fail Safe* permettant de transformer un système de traitement quelconque en système *Fail Safe*.

Ce chapitre présente donc le principe général des systèmes *Fail Safe* avant d'analyser la structure et les propriétés de l'interface proposée. Les problèmes liés à l'implémentation de cette interface et la présentation des différentes cellules réalisées en cellules standards constituent la seconde partie du chapitre.

2. Principe Général des circuits *Fail Safe*

Les circuits *Fail Safe* garantissent, qu'en présence d'une panne, la sortie du système doit prendre soit la valeur correcte, soit une valeur dite *sûre* qui ne peut en aucun cas entraîner de situation dangereuse. Cette propriété est, par exemple, indispensable pour le pilotage d'actionneurs se trouvant en bout de chaîne de nombreux systèmes de surveillance utilisés dans l'industrie chimique et des transports. C'est également le cas des systèmes de signalisation routière et ferroviaire.

Concrètement, un système de commande de passage à niveau *Fail Safe* implique qu'en présence d'une panne, la barrière doit obligatoirement être abaissée, empêchant les véhicules de traverser même en l'absence de train.

Cet exemple met également en évidence le problème de la disponibilité de tels systèmes. En effet, le respect de critères de sécurité très stricts est souvent pénalisant pour l'usager. Ce prix à payer pour une sécurité totale peut être atténué en utilisant des systèmes tolérants aux pannes, qui permettent donc de maintenir le système en fonctionnement pour un nombre donné de pannes.

Le développement des systèmes *Fail Safe* a été pendant longtemps freiné par leur principe de conception. Seul l'utilisation de composants discrets spécifiques convenait à leur réalisation. Les circuits VLSI classiques ne délivrent, en effet, pas de données conformes à la propriété *Fail Safe*. L'occurrence de pannes de collage ou de courts-circuits, par exemple, est susceptible de générer des réponses erronées dont l'état n'est pas maîtrisable à la conception.

Récemment ***NICOLAIDIS*** **[17]** a repris la théorie des systèmes *Fail Safe* en la généralisant afin de pouvoir l'appliquer aux circuits intégrés. Les principes définis dans **[17]** et **[30]** repris dans la suite autorisent l'intégration de systèmes *Fail Safe* au moyen d'une interface. Cette interface permet la discrimination entre les signaux corrects et les signaux erronés et transforme ces derniers en signaux dit *sûrs*.

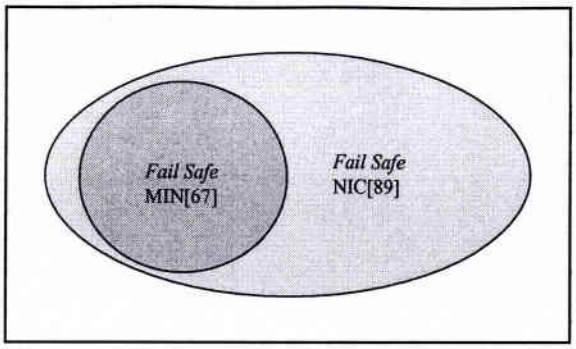


Figure II.2 : Relations entre les différentes définitions des circuits *Fail Safe*

Rappelons la définition d'un circuit *Fail Safe* :

Un système est *Fail Safe* pour un ensemble de pannes prédéfini F, si en présence d'une de ces pannes, la sortie du système prend soit une valeur correcte, soit une valeur de secours dite *sûre*.

Cette définition n'est cependant pas suffisante pour concevoir des systèmes à très haute sûreté de fonctionnement. Si l'on considère un système *Fail Safe* pour un ensemble de panne F, chacune des pannes appartenant à F ne peut produire de situation dangereuse, cependant l'apparition successive de deux de ces pannes peut produire une panne combinée n'appartenant peut-être pas à F. Le système perd alors sa propriété *Fail Safe*.

Il est donc très important qu'un système *Fail Safe* possède des capacités de détection de panne afin de pouvoir les signaler. De tels circuits vérifient donc la définition des circuits *Strongly Fail Safe* :

Un circuit est *Strongly Fail Safe* si il est *Fail Safe* et que toute apparition de panne puisse être détectée.

La détection de ces pannes permet donc l'activation de mécanismes visant à verrouiller le système dans un état sûr. Pour que le système soit rigoureusement sur en présence de panne, il doit donc vérifier la définition de l'*Ultimate Fail Safe Goal* suivante:

Aussi longtemps qu'un circuit délivre des services, il ne doit pas produire de sortie erronée n'appartenant pas à l'espace des sorties sûres.

3. Principe général des interfaces réalisées

3.1. Hypothèses de fonctionnement

L'interface présentée dans la suite permet donc la réalisation d'un système intégré *Fail Safe* à partir d'un système de traitement préalablement dupliqué. Cette interface autorise la réalisation d'un système conforme à l'*Ultimate Fail Safe Goal* en posant les hypothèses suivantes :

Hyp1 : Les pannes apparaissent une à une dans le circuit

Hyp2 : Entre l'occurrence de deux pannes consécutives, il s'écoule un laps de temps suffisamment long pour que, pendant ses différents modes de fonctionnement, le circuit soit soumis à un ensemble d'entées permettant de détecter toute panne.

Etant donné que les mécanismes des défaillances liés à la durée de vie des circuits sont en général très lents, la probabilité d'occurrence simultanée de plusieurs pannes est négligeable. C'est pourquoi la première hypothèse est très souvent utilisée.

La deuxième hypothèse dépend de la structure de l'interface ainsi que de l'application choisie. La conception du système devra permettre l'application des vecteurs de test à des intervalles de temps réguliers même dans le cas où les entrées du système n'évoluent que très rarement.

Le système est réalisé à partir de cellules *Fail Safe* auto-contrôlables afin d'obtenir la propriété *Strongly Fail Safe*. De plus, pour aboutir à l'*Ultimate Fail Safe Goal*, il faut, lors de la détection d'une panne, activer un mécanisme permettant le verrouillage irréversible du système dans un état *sûr*.

Pour obtenir un système dont les sorties sont *Fail Safe* individuellement, on utilise la technique de codage en fréquence. Celle-ci consiste à faire correspondre à un état *non sûr* une fréquence ou un intervalle de fréquences, et à considérer tous les autres états électriques comme des états sûrs. Une telle représentation a l'avantage de convenir parfaitement à la commande des actionneurs se trouvant généralement placés en bout de chaîne des fonctions

critiques. Par ailleurs, en choisissant de manière judicieuse la fréquence correspondant à l'état *non sûr*, il est possible d'éviter qu'elle ne soit produite par des perturbations extérieures.

La solution la plus courante pour réaliser l'auto-test de l'interface consiste à utiliser la technique BIST. Celle-ci présente cependant deux inconvénients :

- D'une part, un accroissement non négligeable de la complexité de l'interface ;
- D'autre part, une panne dans la circuiterie BIST peut générer accidentellement la fréquence correspondant à l'état *non sûr* et provoquer la perte de la propriété *Fail Safe* du circuit.

Pour ces raisons, l'interface développée est basée sur le contrôle concurrent. Cette interface a été conçue de manière à stimuler de façon exhaustive toutes ses parties critiques par les valeurs requises, pendant le fonctionnement normal du système.

3.2. Interface *Fail Safe*

La figure suivante présente le diagramme fonctionnel de l'interface réalisant l'*Ultimate Fail Safe Goal* dans le cas de systèmes de traitement dupliqués.

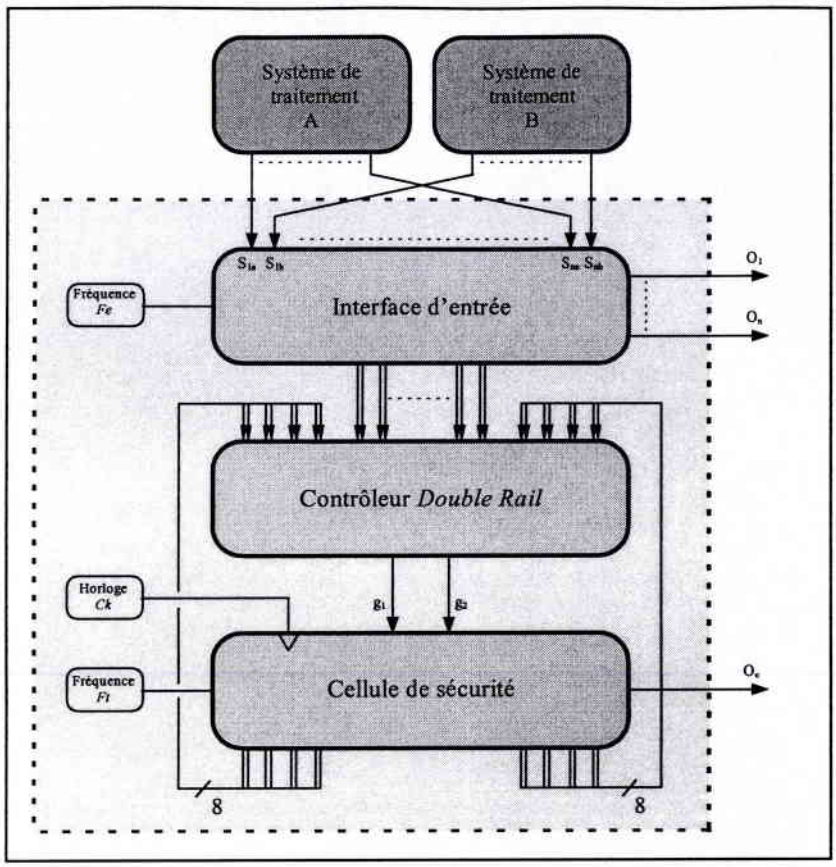


Figure II.3 : Schéma fonctionnel de l'interface *Fail Safe*

On voit clairement apparaître un découpage en trois sous-parties, à savoir, l'interface d'entrée, le contrôleur *Double-Rail* qui assure la propriété *Fail Safe*, et la cellule de sécurité qui permet d'aboutir à l'*Ultimate Fail Safe Goal*.

3.2.1. L'interface d'entrée

En fonctionnement normal (c'est à dire sans panne) les valeurs $S_{ia}=S_{ib}=1$ propagent la fréquence *Fe* jusqu'à la sortie O_i. Cela correspond à l'état *non sûr*. Les valeurs $S_{ia}=S_{ib}=0$ isolent la sortie O_i de la fréquence *Fe* en deux points. Il n'y a donc pas de panne unique qui permette la connexion entre la sortie O_i et la fréquence *Fe* lorsque les sorties du système de traitement sont à zéro. L'interface est donc *Fail Safe* pour les pannes uniques.

Pour rendre l'interface *Strongly Fail Safe*, les pannes internes doivent y être détectées. Les sorties de contrôle codées en *Double-Rail* permettent de détecter toutes les pannes invalidant la propriété. Ainsi, le système global constitué des systèmes de traitement et de l'interface d'entrée est *Strongly Fail Safe*.

3.2.2. Le contrôleur *Double-Rail*

Le rôle du contrôleur *Double-Rail* est de fournir une indication d'erreur sur ses deux sorties lorsqu'une des paires d'entrées signale une erreur.

Le contrôleur vérifie les propriétés d'auto contrôle. Bien qu'il y ait un grand nombre d'ensemble de vecteurs d'entrée, chaque ensemble, composé de quatre mots de code *Double-Rail*, permet à lui seul, de détecter toute panne dans le contrôleur. Si ces quatre vecteurs apparaissent pendant le fonctionnement normal, le contrôleur sera entièrement testé.

3.2.3. La cellule de sécurité

La cellule de sécurité laisse passer la fréquence *Ft* vers la sortie *Oe* tant que les signaux *g1* et *g2* n'indiquent pas la présence d'une panne. Lorsqu'une indication d'erreur apparaît sur les sorties du contrôleur *Double-Rail*, cette indication est mémorisée à l'intérieur de la cellule de sécurité et le reste définitivement, quelles que soient les valeurs ultérieures de *g1* et *g2*. De plus, lorsqu'il y a mémorisation de l'indication d'erreur, la fréquence *Ft* est isolée de la sortie *Oe*. Cette disparition de la fréquence en *Oe* déclenche un mécanisme de coupure d'alimentation qui verrouille le système dans un état *sûr*.

3.2.4. Propriété du système tout entier

Dans le système tout entier, aucune panne unique ne peut détruire la propriété *Strongly Fail Safe* du système. Celui-ci accomplit l'*Ultimate Fail Safe Goal* puisque, après l'occurrence d'une panne, le circuit est bloqué de manière irréversible dans un état *sûr*.

L'interface présentée peut également être utilisée avec des systèmes de traitement auto-contrôlables basés sur des codes de détection d'erreur.

Enfin, sur la base du montage présenté, de nombreuses variantes permettent, soit d'augmenter le nombre de pannes simultanées pour lesquels le système est *Strongly Fail Safe*, soit de réaliser un système tolérant aux pannes.

3.3. Interface tolérante aux pannes

Les circuits *Fail Safe* ont pour fonction de garantir un degré de sécurité maximum en verrouillant le système dans un état *sûr* dès la première anomalie de fonctionnement. Cet objectif est atteint en utilisant l'interface *Fail Safe* présentée précédemment.

Toutefois, pour certaines applications, le verrouillage du système dans l'état sûr, bien que nécessaire, entraîne une situation perturbante. Il serait donc intéressant d'améliorer la disponibilité du système. Cette amélioration ne peut se faire qu'en augmentant le nombre de pannes à partir duquel le système se verrouille.

L'interface présentée ci-dessous apporte une solution à ce problème.

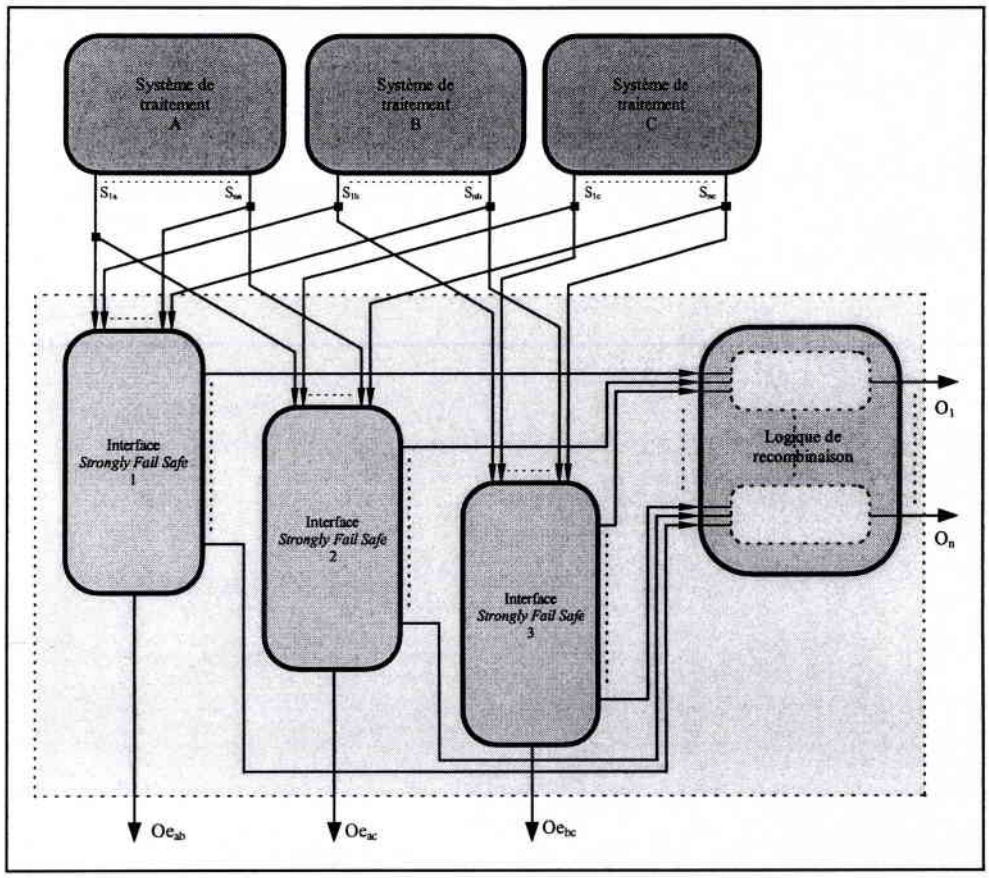


Figure II.4 : Schéma fonctionnel de l'interface tolérante aux pannes

Cette interface est donc basée sur la triplication des systèmes de traitement et présente donc l'inconvénient d'augmenter considérablement la surface du système. La triplication est souvent utilisée pour l'implémentation de systèmes tolérants aux pannes. Un des principes consiste à utiliser un voteur (logique majoritaire) qui fournit un signal de sortie égal à la majorité de ceux présents à son entrée.

Le principe de l'interface présentée à la figure II.4 est quelque peu différent. On utilise trois interfaces identiques à celle présentée au paragraphe précédent, chacune d'elles possédant une ligne d'alimentation spécifique. Chacune des interfaces est reliée à deux

systèmes de traitement, de façon à ce que chaque système de traitement soit contrôlé par deux interfaces.

Lors de la détection d'une panne interne à l'interface, celle-ci se verrouille dans un état *sûr* par une coupure de son alimentation. La logique de recombinaison laisse passer la fréquence *Fe* correspondant à l'état *non sûr* si celui-ci est généré par l'une au moins des trois interfaces.

Ainsi, il faudra deux pannes dans les systèmes de traitement ou trois pannes dans les interfaces pour verrouiller le système entier dans l'état sûr.

4. Mise en œuvre

4.1. Introduction

Cette partie est consacrée à la mise en œuvre de l'interface *Fail Safe*. L'objectif est de réaliser physiquement l'interface afin d'évaluer les problèmes que poseraient l'industrialisation d'un tel circuit. Chacune des trois parties de l'interface est étudiée de façon très détaillée afin de donner un aperçu de la méthodologie à suivre pour la conception de circuits *Fail Safe*. L'étude théorique, réalisée pour un nombre donné de sorties du système de traitement, reste valable pour un nombre quelconque.

Il n'en est pas de même pour la mise en œuvre, où le nombre de sorties du système de traitement détermine l'ordre de connexion entre les différentes fonctions de l'interface. Ces contraintes sont détaillées pour chacune des sous-fonctions concernées. La relation entre le nombre de sorties du système de traitement et l'organisation de la cellule est également précisée.

Cette section s'attache donc à présenter les propriétés de chacune des parties de l'interface et d'en préciser les limites.

4.2. L'interface d'entrée

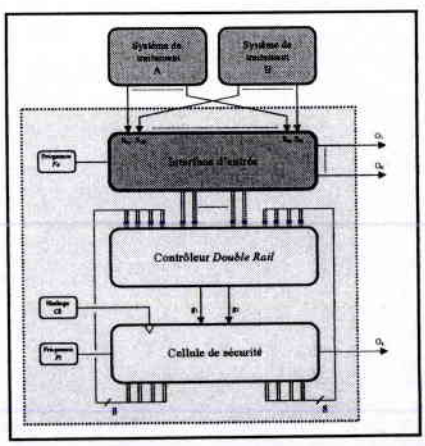


Figure II.5 : Localisation de l'interface d'entrée

La technique du codage en fréquence consiste à générer une fréquence lorsque l'actionneur doit être commandé, et de ne générer aucune fréquence dans le cas contraire.

C'est cette fonction qui est assurée par l'interface d'entrée. Celle-ci laisse passer une fréquence *Fe* lorsque la sortie du système de traitement est à l'état logique 1, et à déconnecter cette fréquence lorsque la sortie du système de traitement est à l'état logique 0. Ainsi, l'état *non sûr* correspondra à la présence de cette fréquence alors que tous les autres états électriques correspondront à l'état *sûr*, c'est à dire l'état qui doit être généré en présence d'une panne.

Le schéma de la figure II.6 permet de réaliser cette fonction. On utilise deux portes logiques connectées en série, pour isoler ou non la sortie, de la fréquence *Fe*. L'occurrence de deux pannes identiques dans deux portes identiques étant plus probable que deux pannes de type différent dans des portes différentes il est donc préférable d'utiliser deux portes logiques différentes (une NOR et une NAND). Une panne dans chacune des portes peut en effet connecter *Fe* à la sortie de l'interface et donc invalider la propriété *Fail Safe* de l'interface.

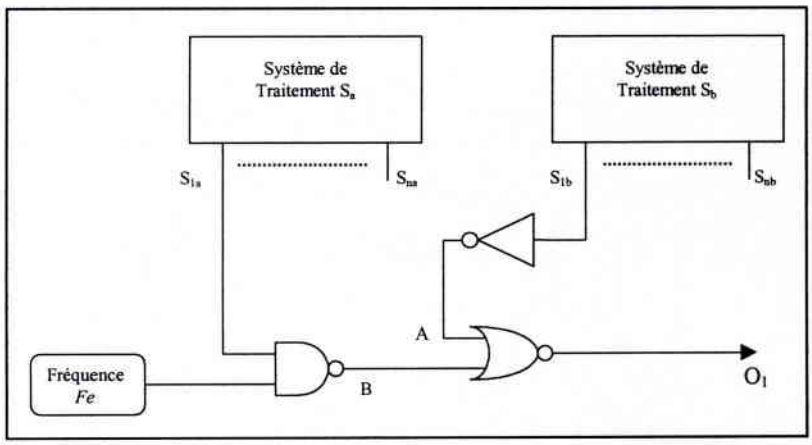


Figure II.6 : Interface *Fail Safe* pour des systèmes dupliqués

En fonctionnement normal (sans panne) les valeurs $S_{1a} = 1$ et $S_{1b} = 1$ propagent la fréquence *Fe* vers la sortie de l'interface, alors que les valeurs $S_{1a} = 0$ et $S_{1b} = 0$ isolent cette fréquence de la sortie (c'est l'état sûr).

Il n'y a donc pas de panne unique, dans l'interface ou dans l'un des systèmes de traitement, qui puisse connecter la sortie à la fréquence *Fe* lorsqu'elle aurait du être isolée ($S_{1a} = S_{1b} = 0$). L'interface est donc *Fail Safe* pour les pannes uniques.

Les chronogrammes de fonctionnement sont représentés à la figure suivante.

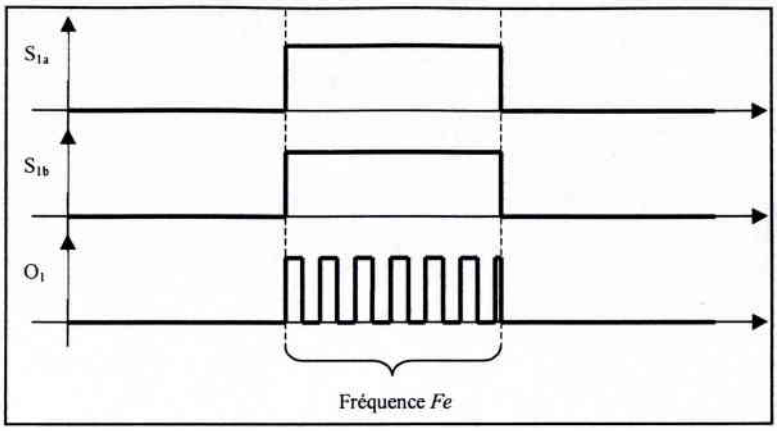


Figure II.7 : Signaux générés par l'interface d'entrée

Pour transformer cette interface en une interface *Strongly Fail Safe*, il faut lui ajouter des mécanismes d'auto-test. Un des moyens d'y parvenir est d'utiliser la technique BIST. Il a cependant été mentionné dans la section précédente qu'il était préférable de remplacer cette technique par une méthode de contrôle concurrent lorsque cette dernière était applicable.

Pour ce faire, l'interface est dupliquée, la détection de panne se faisant alors par comparaison des deux branches de l'interface. La validité de cette technique repose sur les deux conditions suivantes :

•Pendant le fonctionnement normal du système, les éléments de l'interface reçoivent toutes les valeurs d'entrée permettant de détecter les pannes invalidant la propriété *Strongly Fail Safe*.

•Les effets des pannes doivent être propagés vers les points observables du circuit (les entrées du contrôleur).

On constate que pour le schéma de la figure II.6, la première condition n'est pas remplie. La porte NAND est en effet testée exhaustivement pendant le fonctionnement normal, alors que la porte NOR ne reçoit que les valeurs 00, 01, et 11, qui ne permettent pas de détecter toutes les pannes possibles dans cette porte.

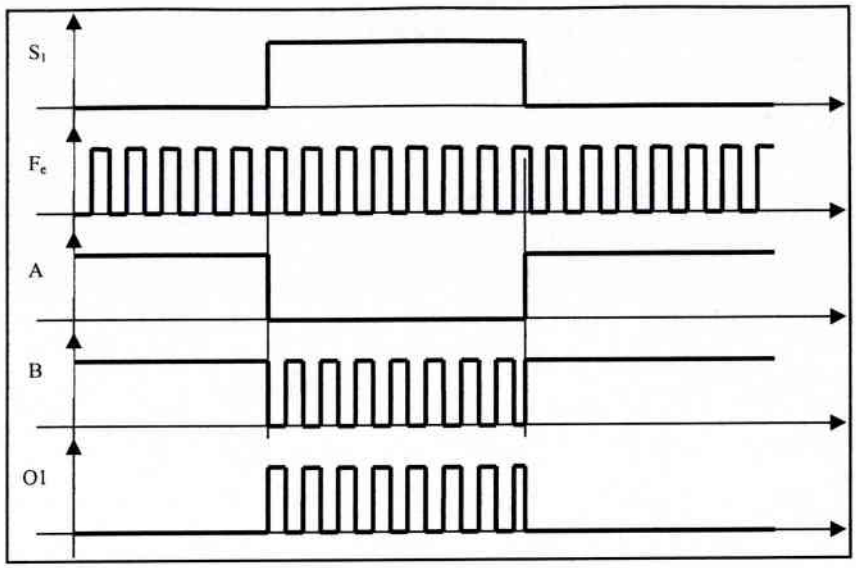


Figure II.8 : Signaux générés dans l'interface d'entrée

Par exemple, la panne de collage à 0 de l'entrée *A* de la porte qui invalide la propriété *Strongly Fail Safe* n'est pas détectée. Pour qu'elle le soit, il faudrait lui appliquer les vecteurs 00, 01, et 10. Ceci est obtenu en inversant la sortie de la porte NAND comme le montre la figure II.9.

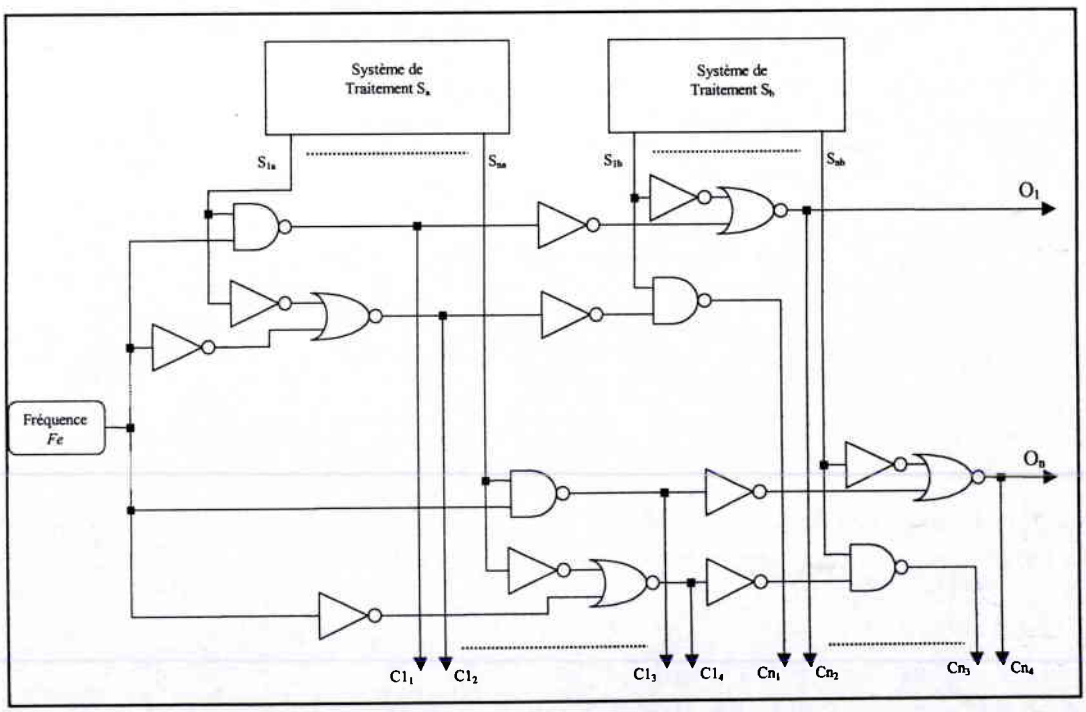


Figure II.9 : Interface *Strongly Fail Safe*

Pour remplir la deuxième condition, l'interface est dupliquée, et les sorties des portes NOR et NAND sont contrôlées par un contrôleur double rail. Toutes les branches générant les sorties *O1*, ... , *On* sont donc contrôlées.

Le problème suivant concerne la détection des pannes de collage fermé et de collage ouvert des transistors. Prenons l'exemple de la figure II.10.

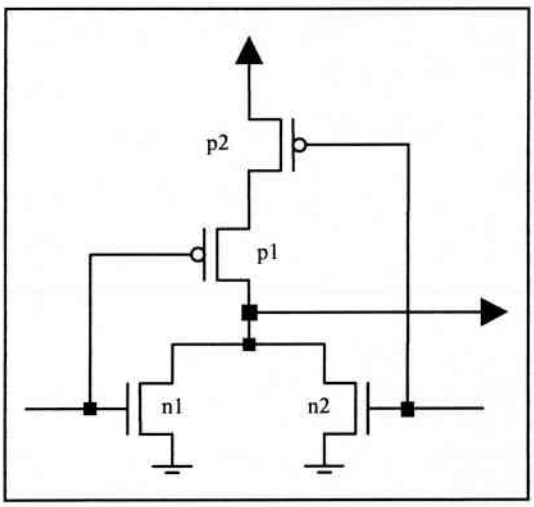


Figure II.10 : Structure d'une porte NOR.

La détection des pannes de collage ouvert nécessite l'application de deux vecteurs, l'un pour initialiser la sortie de la porte et le deuxième pour détecter la panne. Les vecteurs à appliquer pour chacun des transistors dans l'exemple de la porte NOR sont alors les suivants :

- transistor *n1* : 00 puis 10
- transistor *n2* : 00 puis 01
- transistors *p1* et *p2* : 01 (ou 10) puis 00

Les pannes de collage fermé dans les portes CMOS statiques produisent des niveaux indéterminés en sortie. L'interprétation de telles valeurs par la logique en aval étant imprévisible, la détection de telles pannes n'est donc pas garantie.

La résistance de conduction d'un transistor de *type n* est cependant très inférieure à celle de deux transistors de *type p*. Ainsi, si les réseaux de *type n* et *p* de la porte NOR conduisent simultanément, la sortie sera à un niveau logique 0. Ce comportement de la porte (appelé *n dominant*) peut être renforcé en augmentant la largeur du transistor de *type n*.

Les pannes de collage fermé des transistors *n1* et *n2* sont donc détectées par le vecteur 00. Les pannes de collage fermé des transistors *p1* et *p2* ne modifient pas la fonction de la porte, (du fait de son comportement *n dominant*) mais ne sont pas détectées. Leur détection n'est pas nécessaire car ces pannes latentes n'affectent pas la détectabilité des autres pannes.

Tous les raisonnements effectués sur la porte NOR sont applicables à la porte NAND qui a un comportement *p dominant*.

Les cellules standards étant réalisées selon ce principe, toutes les pannes susceptibles d'invalider la propriété *Fail Safe* sont donc détectées. L'interface est donc *Strongly Fail Safe* pour toutes ses pannes internes.

Cependant, la structure de la figure II.9 ne permet pas la détection des pannes dans les systèmes de traitement. La propriété *Fail Safe* y est assurée, car si l'un des systèmes de traitement délivre une valeur erronée, la sortie correspondante sera dans un état *sûr*. Cependant, la panne ne sera pas détectée et la propriété *Fail Safe* de l'interface sera perdue en cas d'apparition d'une autre panne. Il serait donc intéressant de détecter l'apparition des pannes dans les systèmes de traitement pour permettre le remplacement du système ou le blocage de l'interface dans un état *sûr*. Ces problèmes sont résolus en modifiant l'interface de la figure II.9 pour obtenir celle de la figure II.11.

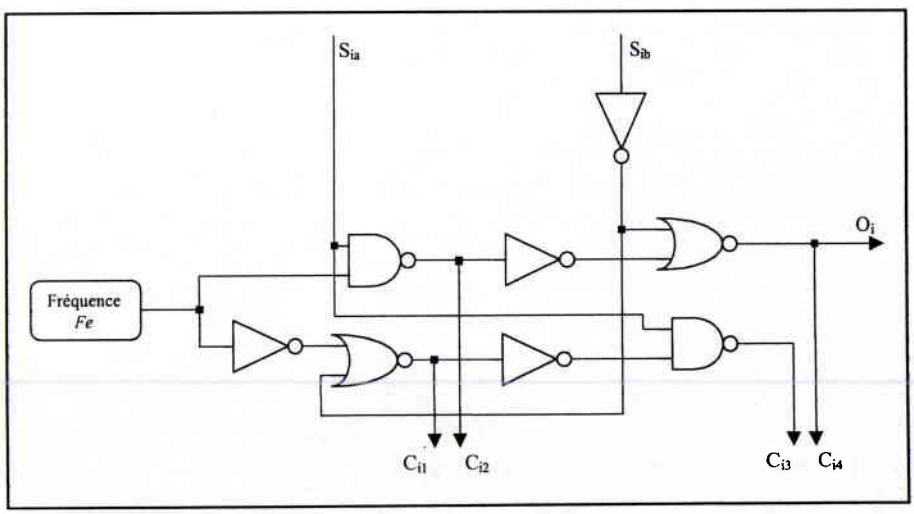


Figure II.11 : Interface permettant le test des systèmes de traitement

Les chronogrammes générés sont représentés à la figure suivante :

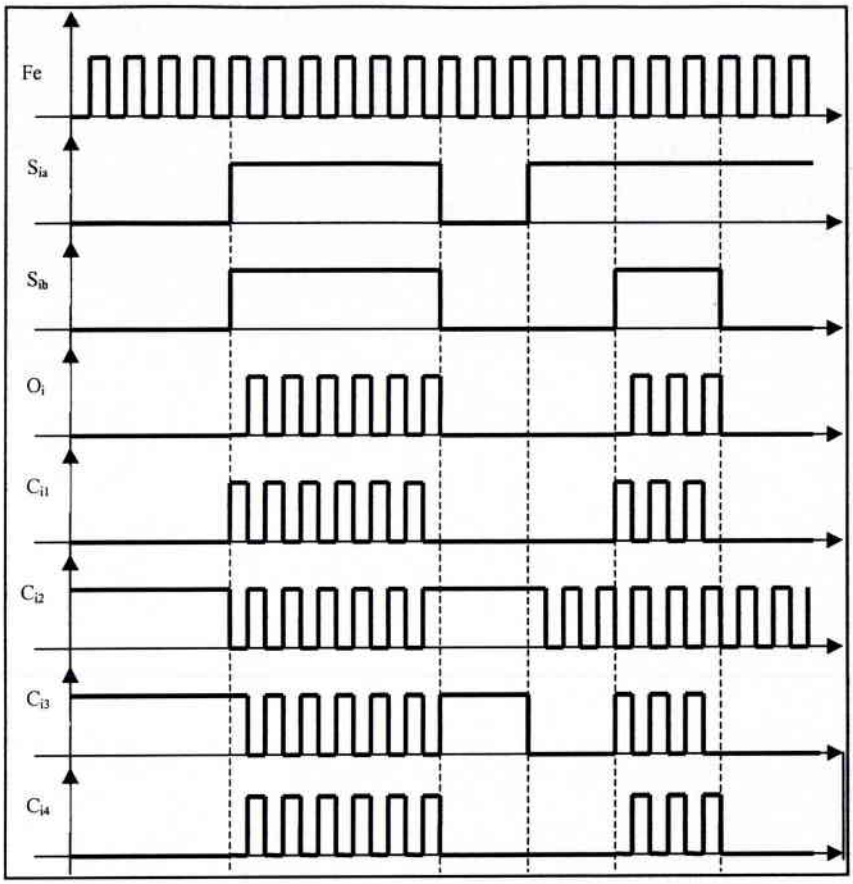


Figure II.12 : Signaux générés dans l'interface d'entrée

L'interface d'entrée peut donc se décomposer en deux sous parties :

•Une chaîne de base qui correspond à l'interface de la figure II.6

•Une chaîne redondante qui fournit en tout point le complément logique de la chaîne de base.

Les signaux présents en différents nœuds de ces deux chaînes fournissent donc un code correspondant au code *Double Rail*. Ils seront donc reliés à un contrôleur *Double Rail* qui détectera toutes les pannes affectant soit l'interface d'entrée soit l'un des deux systèmes de traitement.

L'interface est donc *Strongly Fail Safe* pour toute panne unique affectant soit l'interface d'entrée soit l'un des systèmes de traitement.

4.3. Le contrôleur *Double Rail*

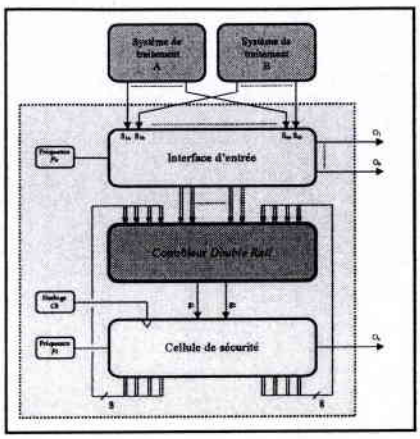


Figure II.13 : Localisation du contrôleur *Double Rail*

4.3.1. Fonction globale du contrôleur *Double Rail*

Le contrôleur *Double Rail* de l'interface étudiée a pour fonction de détecter l'apparition d'une panne, soit dans l'interface d'entrée, soit dans la cellule de sécurité. Pour que la propriété *Strongly Fail Safe* de l'interface soit vérifiée, il faut également garantir que le contrôleur *Double Rail* puisse détecter ses propres pannes. C'est sur ce point que la mise en œuvre du contrôleur est délicate.

Un contrôleur *Double Rail* compare constamment les états de ces m paires d'entrées codées en *Double Rail*, et signale, sur ces sorties, toute manifestation d'erreur sur l'une de ses entrées.

Dans un codage *Double Rail*, les deux signaux logiques formant une paire *Double Rail* doivent toujours être dans un état opposé. Une erreur se manifeste par la présence de deux états identiques sur une même paire. Ainsi, tant que chaque paire d'entrée du contrôleur *Double Rail* présente deux états opposés, les deux sorties sont dans un état opposé. Si au moins une des paires d'entrée présente deux états identiques, alors les sorties du contrôleur *Double Rail* sont dans un même état, permettant donc la détection de la panne. La structure du contrôleur utilisé (pour $m = 2$) est présentée à la figure II.14.

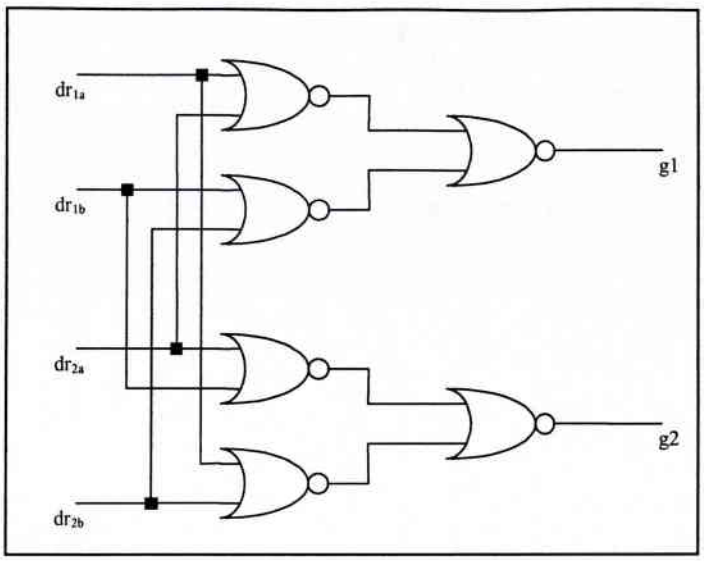


Figure II.14 : Structure du contrôleur *Double Rail* (m=2)

En fonctionnement sans panne, les équations de $g1$ et $g2$ sont les suivantes :

- $g1 = dr1a \oplus dr2a = dr1b \oplus dr2b$
- $g2 = \overline{g1}$

En présence d'une erreur sur l'une des paires d'entrée :

- $g1 = g2$

Le contrôleur *Double Rail* permet donc de signaler instantanément, sur ses sorties, l'apparition d'une panne dans l'interface d'entrée ou dans la cellule de sécurité. Pour garantir la propriété *Strongly Fail Safe* de l'interface globale, il faut en outre, que le contrôleur soit capable de détecter ses propres pannes. Une panne interne pouvant, par exemple, empêcher la détection d'une erreur présente à l'une des paires d'entrée.

4.3.2. Auto-test du contrôleur *Double Rail*

Pour garantir la propriété *Fail Safe* de l'interface entière, le contrôleur *Double Rail* ***CARTER*** **[29]** doit donc être capable de détecter ses propres pannes. Un grand nombre d'ensembles de quatre vecteurs d'entrée (4^{m-2}, m étant le nombre de paires d'entrées) permet de détecter une panne unique dans le contrôleur *Double Rail*.

Il faut donc s'assurer que ces quatre vecteurs apparaissent au moins une fois à l'entrée du contrôleur *Double Rail* pendant le fonctionnement de l'interface.

4.3.2.1. Etude d'un cas général

Pour un contrôleur à deux paires d'entrée A et B, cet ensemble de vecteurs est donné dans le tableau suivant où S représente l'état de la paire de sortie :

A	B	S
01	01	01
01	10	10
10	01	10
10	10	01

Tableau n°1

Si l'on connecte la sortie de ce contrôleur sur un second contrôleur *Double Rail* dont l'autre paire d'entrée est connectée à une paire d'entrée C, on forme un contrôleur *Double Rail* à trois paires d'entrées (m=3). Pour que le contrôleur dont les entrées sont S et C soit entièrement contrôlé, (en gardant les mêmes vecteurs A et B) l'entrée C doit prendre un des quatre ensembles de valeurs suivants :

S	C
01	01
10	01
10	10
01	10

Tableau n°2

S	C
01	10
10	10
10	01
01	01

Tableau n°3

S	C
01	01
10	10
10	01
01	10

Tableau n°4

S	C
01	10
10	01
10	10
01	01

Tableau n°5

Quatre ensembles de quatre vecteurs d'entrée permettent ainsi l'auto-test du contrôleur avec $m = 3$.

Prenons l'exemple d'un contrôleur *Double Rail* d'ordre 3 (m = 3) connecté selon la figure suivante où A, B et C prennent les états du tableau 6 :

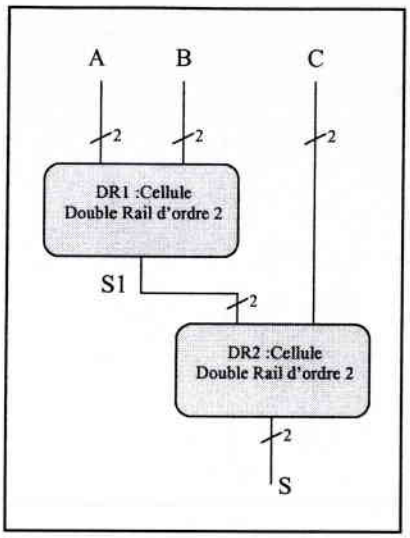


Figure II.15 : Cellule Double Rail d'ordre 3 exemple 1

A	B	C	S1
01	01	01	01
01	10	01	10
10	01	10	10
10	10	10	01

Tableau n°6

Le contrôleur DR1 voit donc les couples de vecteurs d'entrée (A, B) correspondant au tableau n°1 et est donc entièrement testé. La sortie intermédiaire S1 délivre alors la séquence correspondant à la sortie S du tableau n°1.

Le contrôleur DR2 voit donc les couples de vecteurs d'entrée (S1, C) correspondants au tableau n°2 et est donc également totalement testé. Le contrôleur *Double Rail* d'ordre 3 est donc entièrement testé.

Reprenons ce contrôleur d'ordre 3 en le connectant cette fois selon la figure II.16 suivante où A, B, C gardent les états du tableau n°6 :

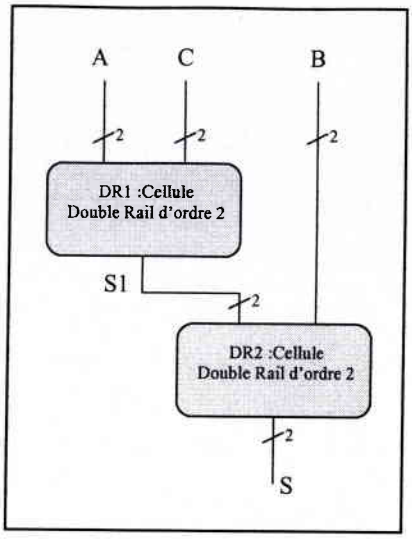


Figure II.16 : Cellule Double Rail d'ordre 3 exemple 2

Le contrôleur DR1 voit cette fois donc les couples de vecteurs d'entrée (A, C) correspondants au tableau n°7. Il manque donc deux vecteurs nécessaires au test complet de contrôleur DR1.

Le contrôleur DR2 voit donc les couples de vecteurs d'entrée (S1, B) correspondants au tableau n°8 et n'est donc pas complètement testé.

Ce contrôleur *Double Rail* d'ordre 3 n'est donc pas entièrement testé, puisque au moins l'un des deux contrôleurs le composant n'est pas exhaustivement testé. Ce contrôleur ne permet donc pas de garantir la propriété *Strongly Fail Safe*.

A	C
01	01
01	01
10	10
10	10

Tableau n°7

S1	B
01	01
01	10
01	01
01	10

Tableau n°8

Cet exemple illustre l'importance de l'ordre de connexion des différentes paires d'entrée au niveau du contrôleur *Double Rail* sur ses capacités d'auto-test. Il faut donc toujours veiller à ce que les vecteurs présents à l'entrée du contrôleur permettent son auto-test.

4.3.2.2. Etude du contrôleur *Double Rail* de l'interface proposée

Dans l'interface étudiée, les lignes de contrôle codées en *Double Rail* proviennent des interfaces d'entrée et de la cellule de sécurité. Le problème de ce montage vient du fait que, pour une interface d'entrée donnée, les deux paires de lignes de contrôle sont corrélées. Ainsi, le vecteur formé de ces deux paires de lignes, ne prend pas toutes les valeurs possibles. Il en est de même pour les lignes de contrôle de la cellule de sécurité. De plus, pour celle-ci, les signaux *Xij* sont identiques aux signaux *Yij*. Par contre, les signaux *Oij* et *Bij* sont indépendants des signaux *Xij* et *Yij*.

Pour que l'interface soit auto-testée, il faut que l'un au moins des ensembles de quatre vecteurs cités précédemment, apparaisse à l'entrée du contrôleur au cours de son fonctionnement. Le problème réside dans le fait que l'on ne peut pas agir sur les états que prennent les différentes lignes du circuit. La seule solution consiste à modifier l'ordre de connexion de ces lignes sur le contrôleur *Double Rail*, en tenant compte des relations les liant.

Il faut donc mélanger les lignes provenant de la cellule de sécurité et celles provenant de l'interface d'entrée afin d'éviter que deux lignes adjacentes soient corrélées.

Il faut ensuite vérifier que, pour chaque configuration choisie, le contrôleur *Double Rail* sera exhaustivement testé.

Cette méthode est applicable pour des contrôleurs *Double Rail* d'ordre m réalisés par la mise en cascade de (m-1) contrôleurs d'ordre 2.

En raison de la structure en cascade du contrôleur *Double Rail*, sa sortie ne se stabilise qu'après un temps τ. Ce temps est fonction du nombre d'entrée du contrôleur et de la technologie utilisée pour l'implémentation de l'interface.

Il doit être pris en compte pour déterminer la fréquence d'utilisation de l'interface ainsi que pour la réalisation de la cellule de sécurité. Pendant ce laps temps, les indications du

contrôleur *Double Rail* ne devront pas être prises en compte puisqu'elles peuvent indiquer la présence d'une panne fictive.

4.4. La cellule de sécurité

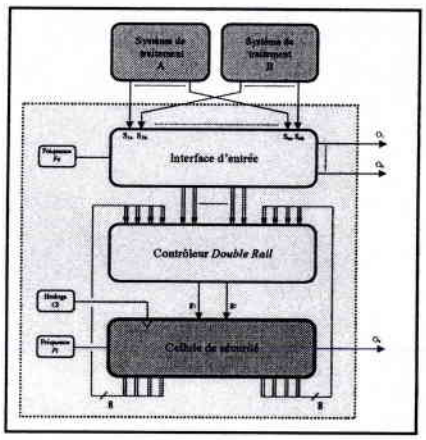


Figure II.17 : Localisation de la cellule de sécurité

La fonction de la cellule de sécurité est de verrouiller, de façon irréversible, l'interface dans un état *sûr*, lors de la détection d'une panne par le contrôleur *Double Rail*. Elle garantit donc l'*Ultimate Fail Safe Goal*.

Les chronogrammes de l'interface sont représentés à la figure suivante :

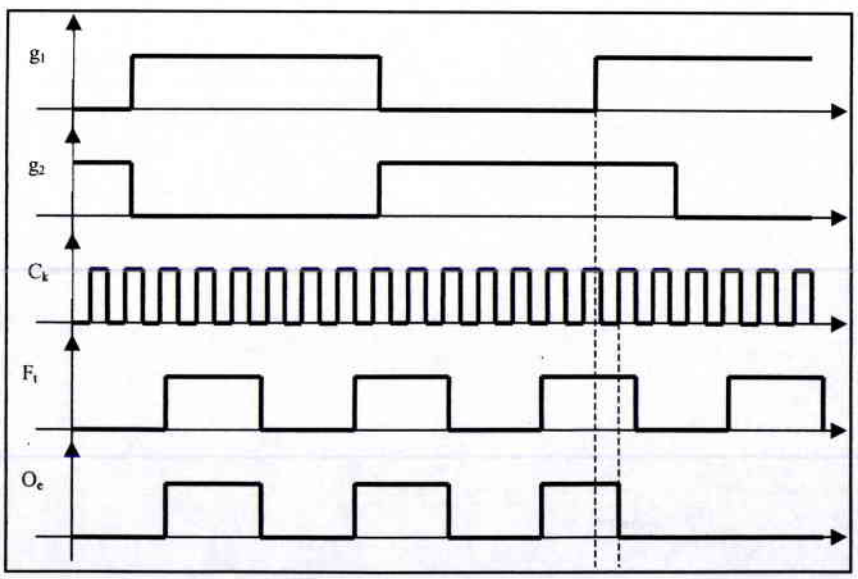


Figure II.18 : Signaux générés par la cellule de sécurité

La cellule de sécurité se décompose en deux sous-fonctions :

- L'indicateur d'erreur
- Le transducteur

L'indicateur d'erreur mémorise la première indication d'erreur issue du contrôleur *Double Rail.* Pour que cette indication ne puisse pas être perdue par une panne dans le mécanisme de mémorisation, il faut alors verrouiller le système dans un état sûr. C'est le rôle du transducteur.

Le transducteur transforme l'indication d'erreur codée dans le code *Double Rail* en une indication codée en fréquence. Le principe de codage en fréquence est le même que celui utilisé dans l'interface *Fail Safe.* Le transducteur est lui-même auto-testé par l'intermédiaire du contrôleur *Double Rail* de sorte que la cellule de sécurité soit *Strongly Fail Safe.*

L'*Ultimate Fail Safe Goal* est assuré par un mécanisme externe dont la fonction est de couper l'alimentation de l'interface lors d'une indication d'erreur issue du transducteur. L'interface est alors verrouillée dans un état *sûr* de manière irréversible.

La figure suivante présente le schéma fonctionnel de la cellule de sécurité :

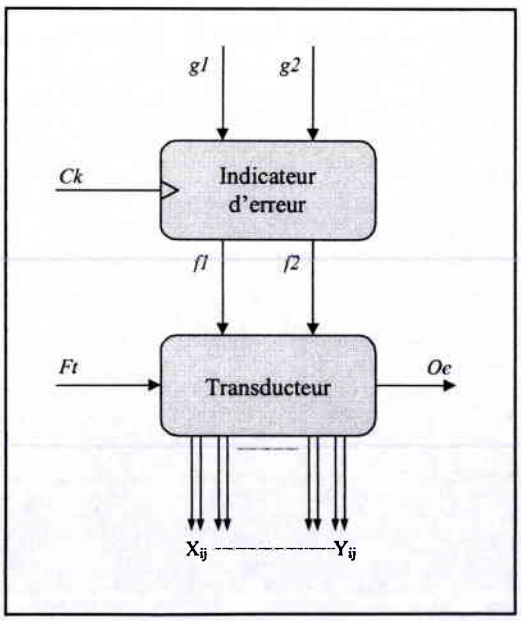


Figure II.19: Schéma fonctionnel de la cellule de sécurité

4.4.1. L'indicateur d'erreur

Les indications d'erreur issues du contrôleur *Double Rail* peuvent être transitoires dans le cas où la panne ne serait détectable que par un faible nombre de vecteurs d'entrée du contrôleur *Double Rail*, ou dans le cas d'une panne de type transitoire. Il faut donc mémoriser la première apparition d'une indication d'erreur pour verrouiller le système dans un état *sûr*. Il a été mentionné précédemment qu'en raison des temps de propagation et de la structure du contrôleur *Double Rail*, des indications d'erreur erronées peuvent apparaître temporairement à la sortie du contrôleur.

Le rôle de l'indicateur d'erreur est également de filtrer ces fausses indications d'erreur.

Sa structure interne est présentée à la figure II.20.

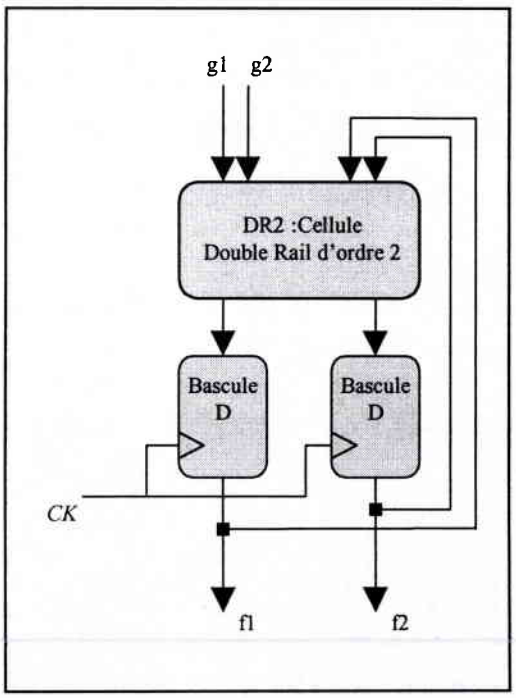


Figure II.20 : Structure de l'indicateur d'erreur

Le principe de fonctionnement de l'indicateur d'erreur est le suivant :

4.4.1.1. Pour les pannes extérieures à l'indicateur d'erreur

Lorsqu'une indication d'erreur apparaît à l'entrée de l'indicateur, (c'est à dire *g1g2* = 00 ou 11) au front montant de l'horloge suivant, cette indication est présente en *f1f2* et réinjectée sur la deuxième paire d'entrée de la cellule *Double Rail*. Ainsi, du fait de la propriété "*code disjoint*" de cette cellule, cette indication d'erreur restera mémorisée en *f1f2* quelles que soient les valeurs ultérieures de *g1g2*.

De plus, en cas de mémorisation d'erreur, la sortie *f1f2* changera au plus une fois d'état, (de 00 à 11 ou inversement selon la nature de la cellule *Double Rail*) si la cellule de contrôleur *Double Rail* est non inversante.

4.4.1.2. Pour les pannes propres à l'indicateur d'erreur

Toutes les pannes affectant, soit la cellule *Double Rail*, soit les bascules de sortie, sont détectables en appliquant les séquences 01,10,01,10.... en entrée. La détection d'une panne se traduit alors par une indication d'erreur en *f1f2* qui est alors mémorisée.

Les valeurs de *f1f2* produites par la détection d'une panne dans l'indicateur ne changent plus.

L'indication d'erreur mémorisée en *f1f2*, utilisée pour verrouiller le système dans un état sûr, ne doit pas être affectée par l'apparition d'une nouvelle panne. L'ajout d'un transducteur permet d'éviter que la mémorisation d'une erreur ne soit annulée par une panne ultérieure.

Pour y parvenir, l'indicateur d'erreur doit être transformé en un indicateur *Strongly Fail Safe*. Il est donc combiné au transducteur décrit ci dessous.

4.4.2. Le transducteur

La fonction du transducteur est de transformer l'indication d'erreur, issue de l'indicateur d'erreur, en un signal *Fail Safe* codé en fréquence. En l'absence de panne dans l'interface, le transducteur génère une fréquence *Ft* sur sa sortie *Oe*. En présence d'une indication d'erreur, aucune fréquence ne doit être présente en *Oe*. Le principe est donc identique à celui de l'interface d'entrée. En outre, pour que l'ensemble de la cellule de sécurité soit *Strongly Fail Safe*, il faut que le transducteur soit capable de détecter ses propres pannes. Comme dans le

cas de l'interface d'entrée, une chaîne redondante permet de disposer de noeuds de test codés en *Double Rail*. Ces lignes de contrôle sont reliées au contrôleur *Double Rail*. Cette boucle permet de garantir la propriété *Strongly Fail Safe* de la cellule de sécurité et donc de l'interface globale.

La structure du transducteur est donnée à la figure suivante :

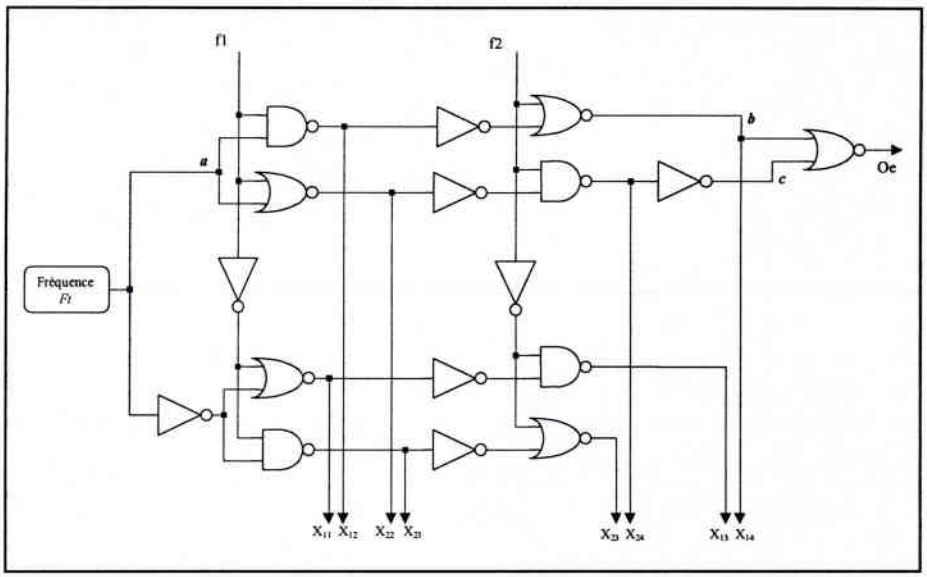


Figure II.21 : Structure du transducteur

La fonction désirée est réalisée par les deux branches *ab* et *ac* similaires à celles proposées figure II.9. Une branche connecte *Ft* à *Oe* pour *f1f2* = 01 et l'autre pour *f1f2* = 10. Ces branches sont testées pendant le fonctionnement normal du système. Les deux branches sont reliées à une porte NAND appelée "logique de recombinaison". Lorsque *f1f2* indique une erreur, les signaux b et c sont déconnectés de la fréquence *Ft*. Cependant, en raison des temps de propagation, la variation périodique de *f1f2* de l'état 00 à l'état 11 et inversement peut conduire à des variations imprévisibles en *Oe*. De telles variations peuvent recréer la fréquence *Ft* en *Oe*. Toutefois, il a été précisé précédemment qu'en cas d'indication d'erreur, les signaux *f1f2* ne peuvent changer au plus qu'une seule fois d'état, empêchant une telle situation.

La structure finale de l'indicateur d'erreur est donc la suivante :

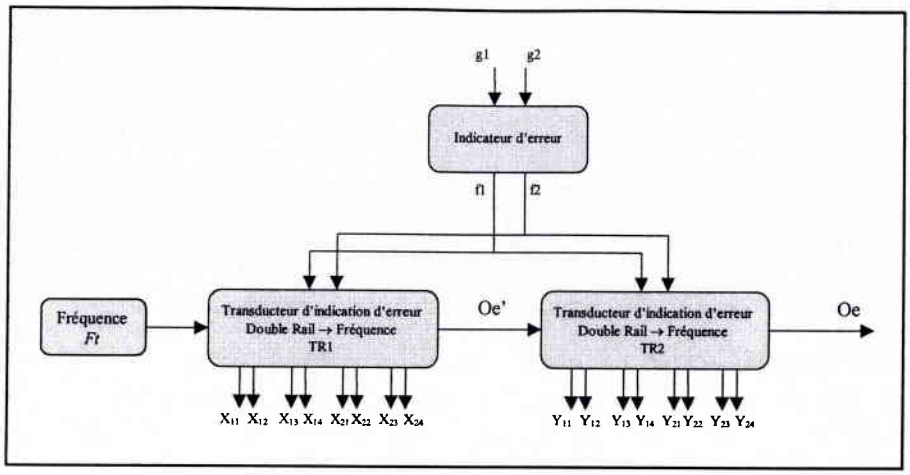


Figure II.22 : Composition de la cellule de sécurité SFS

La présence de deux transducteurs en série est imposée par le fait qu'une panne dans le transducteur peut permettre la génération de *Ft* en *Oe*. Cette panne serait obligatoirement détectée et provoquerait une indication d'erreur, mais elle pourrait rendre impossible la déconnexion de *Ft* à *Oe'*. L'utilisation d'un second transducteur en série permet donc de pallier ce problème.

4.4.3. Le mécanisme de coupure d'alimentation

Ce mécanisme externe, présenté à la figure II.23, permet de verrouiller le système dans un état sûr de manière irréversible, en coupant l'alimentation. Cette coupure s'effectue lorsque la sortie *Oe* est déconnectée de la fréquence *Ft*. Ce mécanisme, couramment utilisé pour ce type d'application, nécessite l'utilisation de composants *Fail Safe* discrets tels que les relais. Le circuit proposé est *Fail Safe* pour toutes les pannes simples ou multiples affectant ses composants.

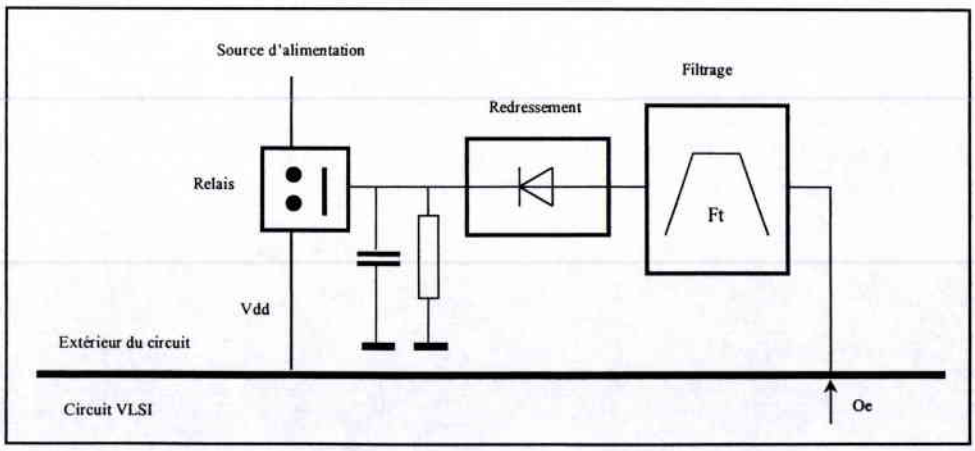


Figure II.23 :Mécanisme de coupure d'alimentation

4.5. Conclusion

Nous venons donc de présenter les différentes parties de l'interface *Fail Safe* conçues dans la perspective d'une réalisation pratique en prenant en compte des contraintes liées à la technologie utilisée.

L'assemblage de ces trois parties n'est toutefois pas suffisant pour garantir le bon fonctionnement du circuit. Cette interconnexion doit, en effet, se faire en respectant un certain nombre de contraintes spécifiques à l'application désirée. De même, le choix des fréquences utilisées et les relations les liants ne peuvent se faire de façon arbitraire. Tous ces choix ont une influence sur l'exhaustivité du test des différentes parties de l'interface.

Les solutions retenues, ainsi que le logiciel et la technologie utilisés pour la réalisation du prototype, sont développées dans la prochaine section.

5. Réalisation physique

5.1. Introduction

Bien que conçue dans l'objectif d'une application physique, l'interface présentée dans le chapitre précédent ne peut être directement implémentée. Sa réalisation nécessite plusieurs contraintes de conception et de connexions, explicitées dans la suite. L'interface *Fail Safe* a donc été réalisée dans une technologie donnée, en tenant compte de ces contraintes. Les résultats obtenus sont présentés dans la suite.

Cette réalisation a nécessité le développement de bibliothèques de cellules *Fail Safe*. Ces cellules ont alors été utilisées pour générer directement l'interface *Fail Safe*. Cet outil a été réalisé dans une seule technologie. Sa portabilité vers une autre technologie n'est toutefois pas assurée. Cette section s'attache donc à étudier les limites que pose cette évolution.

Nous présentons d'abord les contraintes environnementales et technologiques retenues avant de présenter les contraintes structurelles.

5.2. Méthodologie

Le processus de conception d'un circuit intégré allant d'une description comportementale de haut niveau vers le dessin des masques est, dans le cas général, difficilement maîtrisable dans sa totalité.

La définition de démarches standards pour la conception de circuits intégrés permet de mieux maîtriser la complexité du processus de conception. Ces démarches sont généralement appelées méthodologie de conception. Elles permettent de limiter les domaines de connaissance requis, en définissant des primitives permettant de cacher les détails de la conception. L'utilisation d'une bibliothèque standard, par exemple, permet de concevoir un circuit sans se préoccuper des détails de fabrication.

C'est cette voie qui a été étudiée ici, en réalisant, dans un premier temps une bibliothèque de cellules comprenant les différentes parties de l'interface *Fail Safe*. Cette solution a ensuite été améliorée en réalisant une bibliothèque de cellule comprenant des interfaces *Fail Safe* complètes dont le nombre d'entrées est paramétrable. Ces cellules sont

alors aisément utilisables par un outil permettant la transformation automatique d'un système quelconque en un système *Fail Safe*.

5.2.1. Full custom ou pré-caractérisé ?

Une méthodologie de conception repose sur un style de conception. A chaque style de conception correspond généralement un mode d'organisation des circuits intégrés. On distingue quatre types d'organisations.

•Les circuits organisés en réseaux programmables : dans cette catégorie on peut citer les PLAs et les ROMs.

•Les circuits organisés en cellules pré-caractérisées : Ces cellules sont indépendantes du circuit qu'elles peuvent constituer. Elles sont généralement peu complexes (une dizaine de transistors) rarement plus qu'une porte ou une bascule. Les cellules sont le plus souvent organisées selon une topologie standard afin de faciliter leur assemblage.

•Les circuits organisés en macro-blocs : Ce sont généralement des circuits qui réalisent des fonctions complexes telles que des mémoires ou des opérateurs. Un macro-bloc peut être un réseau programmable tel qu'une ROM ou un PLA. Il peut être aussi une simple cellule. Aucune restriction n'est faite sur l'organisation et la taille de ces macro-blocs. Leur structure régulière permet leur génération par un compilateur de silicium.

•Les circuits organisés à la demande (full custom) : Le processus de conception dans ce style n'utilise pas nécessairement des modèles pré-définis. Les éléments de base qui constituent le circuit sont définis durant le processus de conception. Ce type d'organisation permet d'obtenir de meilleures performances (en terme de compromis surface - vitesse - consommation) et par la suite de concevoir des circuits plus complexes. Par contre le coût de conception est plus élevé que pour les autres styles.

Aujourd'hui, les méthodes de conception des circuits intégrés permettent de mélanger tous ces styles dans un même circuit. Pour créer les différentes bibliothèques nécessaires à la

réalisation d'un outil de génération automatique d'interfaces *Fail Safe,* deux approches sont envisageables :

5.2.1.1. L'utilisation exclusive de cellules standards

Cette solution consiste à réaliser des blocs élémentaires constitués de cellules standards puis d'effectuer leur assemblage par programmation pour obtenir des interfaces complètes. Dans cette solution, la flexibilité, et par conséquent une certaine optimisation vont dépendre de la diversité de la bibliothèque employée. La génération des modèles logiques et électriques est immédiate puisque les cellules ont été caractérisées auparavant. Avec cette approche, on perd toutefois beaucoup de surface. Certaines cellules pourraient, en effet, être réalisées en full custom avec un nombre de transistors plus réduit, et donc une surface plus faible, que celles réalisées en cellules standards.

5.2.1.2. L'utilisation combinée de cellules standards et full customs :

Cette approche consiste à ne réaliser en layout que les cellules inexistantes dans la bibliothèque de cellules standard du fondeur. Les cellules seront taillées de façon à intégrer la bibliothèque standard, donc avec les mêmes contraintes topologiques que celles-ci. Les macro-cellules générées seront ainsi faites d'une combinaison de cellules standards et cellules full custom, ce qui représente un gain en surface non négligeable.

C'est donc cette deuxième solution qui sera retenue à terme, puisqu'elle permettra de générer des interfaces *Fail Safe* de façon automatique à partir de bibliothèques de cellules standard, full custom ou semi-custom, en offrant le meilleur compromis optimisation/flexibilité.

5.2.2. Logiciel utilisé

L'objectif étant de réaliser les différentes cellules dans une technologie donnée afin de valider le concept de l'interface, le choix du logiciel s'est porté sur celui disponible alors : le logiciel SOLO 1400 de la société ES2.

5.3. Contraintes d'implémentation

Les contraintes environnementales étant définies, l'ultime étape consiste donc à assembler les différentes parties étudiées précédemment afin de créer l'interface *Strongly Fail Safe*. Les connexions entre les différents blocs doivent toutefois respecter certaines règles visant à garantir la détection de toutes les pannes de l'interface. De même, le choix des différentes fréquences utilisées peut également modifier la propriété du système.

L'origine de ces différentes contraintes est donc étudiée dans ce paragraphe.

5.3.1. Connexion au niveau du contrôleur *Double Rail*

L'influence de l'ordre des connexions au niveau du contrôleur *Double Rail* a été mis en évidence à la section 4.3. Il a été constaté que certaines lignes de contrôle étaient corrélées entre elles. Le test exhaustif du contrôleur *Double Rail* nécessite donc un agencement des lignes provenant de l'interface d'entrée et de la cellule de sécurité, de façon à éviter que deux lignes adjacentes soient corrélées. Une vérification du test complet du contrôleur *Double Rail* est alors nécessaire pour chaque configuration. Ce test exhaustif repose également sur l'hypothèse suivante :

•Pendant le fonctionnement de l'interface, les sorties du système de traitement prennent toutes les valeurs possibles.

La méthode utilisée pour la recherche de la combinaison optimale permettant l'auto-test du contrôleur *Double Rail* est la suivante.

Les différentes lignes de contrôle sont connectées au contrôleur *Double Rail* de manière à créer le plus grand nombre de vecteurs différents à son entrée.

Cette disposition résulte d'une étude des corrélations existant entre les différentes lignes. Une vérification est faite au niveau de chaque sous cellule *Double Rail* d'ordre 2, pour chacune des configurations retenues. Chacune de ces sous cellules doit être stimulée par les quatre vecteurs nécessaires à son auto-test complet. La simulation est effectuée en appliquant tous les vecteurs possibles à l'entrée de l'interface globale.

La réussite de la méthode dépend de la rigueur des simulations effectuées. Il faut impérativement veiller à ne pas générer un cas particulier qui ne se reproduirait pas avec certitude dans le cas d'une utilisation réelle.

Un certain nombre de vecteurs de test varient, par exemple, en fonction de l'état des signaux *f1* et *f2*. On peut constater que *f1* et *f2* prennent soit une valeur constante pour un état donné des entrées, soit oscillent à une fréquence moitié de celle du signal *CK*. La valeur constante dépend de la valeur de *f1* ou *f2* précédant un changement de S_{ia} et S_{ib} (donc de g1 et g2) et le front montant du signal d'horloge, comme l'illustre la figure II.24.

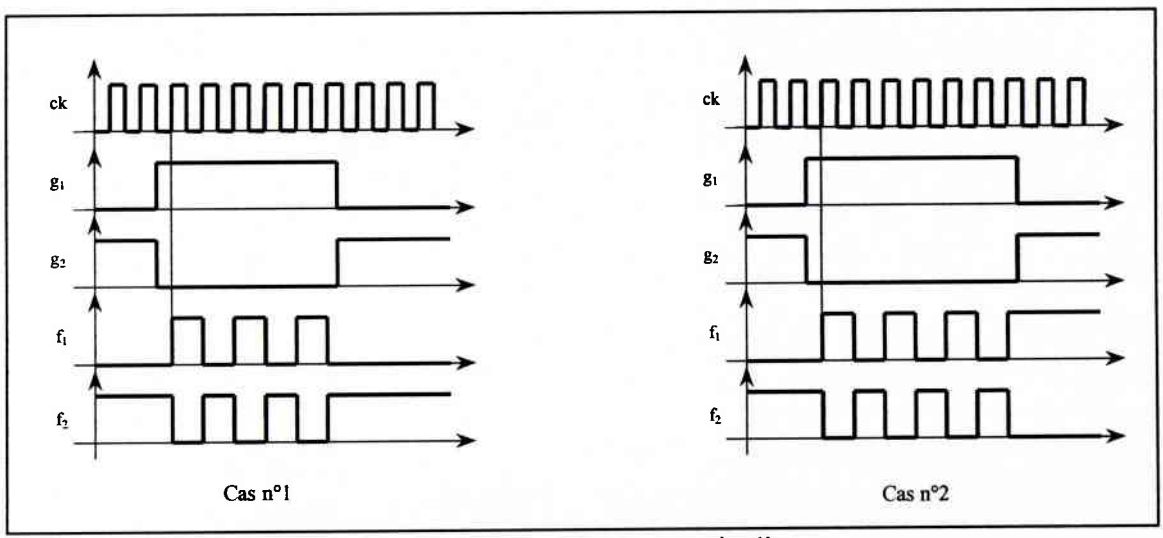


Figure II.24 : Illustration des cas particuliers

- Si $f_1f_2 = 01$ quand g_1g_2 passe à *01* alors f_1f_2 reste à *01* jusqu'au prochain passage à *10* de g_1g_2.
- Si $f_1f_2 = 10$ quand g_1g_2 passe à *01* alors f_1f_2 reste à *10* jusqu'au prochain passage à *10* de g_1g_2.

La valeur prise alors par *f1* et *f2* est donc totalement aléatoire.

Ces vecteurs, présents à l'entrée du contrôleur *Double Rail* dans cette zone, ne peuvent donc pas être pris en compte en raison du risque que cet état n'apparaisse jamais lors d'une utilisation sur site.

Les vecteurs nécessaires à l'auto-contrôle doivent donc apparaître pendant la phase où *f1* et *f2* varient à chaque front actif du signal d'horloge.

Cette méthode a donc permis de trouver une configuration pour laquelle le contrôleur *Double Rail* est entièrement testé à la seule condition que les vecteurs d'entrée de l'interface prennent toutes les valeurs possibles (figure II.25).

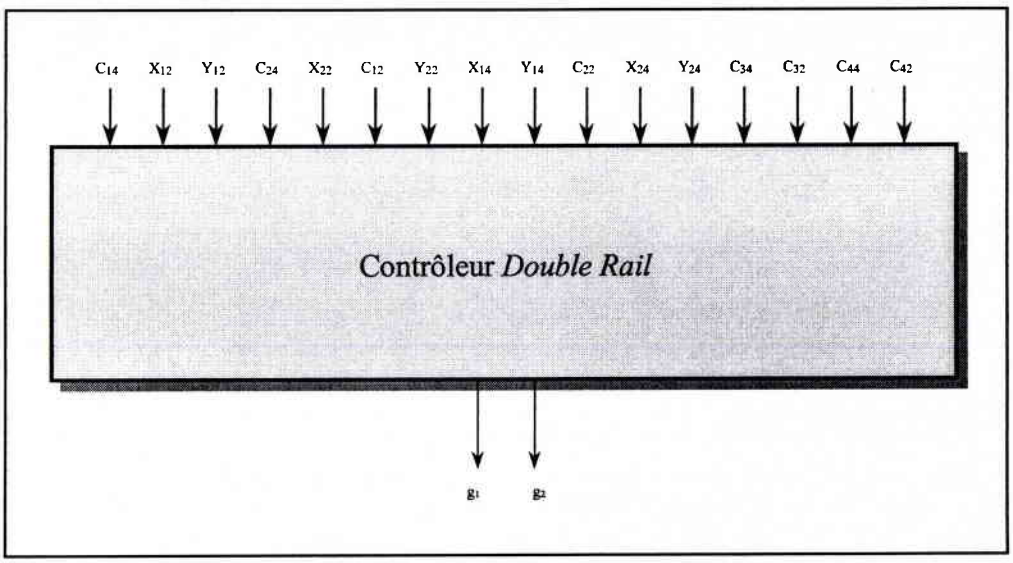


Figure II.25 : Exemple de configuration pour un système de traitement à 4 sorties

Cette configuration est valable pour une interface possédant deux entrées. Elle peut toutefois être utilisée pour des interfaces possédant un nombre d'entrée supérieur à deux. Chaque entrée supplémentaire crée deux paires de lignes supplémentaires. Il a été démontré qu'il suffisait d'ajouter les nouvelles lignes à la suite pour garantir que la propriété d'auto-test ne soit pas remise en cause.

5.3.2. Choix des fréquences

Les différentes fréquences de l'interface ne peuvent être choisies de façon arbitraire. Chacune d'elles a, en effet, une influence sur les lignes de contrôle. La valeur absolue de ces fréquences doit être choisie en fonction de l'environnement de l'application (sources de perturbation à proximité) et dépend également de la technologie. Celle-ci a une influence sur les temps de propagation qui déterminent la fréquence maximale utilisable. Ces choix dépendent donc directement de l'application et sont donc choisis au cas par cas.

Par contre, les valeurs relatives de ces fréquences les unes par rapport aux autres ne dépendent pas de l'application et peuvent donc être imposées.

5.3.2.1. Choix de *Fe* et *Ft*

Ces deux fréquences doivent être choisies de manière à créer un maximum de vecteurs à l'entrée du contrôleur *Double Rail*. Compte tenu des corrélations entre les différentes lignes, le rapport entre ces deux fréquences doit être supérieur ou égal à deux quel que soit le sens.

5.3.2.2. Choix de *Ck*

La mémorisation d'éventuelles indications d'erreur en sortie du contrôleur *Double Rail* s'effectue à la fréquence de *Ck*. Pour garantir la détection de toutes les pannes de l'interface, il faut donc qu'un front actif du signal d'horloge apparaisse pour chaque vecteur d'entrée du contrôleur *Double Rail*. Il se peut en effet, qu'une panne ne provoque une indication d'erreur uniquement pour un seul vecteur d'entrée. Il faut donc que la fréquence du signal *Ck* soit la plus élevée.

Les variations à l'entrée du contrôleur *Double Rail* étant fonction de *Fe*, *Ft*, et *Ck/2*, la valeur de *Ck* doit être supérieure à *Fe*. Pour créer un maximum de vecteurs d'entrée, il faut que la fréquence *Ck/2* soit au moins deux fois supérieure à celle de *Fe*.

5.3.2.3. Contraintes dues aux temps de propagation

Il a été mentionné précédemment que la structure en cascade du contrôleur *Double Rail* imposait des temps de propagation non négligeables. Pendant ce temps, les sorties du contrôleur *Double Rail* peuvent indiquer des erreurs fictives. Il ne faut donc pas qu'un front actif du signal Ck apparaisse pendant ce laps de temps. Ainsi, la fréquence du signal Ck devra être inférieure à une valeur fonction de ce temps de propagation. De même les fréquences Fe et Ft doivent être déphasées d'un temps supérieur à ce temps de propagation.

5.3.3. Résultats de simulation

L'interface a donc été réalisée en respectant les contraintes mentionnées dans le paragraphe précédent.

Les résultats de simulations en fonctionnement normal, c'est à dire sans panne, sont fournis dans l'annexe 1. Les résultats de simulation d'une panne dans les systèmes de traitement y sont également présentés.

Une panne dans le système de traitement est simulée par deux niveaux différents sur les entrées S_{ia} et S_{ib}. A l'apparition de la panne, la sortie correspondante de l'interface est déconnectée de Fe (état sûr), et les lignes *f1* et *f2* prennent alors le même état logique au front actif suivant du signal *Ck*. La fréquence *Ft* n'est alors plus présente en *Oe*, impliquant une coupure d'alimentation du circuit et son basculement dans un état *sûr*.

Afin de vérifier la propriété de l'interface, des apparitions de panne ont été simulées dans les différentes parties de l'interface. Les pannes simulées sont des pannes de collage logique et des pannes de circuits ouverts. Une entrée supplémentaire (*Panne*) a donc été ajoutée pour déclencher la manifestation de la panne.

Les résultats de simulation sont donnés dans l'annexe 2 pour des pannes de différents types dans l'interface d'entrée, le contrôleur *Double Rail* et la cellule de sécurité.

n	2	3	4	5
τ (ns)	52	60	68	76
Surface (mm²)	0.27	0.3	0.34	0.40

Tableau n°9 : Temps de propagation en g1,g2 et taille de l'interface en fonction de n dans une technologie 1µm.

6. Bibliothèques développées

La réalisation d'une interface *Strongly Fail Safe* passe donc par l'assemblage des différentes cellules étudiées en respectant les contraintes citées précédemment.

Certaines de ces cellules dépendent du nombre d'entrée que doit posséder l'interface, alors que d'autres sont figées.

L'interface d'entrée dépend évidemment du nombre d'entrées dont l'interface globale doit disposer. Elle est composée d'autant de cellule de base que l'interface compte d'entrées.

La structure du contrôleur *Double Rail* dépend directement du nombre d'entrées de l'interface. Le contrôleur *Double Rail* doit donc être recréé pour chaque taille d'interface.

Enfin, la cellule de sécurité ne dépend pas du nombre d'entrées et sera donc toujours la même pour toutes les applications.

Considérons une interface *Strongly Fail Safe* possédant *n* entrées. Une solution consiste à créer une bibliothèque comprenant des cellules *Double Rail* correspondant à différentes valeurs de *n*. Cette solution présente cependant deux inconvénients.

Elle nécessite la création d'un très grand nombre de cellules pour répondre à tous les types de besoin. D'autre part, elle oblige le concepteur à connaître les règles de connexion développés précédemment.

Le premier problème a pu être résolu en créant des cellules dont le nombre d'entrées est parametrable par *n*. Ce travail a été effectué pour le contrôleur *Double Rail,* puis pour l'interface d'entrée, avant d'être étendu à l'interface entière.

L'interface globale peut donc être directement générée en parametrant simplement le nombre d'entrée dont elle doit disposer. Cette étape a pu être réalisée grâce à la relation que doivent respecter les connexions entre les lignes de contrôle et les entrées du contrôleur *Double Rail*. Cet algorithme découlant des travaux exposés dans la section 5.3 a été inclus dans la description de la cellule.

Une génération semi-automatisée de l'interface *Strongly Fail Safe* a ainsi pu être réalisée. Ces cellules ont été modélisées dans le langage de description du logiciel utilisé.

7. Conclusion

Ce chapitre présente les travaux ayant abouti à la fabrication d'une interface permettant la transformation d'un système de traitement quelconque en un système *Fail Safe*.

Ces travaux ont permis de valider le principe de ce type d'interface, tout en mettant en lumière un certain nombre de contraintes relatives à la spécificité de ces circuits, et la rigueur de conception requise. Ces contraintes dépendent également du type d'application que l'on désire sécuriser.

Cette étude a abouti à la création de bibliothèques de cellules pré-caractérisées permettant la conception semi-automatique de l'interface en parametrant le nombre d'entrée désiré.

Ces cellules pourraient aisément être intégrées à un outil permettant la transformation automatique d'un système quelconque en un système *Fail Safe* par l'adjonction de cette interface. Tous ces travaux ont été réalisés dans une seule technologie, afin de valider le concept. Ils peuvent être utilisés pour une implémentation dans une autre technologie avec d'autres logiciels en redéfinissant les bibliothèques dans un langage moins spécifique (VHDL). L'adaptation dépendra toutefois du type de cellules de base disponibles dans la technologie désirée.

Ce chapitre permet ainsi de conclure positivement quant à l'intégration d'interfaces *Fail Safe* en circuit VLSI. Le talon d'Achille de cette interface concerne toutefois la dépendance de la propriété *Fail Safe* à sa structure. L'utilisation d'autres composants élémentaires pour la réalisation d'une des cellules de l'interface peut donc invalider la propriété *Fail Safe*.

Cette caractéristique constitue un frein à l'implémentation de cette interface en circuits programmables. Ces types de circuits se caractérisent en effet, par l'absence de contrôle de leur structure au moment de leur programmation.

Le prochain chapitre s'attache donc à concilier ces deux antagonismes afin de faciliter l'accès aux systèmes *Fail Safe* à un plus grand nombre de concepteurs.

Chapitre III :

Implémentation d'interfaces *Fail Safe* en circuits programmables.

Chapitre III : Implémentation d'interfaces *Fail Safe* en circuits programmables.

1. Introduction

Les circuits *Fail Safe* ont été conçus pour éviter qu'une panne dans un système produise des conséquences dramatiques pour les utilisateurs.

Ces systèmes, en raison de leur définition, étaient principalement basés sur les techniques de duplication combinées à l'utilisation de composants discrets spécifiques. Tout composant matériel étant sujet aux risques de pannes, un système sera d'autant plus vulnérable à ces risques qu'il est complexe. La croissante complexité des systèmes représente une contrainte supplémentaire pour atteindre les objectifs *Fail Safe*.

Cependant, les travaux réalisés ces dernières années et ceux présentés dans le chapitre précédent tendent à concilier ces deux aspects. Il est désormais envisageable d'intégrer totalement un système *Fail Safe* sous certaines hypothèses.

Cette intégration présente un grand nombre d'avantages par rapport aux systèmes *Fail Safe* classiques, tant en ce qui concerne la place, le coût, que les perspectives d'évolution vers des systèmes très complexes.

Il serait toutefois intéressant d'étudier l'implémentation de circuits *Fail Safe* en circuits programmables de type FPGA, afin d'en réduire encore les coûts de développement et de les rendre plus accessibles.

Dans l'optique d'une production à petite échelle de systèmes *Fail Safe* de moyenne complexité, les circuits programmables semblent être la meilleure solution tant en ce qui concerne les coûts de développement qu'en terme de temps de conception.

La réalisation de systèmes *Fail Safe* en FPGA pose toutefois un certain nombre de problèmes. Le principal problème réside dans l'absence de contrôle des structures internes utilisées pour la réalisation de la fonction désirée. Cette absence de contrôle et de connaissance de la structure interne peut compromettre l'utilisation de ces circuits pour la réalisation d'interfaces *Fail Safe* telles que celles décrites au chapitre précédent.

Ce chapitre s'efforce donc d'analyser les problèmes liés à une implémentation en circuits programmables et de proposer des solutions pour y remédier **[36]**.

2. Nécessité industrielle et contraintes des circuits programmables

Le chapitre précédent présente les nombreux avantages qu'offre les circuits VLSI pour la réalisation de systèmes *Fail Safe*. Cette comparaison a été effectuée par rapport aux applications *Fail Safe* utilisant des composants discrets spécifiques. Sans remettre en cause les arguments cités, bien réels et très importants, les circuits VLSI peuvent, dans certains cas particuliers, présenter quelques contraintes plus ou moins pénalisantes.

Compte tenu, des moyens à mettre en œuvre pour la fabrication d'un circuit VLSI, et de la rapide évolution des technologies, la fabrication du circuit ne peut être effectuée dans l'entreprise. Ainsi, le délai de fabrication d'un circuit pendant la phase de développement du prototype dépend notamment de la disponibilité du fabricant (fondeur). Le délai de conception est donc relativement long et soumis à des contraintes de délais indépendantes du concepteur.

De plus, en raison des moyens technologiques requis pour la fabrication d'un circuit VLSI, le coût de fabrication d'une série de prototype est très élevé. Le circuit ne devra être fabriqué qu'après un travail de simulation et de conception très rigoureux. Le droit à l'erreur est donc très restreint.

Enfin, lorsque la phase d'élaboration du prototype est terminée, le circuit est fabriqué en série et son application est définitivement figée. Le circuit ne peut pas être réutilisé pour une autre application, et toute évolution du système oblige le concepteur à recommencer la phase d'essai en vue de la fabrication en série de la nouvelle évolution. Les systèmes VLSI ne sont donc pas suffisamment flexibles.

Les entreprises de taille et de budget relativement modestes sont généralement soumises à des contraintes en opposition avec les inconvénients qui viennent d'être cités.

Les délais entre la demande d'une application et la fabrication du premier prototype conforme au besoin doivent être les plus courts possibles.

Le coût de l'étude doit être réduit au minimum. Les moyens logiciels et matériels pour la conception d'un circuit doivent être financièrement supportables par l'entreprise.

Enfin, le système doit pouvoir constamment évoluer et les prototypes doivent pouvoir être réutilisés.

Ainsi, l'exploitation des nombreux avantages qu'offrent les circuits VLSI est pour l'instant réservée aux grands industriels.

Les circuits programmables semblent toutefois particulièrement adaptés aux exigences des petites entreprises. Ils présentent par ailleurs des qualités comparables à celles des circuits VLSI pour la réalisation de systèmes de taille moyenne. Ils constituent donc une alternative intéressante aux circuits VLSI dans les deux cas suivants :

- L'élaboration de prototypes à des coûts relativement faibles.
- La fabrication de petites séries.

Les circuits programmables présentent toutefois quelques inconvénients.

La fréquence maximale de fonctionnement est inférieure à celle des circuits VLSI. De plus, le nombre maximum de portes logiques utilisables, bien qu'en constante augmentation, limite la complexité de l'application. Enfin, lors de la phase de placement routage, le concepteur n'a pas forcement de contrôle sur la structure retenue pour l'implémentation d'une fonction.

Dans l'optique d'une fabrication en petite série de systèmes *Fail Safe* de moyenne complexité, l'utilisation de circuits programmables semble être la solution la mieux adaptée. Comme mentionné précédemment, le seul inconvénient de ce type de circuit réside dans l'absence de contrôle de la structure interne lors de l'implémentation.

La section suivante s'attache donc à étudier l'impact que produirait cette caractéristique en cas d'implémentation de l'interface *Fail Safe* proposée dans le chapitre précédent en FPGA.

3. Implémentation de l'interface *Fail Safe* en FPGA

3.1. Principe de l'interface

L'interface *Fail Safe* dont l'implémentation en FPGA est étudiée dans cette section reprend le principe de celle étudiée pour une application VLSI.

Le concept *Fail Safe* n'étant pas lié à une technologie d'intégration spécifique, l'aspect fonctionnel de l'interface restera donc le même.

La figure suivante reprend donc l'interface présentée précédemment pour des systèmes de traitement dupliqués.

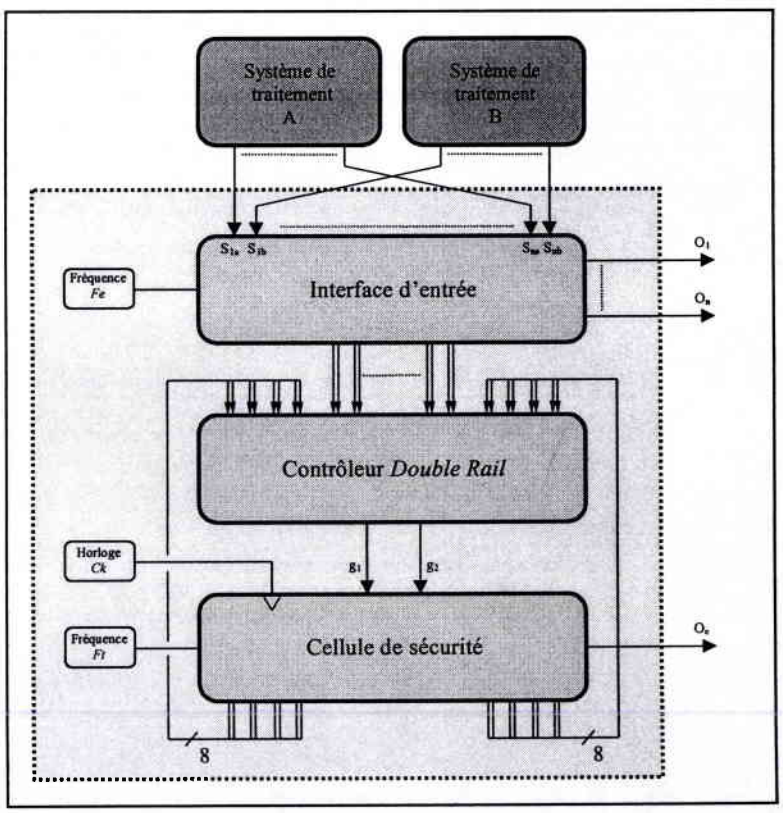


Figure III.1 : Interface *Strongly Fail Safe* FPGA

Cette interface, qui respecte l'*Ultimate Fail Safe Goal* est basée sur les mêmes techniques à savoir :

- Le codage en fréquence

- L'indépendance des chemins
- Le contrôle concurrent

Les hypothèses de fonctionnement sont identiques à celles de l'interface VLSI :

Hyp1 : Les pannes apparaissent une à une dans le circuit.

Hyp2 : Entre l'occurrence de deux pannes consécutives, il s'écoule un laps de temps suffisamment long pour que, pendant ces différents modes de fonctionnement, le circuit soit soumis à un ensemble d'entées permettant de détecter toute panne.

L'interface d'entrée réalise le codage en fréquence, le contrôleur *Double Rail* détecte l'apparition des pannes dans toute l'interface, et la cellule de sécurité verrouille le système dans un état sûr lors de la détection d'une panne.

Dans chacune des parties de l'interface, la propriété *Fail Safe* est obtenue en comparant les valeurs fournies par un chemin de données aux valeurs complémentées fournies par un chemin redondant.

La principale difficulté d'une implémentation FPGA réside dans le respect de cette propriété.

3.2. Implémentation des différentes parties de l'interface

L'implémentation en FPGA pose deux problèmes pouvant remettre la propriété *Fail Safe* de l'interface en cause.

Le premier repose sur l'absence de contrôle et de connaissance de l'implémentation physique d'une fonction. Une porte logique pourra en effet être réalisée physiquement en utilisant plusieurs portes logiques de différents types ainsi que des multiplexeurs ou des éléments séquentiels. Une cellule conçue au niveau structurel ne peut donc pas être directement implantée sur un FPGA.

Le deuxième problème est dû aux outils d'optimisation du nombre de connexions ou du nombre de portes. Deux chemins de données conçus de manière indépendante risquent de

perdre leur propriété d'indépendance s'ils possèdent des équations logiques communes en certains points.

Ces deux caractéristiques des circuits FPGA imposent de reprendre chacune des parties fonctionnelles de l'interface et d'étudier si leur implémentation est possible sans que leur propriété en soit affectée.

3.3. Interface d'entrée

La structure de l'interface d'entrée est rappelée dans la figure suivante :

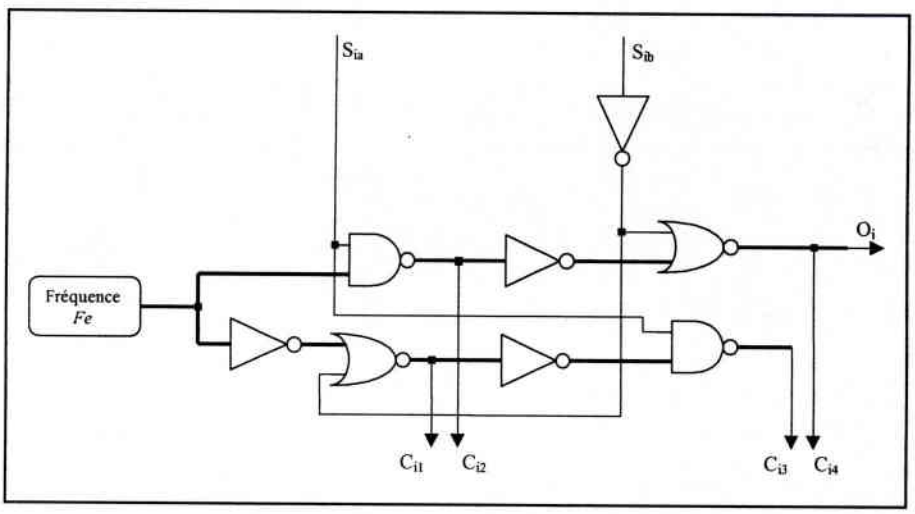


Figure III.2 : Structure de l'interface d'entrée

Cette interface est basée sur le principe cité précédemment utilisant la redondance des chemins. Le chemin principal est comparé en différents nœuds au chemin dupliqué. Les données fournies par les deux chemins en ces différents nœuds doivent être complémentées.

L'implémentation FPGA ne permettant pas de raisonner sur la structure d'un élément, il faut analyser le fonctionnement de l'interface d'entrée à partir de blocs fonctionnels. Les différents blocs présents entre deux noeuds de contrôle sont représentés à la figure III.3.

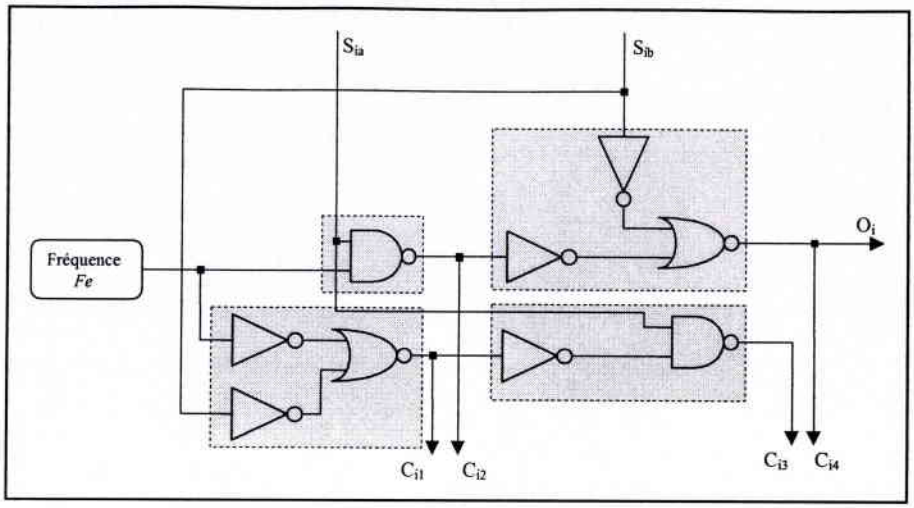


Figure III.3 : Découpage en blocs fonctionnels de l'interface d'entrée

Pendant la phase de placement routage, l'implantation des blocs n'est pas toujours maîtrisable. Les blocs peuvent alors être construits de manière différente et risquent de remettre en cause la propriété *Fail Safe* de l'interface d'entrée.

L'apparition d'une panne dans l'un des blocs entraînera l'une des deux situations suivantes :

- La panne provoque une erreur en sortie du bloc
- La panne ne provoque pas d'erreur en sortie du bloc

Dans le premier cas, la panne sera immédiatement détectée. La comparaison des sorties des différents blocs avec celles du chemin dupliqué permettra, en effet, la détection de la panne, et donc le verrouillage du système dans un état *sûr*. La propriété est garantie par le fait qu'une seconde panne ne peut pas se produire avant que la première ne soit détectée (selon les hypothèses utilisées).

Dans le second cas, en l'absence d'apparition d'une seconde panne, le circuit fonctionne normalement. Lors de l'apparition d'une panne ultérieure, soit celle-ci provoque une erreur en sortie du bloc et est alors détectée (premier cas), soit elle ne peut être détectée (deuxième cas).

Si cette seconde panne non détectée se produit dans un autre bloc, le circuit continu de fonctionner normalement. Dans le cas où les deux pannes apparaîtraient dans le même bloc, là encore, la panne combinée produirait une erreur en sortie du bloc (premier cas) ou non (deuxième cas). En conséquence, en raison de l'indépendance des blocs, toute panne pouvant

endommager la propriété *Fail Safe* de l'interface d'entrée sera donc détectée. Cette propriété provient de l'hypothèse impliquant l'application de tous les vecteurs de test nécessaires entre l'occurrence de deux pannes.

Le second problème est relatif à l'indépendance des chemins. Cette condition est indispensable pour garantir la propriété du système. Si elle ne peut être obtenue par les utilitaires de placement routages, il faut alors définir deux entrées externes différentes Fe_1 et Fe_2 transportant les mêmes signaux comme l'illustre la figure III.4.

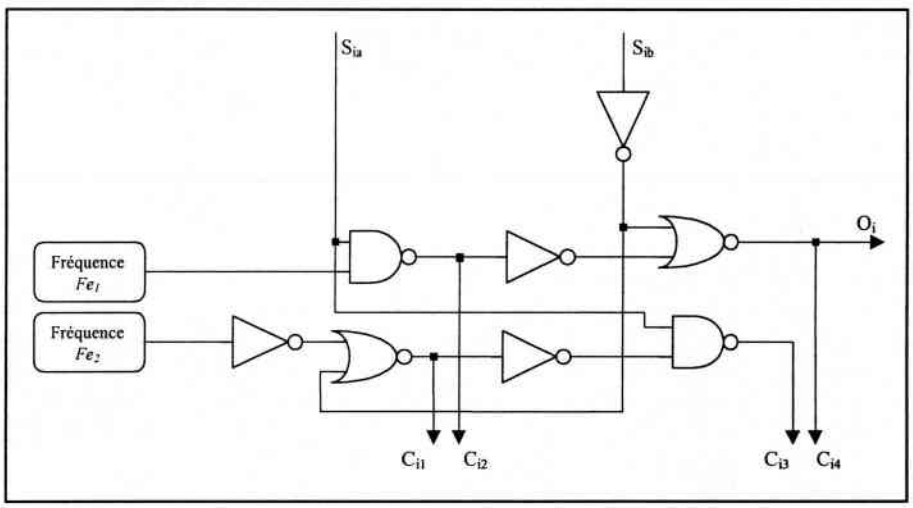


Figure III.4 : Interface permettant de garantir l'indépendance des chemins

3.4. Contrôleur *Double Rail*

La fonction du contrôleur *Double Rail* est d'indiquer une erreur sur ses deux sorties, en présence d'une indication d'erreur sur l'une de ses paires d'entrées, ou en cas d'apparition d'une panne interne au contrôleur.

La figure suivante présente une cellule *Double Rail* élémentaire, possédant deux sorties complémentées et deux paires d'entrées. Un contrôleur *Double Rail* d'ordre plus élevé sera réalisé par la mise en cascade de plusieurs cellules élémentaires.

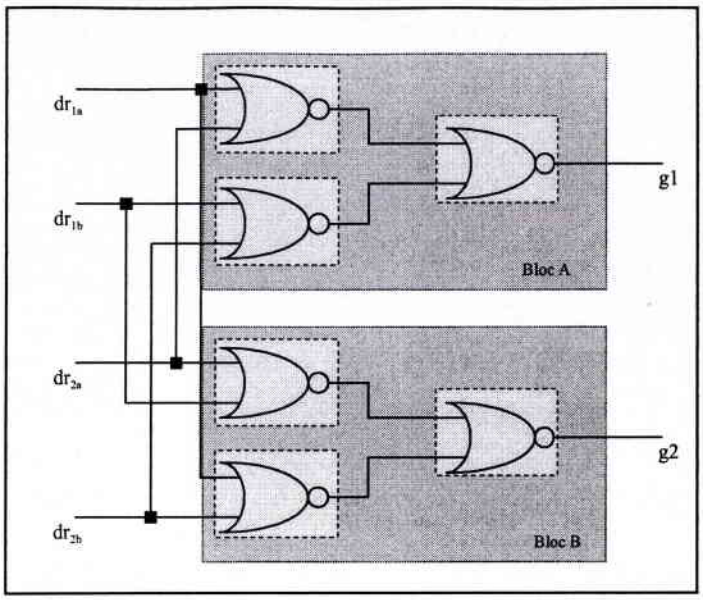


Figure III.5 : Structure d'une cellule *Double Rail* d'ordre 2

Chacun des deux blocs réalise la même fonction logique. L'ordre de connexion des entrées permet de réaliser la fonction du contrôleur *Double Rail*. Les deux blocs doivent rester indépendants afin de garantir que l'apparition d'une panne dans un bloc ne puisse affecter le deuxième bloc.

Compte tenu des propriétés d'auto-test que doit vérifier le contrôleur *Double Rail*, l'indépendance de ces deux blocs n'est pas suffisante. Cette indépendance permet de garantir la propagation d'une indication d'erreur à l'entrée du contrôleur vers sa sortie, mais n'assure pas nécessairement la propagation d'une panne interne au contrôleur.

Pour que les propriétés du contrôleur *Double Rail* exposées au chapitre II soient respectées, il faut garantir l'indépendance de chacune des sous cellules des deux blocs.

Le fonctionnement en cas d'apparition d'une erreur dans l'une des sous cellules sera identique à celui détaillé pour l'interface d'entrée.

Cette indépendance permet ainsi de conserver la propriété du contrôleur *Double Rail* quelle que soit la structure interne du bloc.

3.5. La cellule de sécurité

La cellule de sécurité mémorise l'indication d'erreur issue du contrôleur *Double Rail*. Lors de cette mémorisation, elle verrouille le système dans un état sûr en utilisant la sortie *Oe* de la fréquence *Ft*.

La cellule de sécurité est composée de deux parties :

•L'indicateur d'erreur, constitué d'une cellule *Double Rail* dont les sorties sont mémorisées. Comme pour la cellule *Double Rail* l'indépendance des blocs est la condition suffisante pour assurer la propriété *Fail Safe* d'une implémentation en FPGA.

•Le transducteur. Son fonctionnement et son architecture sont comparables à ceux de deux interfaces d'entrée mises en parallèle. Ainsi, les contraintes d'implémentation en FPGA seront identiques à celles de l'interface d'entrée. Comme dans le cas de l'interface VLSI, le transducteur doit être dupliqué pour garantir le verrouillage du système même en cas d'apparition d'une panne interne.

Le transducteur est donc composé de neuf blocs représentés à la figure III.6.

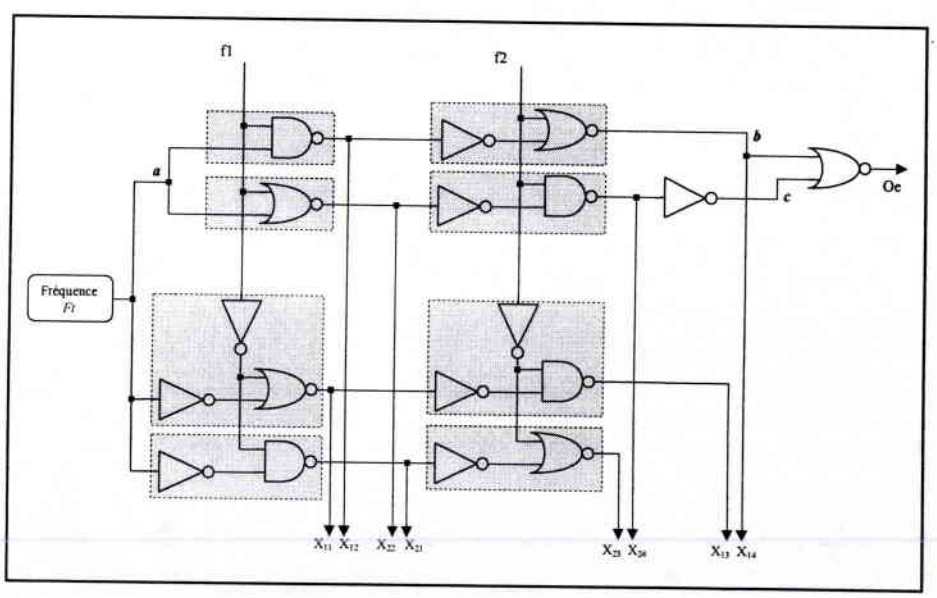


Figure III.6 : Organisation du transducteur FPGA

4. Simulations

Les différentes cellules de l'interface ont été créées à l'aide du logiciel COMPASS. Ces cellules ont alors été implémentées sur différents types de FPGA. L'analyse de la structure proposée par le logiciel confirme l'indépendance des chemins. Les simulations fournies par le logiciel sont similaires à celles obtenues pour l'interface du chapitre II. Les simulations en présence d'une panne dans l'un des systèmes de traitement sont données à la figure III.7.

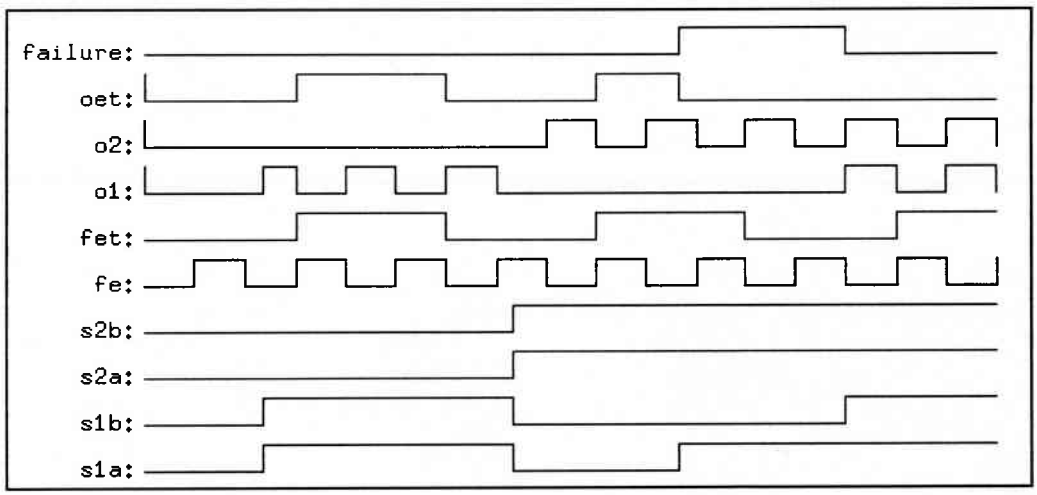


Figure III.7 : Simulation d'une panne dans l'un des systèmes de traitement.

Il convient toutefois de s'assurer que le logiciel de programmation du circuit FPGA choisi ne remet pas en cause l'indépendance des blocs lors de la phase de placement routage.

5. Conclusion

L'objectif de ce chapitre était d'étudier la possibilité d'implémentation d'interfaces *Fail Safe*, telles que celles présentées au chapitre II, en circuits programmables.

L'intérêt des circuits programmables est principalement d'ordre économique. Ils nécessitent en effet une infrastructure beaucoup plus modeste que celle requise pour la conception de circuits VLSI. Ils sont donc accessibles à un plus grand nombre d'entreprises et présentent l'avantage d'être réutilisables. Ces circuits programmables ne peuvent donc pas être ignorés pour la fabrication de petites séries ou la réalisation de prototypes.

L'utilisation de ces circuits pour la réalisation de circuits *Fail Safe* n'est toutefois pas aisée, en raison de l'absence de contrôle de la structure utilisée par le circuit programmable, lors de la phase de placement routage.

Ce chapitre s'attache donc à rappeler l'importance de ces paramètres pour la réalisation d'une interface *Fail Safe*. Une telle implémentation nécessite quelques aménagements de l'interface présentée au chapitre II, pour garantir l'indépendance de certains chemins de données et des blocs les constituant. Le travail présenté consistait à repérer ces différents chemins et à veiller à garantir leur indépendance après la phase de placement routage, en ayant, au besoin, recours à certains artifices. L'étude théorique est appuyée par quelques simulations.

La génération automatique de la « netlist » pour une implémentation FPGA, selon le principe développé au chapitre II est envisageable, après localisation des chemins de données devant obligatoirement rester indépendants.

Afin de se soustraire d'une étude comparable à celle de ce chapitre, il serait intéressant de disposer d'interfaces *Fail Safe* définies de façon totalement indépendante d'une structure ou d'une technologie. La conception d'une telle interface, qui serait alors totalement portable, fait l'objet du chapitre suivant.

Chapitre IV :

Conception d'interfaces fortement sûres indépendantes d'une technologie

Chapitre IV : Conception d'interfaces fortement sûres indépendantes d'une technologie

1. Introduction

Les progrès effectués dans le domaine de la micro-électronique ces dernières années ont abouti à la fabrication, sur une seule puce, de systèmes intégrés très complexes, à des coûts relativement bas.

Cependant, certaines applications, exigent en priorité des garanties concernant la sécurité de fonctionnement plutôt qu'une augmentation de la densité d'intégration. Ces systèmes requièrent, au niveau de leur sortie, un taux de sûreté plus important que celui proposé par les circuits standards.

Un des moyens d'y parvenir consiste à utiliser plusieurs circuits identiques et à utiliser une stratégie de «vote». L'autre moyen consiste à utiliser les propriétés des circuits auto-contrôlables.

Pour certains de ces circuits, les sorties sont utilisées pour le pilotage d'actionneurs. Les valeurs de sorties sont alors dissociées en un ensemble de valeurs *sûres* et un ensemble de valeurs *non sûres*. Les valeurs *sûres* sont celles déclenchant les actionneurs. En fonctionnement sans panne, les sorties peuvent donc être *sûres* ou *non sûres*. En cas d'apparition d'une panne dans le circuit, les sorties ne doivent jamais prendre la valeur *non sûre*, si celle-ci n'est pas la valeur correcte. Un circuit est donc sûr si ses sorties prennent soit la valeur correcte soit la valeur *sûre*. Si cette propriété est garantie, le circuit est dit *Fail Safe*. Dans le cas contraire, le taux de sûreté du circuit peut-être calculé à partir de la probabilité de génération de sorties *sûres*.

Les interfaces *Fail Safe* étudiées dans les chapitres précédents présentent l'inconvénient d'être liés à une structure. Ils ne permettent donc pas une implémentation sur tous les types de support sans que leur propriété ne puisse être remise en cause. La définition de tels systèmes à un niveau d'abstraction indépendant de la structure permettrait leur implémentation sur n'importe quel support et dans n'importe quelle technologie.

C'est vers cet objectif que s'articule ce chapitre, en essayant, dans un premier temps de réaliser une interface disposant d'un taux de sûreté élevé puis en étudiant les possibilités d'évolution vers la propriété *Fail Safe*.

La description de ces interfaces est faite à partir du langage de description VHDL autorisant leur simulation et leur réalisation sur la majorité des outils logiciels existants.

Ces interfaces sont basées sur une technique de codage hybride que nous présentons dans ce chapitre. Ce type de codage combine l'utilisation des codes statiques existants, au principe des codes dynamiques basés sur l'évolution temporelle du code.

Ce chapitre présente, dans un premier temps, le principe général d'une interface permettant de valider le concept retenu **[37], [38]**. Cette section étudie l'architecture de l'interface, ses modifications en fonction de la taille du système de traitement et donne les résultats de simulation. La seconde partie du chapitre présente une autre interface permettant d'augmenter sensiblement le taux de sûreté, et de pallier certaines contraintes de l'interface précédente. L'étude théorique est appuyée par la simulation d'une implémentation FPGA de l'interface. Le taux de sûreté de chacune des interfaces est calculé ainsi que leurs possibilités d'évolution vers la propriété *Fail Safe*.

2. Principe général de l'interface

L'interface étudiée a pour but d'augmenter le taux de sûreté d'une application. Cette interface est donc située entre le système de traitement générant les sorties que l'on souhaite sécuriser, et les organes commandés : les actionneurs.

Le principe est représenté à la figure suivante :

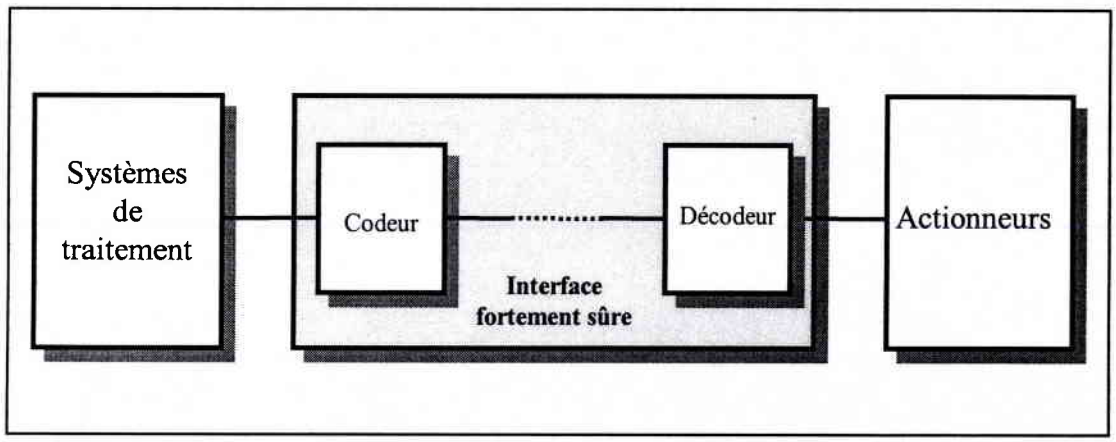


Figure IV.1 : Principe de l'interface

L'interface se décompose en deux parties :

•Le codeur, placé à la sortie des systèmes de traitement, qui transforme les signaux binaires du système de traitement en un mot de code.

•Le décodeur, placé devant les actionneurs, qui transforme le mot de code généré par le codeur en un signal compatible avec les actionneurs.

Dans la suite de ce chapitre, les différentes valeurs prisent par les sorties du codeur pendant un instant donné seront appelées «mot de code».

Afin d'autoriser la détection d'une panne dans le système de traitement, l'interface nécessite sa duplication. La comparaison des signaux générés par les deux systèmes de traitement permet alors la détection d'une panne dans l'un d'eux.

L'interface utilise une technique de codage hybride basée sur l'utilisation d'un code statique non ordonné combiné à des contraintes dynamiques.

Le code statique est un sous-ensemble de code *n parmi 2n*, la valeur de *n* étant fonction du nombre de sorties du système de traitement.

L'aspect dynamique du code exige que deux mots de code successifs soient toujours différents, même si les sorties du système de traitement n'évoluent pas. De plus, seules

certaines transitions entre codes statiques différents sont permises. Une erreur sera donc détectée dans l'un des cas suivants :

- Le code statique est erroné
- La transition entre deux mots de code n'est pas une transition autorisée
- Il n'y a pas de transition entre deux périodes d'horloge

La taille du mot de code, et, par conséquent, le nombre de lignes de transmission entre le codeur et le décodeur, sont proportionnels au nombre de sorties du système de traitement, comme l'illustre la figure suivante :

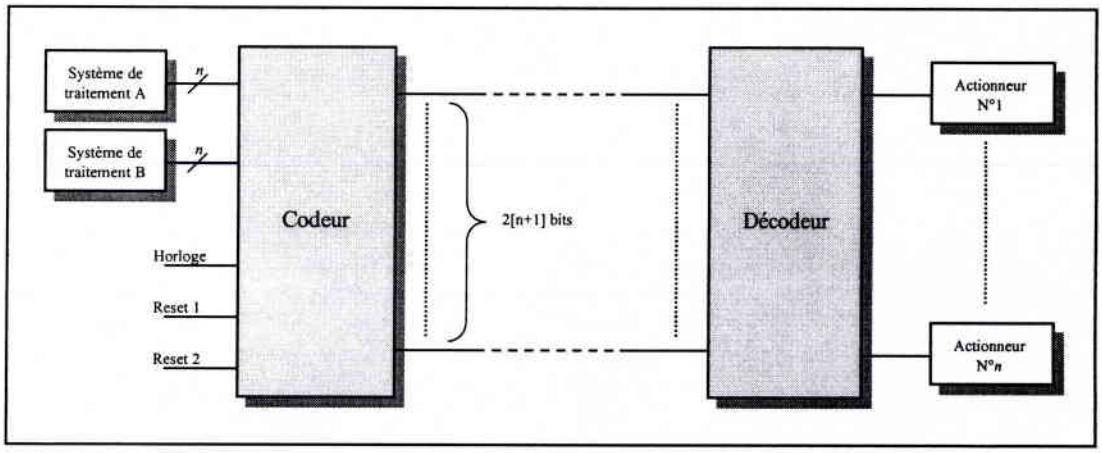


Figure IV.2 : Relation entre le nombre d'entrée et le nombre de lignes de transmission

Le fonctionnement ainsi que les propriétés des différentes parties de l'interface sont présentés dans les sections suivantes.

3. Présentation de l'interface

3.1. Etude du codeur

3.1.1. Principe

Le codeur a pour fonction de transformer les sorties binaires générées par le système de traitement en un mot de code permettant d'accroître la sûreté des données générées par le système. Afin de permettre la détection de pannes dans le système de traitement, celui-ci sera dupliqué.

Le schéma fonctionnel pour un système de traitement à une seule sortie est présenté à la figure suivante :

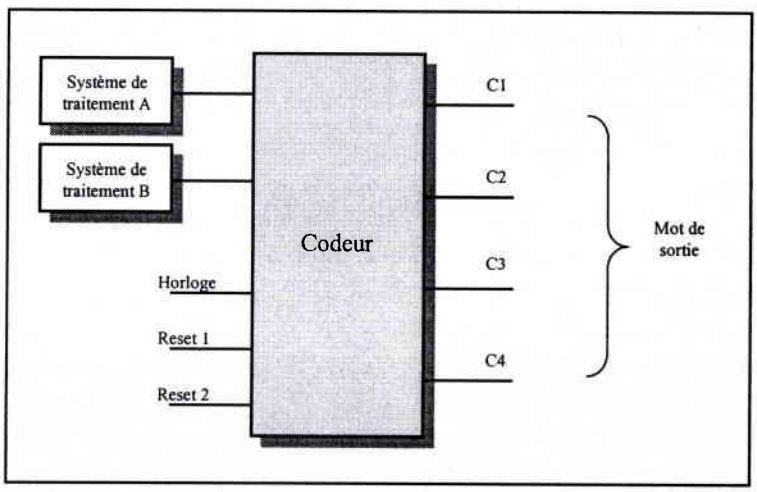


Figure IV.3 : Schéma fonctionnel pour un système de traitement à une seule sortie

3.1.2. Aspect statique du codage

Le principe du codeur est celui d'un système auto-contrôlable. L'ensemble des mots de sortie est divisé en deux sous-ensembles :

- L'ensemble *A* des mots corrects, qui appartiennent au système de codage retenu
- L'ensemble *B* des mots incorrects, qui sont en dehors du système de codage choisi

En fonctionnement normal (sans panne), les mots de sorties appartiennent tous à l'ensemble A. En présence d'une panne dans le codeur ou dans l'un des systèmes de traitement, les mots de sortie doivent appartenir à l'ensemble B.

Le code choisi doit donc permettre au décodeur de détecter toutes les pannes susceptibles de nuire au bon fonctionnement de l'interface ainsi que des pannes dans l'un des systèmes de traitement.

3.1.3. Aspect dynamique du codage

Pour améliorer la sûreté de fonctionnement de l'interface, il est indispensable d'introduire un aspect temporel au codage retenu. Il a été souligné au chapitre II que la plupart des systèmes *Fail Safe* utilisent la technique de codage en fréquence. Cet aspect temporel permet, d'une part, de s'affranchir plus facilement des problèmes de collage logique, et, d'autre part, de verrouiller le système dans l'état *sûr* en cas d'absence de cette fréquence.

Le codeur présenté dans la section suivante utilise ce principe, mais ne nécessite pas l'utilisation d'une entrée fournissant cette fréquence.

Ce codeur est conçu de manière à ce que deux mots de code consécutifs soient différents, même dans le cas où l'entrée du codeur resterait inchangée. Ce principe permet d'augmenter la sûreté de fonctionnement du système tout en se passant d'une entrée supplémentaire générant la fréquence correspondant à l'état *non sûr*.

Par ailleurs, seules quelques transitions entre les différents mots de code sont autorisées pendant le fonctionnement normal de l'application. L'apparition de transitions incorrectes entre deux mots de code corrects serait traitée comme un mot n'appartenant pas au code. Il s'agit de l'aspect dynamique du codage. Ainsi, les mots de code à la sortie du codeur seront corrects si ils appartiennent à l'ensemble des mots de code statiques, et, si la transition par rapport au code précédent appartient à l'ensemble des transitions correctes. Dans tous les autres cas, l'interface est en présence d'une panne.

Le schéma fonctionnel du codeur, dans le cas d'un système de traitement possédant m sorties, est le suivant :

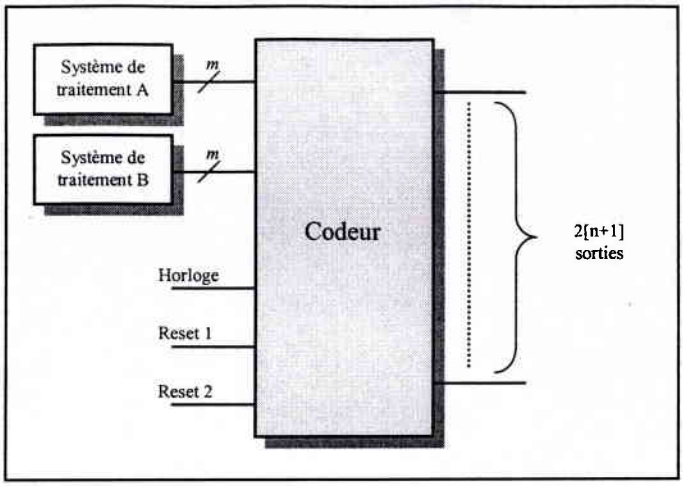


Figure IV.4 : Schéma fonctionnel du codeur

Le codeur possède alors *2*(m+1)* sorties codées.

3.1.4. Etude du fonctionnement dans le cas ou m=1

Le schéma fonctionnel du codeur est alors celui de la figure IV.5.

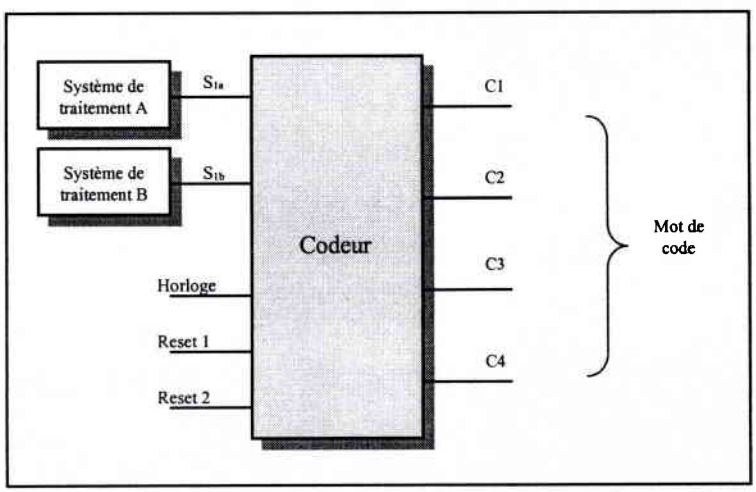


Figure IV.5 : Schéma fonctionnel du codeur pour *m*=1

Le code de sortie utilisé par le codeur est un sous-ensemble du code *2 parmi 4*, dans lequel, seuls quelques-uns des codes disponibles sont utilisés. Ces codes, présentés dans la suite, sont les codes statiques. Les variations du mot de code résultant de la variation d'une entrée ne se feront qu'après le prochain front actif du signal d'horloge.

Nous avons montré que, dans le but d'accroître la sûreté de fonctionnement, le mot de code doit changer à chaque front actif du signal d'horloge. Cette variation doit avoir lieu

même dans le cas où l'entrée du codeur (donc la sortie du système de traitement) reste inchangée.

Il en résulte qu'il y aura plusieurs codes statiques correspondant à un même niveau logique de l'entrée S_1. Seules quelques-unes des transitions possibles entre les différents codes statiques sont toutefois autorisées. Ces transitions autorisées constituent le code dynamique.

En résumé, le mot de code de sortie du codeur change donc à chaque front actif du signal d'horloge et il y a détection d'une erreur dans chacun des trois cas suivants :

- Le code statique de sortie est erroné
- La transition n'est pas une des transitions autorisées
- Il n'y a pas de transition

3.1.4.1. Etude du code statique

Le code statique utilisé est un sous-ensemble de code 2 parmi 4. Les mots de code retenus sont : (C1,C2,C3,C4) = (0011), (0110), (1001), (1100).

Le niveau logique 0 en S_{11} et S_{21} est codé par (0011) ou (1100)

Le niveau logique 1 en S_{11} et S_{21} est codé par (0110) ou (1001)

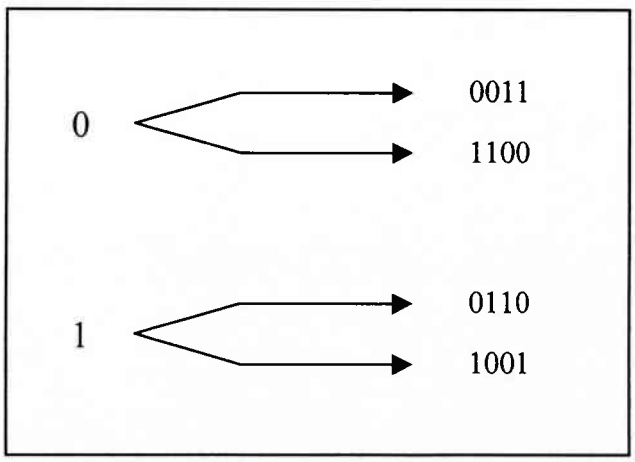


Figure IV.6 : Codage statique

Les codes statiques ne peuvent évoluer qu'à chaque front actif du signal d'horloge.

3.1.4.2. Etude du code dynamique

Deux codes statiques consécutifs doivent toujours être différents. Les transitions autorisées entre ces différents codes statiques sont représentées à la figure suivante :

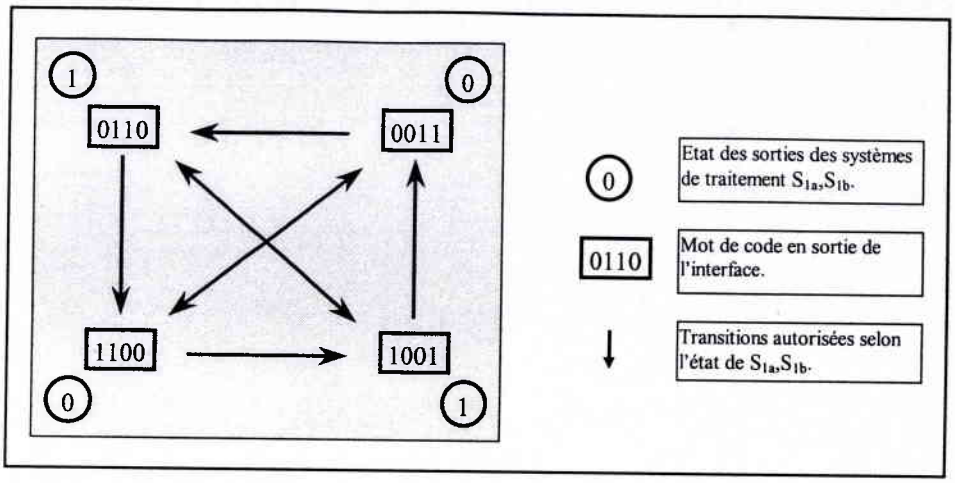


Figure IV.7 : Transitions autorisées par le codage dynamique

A chaque niveau logique de S_{1a}=S_{1b} correspond un mot de code. La valeur que prendra S_{1a}=S_{1b} au front d'horloge suivant modifiera le mot de code selon le diagramme de la figure IV.7.

On constate que seules certaines transitions entre les différents mots de code sont possibles. Par exemple, le mot de code ne peut pas passer de l'état (0011) à l'état (1001), ce qui correspondrait (en théorie, du point de vue statique) à une transition de S_{1a}=S_{1b} de 0 à 1. Pour une telle transition, la seule transition autorisée du mot de code est celle de l'état (0011) à l'état (0110).

L'interdiction de certaines transitions permet d'accroître la sûreté de l'interface.

3.1.4.3. Contrôle du fonctionnement

Le contrôle du bon fonctionnement du codeur est effectué en vérifiant la conformité du code statique, mais également la transition entre le code actuel et le précédent comme l'illustre la figure suivante :

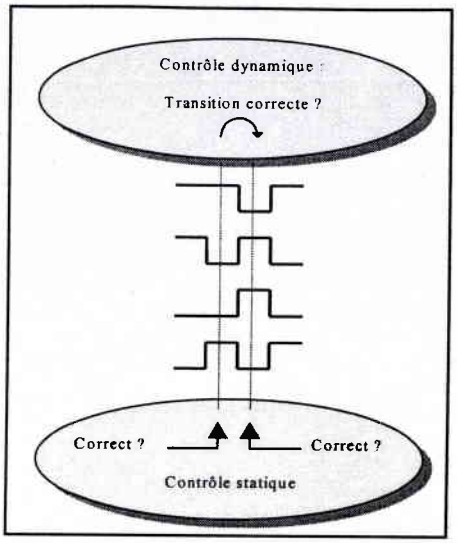


Figure IV.8 : Contrôle statique et dynamique du code de sortie

3.1.5. Fonctionnement de la structure

La structure du codeur est présentée à la figure suivante :

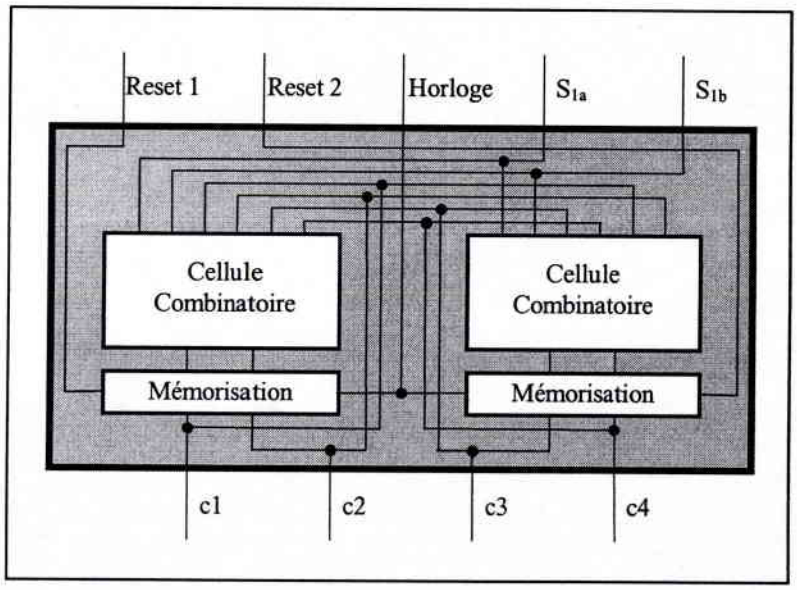


Figure IV.9 : Organisation interne du codeur

Il est composé de deux cellules combinatoires et de deux cellules de mémorisation. Chacune des deux cellules combinatoires réalise les deux fonctions suivantes :

- Calculer le prochain demi code de sortie correct
- Vérifier la conformité du demi code généré par l'autre cellule combinatoire

Appelons bloc, l'ensemble formé par une cellule combinatoire et la cellule de mémorisation correspondante. Chaque bloc génère un demi code (2 des 4 bits du code de sortie). Pendant le fonctionnement normal, le demi code généré par un bloc ne dépend que de son demi code précédent et de la nouvelle valeur de S_{1a}=S_{1b}.

Chaque bloc possède donc six entrées. Les deux entrées provenant des systèmes de traitement, les deux sorties précédentes du bloc permettant de calculer le demi code suivant, et les deux sorties de l'autre bloc permettent le contrôle du bon fonctionnement de ce dernier.

La structure des deux blocs est identique. Seule l'initialisation des cellules de mémorisation est différente. A l'initialisation, l'un des blocs génère le demi code *00* alors que l'autre génère le demi code *11*. Le code de départ est donc *0011*.

En reprenant l'hypothèse d'occurrence de pannes simples, une panne ne peut affecter qu'un seul bloc à la fois. Ainsi, le bloc sain reste capable de détecter la panne du bloc affecté, et peut produire une fausse transition, et un faux code, en sortie de l'interface.

Etant donné que le code de sortie de l'interface varie à chaque front actif du signal d'horloge, l'utilisation d'une entrée générant une fréquence (utilisée dans la plupart des applications *Fail Safe*) n'est pas nécessaire.

3.2. Taux de sûreté du codeur

Pour estimer le niveau de sûreté du codeur, il est important de calculer la probabilité d'occurrence d'une panne non détectable.

Le fonctionnement du codeur peut se décomposer en trois états fondamentaux O1, O2, O3.

- L'état O1 correspond au fonctionnement normal du codeur, c'est à dire sans panne.

- L'état O2 correspond à l'apparition d'une panne détectée par le codeur, et signalée sur ses sorties.

- L'état O3 correspond à l'état indésirable où l'occurrence d'une panne n'est pas détectée. L'état O3 ne peut se produire qu'à la suite d'une panne dans le codeur. En

raison de l'hypothèse d'apparition de pannes simples, l'occurrence d'une panne dans l'un des systèmes de traitement est, en effet, obligatoirement détectée.

La chaîne de MARKOV de la probabilité de transition, à chaque cycle d'horloge, entre ces trois états, est présenté à la figure IV.10.

- p_1 représente la probabilité d'apparition d'une panne dans l'un des systèmes de traitement.
- p_2 représente la probabilité d'apparition d'une panne détectée dans l'un des deux blocs.
- p_3 représente la probabilité d'apparition d'une panne non détectée dans l'un des deux blocs.

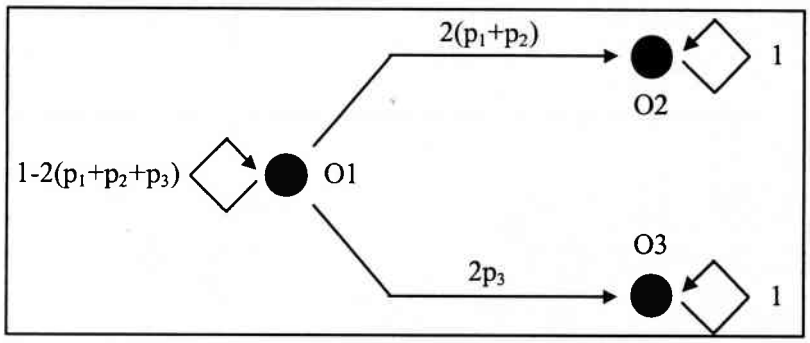


Figure IV.10 : Graphe de transition de MARKOV dans le codeur

La probabilité P de détection de l'apparition d'une panne peut être calculée à l'aide des chaînes de MARKOV. Le détail des calculs est donné dans l'annexe 3.

On obtient donc : $P = \dfrac{p_1 + p_2}{p_1 + p_2 + p_3}$

La probabilité Q de non détection de cette panne est alors : $Q = \dfrac{p_3}{p_1 + p_2 + p_3}$

Notons que la probabilité d'apparition d'une panne dans un bloc est grossièrement fonction de la surface de silicium nécessaire à son implémentation.

On peut, en outre, aisément supposer, que pour un grand nombre d'applications, la surface requise pour chacun des systèmes de traitement, est sensiblement supérieure à celle requise par l'interface. Ainsi, dans la plupart des cas, nous avons $p_2 << p_1$ et $p_3 << p_1$.

Ce qui implique $P \approx 1$ et $Q \approx 0$

Il est important de signaler que, si les blocs du codeur peuvent être réalisés de façon à ce que l'apparition d'une panne dans un bloc provoque obligatoirement une modification de sa sortie, la probabilité p_3 est nulle.

On obtient dans ce cas P=1 et Q=0, qui correspond à la propriété *Fail Safe*.

3.3. Structure du codeur pour un nombre quelconque d'entrées

Pour un système de traitement possédant *m* sorties, le code utilisé sera un sous-ensemble de *[m+1]* parmi *2*[m+1]*. Prenons l'exemple d'un codeur pour lequel *m=2*, le code utilisé est un code 3 parmi 6 et la structure du codeur est celle de la figure suivante :

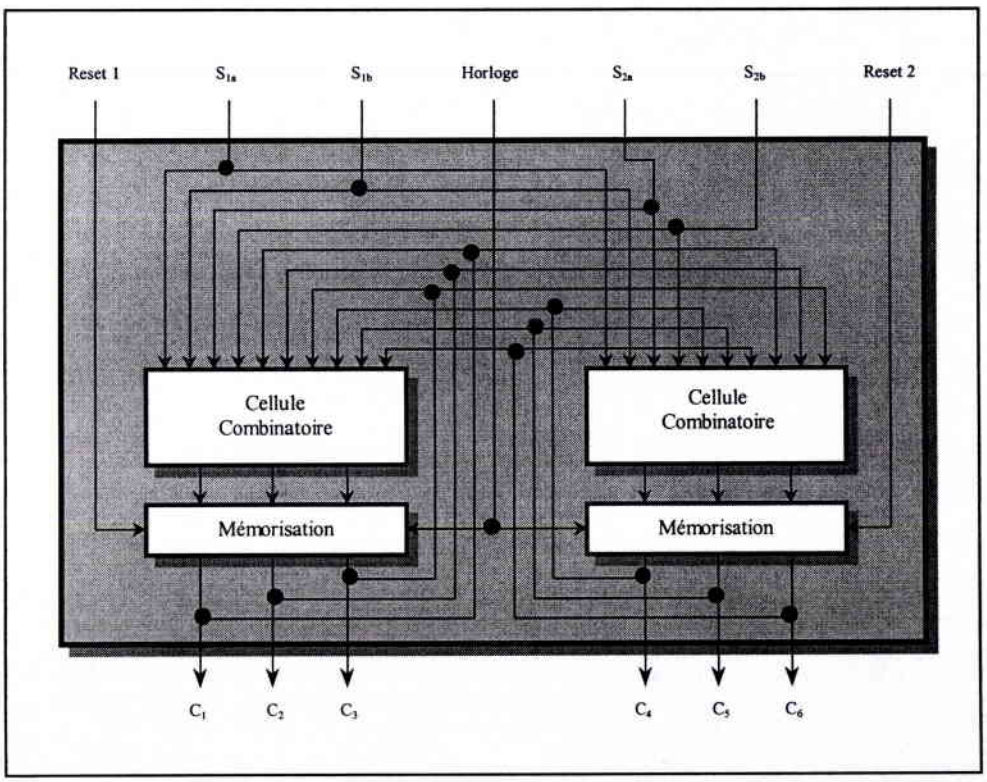


Figure IV.11 : Structure du codeur avec m=2

3.3.1. Choix des mots de code

Le choix des mots de code de sortie se fait parmi les éléments du code 3 parmi 6. Le codeur est toujours réalisé à partir des deux blocs générant chacun le complément de l'autre en fonctionnement normal. Ainsi, les mots du code de sortie correct seront toujours de la forme $(a_1a_2a_3a_4a_5a_6)$ avec $a_1 = \overline{a_4}$; $a_2 = \overline{a_5}$; $a_3 = \overline{a_6}$

D'une manière générale, pour un code *n* parmi *2n*, les mots de code retenus seront $(a_1a_2a_3...a_n...a_{2n})$ tels que $a_1 = \overline{a_{(n+1)}}$; $a_2 = \overline{a_{(n+2)}}$;.....; $a_n = \overline{a_{2n}}$

Il y a donc 2^n mots de code utilisés

3.3.2. Choix des transitions autorisées

Dans le cas de notre exemple de code 3 parmi 6, il y a 2^3=8 mots de code de sortie utilisés. Ces codes peuvent être placés sur un cercle de la manière présentée à la figure suivante :

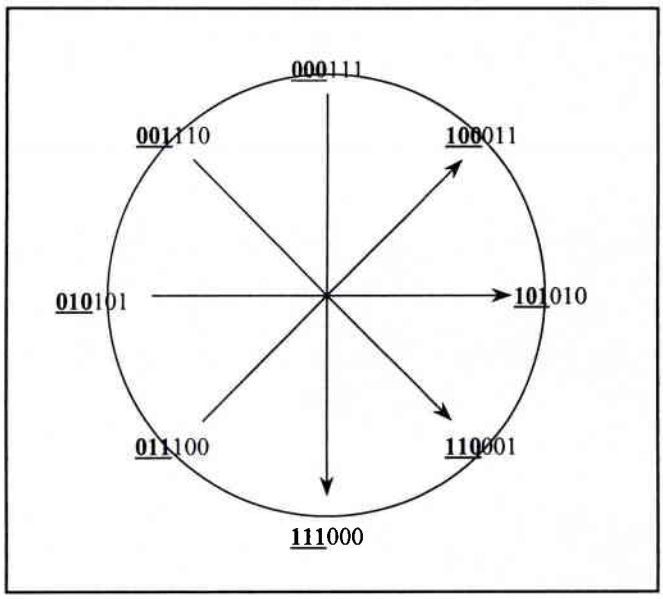


Figure IV.12 : Positionnement des mots de code sur le cercle des transitions

Chaque mot est constitué de 3 bits de poids forts, et trois bits de poids faibles représentant le complément logique des bits de poids forts. Le cercle peut être construit de la manière suivante.

On ne se préoccupe au départ que des trois bits de poids forts.

- On place le premier mot 000 sur le cercle
- On place son complément logique diamétralement opposé
- On place ensuite, en se déplaçant toujours dans le même sens (sens trigonométrique par exemple), le code binaire suivant, en plaçant toujours son complément logique diamétralement opposé.
- On continue ainsi tant que le bit de poids fort reste à 0

On aura alors placé $2^{(3-1)} = 2^2=4$ mots et leurs 4 compléments logiques, soit 8 mots, ce qui correspond à la totalité des mots de codes.

Il suffit alors d'ajouter les 3 bits de poids faibles à chacun des mots, pour obtenir les mots de code de sortie.

Ce cercle permet désormais de déterminer les transitions autorisées. A chacun des codes placés sur le cercle doit correspondre un état du système de traitement. Cet état des sorties du système de traitement (S_{ia},S_{ib}) sera appelé combinaison.

On attribue, au premier code placé sur le cercle, la combinaison 00, puis 01 au code suivant, et ainsi de suite jusqu'à la combinaison 11. On attribue ensuite les mêmes combinaisons aux codes diamétralement opposés. Chaque combinaison correspond alors à 2 mots de code.

Il reste alors à flécher les transitions autorisées entre les différents codes, en fléchant chaque mot de code vers les mots de code suivants, dans le sens trigonométrique, jusqu'à flécher le code diamétralement opposé.

On passe alors au code suivant en répétant la même opération, comme l'illustre la figure suivante :

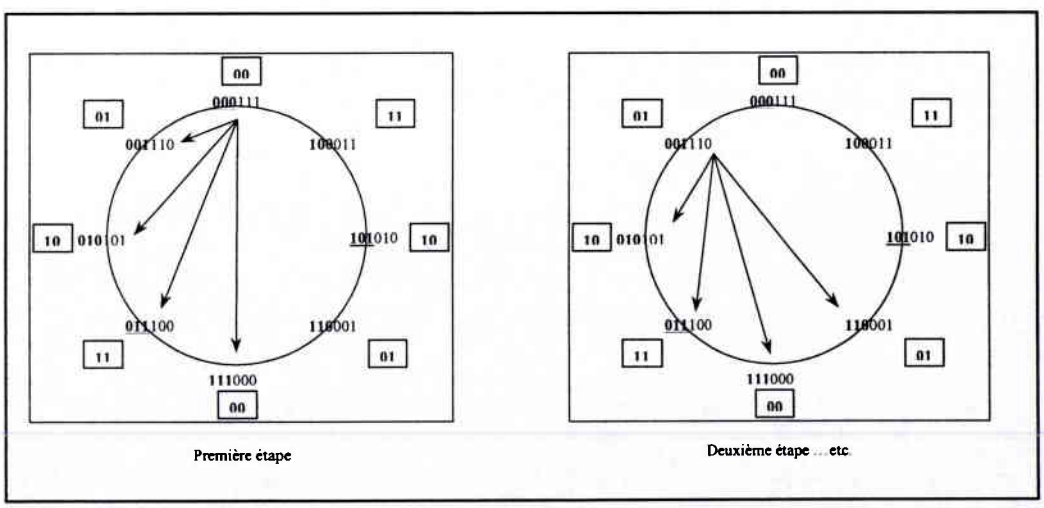


Figure IV.13 : Construction des transitions autorisées

Chaque transition correspond à un angle compris dans l'intervalle]0 ;π]. On obtient alors toutes les transitions autorisées, représentées à la figure IV.14.

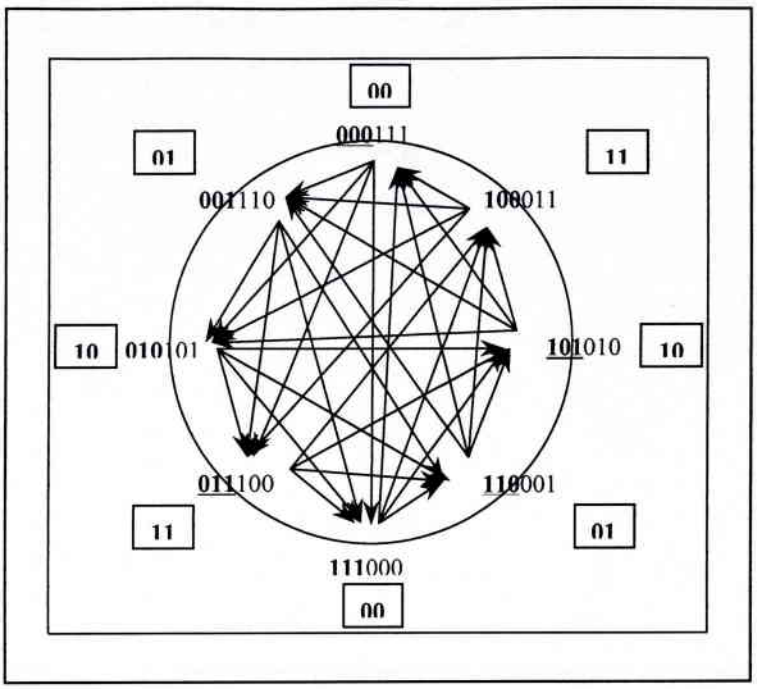


Figure IV.14 : Graphe des transitions autorisées pour m=2

Pour le cas général d'un code *n* parmi *2n*, la méthode de construction du cercle sera identique. Le cercle sera alors composé de 2^n mots de codes, et construit en plaçant les $2^{(n-1)}$ demi mots de code puis leurs compléments logiques diamétralement opposés et enfin en complétant les demis codes en ajoutant les *n* bits de poids faible à chaque demi mot.

Les transitions seront obtenues par la méthode détaillée précédemment.

Le fait de faire évoluer les demi codes selon le code binaire lors du placement sur le cercle permet de déterminer automatiquement la validité d'une transition.

Exemple :

Considérons les deux mots de code consécutifs suivants : $(a_1a_2a_3....a_n....a_{2n})$ et $(b_1b_2b_3....b_n....b_{2n})$.

Pour que la transition soit une des transitions autorisées, il faut que le demi code $(b_1b_2b_3....b_{(n-1)})$ soit l'un des $2^{(n-1)}$ mots binaires suivant $(a_1a_2a_3....a_{(n-1)})$ en évoluant selon le code binaire.

3.3.3. Structure du codeur pour *m* quelconque

Chaque bloc dispose, en entrée, des *m* sorties de chaque système de traitement, ses *(m+1)* sorties générées et les *(m+1)* sorties du bloc complémentaire. Ainsi, chaque bloc

possède *(4m+2)* entrées et *(m+1)* sorties, en plus du signal d'horloge et de l'entrée d'initialisation.

L'organisation interne du codeur est alors la suivante :

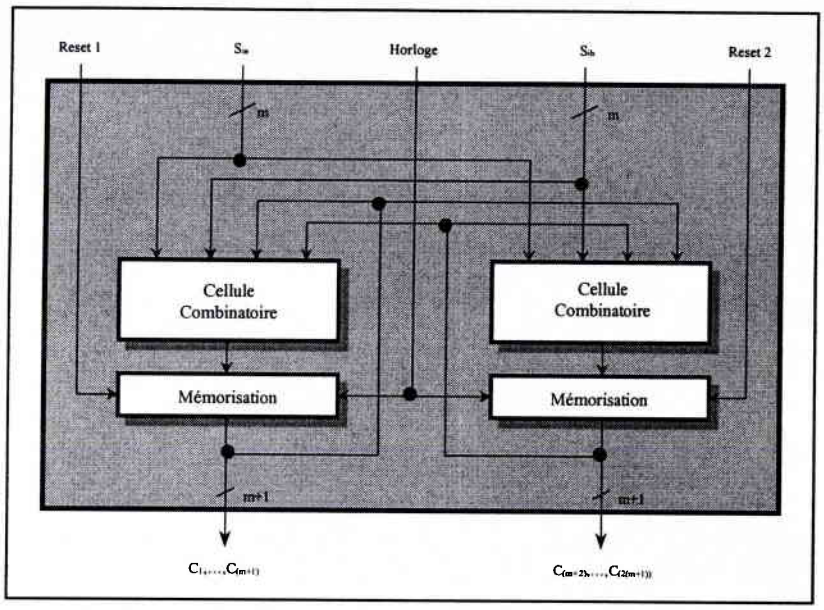


Figure IV.15 : Organisation interne du codeur en fonction de m

3.4. Résultats de simulation

La validation du principe retenu nécessite une phase de simulation par un outil adapté à la structure du système. Dans le cas du codeur étudié précédemment, cette simulation ne peut être effectuée qu'à partir d'une définition dans un langage de description de haut niveau (VHDL). L'objectif de ce chapitre étant de réaliser une interface totalement portable, la définition du codeur n'a, à aucun moment, été liée à une structure ou une fonction électronique de bas niveau. Les simulations du fonctionnement du codeur ont donc, dans un premier temps, été effectuées à l'aide d'un logiciel de simulation purement informatique, et par conséquent indépendant d'un type de support ou d'une technologie.

Il est toutefois possible de simuler les conséquences de l'apparition d'une panne dans le codeur en forçant, de façon aléatoire, une ligne à un état logique quelconque.

Les différentes simulations effectuées à partir de la description VHDL (Annexe 5) du codeur en fonctionnement normal, puis en présence de pannes sont présentées dans cette section. Ces simulations ont été effectuées à l'aide du logiciel SYSTEM HILO de la société VEDA.

3.4.1. Simulation du fonctionnement sans panne

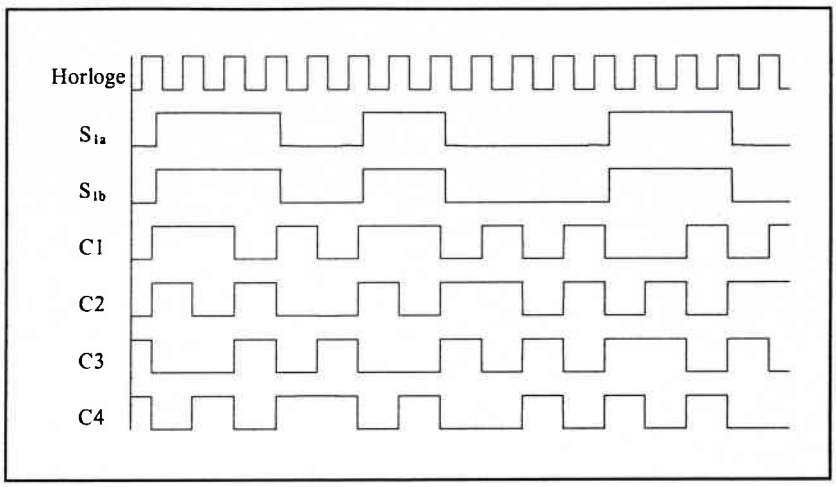


Figure IV.16 : Simulation du codeur en fonctionnement normal

On constate que le mot de code généré par le codeur évolue à chaque front montant du signal d'horloge, et ce, même lorsque les entrées restent inchangées. Les sorties du premier bloc sont le complément logique des sorties du second bloc. Chacune des sorties du codeur change d'état au moins une fois tous les deux cycles d'horloge.

3.4.2. Simulation d'une panne dans le système de traitement

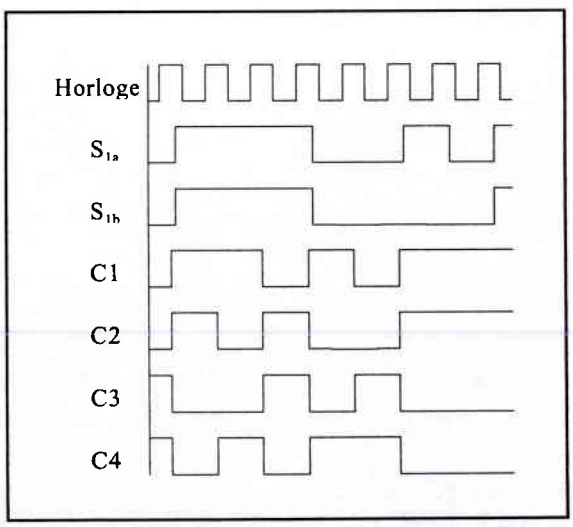


Figure IV.17 : Simulation d'une panne dans l'un des systèmes de traitement

Lors de l'apparition d'une panne dans le système de traitement, les sorties de l'interface conservent l'état dans lequel elles se trouvaient au moment de la détection de la panne. Aucune transition ne se produisant alors en sortie, le système reste bloqué dans un état *sûr*.

3.4.3. Simulation d'une panne dans le codeur

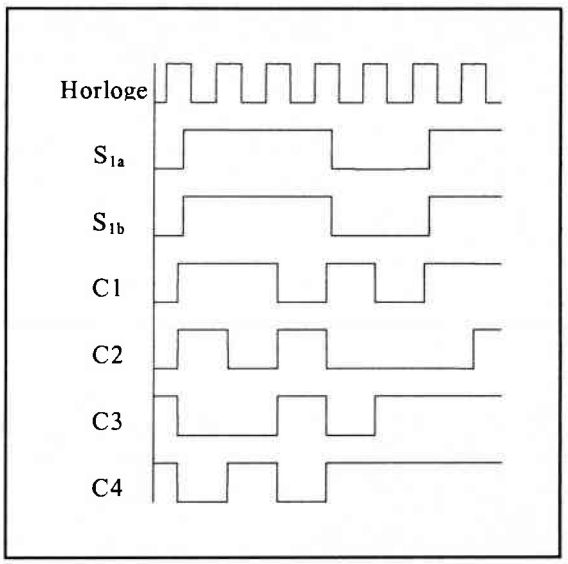


Figure IV.18 : Simulation d'une panne dans le codeur

Lors de l'apparition d'une panne dans le codeur, seul un des blocs est capable de détecter la panne. Le mot de code statique généré par le codeur est alors erroné car le demi code produit par le bloc défectueux est modifié. Pour des raisons de sécurité, le bloc sain bloque ses sorties, maintenant ainsi la sortie du codeur dans un état *sûr*.

3.5. Possibilité d'évolution vers un codeur *Fail Safe*

Les calculs effectués à la section 3.2 attestent du haut niveau de sûreté offert par le codeur. Cependant, pour une utilisation dans un système critique, l'interface présentée dans ce chapitre doit disposer de la propriété *Fail Safe*. Cette propriété exige dans un premier temps que le codeur soit *Fail Safe*. Pour atteindre cet objectif, le codeur doit respecter un certain nombre de contraintes que nous analyserons dans ce paragraphe.

3.5.1. Contrainte de verrouillage irréversible

Une première contrainte consiste à verrouiller définitivement le codeur dans un état sûr. Un des principes fondamentaux de l'interface repose sur la variation régulière (à chaque cycle d'horloge) du mot de code en sortie du codeur. Les simulations confirment cette propriété et illustrent le blocage des sorties du codeur en cas de détection d'une panne. Pour garantir la propriété *Fail Safe* de l'interface, ce blocage ne doit pas être remis en cause par l'occurrence d'une panne ultérieure. Il faut donc ajouter un mécanisme de verrouillage définitif du codeur dans cet état. Pour ce faire, une des propriétés intéressantes du codeur, mise en évidence par les simulations (et aisément démontrable), est la variation de chacune des sorties du codeur au minimum tous les deux cycles d'horloge. Il suffit d'ajouter un dispositif externe de coupure d'alimentation, tel que celui présenté au chapitre II, connecté à une ou plusieurs sorties du codeur. L'alimentation sera alors coupée lorsque la sortie ne générera plus une fréquence comprise entre $f_H/2$ et f_H, où f_H correspond à la fréquence du signal d'horloge.

3.5.2. Contraintes de test exhaustif des blocs du codeur

La tâche la plus délicate consiste à rendre le codeur *Fail Safe*. Le seul moyen d'y parvenir consiste à garantir que l'apparition d'une panne dans l'un des blocs sera propagée sur ses sorties.

Une solution consiste donc à rechercher une structure permettant de garantir que les vecteurs présents à l'entrée du bloc pendant le fonctionnement normal du codeur propagent l'erreur produite par la panne vers l'une des sorties. Une telle structure est réalisable en utilisant une méthode de conception similaire à celle du chapitre II mais serait alors obligatoirement liée à une structure. Ce procédé est donc en contradiction avec les objectifs de ce chapitre.

L'autre solution nécessite l'utilisation d'un module de test permettant de garantir le contrôle complet du bloc. En appliquant tous les vecteurs possibles à l'entrée du bloc, la panne doit obligatoirement apparaître sur les sorties du codeur. Dans le cas contraire, cette panne n'est pas dangereuse et n'a pas de conséquences sur le fonctionnement du codeur. Le principe d'un tel codeur est présenté à la figure suivante :

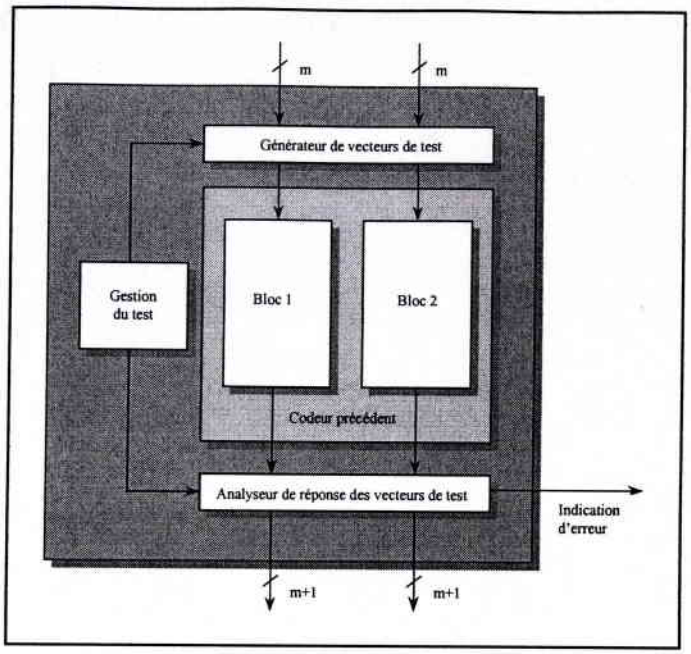


Figure IV.19 : Principe d'un codeur *Fail Safe*

Le générateur de vecteurs de test et l'analyseur doivent être transparents en dehors des phases de test.

Cette solution est cependant lourde à mettre en œuvre et doit obéir à des contraintes permettant d'éviter qu'une panne dans l'un de ces mécanismes ne puisse générer accidentellement des sorties erronées mais compatibles avec le code choisi.

La transformation du codeur en un codeur *Fail Safe* est donc relativement lourde à mettre en œuvre si l'on désire rester à un niveau d'abstraction indépendant de la structure.

3.6. Etude du décodeur

Le but du décodeur est de convertir les codes générés par le codeur en un signal compatible avec l'actionneur. Pour ce faire, le décodeur doit réaliser l'opération inverse du codeur, tout en s'assurant que la commande envoyée à l'actionneur sera celle correspondant au code reçu.

Le principe, illustré à la figure IV.20, consiste à décoder le signal issu du codeur et à l'analyser.

- S'il correspond à un niveau logique 1, la sortie du décodeur change d'état logique.
- S'il correspond à un niveau logique 0, la sortie doit conserver l'état précédent.

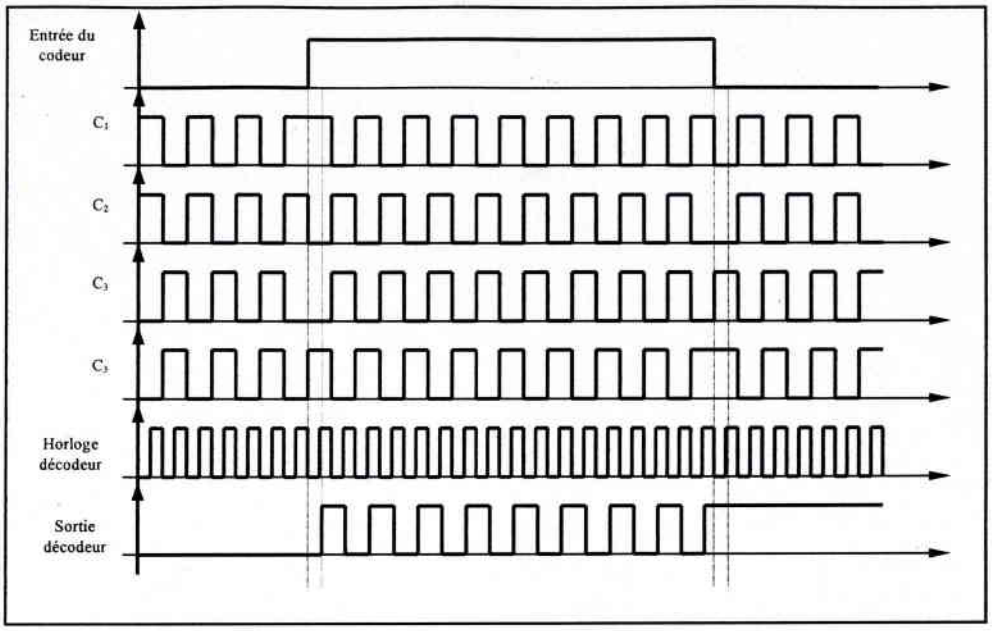


Figure IV.20 : Principe du décodeur pour m=1

Ainsi, une fréquence moitié de celle du signal d'horloge du codeur correspondra au niveau logique 1 (état *non sûr*), alors que les autres états électriques correspondent à l'état sûr.

En plus du décodage du code reçu, le décodeur doit également contrôler la validité des transitions reçues. En cas de détection d'une transition interdite, la sortie du décodeur doit être bloquée dans un état quelconque de façon irréversible.

De plus, pour garantir une grande sûreté de fonctionnement, le décodeur doit être capable de détecter ses propres pannes. Il doit donc être composé de deux blocs identiques travaillant sur des données dupliquées. Chacun des blocs vérifiant le bon fonctionnement de l'autre, de la même manière que dans le cas du codeur.

Le décodeur possède les mêmes problèmes que le codeur, c'est à dire sa grande complexité pour un nombre élevé d'actionneurs. Comme dans le cas du codeur, l'apparition d'une donnée erronée sur l'une des entrées du décodeur entraîne le blocage dans un état sûr de tous les actionneurs.

Ce décodeur, combiné au codeur précédent, permet donc d'obtenir une interface fortement sûre indépendante d'une technologie. Il est cependant plus intéressant, en terme de coût de conception ainsi qu'en terme de disponibilité du système d'éviter l'utilisation d'un décodeur. L'interface proposée dans la seconde partie de ce chapitre répondant à ce besoin, la simulation du décodeur n'a pas été effectuée.

3.7. Conclusion

Les travaux présentés dans cette section ont abouti à la conception d'une interface fiable basée sur des techniques de codage statiques et dynamiques à la fois. Ce type de codage permet de concevoir l'interface à un niveau d'abstraction très élevé, c'est à dire sans s'intéresser à la structure au niveau porte logique ou transistor.

Une telle interface peut donc être implémentée dans n'importe quelle technologie en utilisant la plupart des supports (FPGA, ASIC, ...etc.). Sa description a été faite entièrement dans le langage de modélisation VHDL au niveau comportemental.

Le taux de sûreté de l'interface peut être calculé à partir des paramètres fournis par le fabriquant selon la technologie utilisée.

L'évolution de cette interface vers la propriété *Fail Safe* a été évaluée. Il en ressort que celle-ci n'est faisable qu'en compliquant singulièrement l'interface par l'adjonction de mécanismes BIST ou en perdant la propriété d'indépendance à la technologie ou au support d'implémentation.

Le second problème posé par l'interface concerne la lourdeur du code utilisé lorsque son nombre d'entrée est élevé. Bien que la méthode de détermination des différentes transitions utilisables puisse facilement être automatisée, un code d'ordre élevé compliquera sensiblement les deux blocs du codeur et augmentera sa taille de façon considérable et donc sa probabilité de défaillance.

De plus, le codage de toutes les entrées de l'interface en un seul code entraîne le blocage de tous les actionneurs, même dans le cas d'une panne affectant la commande d'un seul actionneur.

Le dernier problème posé par ce code concerne le nombre de sorties du codeur ; et donc de liaisons entre le système de traitement et l'actionneur. Le nombre de fils deviendra rapidement dissuasif selon le niveau de complexité du système de traitement.

L'utilisation d'une transformation série-parallèle permettrait de réduire le nombre de lignes de liaison, mais compliquerait le système et risquerait de dégrader la sûreté de l'interface.

Cette interface répond donc aux objectifs initiaux mais n'est que difficilement applicable à un système réel, en raison des problèmes énoncés précédemment. Son coût de développement et sa mise en œuvre se révèlent en effet pénalisant.

Une seconde interface, reprenant les propriétés de base de celle-ci, est présentée dans la section suivante. Cette interface est beaucoup plus adaptée à un contexte réel.

4. Présentation de l'interface améliorée

4.1. Objectif

L'interface proposée précédemment a permis de valider le principe d'un système fiable basé sur une technique de codage hybride (statique et dynamique). Cette technique permet la description du système à un niveau d'abstraction élevé (VHDL). L'interface n'est pas liée à une technologie ou une structure et, par conséquent, à un type de support d'implémentation.

Toutefois, l'architecture proposée et l'utilisation du code de sortie ne permettent pas aisément l'évolution de cette interface vers un système *Fail Safe* sans se soucier de la structure interne de certains blocs de l'interface. Cette démarche permet d'atteindre l'objectif *Fail Safe*, mais n'autorise plus, dans ce cas, la définition de l'interface dans un langage de modélisation de haut niveau.

L'autre inconvénient de l'interface précédente concerne le nombre élevé de ses sorties pour un nombre important d'entrées ainsi que la lourdeur du code de sortie dans ce cas. Le type de code choisi nécessite, en outre, le décodage des commandes en ajoutant une cellule, relativement complexe, de commande des actionneurs.

L'objectif de l'interface, présentée dans la section suivante **[39]**, est donc de réduire les différents inconvénients de l'interface précédente, tout en conservant le principe du codage hybride. Cette interface est également décrite au niveau comportemental.

4.2. Principe

L'interface proposée est dédiée à des systèmes de traitement dupliqués afin de couvrir les pannes affectant l'un de ses systèmes. Elle possède *(2m+4)* sorties où *m* représente le nombre de sorties du système de traitement. Notons toutefois que seuls *2m* sorties sont transmises vers les actionneurs, et seulement deux à chaque actionneur. Les quatre autres sorties sont des sorties d'indication d'erreur, permettant le verrouillage de l'interface dans un état *sûr*.

Le schéma de principe de l'interface complète est présenté à la figure suivante, dans le cas de systèmes de traitement possédant deux sorties.

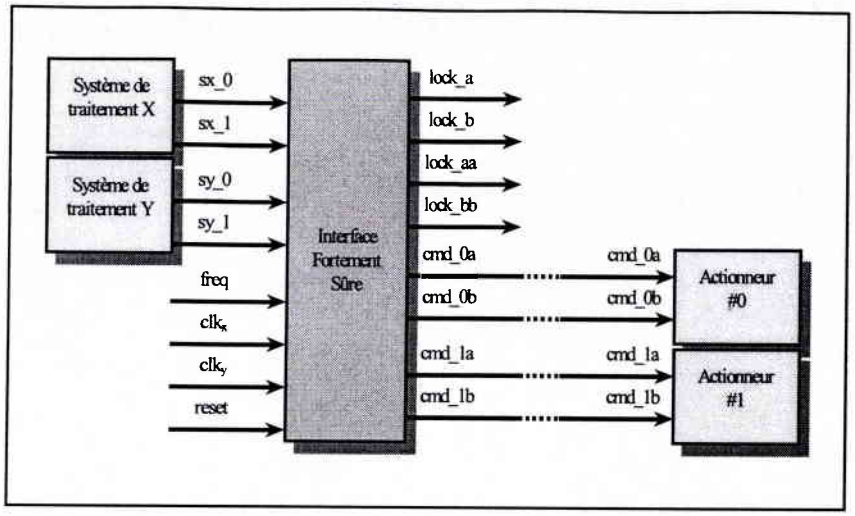


Figure IV.21 : Synoptique de l'interface complète

Les sorties *lock_a*, *lock_b*, *lock_aa*, *lock_bb*, sont les sorties de verrouillage de l'interface. Les sorties de données correspondent aux sorties *cmd_0a*, *cmd_0b*, *cmd_1a*, *cmd_1b*.

Ces sorties utilisent le principe du codage en fréquence présenté au Chapitre II où :

- La présence d'une fréquence *Freq* correspond au niveau logique 1 de la sortie du système de traitement
- Tous les autres états électriques sont considérés comme un niveau logique 0 en sortie du système de traitement.

La plupart des systèmes critiques utilisant ce principe, la présence d'une cellule de commande des actionneurs n'est donc pas nécessaire. Les sorties sont donc indépendantes, dans le sens où elles ne forment pas un mot de code comme dans le cas de l'interface précédente. La disponibilité du système en cas d'erreur de transmission est donc améliorée.

L'interface nécessite l'utilisation d'une entrée fournissant la fréquence correspondant à l'état *non sûr* ainsi que les signaux d'horloge (identiques) des deux systèmes de traitement.

4.3. Fonctionnement de l'interface

Le schéma fonctionnel de l'interface pour un nombre quelconque de sortie du système de traitement est présenté à la figure suivante :

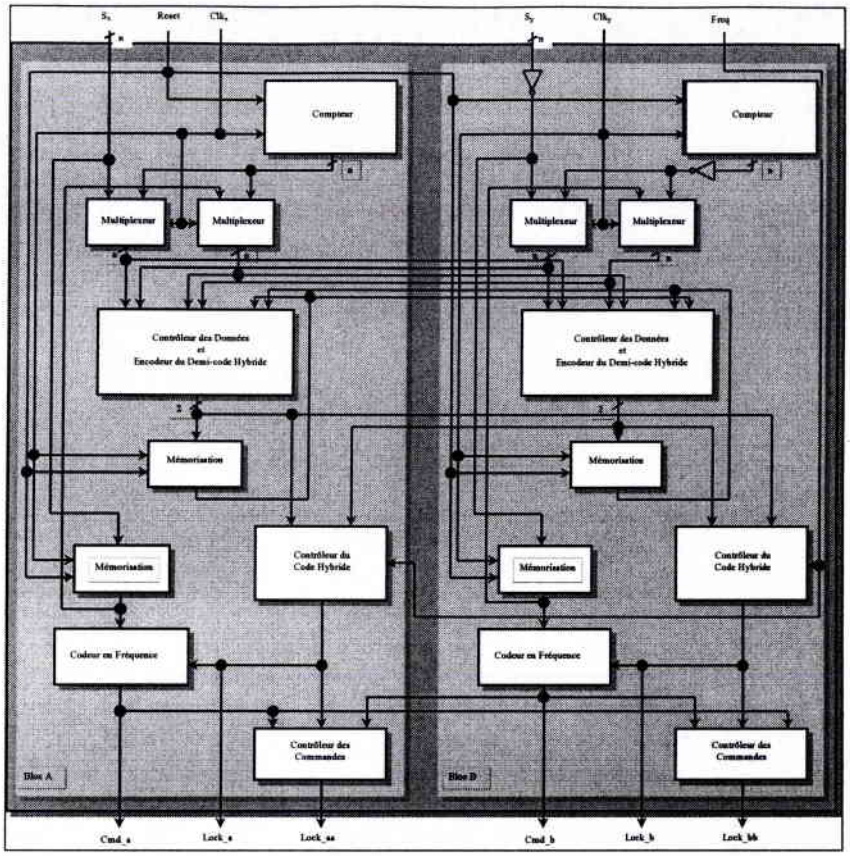


Figure IV.22 : Organisation interne de l'interface

L'interface est constituée de deux parties identiques. La fonction de chaque partie est de générer les sorties codées en fréquence correspondant à l'un des systèmes de traitement ainsi que deux indications de verrouillage de l'interface. Chacune des deux parties contrôle le bon fonctionnement de l'autre. Nous utiliserons le terme de bloc pour chacune des deux parties, et le terme de cellule pour les constituants d'un bloc.

Les deux blocs travaillent sur des niveaux logiques complémentés l'un par rapport à l'autre.

Afin d'améliorer la sûreté de fonctionnement de l'interface, un test en-ligne continu est utilisé pour chacun des deux blocs. Le test est activé par le signal d'horloge (clk_x ou clk_y). Le test est effectué pendant la durée du signal d'horloge à l'état haut, le bloc fonctionnant normalement pendant l'état bas de ce signal.

4.3.1. Fonctionnement normal (sans test)

Considérons dans un premier temps, le fonctionnement de l'interface pendant l'état bas du signal d'horloge. Le schéma fonctionnel simplifié de l'interface est alors le suivant :

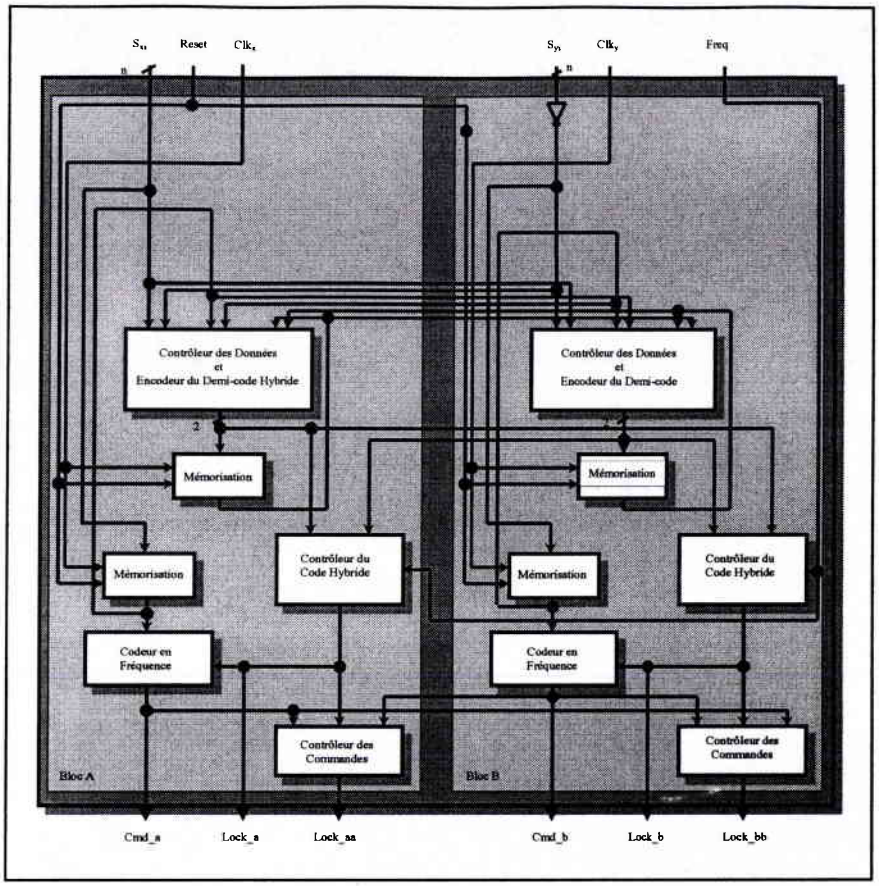


Figure IV.23 : Schéma simplifié de l'interface pendant la phase de traitement des données

Le codage en fréquence des sorties de données est effectué par la cellule «Codeur en Fréquence» (CF) en fonction des valeurs des entrées issues du système de traitement et de l'entrée fournissant la fréquence correspondant à l'état *non sûr*. Cette entrée est déconnectée de la fréquence si le «Contrôleur du Code Hybride» (CCH) détecte la présence d'une panne. Remarquons que la cellule CF est la seule qui réalise le codage en fréquence des sorties de données de l'interface. Toutes les autres cellules permettent le contrôle du bon fonctionnement de l'interface et la signalisation de l'occurrence d'une panne.

La cellule «Contrôleur des Données et Encodeur du Demi code Hybride» (CDEDH) génère le code hybride permettant la détection d'une panne dans le système. Elle utilise les données issues du système de traitement avant et après mémorisation, ainsi que ses propres sorties en comparant toutes ses données avec celles traitées par l'autre CDEDH. Elle génère alors une séquence de codes statiques dans un ordre prédéfini tant que les données contrôlées sont correctes.

La cellule CCH compare les sorties des deux CDEDH et vérifie leur conformité statique et dynamique. Le CCH génère la fréquence sur sa sortie tant que les données sont correctes. Cette sortie constitue l'une des commandes de verrouillage de l'interface.

La cellule «Contrôleur des Commandes» (CC) compare les sorties des deux CF et connecte la fréquence issue du CCH tant que les données sont correctes. Sa sortie constitue la deuxième commande de verrouillage.

4.3.2. Présentation des différentes cellules

4.3.2.1. Cellule «Contrôleur des Données et Encodeur du Demi code Hybride» (CDEDH)

Cette cellule constitue le cœur du dispositif de contrôle du bon fonctionnement de l'interface. Elle compare les signaux générés aux points stratégiques du bloc (y compris ses propres sorties) à ceux générés aux même endroits du second bloc. En cas de bon fonctionnement, elle génère un code de sortie correct. Le code produit par l'ensemble des deux CDEDH est un code hybride tel que celui utilisé dans l'interface de la section précédente. Il s'agit d'un sous-ensemble de code 2 parmi 4 pour lequel seuls les codes statiques *(0011)*, *(0110)*, *(1100)*, et *(1001)* sont utilisés.

Les structures des deux cellules CDEDH sont identiques, chacune générant la moitié du code 2 parmi 4. Les sorties d'une cellule correspondent toujours au complément logique des sorties de la deuxième cellule (en fonctionnement sans panne).

Une erreur sera donc détectée dans les deux cas suivants :

- Un code statique est en dehors des codes autorisés
- La transition entre deux codes statiques corrects n'est pas une de celles autorisées

En fonctionnement normal, les transitions autorisées entre les différents codes statiques sont représentées à la figure suivante.

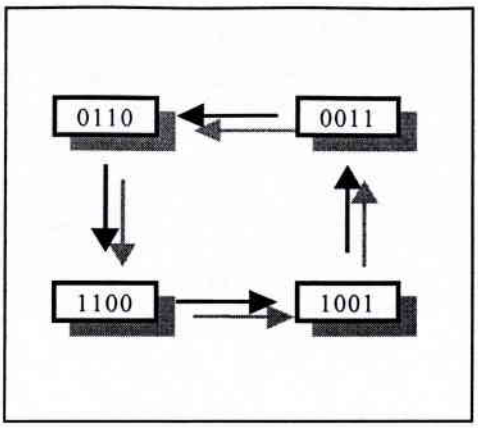


Figure IV.24 : Graphe des transitions autorisées du code 2 parmi 4

Les sorties de chacun des CDEDH évoluent selon le code Gray. En cas d'apparition d'une erreur en entrée du CDEDH, celui-ci interdit toute évolution de ses sorties, entraînant une transition erronée et un demi code erroné par rapport à celui généré par le second CDEDH. Le second CDEDH détecte alors la panne du premier et permet le verrouillage de l'ensemble de l'interface.

4.3.2.2. La cellule «Contrôleur du Code Hybride» (CCH)

Elle vérifie, d'une part, la complémentarité des sorties des deux CDEDH (donc la validité du code statique) et d'autre part la transition entre deux codes consécutifs.

Si les codes générés par les CDEDH sont corrects, le CCH génère en sortie la fréquence correspondant à l'état *non sûr*.

Dans le cas contraire, la sortie du CCH doit prendre un état logique quelconque différent de cette fréquence.

Cette sortie est utilisée par la cellule CF pour générer les sorties de données de l'interface. Elle est également propagée vers la sortie de l'interface pour verrouiller le système dans un état sûr, lors de sa disparition.

4.3.2.3. La cellule «Codeur en Fréquence» (CF)

Elle effectue le codage en fréquence des sorties mémorisées du système de traitement. Elle utilise la fréquence générée par la cellule CCH qui n'est plus présente en cas de détection d'une panne par cette dernière.

La présence d'un niveau logique 1 sur l'entrée provenant du système de traitement provoquera la génération de la fréquence lock_a sur la sortie correspondante de l'interface.

La présence d'un niveau logique 0 sur l'entrée provenant du système de traitement forcera la sortie correspondante de l'interface à un niveau logique 0.

En cas de détection de panne par le CCH, la fréquence lock_a ne pourra pas être générée sur l'une des sorties de l'interface, même si l'entrée correspondante est au niveau logique 1, car elle n'est plus générée par le CCH. Les sorties de l'interface sont alors dans un état *sûr*.

4.3.2.4. La cellule «Contrôleur des Commandes» (CC)

Elle compare les sorties de données de l'interface et connecte la fréquence lock_a issue du CCH, vers sa sortie, si les données contrôlées sont correctes. Dans le cas contraire, la fréquence lock_a n'est pas propagée vers la sortie.

Notons qu'en cas de détection d'erreur par la cellule CCH, la fréquence lock_a n'est plus disponible en entrée du CC et ne peut donc plus être générée en sortie. L'absence de fréquence sur la sortie *lock_a* entraîne obligatoirement son absence en *lock_aa*.

4.3.2.5. Les Cellules de mémorisation

Pendant la phase de traitement des données générées par les systèmes de traitement, ces données doivent être mémorisées, pour éviter qu'elles évoluent au cours de leur traitement. Cette fonction est réalisée par les cellules de mémorisation.

4.3.2.6. Propriété globale

L'interface est conçue de manière à ce que tous les tests effectués dans un bloc le soient également dans le second bloc. Chacune des cellules d'un bloc teste également sa cellule symétrique de l'autre bloc. Ainsi, l'occurrence d'une panne dans une cellule d'un bloc affecte obligatoirement une des données contrôlées par l'autre bloc et provoque automatiquement une détection d'erreur par l'une de ses cellules. Cette détection provoque obligatoirement un code erroné dans le bloc sain, et active ses sorties de verrouillage. Le bloc en panne en fera alors de même. Les blocs sont conçus de manière à ce que qu'une détection de panne provoque une indication d'erreur au niveau de toutes les cellules. Aucune panne unique ne peut alors empêcher le verrouillage de l'interface dans un état sûr, suite à sa détection.

4.3.3. Fonctionnement lors de la phase de test

Afin de garantir que l'occurrence d'une panne soit obligatoirement signalée à la sortie de l'interface, celle-ci est équipée de mécanismes permettant le test exhaustif du cœur de l'interface : le CDEDH.

Ce test est activé pendant l'état haut du signal d'horloge et le schéma simplifié de l'interface est alors le suivant :

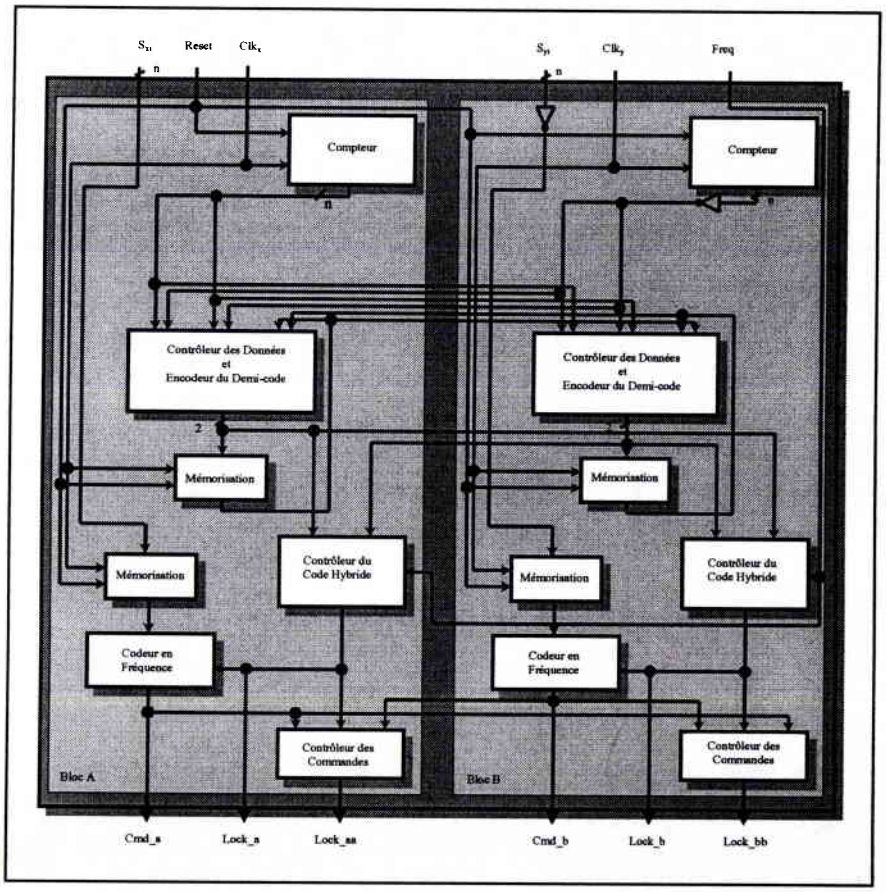


Figure IV.25 : Organisation interne de l'interface pendant la phase de test

A chaque phase de test (état haut du signal d'horloge), le compteur prend une valeur différente, qui est appliquée sur les entrées de la cellule CDEDH. Ainsi, pour un système de traitement disposant de n sorties, le CDEDH sera testé de façon exhaustive après 2^n cycles d'horloge. Pendant le test de l'interface, la cellule CF continue à générer les sorties correctes, car l'état des entrées de l'interface a été mémorisé par le registre. Si l'une des valeurs générées par le compteur provoque l'apparition d'une erreur en sortie de la cellule CDEDH, le fonctionnement sera identique à celui pendant la phase hors test.

Cette erreur sera détectée par la 2ème cellule CDEDH et les deux CCH. La fréquence lock_a ne sera plus générée par les CCH, et les sorties de données seront bloquées dans un état *sûr*. Le système sera alors verrouillé dans un état *sûr*, par l'intermédiaire des quatre sorties de verrouillage.

Le rôle des multiplexeurs est de placer, sur les entrées du CDEDH, soit les données issues des systèmes de traitement (fonctionnement normal, figure IV.23), soit les données de test de l'interface issues des compteurs (fonctionnement en test, figure IV.25).

4.4. Taux de sûreté de l'interface

Comme dans le cas de l'interface précédente, l'interface étudiée ci-dessus ne peut être que dans l'un des trois états suivants :

- L'état O1 correspond au fonctionnement normal de l'interface, c'est à dire sans panne.
- L'état O2 correspond à l'apparition d'une panne détectée par l'interface, et signalée sur ses sorties.
- L'état O3 correspond à l'état indésirable où l'occurrence d'une panne n'est pas détectée. Celui-ci est obligatoirement dû à une panne interne à l'interface, car l'occurrence d'une panne dans l'un des systèmes de traitement est obligatoirement détectée.

La figure suivante représente le graphique de transition de MARKOV entre ces trois états.

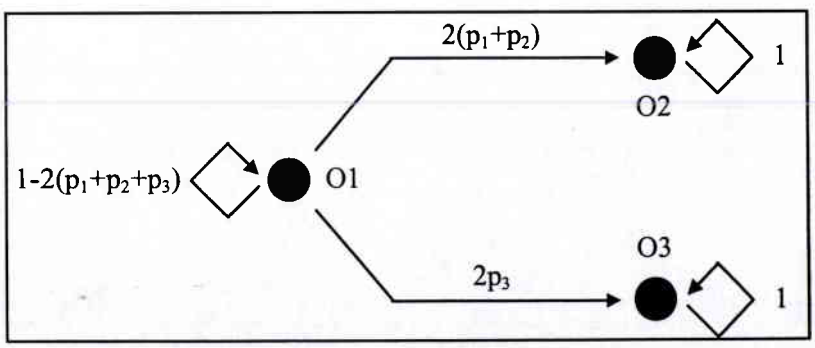


Figure IV.26 : Graphe des transitions entre les trois états de l'interface

La probabilité P de détection, et Q de non détection d'une panne dans l'interface est alors :

$$P = \frac{p_1 + p_2}{p_1 + p_2 + p_3} \quad \text{et} \quad Q = \frac{p_3}{p_1 + p_2 + p_3}$$

Où :

- p_1 représente la probabilité d'apparition d'une panne dans l'un des systèmes de traitement.
- p_2 représente la probabilité d'apparition d'une panne détectée dans l'un des deux blocs, donc propagée vers les sorties de verrouillage.
- p_3 représente la probabilité d'apparition d'une panne non détectée dans l'un des deux blocs.

La structure de l'interface permet d'obtenir $p_3 << p_2$ et donc $P \approx 1$.

Pour obtenir la propriété *Fail Safe*, il faut cependant que $P=1$, ce qui implique que $p_3=0$. Ce résultat peut être obtenu en garantissant que toutes les pannes affectant le CDEDH et le CCH seront détectées. Le test exhaustif du CDEDH est réalisé par la phase de test, mais le CCH n'est pas testé exhaustivement en raison du type de code utilisé par le CCH.

Il est possible de trouver une implémentation du CCH permettant de garantir son test exhaustif pendant le fonctionnement de l'interface, mais celle-ci serait dépendante d'une structure, donc en contradiction avec l'objectif de l'interface.

4.5. Résultats de simulation

L'implémentation FPGA de l'interface fournie les résultats de la figure 27. Cette simulation a été éffectuée pour des systèmes de traitement diposant de quatre sorties. L'interface possède alors 11 entrées (Reset, Clkx, Clky, Sx[0:3], Sy[0:3]) et 12 sorties (Cmd_a[3:0], lock_a, lock_b, lock_bb, lock_aa, Cmd_b[3:0]).

Le circuit utilisé est le FPGA « Altera-EPF10K10LC84 » basé sur une technologie SRAM. La fréquence d'horloge maximale de Clk_x et Clk_y est d'environ 50Mhz.

Le fonctionnement est normal jusqu'à 140µS. Une erreur dans l'un des systèmes de traitement est alors simulée, entraînant le blocage des sorties de verrouillages et des sorties de données.

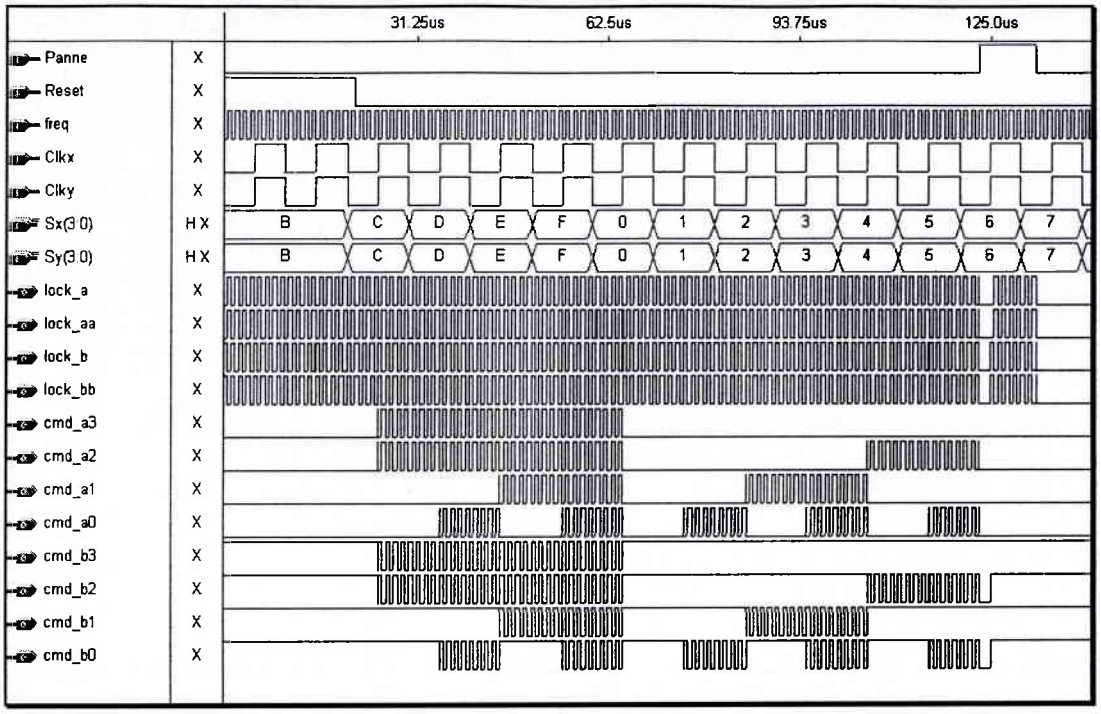


Figure IV.27: Simulation de l'interface avec une panne à 140µs

4.6. Conclusion

L'objectif de cette seconde interface était de reprendre le principe validé par la première interface, en essayant toutefois d'en éliminer quelques contraintes. Cette interface fortement fiable est également basée sur l'utilisation d'une technique de codage hybride. Sa description est faite dans un langage de haut niveau (VHDL) permettant son implémentation sur n'importe quel support.

La première amélioration par rapport à l'interface précédente concerne la simplification du codage des sorties. Le codage en fréquence utilisé permet d'éviter l'adjonction d'un décodeur au niveau de l'actionneur et permet un gain en nombre de fils transmis. L'absence de décodeur en fin de transmission participe également à l'amélioration de la fiabilité. Le codage utilisé à l'intérieure de l'interface est plus facile à mettre en œuvre.

La seconde amélioration concerne le test interne de l'interface. Celle-ci est désormais équipée de moyens de test permettant de garantir la détection de toutes les pannes pouvant affecter certaines cellules. L'agencement des différentes fonctions et le test exhaustif de certaines cellules garantissent une fiabilité plus élevée que pour la première interface. Sa transformation en interface *Fail Safe* est, de plus, beaucoup plus aisée, et indépendante du nombre d'entrées de l'interface. Elle nécessite toutefois toujours une dépendance à la structure.

5. Conclusion

L'objectif de ce chapitre était d'étudier la possibilité de réalisation d'interfaces fortement sûres, à partie d'une description de haut niveau. Cette conception à un niveau d'abstraction très élevé n'est pas dépendante d'une structure ou d'un composant, et autorise une implémentation de l'interface dans toutes les technologies, et sur n'importe quel support (ASIC, FPGA, etc.). L'utilisation du langage de description VHDL assure une compatibilité avec de nombreux logiciels de conception.

Deux types d'interfaces basées sur ce concept sont proposés. Toutes les deux utilisent le principe de codage hybride présenté dans ce chapitre, et basé sur les aspects statiques et dynamiques des codes utilisés jusqu'alors. La première interface permet de valider les propriétés de ce code pour la réalisation d'interfaces fortement sûres. Celle-ci présente un certain nombre de contraintes quant au nombre de sorties nécessaires, ainsi qu'à l'utilisation d'un décodeur au niveau des actionneurs.

La seconde interface proposée, permet, tout en réutilisant le principe du codage hybride, de se soustraire de ces inconvénients. Elle ne nécessite plus l'utilisation d'un décodeur au niveau de l'actionneur et permet de réduire le nombre de lignes de transmission. L'autre paramètre important de cette interface réside dans son architecture autorisant le test exhaustif de ces différents constituants. L'interface est donc plus fiable que la précédente, et permet une évolution beaucoup plus aisée vers la propriété *Fail Safe*.

Les deux interfaces proposées peuvent, en effet, toutes deux être transformées en interface *Fail Safe*, mais nécessitent dans ce cas, une dépendance à une structure. Leur implémentation peut alors ne plus être possible pour certains supports. Cette évolution est beaucoup moins contraignante pour la seconde interface. Notons enfin que le taux de sûreté des deux interfaces peut être calculé à partir des paramètres technologiques fournis lors de son implémentation.

Conclusion

Conclusion

Les travaux de cette étude présentent les différentes approches envisagées pour aboutir à la réalisation de systèmes intégrés à défaillances sûres (*Fail Safe*).

Cette étude découle d'un constat lié à la formidable évolution des techniques d'intégration ces dernières années. Ces progrès ont permis d'accroître sensiblement la complexité des systèmes et d'en diminuer la taille, mais n'ont pas permis de réduire le risque de panne global dans les mêmes proportions.

Afin d'inverser cette tendance, le domaine du test des systèmes intégrés fait l'objet d'une grande activité de recherche, et progresse très rapidement. La conception en vue du test (*Design for test*) est devenue incontournable pendant la phase de réalisation d'un système.

Les principales méthodes de test utilisées qui permettent actuellement d'améliorer la sûreté de fonctionnement des systèmes ont donc été recensées. Ces techniques de test, principalement les circuits auto-contrôlables, constituent les éléments de base pour la réalisation de systèmes *Fail Safe* intégrés. Ces systèmes permettent la détection immédiate de toute anomalie de fonctionnement pendant le déroulement de l'application. La récente généralisation du concept des systèmes *Fail Safe* autorise l'intégration de ces systèmes, en combinant l'utilisation des circuits auto-contrôlables à des critères rigoureux liés aux caractéristiques des systèmes critiques. Ces caractéristiques, détaillées au deuxième chapitre, rendent la réalisation de systèmes de traitement *Fail Safe* complexes très délicate voire impossible.

La solution proposée consiste donc à utiliser une interface permettant la transformation d'un système de traitement quelconque en un système *Fail Safe* en sécurisant les sorties générées par l'interface.

Le travail présenté a donc permis de valider le principe d'une telle interface. Ce travail est complété par la création d'une bibliothèque paramétrable permettant la génération automatique de l'interface *Fail Safe* en fonction du nombre de ses entrées. Les différentes parties de l'interface ont fait l'objet d'une analyse détaillée afin de mettre en évidence les problèmes liés à une réalisation pour une utilisation sur site. Cette implémentation, réalisées dans une seule technologie, peut toutefois être adaptée à d'autres technologies, selon la disponibilité de certaines cellules élémentaires.

Afin de faciliter l'accessibilité des circuits *Fail Safe*, leur implémentation en circuits programmables a été étudiée. Ces circuits présentent de nombreux avantages liés à l'infrastructure nécessaire à leur utilisation ainsi qu'en terme de délais de fabrication. Leur principal inconvénient, c'est à dire l'absence de contrôle de la structure utilisée lors de la programmation, a fait l'objet d'une étude approfondie. L'interface validée précédemment étant fortement liée à une structure, son implémentation en FPGA était fortement compromise. L'utilisation de quelques artifices permettant de garantir l'indépendance des chemins de données ont permis de résoudre ce problème. Ce travail nécessite toutefois un contrôle rigoureux de la structure générée par le circuit programmable.

Pour s'affranchir de tous les problèmes liés à la structure de l'interface, un nouveau type d'interface est proposé. Cette interface n'est plus liée à une structure mais définie uniquement au niveau comportemental. Les interfaces ainsi présentées, sont basées sur l'utilisation d'un code hybride, et permettent de garantir une sûreté de fonctionnement très élevée. Ce taux de sûreté peut être calculé à partir des données liées à la technologie choisie pour l'implémentation.

Ces interfaces peuvent évoluer vers la propriété *Fail Safe* en respectant certaines contraintes fonctionnelles. Le respect de ces contraintes nécessite toutefois la dépendance partielle à une structure.

Ces interfaces fortement sûres peuvent donc être implémentées sur tous les types de supports, quelle que soit la technologie. Leur propriété *Fail Safe* ne peut toutefois être garantie qu'après adaptation à la technologie choisie.

Ce travail permet avant tout de montrer qu'il est envisageable de réaliser des systèmes *Fail Safe* sur tous les types de support. La difficulté de réalisation dépend toutefois de ce support. Les interfaces présentées au dernier chapitre constituent un premier pas vers la réalisation de systèmes fortement sûrs indépendants d'une technologie. L'évolution vers des systèmes *Fail Safe* implémentables sur tous types de support, sur la base de ces interfaces, est envisageable.

Annexes

Annexes

Annexe 1 : Simulations de l'interface VLSI en fonctionnement normal (figure A.1) puis en présence d'une panne dans l'un des systèmes de traitement (figure A.2)

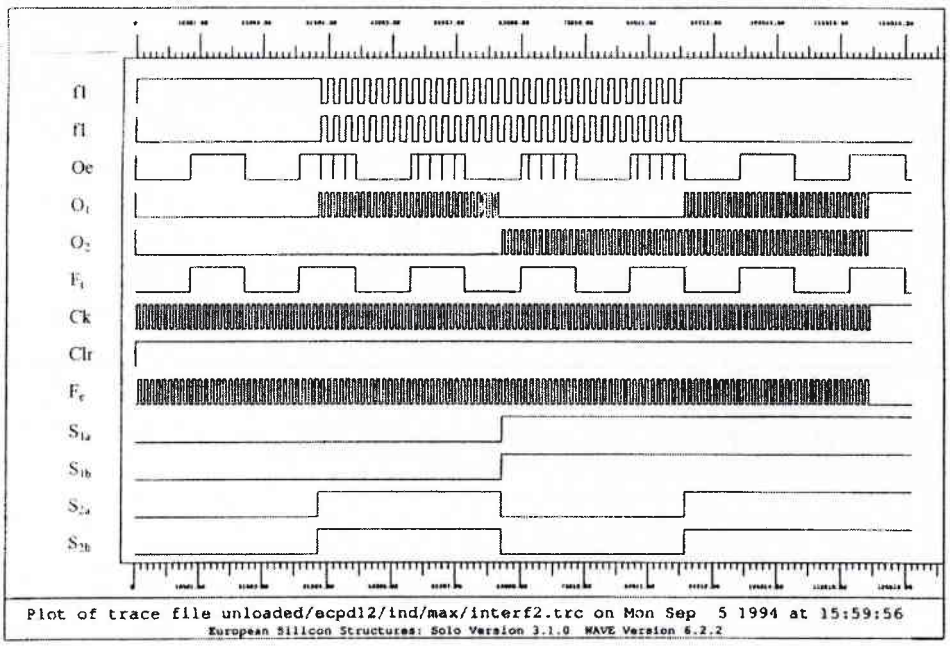


Figure A.1: Simulation de l'interface en fonctionnement normal

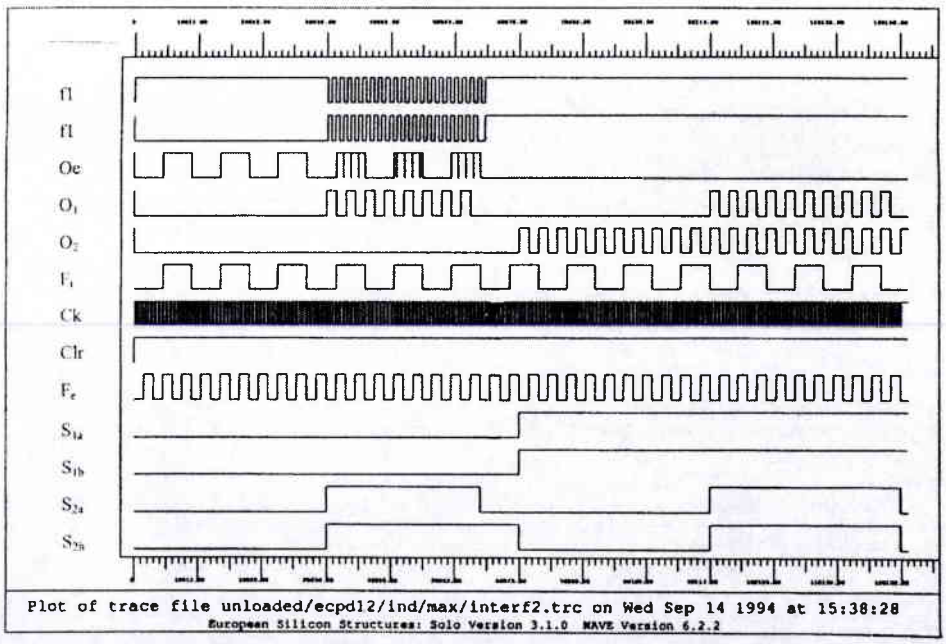


Figure A.2 : Simulation d'une panne dans l'un des systèmes de traitement

Annexe 2 : Simulations de pannes internes à l'interface

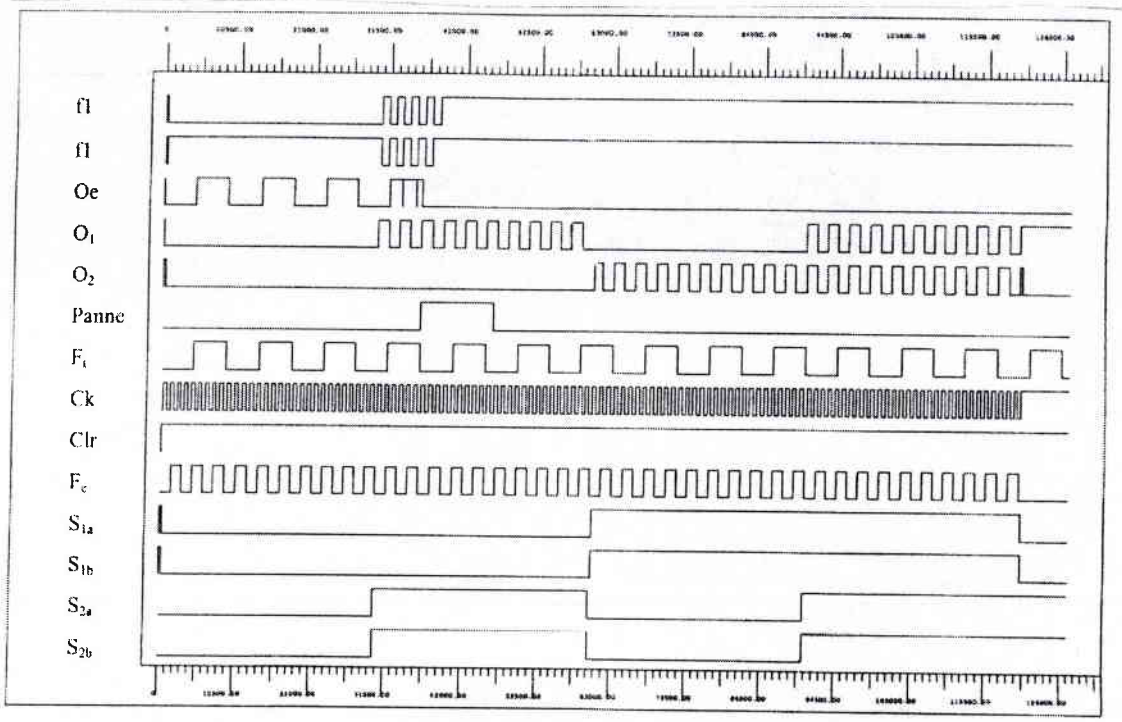


Figure A.3 : Simulation d'une panne de collage dans le contrôleur *Double Rail*

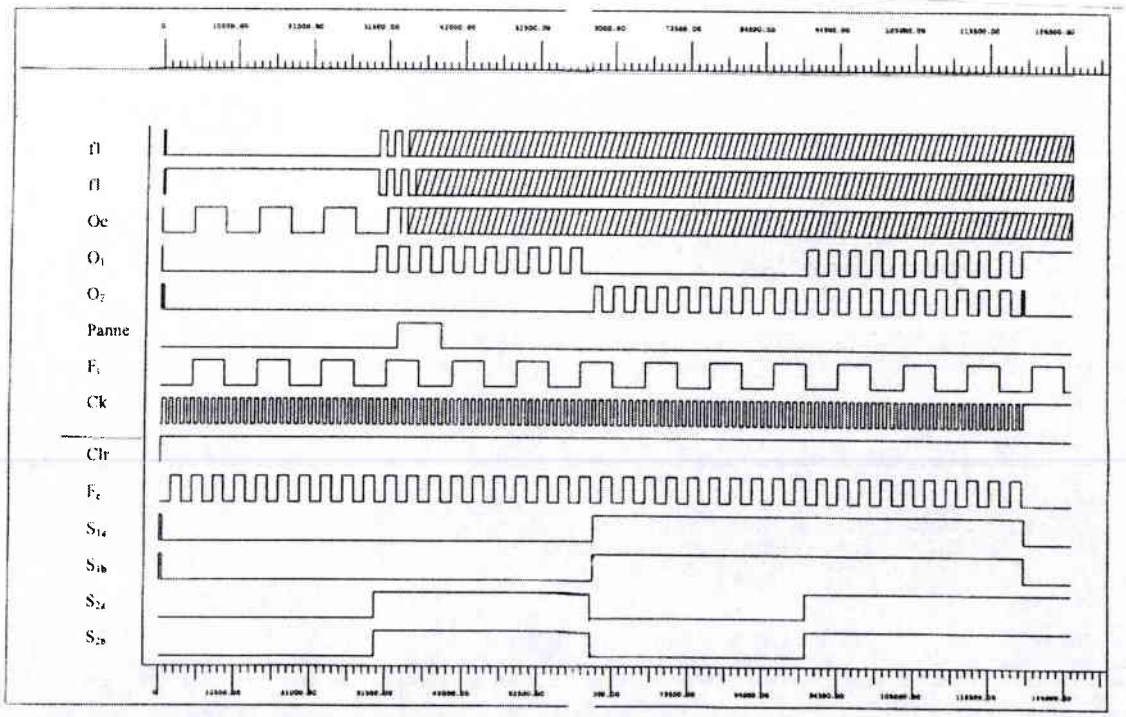
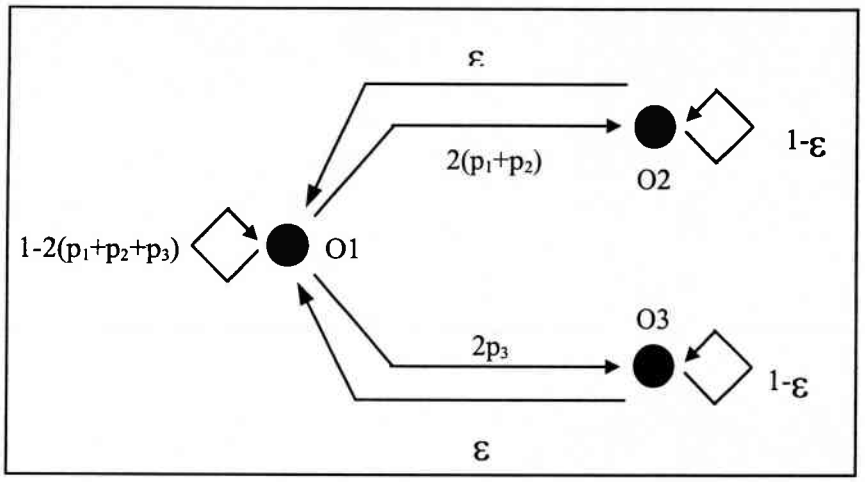


Figure A.4 : Simulation d'une panne de circuit ouvert dans le contrôleur Double Rail

Annexe 3 : Calcul de la probabilité de détection de panne dans le codeur à l'aide des chaînes de MARKOV.

La chaîne de MARKOV permettant le calcul des probabilités de détection de pannes est la suivante où $\varepsilon \to \infty$.



La matrice de transition de la chaîne de MARKOV est alors la suivante :

$$M = \begin{bmatrix} 1-2(p1+p2+p3) & 2(p1+p2) & 2p3 \\ \varepsilon & 1-\varepsilon & 0 \\ \varepsilon & 0 & 1-\varepsilon \end{bmatrix}$$

Considérons les probabilités suivantes :

- R probabilité de se trouver dans l'état O1
- P probabilité de se trouver dans l'état O2
- Q probabilité de se trouver dans l'état O3

Le calcul de ces probabilités consiste à résoudre le système suivant :

$$\begin{cases} [R+P+Q]=[R+P+Q]*M \\ R+P+Q=1 \end{cases} \text{ donc } \begin{cases} R+P+Q=1 \\ R=(1-2(p1+p2+p3)]R+\varepsilon P+\varepsilon Q \\ P=2(p1+p2)R+(1-\varepsilon)P \\ Q=2p3R+(1-\varepsilon)Q \end{cases}$$

On obtient alors les résultats suivants :

$$\begin{cases} R+P+Q=1 \\ R=\dfrac{\varepsilon Q}{2p3} \\ P=\dfrac{p1+p2}{p1+p2+p3} \\ Q=\dfrac{p3}{p1+p2+p3} \end{cases}$$

La probabilité de détection d'une panne est donc $\boxed{P=\dfrac{p1+p2}{p1+p2+p3}}$

La probabilité de non détection de panne est alors $\boxed{Q=\dfrac{p3}{p1+p2+p3}}$

RESUME

Le principal problème lié à la sûreté de fonctionnement d'un système se trouve localisé au niveau des étages de commandes unidirectionnelles. Il n'y a plus, à ce niveau, de retour d'information permettant de vérifier la bonne interprétation des commandes.

Les systèmes *Fail Safe* ont été conçus pour pallier ce problème en garantissant qu'en cas de panne, le système n'engendrera jamais de situation dangereuse. La majorité des systèmes *Fail Safe* sont ainsi réalisés à partir de composants discrets spécifiques, qui brident toutefois leur degré de complexité.

Par ailleurs, les modèles de défaillance des circuits intégrés rendent leur utilisation très délicate, pour la réalisation de systèmes *Fail Safe*. Ce constat est accentué par la densité d'intégration des circuits actuels, qui ne permet plus de couvrir l'ensemble des pannes susceptibles de se produire.

Des travaux récents permettent de concilier ces deux aspects pour aboutir à l'intégration de systèmes *Fail Safe*. Cette étude propose différentes approches permettant d'atteindre cet objectif. Ces approches sont toutes basées sur l'utilisation d'une interface dédiée permettant de transformer un système de traitement quelconque en un système *Fail Safe*.

Après un rappel des principales techniques de test en-ligne, les étapes aboutissant à la validation d'une interface *Fail Safe* intégrable en circuit VLSI sont exposées. Dans un souci d'alléger les contraintes financières et matérielles inhérentes à ce type de circuits, l'implémentation de l'interface en circuits programmables est étudiée.

La dépendance de ces interfaces à une technologie, combinée à l'évolution perpétuelle de cette dernière, nécessite leur constante réadaptation. L'étude d'une interface définie à un niveau d'abstraction plus élevé est donc proposée. Cette interface, basée sur un codage particulier, est totalement indépendante de la technologie et permet d'obtenir un système de traitement fortement sûr, voire *Fail Safe* sous certaines conditions.

MOTS CLES

Sûreté de fonctionnement – Circuits auto-contrôlables – Test en-ligne – Fail Safe – Strongly Fail Safe – Conception VLSI – Interfaces Fail Safe – Codes détecteurs d'erreurs.

ABSTRACT

The main problem concerning the system's dependability is located at the unidirectional control stages. Actually, there is no more information feedback allowing to check the correctness of the controls.

Fail Safe systems have been conceived to avoid this problem as they will never involve dangerous situations in case of failure occurrence. Most of the *Fail Safe* systems are realised from discrete components that limit their complexity degree.

Besides, the fault models of integrated circuits make it hard to realise *Fail Safe* systems. Moreover the raising number of functions assembled on a same chip increases the probability for failure occurrence while reducing the means to detect all the potential failures.

Some recent research results reconcile with both these aspects, allowing full integration of *Fail Safe* systems.

This study proposes several approaches to reach this goal. These approaches are based on the use of a dedicated interface that transforms a processing system into a *Fail Safe* one.

After a general survey of the main on-line testing techniques the validation steps of *Fail Safe* interfaces integrable into VLSI circuits are presented.

In order to lighten the financial and the equipment requirements for these systems, the implementation on programmable circuits is studied.

The dependence of these interfaces on a frequently and quickly evolving technology forces their continual readjustment. The modelisation of an interface at a higher level of abstraction is proposed. This interface, relying on a specific coding technique, is totally independent of technological choices and allows to design highly secure processing systems or even *Fail Safe* systems under some restrictions.

KEYWORDS

Dependability – Self Checking – On-line Testing – Fail Safe – Strongly Fail Safe –VLSI design –Fail Safe Interfaces – Error detecting codes.