Diagnostic automatisé en environnement de production à faible volume et haute complexité

par

Simon PICHETTE

THÈSE PRÉSENTÉE À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION
DU DOCTORAT EN GÉNIE
Ph. D.

MONTRÉAL, LE 16 SEPTEMBRE 2022

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CETTE THÈSE A ÉTÉ ÉVALUÉE

PAR UN JURY COMPOSÉ DE :

M. Claude Thibeault, directeur de thèse
Département de génie électrique à l'École de technologie supérieure

M. Stéphane Coulombe, président du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

M. Pascal Giard, membre du jury
Département de génie électrique à l'École de technologie supérieure

Mme. Catherine Laporte, membre du jury
Département de génie électrique à l'École de technologie supérieure

M. Pierre Langlois, examinateur externe
Département de génie informatique et génie logiciel à Polytechnique Montréal

ELLE A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 31 AOÛT 2022

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

J’aimerais d’abord remercier le professeur Claude Thibeault pour sa patience, sa générosité et sa sagesse. J’ai beaucoup appris à son contact, qui a contribué à faire de moi un meilleur ingénieur et une meilleure personne.

Je souhaite également exprimer ma gratitude envers mes collègues étudiants et chargés de cours, en particulier Vincent, Francis, Stéphane et Louis-Philippe, pour les discussions intéressantes et les encouragements. Merci aussi à ma collègue Christelle pour une importante leçon de persévérance appliquée.

Merci à mes parents, Suzanne et Serge, pour leur amour inconditionnel et pour m’avoir transmis le goût de la découverte et l’amour des sciences et du savoir.

Et finalement, mille mercis à mes filles Roxanne et Marianne, ainsi qu’à ma merveilleuse Manon. Vous êtes magnifiques, la meilleure famille dont un homme puisse rêver.

Diagnostic automatisé en environnement de production à faible volume et haute complexité

Simon PICHETTE

RÉSUMÉ

Les impératifs commerciaux ayant déplacé la production de cartes électroniques à grand volume vers l'Asie, les fabricants nord-américains n'ont d'autre choix que de se positionner dans les créneaux plus difficiles de la production à bas volume, de systèmes plus complexes, dits « Low-Volume, High-Mix » (LVHM).

Ces produits souvent destinés aux secteurs industriel, médical ou militaire intègrent des dispositifs dispendieux et leur coût unitaire justifie le déploiement d'un effort de diagnostic et de réparation important lors de la détection d'une défectuosité pendant la production. Bien que l'automatisation du diagnostic soit souhaitable, le faible volume limite la quantité de données disponibles et rend impraticables les techniques d'apprentissage machine conventionnelles.

Cette thèse propose une nouvelle approche basée sur la modélisation des connaissances et le raisonnement par cas pour le diagnostic automatisé de cartes de circuits imprimés adapté à l'environnement de production LVHM. Notre approche hybride, dont l'efficacité a été démontrée à l'aide d'un prototype que nous avons développé, permet de surmonter le goulot d'étranglement de l'acquisition des connaissances (*knowledge-acquisition bottleneck*) malgré le fait qu'un environnement pauvre en données soit ciblé. L'approche proposée ne nécessite pas la contribution des concepteurs ou d'experts du produit et est conçue pour opérer uniquement avec l'information disponible lors de la fabrication.

Notre approche repose sur l'utilisation d'un système raisonnant accumulant de l'expérience jumelé à un dépôt de connaissances du produit et du domaine. Afin d'accélérer l'acquisition d'expérience sur un nouveau produit, nous procédons à la génération de cas synthétiques à l'aide d'un émulateur de carte au niveau *boundary-scan* branché à l'équipement de test utilisé en production. Les résultats de nos essais démontrent que ces cas synthétiques permettent au système diagnostic de détecter, localiser et classer toutes les pannes simples ainsi que les pannes multiples touchant jusqu'à trois nœuds voisins avec un meilleur taux de succès que l'outil commercial de référence.

De plus, les données de la base de cas, incluant les informations de disposition physique des cartes (*board layout*) ainsi que les rétroactions de l'utilisateur après chaque réparation sont utilisées pour alimenter un système recommandeur chargé de faire l'analyse des causes racines des défectuosités pour identifier le composant fautif et proposer une réparation. Un simulateur de données de production et de réparation est décrit et intégré au prototype pour faire la vérification du système recommandeur, dont la performance est évaluée. Les résultats obtenus sont prometteurs et permettent d'envisager l'utilisation du système proposé dans un environnement commercial.

Mots clés : diagnostique automatisé, émulation matérielle, dictionnaire de pannes, test au niveau carte, modélisation des connaissances, raisonnement par cas, test JTAG.

Automated Diagnostic in a Low-Volume, High-Mix Production Environment

Simon PICHETTE

ABSTRACT

Commercial pressures having resulted in the transfer of most high-volume printed circuit board assembly operations to Asia, remaining North-American manufacturers have had little choice but to reposition themselves in the more difficult market segment of lower volume, higher complexity products, also called Low-Volume, High-Mix (LVHM).

These products, commonly targeted to the industrial, medical or military sectors, integrate expensive components and their high unit costs justify substantial diagnosis and repair efforts when defects are detected in production. Although automated diagnostics is desirable, the low production volumes impose severe limits on available data and make conventional machine learning techniques impractical.

In this thesis, we propose a novel approach based on knowledge modeling and case-based reasoning for automated diagnosis of printed circuit boards in an LVHM production environment. Our hybrid approach, whose effectiveness we have demonstrated using a prototype we developed, can overcome the knowledge-acquisition bottleneck even though it is targeting a data-poor environment. The proposed approach does not require a contribution from product designer or experts and is designed to operate using information available during manufacturing only.

Our approach is based on using a reasoning system to accumulate experience coupled with a repository of domain and product knowledge. In order to accelerate the acquisition of experience on a new product, we generate synthetic cases using a boundary-scan level board emulator connected to the same test equipment used on the real board in production. Our test results demonstrate that these synthetic cases allow our diagnostic system to detect, locate and classify all single faults and multiple faults affecting up to three neighboring nodes with a better success rate than the reference commercial tool.

Moreover, case base data, including board layout information and user feedback from previous repairs, are used to feed a recommender system tasked with performing root-cause analysis to identify faulty components and suggest repairs. A production and repair data simulator is described and integrated with our prototype system in order to verify functionality of the recommender system and evaluate its effectiveness. Results are promising and allow us to consider using the proposed system in a commercial environment.

Keywords : automated diagnostics, hardware emulation, fault dictionary, board-level testing, knowledge modeling, JTAG boundary scan testing.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

INTRODUCTION

Dans le contexte actuel de mondialisation, les équipementiers (« *Original Equipment Manufacturer* », OEM) confient de plus en plus la fabrication et le test de leurs produits à des sous-traitants appelés EMS (« *Electronic Manufacturing Service* »). Les EMS asiatiques, bénéficiant d'avantages majeurs au niveau des coûts de main-d'œuvre ainsi que de la réglementation environnementale et industrielle, sont spécialisés dans la production à haut volume et à bas prix. Les EMS nord-américains, qui détiennent toujours environ 17% du marché mondial, n'ont d'autre choix que de se positionner dans les créneaux plus difficiles de la production à bas volume, de systèmes plus complexes, dits « *Low Volume, High Mix* » (LVHM) (Sprovieri, 2004). En production LVHM, un grand nombre de produits différents est fabriqué sur les mêmes lignes de production, en quantités variant de quelque centaines à quelque milliers d'unités par produit par an. Une différence importante entre ce mode d'opération et la fabrication en grande série réside dans le fait qu'il soit nécessaire de reconfigurer rapidement et fréquemment l'équipement de production. La nécessité de fabriquer de nombreux produits en quantités restreintes sur de l'équipement partagé touche aussi le domaine des semiconducteurs. Les défis de prédiction du rendement et de la qualité (Okusa et al., 2020) ainsi que d'ordonnancement de la production (Boydon et al., 2021) s'appliquent également au domaine de l'assemblage des carte.

La différence de coût de la main-d'œuvre, même hautement qualifiée, dans les pays asiatiques par rapport à l'Amérique du Nord ou à l'Europe, peut atteindre un facteur de dix (Tourangeau, 2006). Cette grande disparité force les EMS nord-américains à utiliser de l'équipement de pointe, plus automatisé, afin de réduire leur besoin de main-d'œuvre au minimum. La taille réduite, le niveau technologique et l'expertise des EMS nord-américains leur confèrent également l'avantage en début et en fin de vie des produits à plus grands volumes. Pour exploiter ces avantages et renforcer leur modèle d'affaires, les EMS nord-américains doivent offrir des services à valeur ajoutée, afin de convaincre leurs clients de se départir d'une partie de leurs activités non stratégiques, par exemple au niveau de la validation et du développement des tests de nouveaux produits.

Contexte et environnement du projet

Ce projet constitue une continuation d'un projet démarré originalement en partenariat avec la société Varitron Technologies, un EMS dont le siège social est situé à St-Hubert en Montérégie et qui se spécialise dans l'assemblage de produits électroniques à faible volume et haute complexité. Le projet hôte consistait à développer une infrastructure de test et de diagnostic définie par logiciel. Cette infrastructure était perçue comme un élément important de la stratégie de Varitron Technologies visant à offrir des services à valeur ajoutée permettant de mieux accompagner leurs clients, en particulier au début du cycle de vie des produits, incluant la phase de déverminage.

Minimiser la phase de déverminage d'un produit signifie, dans un premier temps, de concevoir un produit qui a de meilleures chances d'être manufacturé avec succès, et, dans un deuxième temps, d'identifier et de corriger rapidement les problèmes causant des pertes de rendement. Ces problèmes peuvent provenir des composants à assembler sur les plaquettes de circuits imprimés ou de l'assemblage lui-même. Lorsque ces problèmes sont liés au procédé d'assemblage, leur identification peut permettre une correction des règles de conception des circuits imprimés afin d'en améliorer ultérieurement la « manufacturabilité ».

Au centre de l'infrastructure envisagée, on retrouvait une station générique de test et de diagnostic qui permettrait d'offrir une interface unifiée pour l'opérateur, combinant toutes les fonctions de test et de diagnostic en un seul outil, et dont l'interface serait commune pour tous les produits fabriqués. Notre contribution au projet original devait être le développement d'un système de diagnostic automatisé adapté à l'environnement de production à faible volume et haute complexité.

Suite à des changements à la tête de l'entreprise, Varitron Technologies a pris la décision à l'été 2016 de ne pas participer à la suite du projet. Puisque nous avions déjà identifié quelques avenues prometteuses, nous avons décidé de continuer le développement sans partenaire industriel, avec la contrainte supplémentaire qu'il nous serait désormais impossible d'accéder aux données ou à l'environnement de production.

Problématique

L'identification de la cause de la défaillance d'un produit est un exercice laborieux qui demande un grand niveau d'expertise et une connaissance approfondie du produit. Chez Varitron Technologies, comme c'est le cas pour la majorité des EMS, cet exercice demande l'implication de techniciens et d'ingénieurs, qui peuvent prendre plusieurs heures voire même parfois des jours avant de mettre le doigt sur la cause du problème. Dans le cas des EMS responsables de l'assemblage, la première impérative est de vérifier si la cause du mauvais fonctionnement n'est pas le procédé d'assemblage lui-même. Pour ce faire, le responsable du procédé d'assemblage (c.-à-d. le directeur de production) doit disposer d'informations très complètes sur son procédé de fabrication, des résultats de tests structuraux et fonctionnels ainsi que des résultats du diagnostic des produits défaillants, indiquant la cause probable des défectuosités. Idéalement, le directeur de production devrait pouvoir accéder à cette information en temps réel, afin de suivre l'évolution de son procédé et d'apporter les modifications nécessaires le plus rapidement possible, minimisant ainsi le nombre de produits défectueux fabriqués.

Malheureusement, les méthodes de diagnostic automatisé existantes nécessitent soit une très grande quantité de données de production et de défaillance, soit une contribution majeure de la part d'un expert du produit pour l'élaboration du système. Une telle contribution n'est pas envisageable pour la production en sous-traitance, à faible volume. Ce genre de production souffre également de la faible quantité de données et de produits défectueux. Ces deux contraintes, le manque de données et la nécessité de recourir au travail d'experts, ne sont pas uniques au domaine de la production à faible volume et haute complexité en électronique, mais constituent en fait l'un des problèmes universels du diagnostic automatisé connu sous le nom de goulot d'étranglement de l'acquisition des connaissances (GEAC), ou « *knowledge acquisition bottleneck* » en anglais (Buchanan et Wilkins, 1993).

Objectifs de la recherche

Ce projet vise le développement et la mise en œuvre de stratégies permettant la diminution du temps de déverminage et l'amélioration du rendement dans un environnement de production LVHM par une identification plus rapide des causes de défaillances.

L'objectif principal repose sur la disponibilité d'un système de diagnostic automatisé qui soit adapté aux contraintes particulières de la production sur une même chaîne de montage de plusieurs produits différents à volume faible ou moyen. Puisqu'il ne pourra reposer ni sur un grand volume de données de production et de défaillance ni sur la contribution d'experts de chaque produit, le système devra nécessairement proposer une solution au problème plus général du GEAC. Le développement d'un tel système constitue l'un des objectifs secondaires de la recherche.

En l'absence d'accès aux données et à l'environnement de production commercial, la validation du système de diagnostic devra être effectuée par simulation. Un simulateur de données de production et de défaillances avancé, capable de modéliser les caractéristiques propres à un produit ou à une ligne de production devra être conçu et réalisé, ce qui constitue un autre objectif secondaire de ce projet.

Contributions de la thèse

Les principales contributions de cette thèse sont les suivantes :

- Une méthode de diagnostic automatisé destinée à un environnement pauvre en donnée et capable de surmonter le GEAC en combinant l'émulation matérielle, la modélisation des connaissances et le raisonnement par cas. Cette méthode comprend :
    - La génération automatique d'un dictionnaire de pannes par émulation matérielle de circuits imprimés au niveau JTAG connecté à l'équipement de test utilisé en production.
    - Un système de diagnostic automatisé capable de détecter, localiser et classifier 100% des pannes de types « *stuck-at* », court-circuit ou pont directionnel

impliquant jusqu'à trois nœuds voisins observables avec l'équipement de test JTAG.

- Un système recommandeur permettant la suggestion de causes physiques probables lors du diagnostic et la détection de défaillance sur l'équipement de production.

- Un simulateur de données de production et de défaillances capable de modéliser les particularités d'un produit et d'une ligne de production.

Les contributions proposées dans cette thèse constituent de fait un système de diagnostic automatisé complet prenant en charge la détection, la localisation, la classification et la recherche des causes racines des défectuosités présentes sur les cartes de circuits imprimés. Le système décrit dans cette thèse a également fait l'objet d'un article de revue (Pichette & Thibeault, 2022).

Organisation de la thèse

Le chapitre 1 présente les bases théoriques du problème du diagnostic ainsi qu'une revue des différentes méthodes de diagnostic automatisé ainsi que de leurs forces et leurs faiblesses respectives. Les résultats de recherche récents sont présentés et classés selon les méthodes auxquels ils se rattachent. L'applicabilité des différentes méthodes au contexte particulier de la production à faible volume et haute complexité est évaluée et les contributions de la thèse sont positionnées par rapport à l'état de l'art.

Le chapitre 2 montre une vue d'ensemble de la méthode que nous proposons, qui est basée sur une approche hybride. Il présente également l'architecture globale du système de diagnostic automatisé que nous avons réalisé pour démontrer la validité de la méthode. Les cartes de circuits imprimés de démonstration, conçues pour ce projet, sont également introduites. La description détaillée des différents sous-systèmes constitue l'objet des chapitres suivants.

Le chapitre 3 présente l'émulateur de cartes de circuit imprimé au niveau JTAG sur dispositif programmable FPGA utilisé pour la génération automatique de dictionnaires de pannes. Il

introduit les notions de base du test par balayage des frontières et de la norme IEEE 1149.1 avant de présenter notre implémentation des éléments de base requis par la norme. L'émulateur est ensuite présenté en suivant une approche ascendante. L'assemblage des composants de base pour la reproduction du fonctionnement de la chaîne JTAG des puces est montré en premier, suivi de l'assemblage de ces puces pour former un modèle de la carte entière. Les types de défectuosités pouvant être émulées, les saboteurs et leur dispositif de contrôle et de reprogrammation sont introduits puis nous amènent à une vue d'ensemble de l'émulateur complet et de son logiciel de contrôle. Finalement, l'utilisation de l'émulateur couplé à l'équipement de test commercial et à sa suite logicielle pour la génération automatique d'un dictionnaire de pannes est montrée.

Le chapitre 4 est consacré à la représentation des connaissances dans notre système hybride. Il introduit d'abord certains éléments du formalisme des diagrammes de classe UML utilisés dans ce chapitre et les suivants. Il montre ensuite les éléments de la représentation logique du modèle de carte, suivis d'une représentation des mesures et des défectuosités. La section suivante introduit la façon dont nous avons intégré les données de fabrication assistée par ordinateur (FAO) pour reconstruire les chemins électriques et élaborer un modèle de la disposition physique des composants sur les cartes, normalement invisible au niveau JTAG.

Le chapitre 5 présente d'abord la technique du raisonnement par cas puis son utilisation dans notre système de diagnostic automatisé. La représentation des cas et les classes de défectuosités sont introduites puis les notions de similarité et de distance entre les cas sont présentées, suivies du choix des critères de sélection et de l'attribution des poids relatifs. Une expérimentation visant à démontrer la capacité du système à détecter, localiser et classifier les défectuosités, en s'améliorant avec l'expérience par apprentissage machine, est présentée. Un autre volet expérimental montre la transférabilité de l'apprentissage effectué sur un produit vers un autre produit. Finalement, l'utilisation de cas synthétiques générés à partir du dictionnaire de panne élaboré au chapitre 3 pour accélérer de façon importante la phase d'apprentissage est présentée puis les performances du système sont comparées à celles d'un outil de diagnostic JTAG commercial.

Le chapitre 6 montre la conception et l'utilisation d'un second système de raisonnement par cas exploitant les mêmes données de façon cette fois à formuler des recommandations plutôt que comme classificateur. Ce sous-système est responsable de la recherche des causes racines des défectuosités. Ce chapitre présente d'abord les classes de causes physiques de défaillance, ainsi que les taux d'occurrence relatifs trouvés dans la littérature. Le tout est suivi par la présentation de deux applications envisagées pour ce système, soient la suggestion de causes physiques au technicien lors du diagnostic et la détection de bris sur une ligne de production par la surveillance des courbes de taux de succès de la recommandation. Afin de valider le fonctionnement du système en l'absence de données de production réelles, nous avons conçu un simulateur de cas capable de modéliser les particularités d'un produit et d'une ligne de production. Ce simulateur est présenté et décrit en détail avant d'être utilisé pour mesurer la performance du système recommandeur pour les deux applications.

Le dernier chapitre présente les conclusions de la thèse ainsi qu'une discussion des travaux futurs envisagés.

CHAPITRE 1

REVUE DE LITTÉRATURE

Le problème de la vérification, du test et du diagnostic des circuits électroniques assemblés est inhérent au domaine et existe depuis aussi longtemps que l'industrie. Chaque avancée technologique permettant une augmentation de la complexité des circuits doit être accompagnée d'avancées technologiques ou méthodologiques complémentaires au niveau de la vérification, du test et du diagnostic afin de maintenir le rendement élevé et le coût unitaire faible.

Dans ce chapitre, nous proposons un survol des notions de base du domaine ainsi qu'une taxonomie des différentes approches de diagnostic automatisé proposées et utilisées dans les quatre dernières décennies. Nous présenterons également les résultats de recherches récentes touchant à chacune des approches. Finalement, nous décrirons comment notre projet se positionne parmi ces recherches récentes.

1.1 Causes de défaillances des circuits électroniques

Selon (Lirov, 1989), il est possible de regrouper les causes de défaillance en trois grandes catégories : (1) les dispositifs défectueux, (2) les défectuosités d'assemblage (ex. ponts de soudure, dispositif manquant, défaut d'alignement, etc.) et (3) les problèmes opérationnels, par exemple des dispositifs en apparence fonctionnels qui entraînent des problèmes de délais, lorsque combinés. En accord avec cette catégorisation, une stratégie de test en trois phases est généralement appliquée dans l'industrie. Les cartes assemblées sont donc inspectées (visuellement ou aux rayons X), testées électriquement puis au niveau fonctionnel.

L'augmentation de la complexité et de la densité des circuits provoque une augmentation parallèle de la difficulté du test. L'intégration de méthodologies de conception en vue du test (*design for test*) est devenue nécessaire et a entraîné à son tour l'émergence de technologies novatrices comme le test par chaîne de balayage et de standards industriels comme la norme

IEEE1149.1 JTAG. L'intégration de la chaîne de balayage dans les dispositifs (puces) et sur les cartes permet de réduire la surface occupée par les éléments nécessaires au test sur les cartes (Parker, 2016).

1.2 Définition du diagnostic

Toujours selon (Lirov, 1989), le diagnostic des circuits électroniques est un processus qui permet de déterminer les causes de comportement erroné (en sortie) du circuit pour une certaine combinaison (valide) de valeurs d'entrées.

Plus formellement, soit un circuit $P=P(S,B)$, modélisé par sa structure S et son comportement B. La structure $S=S(C,T)$ est définie par l'ensemble C des composants et la topologie T sur l'ensemble C. Le comportement $B=B(IO,R)$ est défini par deux modèles mathématiques: (1) le modèle IO décrivant les relations entrée-sortie de chacun des composants $c \in C$ et (2) le modèle R décrivant la fiabilité de chaque composant $c \in C$.

Un composant $c \in C$ est dit **défaillant** si son comportement observé b_o est différent de b_e, le comportement attendu selon le modèle IO ($b_e \in IO$).

Problème du diagnostic

Soit un circuit à composants multiples P et un syndrome indiquant une anomalie par rapport au comportement IO attendu. Le problème du diagnostic se résume à identifier le sous-ensemble $C_f \in C$ des composants défaillants.

Puisque l'opération d'identification du problème ci-dessus est elle-même un problème complexe, (Lirov, 1989) identifie six sous-problèmes que doivent surmonter tous les systèmes de diagnostic automatisé :

1. **Génération d'hypothèse**

   À partir d'un syndrome et des résultats des tests ou mesures déjà effectués, obtenir une estimation du sous-ensemble défaillant C_f.

2. **Planification**

   À partir d'une estimation de C_f, obtenir la séquence de mesures ou tests supplémentaires de coût minimal permettant d'affirmer ou d'infirmer l'hypothèse. Puisque le problème de la génération d'une séquence idéale est NP-complet (Ibarra et Sahni, 1976) (Hyafil et Rivest, 1976), de nombreuses approches par heuristique ou inférence ont été et continuent d'être proposées (Amati, 2009) (Bolchini, 2013, 2014).

3. **Raisonnement**

   Trouver une méthode efficace de raisonner à propos des circuits électroniques. Ce sous-problème implique la manipulation de structures de données qui représentent à la fois les informations structurelles et comportementales et le raisonnement sur ces deux axes.

4. **Représentation**

   Trouver une façon efficace de représenter le circuit $P=P(S,B)$. L'efficacité est ici déterminée par la complétude, la régularité, la transparence et la calculabilité (Winston, 1984).

5. **Acquisition et modélisation des connaissances**

   Trouver une méthode efficace pour acquérir et emmagasiner les connaissances d'experts du domaine à propos du diagnostic des circuits électroniques. Aussi appelée ingénierie des connaissances (*knowledge engineering)*.

6. **Interface utilisateur**

   Trouver une façon efficace de conduire un dialogue fructueux entre le système et l'opérateur.

La définition de Lirov décrit le problème du diagnostic de façon abstraite, du point de vue du chercheur. On retrouve aussi dans la littérature une description centrée sur les tâches à accomplir par un système de diagnostic automatisé, telles que perçues par l'utilisateur.

1.3 Tâches d'un système de diagnostic automatisé

Selon (Fenton et al, 2001), du point de vue de l'utilisateur, les systèmes de diagnostic automatisés utilisés en contexte industriel sont chargés de 3 tâches :

1. Détecter les défaillances,
2. Localiser les défaillances,
3. Faire la classification des défaillances.

Les systèmes de pointe, tels que décrits dans (Ye et al., 2015) doivent de plus faire l'analyse des causes racines pour :

4. Proposer des suggestions de réparation à l'utilisateur.

La réalisation de ces tâches étant un des objectifs de ce projet, il est important de considérer les facteurs propres à l'environnement de production visé avant l'évaluation des méthodes et approches proposées dans la littérature.

1.4 Défis particuliers de la production LVHM

Les caractéristiques particulières de la production à faible volume et haute complexité entraînent les conséquences suivantes au niveau du diagnostic :

- La complexité des produits rend leur diagnostic plus difficile et coûteux.
- Le petit nombre d'unités défaillantes augmente la difficulté pour les techniciens et ingénieurs assignés au diagnostic, car ils ne peuvent accumuler beaucoup d'expérience avec un produit.

- Le faible volume réduit aussi considérablement la génération de données à propos des produits et du procédé de fabrication. De fait, c'est un environnement qui sera considéré comme pauvre en données (*data poor*).
- Le recours fréquent à la sous-traitance a comme conséquence que les données de CAO ne sont pas toujours disponibles pour l'assembleur. Seul un ensemble plus restreint, appelé données de « fabrication assistée par ordinateur » (FAO) est généralement disponible.

1.5 Taxonomie des principaux systèmes de diagnostic automatisé en électronique

Il existe de nombreuses méthodes de diagnostic automatisé. Afin de pouvoir les comparer, nous présentons brièvement dans cette section les plus importantes d'entre elles et nous identifions leurs forces et leurs faiblesses, en tenant compte des particularités de notre environnement.

Il est possible de regrouper ces méthodes en trois catégories ou générations (Fenton et al. 2001, 2002). On retrouve d'abord les approches traditionnelles, qui n'utilisent ni modélisation ni apprentissage machine, ensuite les approches basées sur des modèles et finalement les approches utilisant l'apprentissage machine. La taxonomie proposée ici est également inspirée de celle proposée par (Binu et Kariyappa, 2017) pour le diagnostic des circuits analogiques. La figure 1.1 illustre la taxonomie retenue. Les trois prochaines sections décrivent plus en détail chacune de ces trois approches. Notons que ces différentes approches ne sont pas mutuellement exclusives et que par conséquent, des approches hybrides sont possibles. Cet aspect sera également abordé à la section 1.9.

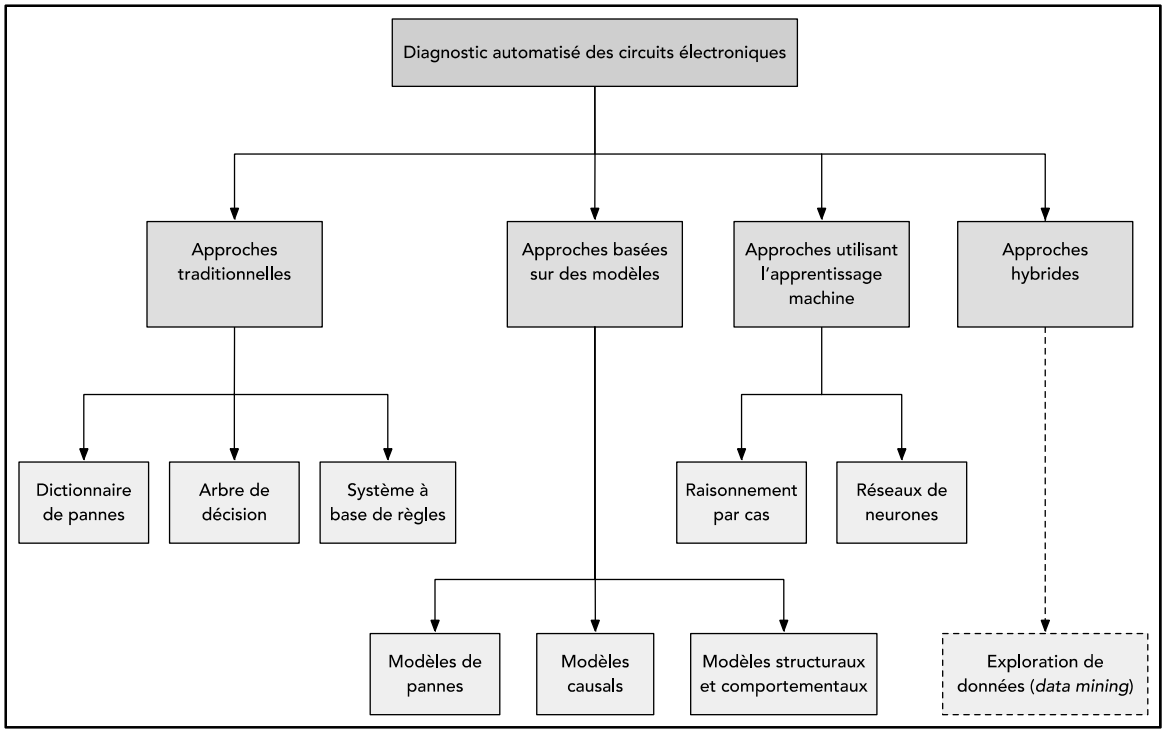


Figure 1.1 Taxonomie des méthodes de diagnostic automatisé

1.6 Approches traditionnelles

Ces approches incluent l'utilisation de dictionnaire de pannes, les arbres de décision informatisés et les systèmes à base de règles. Notons que ces approches sont apparues dès le début de la recherche sur l'intelligence artificielle dans les années 1960, et sont considérées comme des approches traditionnelles pour le diagnostic automatisé (Fenton et al. 2001).

1.6.1 Dictionnaire de pannes

Le dictionnaire de pannes est un dépôt de connaissances où sont associés les comportements défaillants du système ainsi que les causes anticipées de ces derniers. Les comportements sont généralement représentés par des échantillons de sorties erronées. Lors de l'utilisation, il suffit théoriquement de retrouver le comportement correspondant dans le dictionnaire pour en obtenir la cause. Les dictionnaires de pannes sont traditionnellement générés par simulation des comportements fautifs les plus probables ou fréquents.

Forces

- Utilisation très simple et rapide.
- Génération relativement simple.
- Requiert peu d'expertise du produit pour la génération.

Faiblesses

- Croissance très rapide du dictionnaire pour les systèmes complexes.
- Généralement incomplets puisqu'il est difficile d'anticiper tous les comportements erronés possibles.
- Souvent ambigus, des comportements similaires pouvant avoir des causes différentes.

1.6.2 Arbre de décision

L'arbre de décision est une méthode traditionnelle et très répandue de documentation des procédures de diagnostic, antérieure à l'informatique. Les symptômes en constituent la racine, et un embranchement d'actions, de résultats et de décisions guide l'utilisateur jusqu'à une recommandation.

Forces

- Utilisation intuitive.
- Modèle traditionnel utilisé dans les manuels de réparation.
- Permet à l'utilisateur non-spécialiste d'arriver à un diagnostic rapidement.

Faiblesses

- Croissance très rapide de l'arbre dans le cas des systèmes complexes.
- Fortement lié au système et tout changement à ce dernier, même mineur, peut entraîner la nécessité de modifications importantes de l'arbre.
- Difficulté d'acquisition des connaissances nécessaires à la construction de l'arbre; c'est le GEAC, qui affecte à différents degrés toutes les méthodes de diagnostic automatisé.

1.6.3 Système à base de règles

Les systèmes à base de règles tentent de représenter les connaissances d'un diagnosticien expérimenté sous une forme propositionnelle : SI symptôme(s), ALORS défaillance(s).

Un grand nombre de règles, de quelques centaines à plusieurs milliers, sont nécessaires pour représenter adéquatement le domaine d'application. La plupart des systèmes de diagnostic automatisé développés dans les années 1970 et 1980 étaient de ce type (Rowland et Jain, 1993). Plusieurs sont toujours utilisés aujourd'hui, dans les réseaux téléphoniques, les disques durs, les ordinateurs et les systèmes de contrôle d'avionique. Les systèmes experts, très populaires dans les années 1980 et 1990, sont des systèmes à base de règles.

Forces

- Simplicité.

Faiblesses

- Très fortement affecté par le GEAC.
- Incapacité à traiter une panne imprévue.
- Dépendant du système. Un nouvel ensemble de règles doit être généré pour tout nouveau type de système.

1.7 Approches basées sur des modèles

À partir du milieu des années 1980, les approches reposant sur la modélisation sont venues remplacer graduellement les approches traditionnelles dans la littérature et dans l'industrie. Nous les avons regroupés selon la nature de la modélisation.

1.7.1 Modèles de pannes

Très utilisée dans le domaine des circuits numériques, et très bien adaptée aux circuits de types combinatoires, la modélisation de panne consiste à effectuer un ensemble de simulations, dans

lesquelles chacun des types de pannes (par exemple, les courts-circuits, les bits « collés » ou les délais excessifs) est inséré tour à tour dans chacun des composants du système.

Chaque simulation produit une description de l'opération du système en présence d'une panne donnée. Les résultats de simulations sont analysés afin de fournir une liste de doublets pannes/symptômes qui serviront de base à la construction d'un dictionnaire de pannes. Le dictionnaire est à son tour utilisé lors du diagnostic pour identifier le composant fautif selon les symptômes rencontrés (Simpson, 1996). Ceci constitue un exemple d'approche hybride.

Forces

- Modélise avec précision les pannes rencontrées dans les circuits combinatoires.

Faiblesses

- Incapable de traiter une panne imprévue (non simulée).
- Plus difficiles à utiliser dans le cas des circuits séquentiels. La division du circuit en portions plus petites, appelée encapsulation, est une solution possible (Simpson, 1996).
- Une très grande quantité de vecteurs de test peut être nécessaire pour les circuits de grande taille, ce qui allonge de façon excessive la durée des tests. La compression de donnée peut être appliquée comme méthode de mitigation (Chess, 1999).

1.7.2 Modèles causals

Le modèle causal est un graphe orienté où les nœuds représentent les variables du système modélisé et les arcs représentent les relations ou les associations entre les variables. Dans le cas d'un modèle utilisé pour le diagnostic, les nœuds correspondent aux symptômes et aux pannes et les arcs représentent les associations symptôme-panne. Le poids relatif ou une probabilité est associé par un expert du domaine à chacun des arcs.

Forces

- Représentation plus compacte des connaissances structurées que l'ensemble de règles menant à des avantages au niveau du temps de calcul.
- Basé sur la théorie mathématique des probabilités.

Faiblesses

- Une grande expertise du domaine d'application est nécessaire pour construire le modèle, c'est à nouveau le GEAC.

1.7.3 Modèles structuraux et comportementaux

Pour ces modèles, une double représentation de la structure et du comportement du système à l'étude est utilisée. La représentation structurelle est une liste de tous les composants ainsi que de leurs interconnexions. Le modèle comportemental de son côté décrit le fonctionnement correct de chacun des composants.

Forces

- Utilisation de modèles corrects permettant théoriquement le diagnostic de toutes les pannes possibles.
- Les données des outils de conception assistée par ordinateur (CAO) peuvent être utilisées pour la génération automatique des modèles.

Faiblesses

- Une grande puissance de calcul est nécessaire dans le cas des systèmes complexes. L'utilisation conjuguée de modèles de pannes peut améliorer l'efficacité.
- La représentation comportementale de composants complexes comme les microprocesseurs avancés et les systèmes sur puces (*System-on-a-Chip*, SoC) est très difficile.
- Si l'autogénération à partir des données de CAO n'est pas possible, le développement des modèles peut s'avérer excessivement long.

- Ne considère pas les défauts physiques. Par exemple un pont de soudure entre deux composants ne sera pas représenté par le modèle structurel.

1.8 Approches utilisant l'apprentissage machine

Les approches conventionnelles ont toutes en commun d'offrir un niveau de performance constant selon l'implémentation. Le succès ou l'échec du diagnostic n'influence pas l'utilisation future.

Les approches utilisant l'apprentissage machine profitent des expériences passées pour informer le système et améliorer ses performances futures. Nous présentons d'abord ici l'approche du raisonnement par cas, qui reproduit le processus naturel de résolution de problème utilisé (souvent inconsciemment) par l'humain, soit de faire appel à une expérience similaire sans être identique et de l'adapter au problème actuel (Richter et Weber, 2013).

Ensuite, nous présentons brièvement les techniques utilisant les réseaux de neurones artificiels comme celle de l'apprentissage profond (*deep learning*), qui sont en plein essor à l'heure actuelle. Très utilisés pour supporter le diagnostic en imagerie médicale (Buettner et al. 2020), de nombreuses applications au domaine du test des puces et des systèmes électroniques sont envisagées, avec une attention particulière au problème de l'explicabilité (*explainability*) des conclusions (Amrouch et al, 2021).

Il est également possible d'utiliser les techniques d'exploration de données (*data mining*) pour l'extraction automatique des connaissances, qui pourront être mises à profit par la suite dans la construction de systèmes utilisant toutes les approches présentées.

1.8.1 Raisonnement par cas

La technique du raisonnement par cas (*case-based reasoning*, CBR) consiste à conserver les données et les résultats des diagnostics précédents pour ensuite les adapter et les réutiliser lors

de diagnostics futurs. La boucle de rétroaction permet d'améliorer rapidement l'efficacité du système. Il s'agit d'une forme de raisonnement inductif.

Une solution CBR est constituée en cinq phases distinctes (Richter et Weber, 2013) :

1. Lors de la **représentation de cas**, les données pertinentes sont sélectionnées, des structures de données appropriées sont choisies ainsi qu'une méthode d'indexation qui favorisera un rappel efficace.
2. Le **rappel de cas** consiste à identifier les éléments qui résument le problème ou le cas actuel, puis à utiliser ces éléments afin de retrouver des cas similaires emmagasinés dans la mémoire. Dans le cas où plusieurs candidats sont disponibles, celui offrant la plus grande similarité doit être sélectionné.
3. Au cours de l'étape de **réutilisation de cas**, les différences entre le cas actuel et le cas passé sélectionné lors du rappel sont identifiées. Le cas passé est ensuite adapté pour qu'il corresponde au cas actuel. Les méthodes d'adaptation incluent la substitution de valeurs, et la transformation basée sur des heuristiques.
4. La **révision de cas** implique l'évaluation de la solution proposée sur le cas réel et sa correction en cas de besoin. La correction constitue un apport de connaissances spécifiques au domaine.
5. L'étape finale est celle de l'**apprentissage ou rétention**, où l'information utile produite pendant la résolution du problème vient bonifier l'ensemble de cas. La conservation des résultats négatifs comme positifs est importante afin d'éviter de reproduire des erreurs déjà corrigées.

Forces

- Permet d'atteindre des taux de succès supérieurs à 90% lorsqu'une base de cas substantielle est constituée.
- La simulation et les données historiques peuvent être réutilisées pour informer la base de cas (Zhang et al., 2011).

- Le développement est plus facile que dans le cas d'un système traditionnel à base de règles puisque l'acquisition de connaissances est incrémentale et que la contribution d'experts du domaine nécessaire est limitée. (Jin et al, 2016).

Faiblesses

- Nécessite un volume de cas important pour atteindre un taux d'efficacité intéressant.
- Une nouvelle base de cas doit être générée pour chaque produit. (Sugimatsu, 1994) propose l'utilisation de deux bases, l'une, générique, partagée par tous les produits et la seconde, plus spécifique, propre à chacun des produits.
- L'efficacité peut être affectée par la qualité des algorithmes de rappel et d'indexation, surtout dans le cas d'une base de cas volumineuse.

1.8.2 Apprentissage machine par réseaux de neurones artificiels

Les réseaux de neurones artificiels sont des systèmes de traitement de l'information dont la structure est inspirée de la structure biologique du cerveau. Il s'agit de graphes dirigés et pondérés où les nœuds sont appelés des « neurones » et où les liens pondérés appelés « synapses » forment les connexions entre les neurones. Les neurones sont organisés en couches. Une fonction de propagation calcule l'entrée d'un neurone à partir des sorties de ses prédécesseurs. En règle générale, les neurones d'une couche sont uniquement connectés aux neurones des couches immédiatement précédente et suivante.

(Watt et al., 2020) présente le pipeline traditionnel de l'apprentissage machine par réseaux de neurones artificiels :

1. Collecte des données (*Data Collection*)
2. Ingénierie des caractéristiques d'intérêt (*Feature Design*)
3. Entraînement du réseau (*Model Training*)
4. Validation du modèle (*Model Validation*)

Bien qu'ils existent depuis des décennies, les applications et la performance des réseaux de neurones artificiels étaient limitées par des contraintes importantes d'espace mémoire et de puissance de calcul disponibles. L'augmentation rapide de la performance des ordinateurs depuis le début du siècle a permis la résurgence de ces techniques ainsi que l'exploitation de réseaux comportant un très grand nombre de couches internes. Ces réseaux, dits d'apprentissage profond (*deep learning*), ont permis une véritable révolution dans les domaines de l'analyse d'image, de la reconnaissance de la voix ou encore dans le traitement d'information en langage naturel.

Forces

- Permet le diagnostic des systèmes très complexes, pour lesquels la construction d'un modèle traditionnel est très coûteuse (Zhang et al., 2011).
- Nécessite peu de contribution d'experts du domaine par rapport aux méthodes traditionnelles ou basées sur des modèles (Totton et Limb, 1991).
- Les données historiques de réparation peuvent être utilisées pour l'entraînement.

Faiblesses

- Nécessite un grand volume de données pour l'entraînement.
- Très sensibles à la qualité des données fournies pour l'entraînement, les réseaux mal entraînés peuvent s'avérer fragiles ou peu performants (Chowdhury et al, 2022).
- Il peut être difficile pour l'opérateur humain de comprendre les causes qui ont mené aux conclusions d'un réseau de neurones multicouches, comme ceux utilisés en apprentissage profond. Ce manque d'explicabilité (ou d'interprétabilité) peut réduire le niveau de confiance envers le système diagnostic (Amrouch et al, 2021).

1.9 Systèmes hybrides

Puisque toutes les méthodes de diagnostic automatisé élaborées jusqu'à présent comportent des forces et des faiblesses, il est rare qu'elles soient utilisées de façon exclusive dans un système de production. La plupart des systèmes de diagnostic automatisé modernes sont donc

des hybrides, qui utilisent une combinaison de techniques pour accomplir les différentes tâches présentées à la section 1.4.

La combinaison du raisonnement par cas et de la modélisation est particulièrement fréquente, puisqu'elle permet de séparer l'algorithme de raisonnement de la représentation du domaine (Dendani et al, 2012).

À la figure 1.1, on peut remarquer la présence d'un bloc, lié par un lien pointillé à celui des approches hybrides : celui de l'exploration de données ou *data mining*. Le lien pointillé est utilisé en raison du fait que l'exploration de données n'est pas une approche complète pour la construction d'un système de diagnostic automatisé, mais plutôt un outil important utilisé dans la réalisation de systèmes hybrides.

Une façon de vaincre le *knowledge acquisition bottleneck* est l'extraction de connaissances ou d'informations de cas complets dans les bases de données existantes, parfois très imposantes. Il existe certains domaines où le volume de données historiques est très important, ce qui favorise ces techniques. Le diagnostic automobile est un bon exemple et de tels systèmes ont été développés chez certains grands constructeurs (Richter & Weber, 2013).

L'utilisation de l'exploration de données peut contribuer à diminuer l'impact du *knowledge acquisition bottleneck* sur un système utilisant l'une des autres méthodes présentées. Il faut alors comparer le coût d'implémentation de l'exploration de données aux coûts reliés à l'implication d'un expert du domaine.

1.10 État de l'art

Dans cette section, nous présentons les résultats de recherche récents concernant les méthodes de diagnostic automatisé, en particulier les méthodes utilisées pour le diagnostic des circuits électroniques au niveau carte.

Nous avons regroupé les publications selon les catégories présentées dans la taxonomie de la section 1.4. Les approches basées sur l'apprentissage machine et les systèmes hybrides, qui combinent plusieurs approches, sont dominantes dans la littérature récente. Nous avons classé les articles décrivant des systèmes hybrides selon l'approche destinée à la contribution principale de l'article.

1.10.1 Approches traditionnelles

(Nigh et al., 2021) propose l'utilisation d'un système expert intégré directement à l'équipement de test automatisé (*Automated Test Equipment*, ATE) des puces électroniques. Le système peut contrôler l'ATE afin de réaliser de façon autonome des tests permettant de valider ses hypothèses et procéder au diagnostic de façon incrémentale. Un prototype simple est construit et comparé au traitement manuel dans le cas de défectuosités touchant la chaîne de balayage des puces. Le système expert présente des avantages intrinsèques au niveau de l'explicabilité du diagnostic. Les auteurs mentionnent les difficultés reliées à l'acquisition des connaissances, mais ne proposent pas de solution.

1.10.2 Approches basées sur des modèles

L'approche incrémentale présentée dans (Amati et al., 2009) vise à réduire le temps de diagnostic en effectuant seulement un sous-ensemble des tests disponibles, en utilisant le syndrome partiel obtenu pour sélectionner le prochain test à effectuer jusqu'à l'identification du composant fautif. Elle repose sur l'utilisation d'un modèle probabiliste. L'approche a été testée sur 10 circuits d'essais et a permis une réduction de l'ordre de 30% du nombre de tests nécessaires pour arriver au diagnostic, par rapport au nombre de tests disponibles.

(X. Zhang et al., 2012) présente une base mathématique théorique pour un système de diagnostic à base de modèles qui soit à la fois sain (« *sound* ») et complet. La méthodologie permet de développer un moteur de diagnostic en sélectionnant des tests dérivés d'un modèle. Comme la méthodologie est cependant très lourde dans le cas d'un système complexe, il est nécessaire d'augmenter le niveau d'abstraction du modèle.

(Bolchini et al., 2013) présente une technique utilisant certains algorithmes de l'exploration de données afin de découvrir les relations entre les vecteurs de test et le composant fautif identifié. La technique d'exploration utilisée est celle de la recherche des règles d'association (*association rule mining*). Les modèles système utilisés sont très simples. Les résultats expérimentaux sont obtenus en testant 5 circuits d'essais synthétiques non décrits. Une réduction du nombre de tests variant de 32% à 88% est annoncée, ainsi qu'un taux de succès de 100%.

1.10.3 Approches utilisant l'apprentissage machine

(Z. Zhang et al, 2011) propose une méthodologie de diagnostic basée sur les réseaux de neurones artificiels. Plutôt qu'un seul réseau multicouche, les auteurs utilisent un ensemble de réseaux à une seule couche, un par composant de la carte. Cette approche réduit l'occupation mémoire et le temps d'entraînement. L'entraînement est effectué à l'aide des données historiques de réparation. Le système est utilisé expérimentalement pour le diagnostic de commutateurs de réseau industriels et comparé à un réseau de neurones à trois couches conventionnel ainsi qu'à un système de raisonnement par cas utilisant un classificateur Bayesien simple et montre un taux de succès amélioré et un temps d'entraînement réduit.

(Ye et al., 2012) présente une implémentation adaptative (utilisant l'apprentissage machine) d'un système de diagnostic automatisé à base d'arbre de décision et compare ses performances à celles d'un système équivalent utilisant les réseaux de neurones et les machines à vecteurs de support. L'implémentation est un arbre binaire où le syndrome jugé le plus discriminatif est placé à la racine et où les recommandations de réparations constituent les feuilles. La technique présentée permet d'atteindre des taux d'efficacité comparables sur les deux produits à grand volume testés tout en diminuant le nombre de syndromes à évaluer par un facteur de dix, ce qui diminue le temps de diagnostic.

(Dendani et al., 2012) propose une architecture hybride pour un système de raisonnement par cas intégrant une modélisation des connaissances sous la forme d'un arbre des termes et

concepts du domaine, aussi appelé « ontologie ». Cette modélisation des connaissances leur permet d'opérer efficacement dans un environnement pauvre en données. Cette approche, nommée « *Knowledge-Intensive Case-Based Reasoning* » repose sur des bibliothèques libres existantes pour former un système en trois parties, soit un dépôt de connaissances, un système d'apprentissage machine utilisant le raisonnement par cas et un module de raisonnement sémantique faisant office de médiateur et assurant le lien entre les deux dépôts de données (ontologie et base de cas). Une preuve de concept est présentée, mais n'inclut pas l'étape cruciale d'adaptation (ou « révision de cas » dans la terminologie du CBR présentée à la section 1.9.1) qui est laissée à des travaux futurs. Bien que l'application ciblée soit le diagnostic des turbines à vapeur utilisées pour la génération électrique, et bien que nous n'ayons utilisé aucune des bibliothèques présentées, l'architecture générale en trois parties formée d'un dépôt de connaissances, d'un système CBR et d'un médiateur central a servi d'inspiration au système que nous proposons et qui sera décrit au chapitre suivant.

(Ye et al., 2013) présente une méthodologie d'évaluation des syndromes et des composantes causales (*root-cause components*) afin de sélectionner un ensemble réduit, mais plus efficace de syndromes pour l'engin diagnostique. La technique vise à améliorer l'efficacité du moteur diagnostique par l'élimination des syndromes redondants et des composantes causales ambigües. L'efficacité de la méthode est ensuite évaluée sous l'angle de la précision et du rappel (*recall*), des mesures statistiques utilisées dans l'évaluation de performance des applications de l'apprentissage machine.

(Ye et al., 2014) propose deux techniques pour accélérer l'acquisition de connaissances d'un moteur de diagnostic automatisé. La première technique, appelée découverte des connaissances (*knowledge discovery*) repose sur l'exploration de données afin d'identifier des relations entre les syndromes et les causes racines. Les probabilités d'occurrences d'un mot clé sont évaluées sur l'ensemble des syndromes afin de découvrir des patrons. Par exemple, les syndromes contenant le mot clé « ECC » ou « DDR » auront tendance à se rapporter à des problèmes causés par une mémoire. Ces relations sont représentées sous la forme d'une distribution de probabilité. C'est une méthode semi-manuelle. La seconde technique, appelée

transfert de connaissance (*knowledge transfer*) et destinée au raisonnement par cas, permet de réutiliser partiellement la base de cas d'un produit pour un nouveau produit similaire.

(Bolchini et al., 2014) présente une comparaison de six techniques d'apprentissage machine (exploration par règles d'association, arbres de décision, réseaux Bayesiens, *k* plus proches voisins, machines à vecteurs de support et réseaux de neurones) pour l'implémentation d'un moteur de diagnostic incrémental. L'évaluation porte sur un système synthétique, mais réaliste, reproduisant un ordinateur personnel typique. Trois scénarios de test sont évalués. L'exploration par règles d'association et les arbres de décisions sont les plus efficaces selon les scénarios évalués, mais aucune méthode n'est dominante pour tous les tests.

(Jin et al., 2016) présente une comparaison de l'impact du choix d'une méthode de gestion des syndromes manquant pour différents systèmes de diagnostic automatisé basés sur l'apprentissage machine. Cinq méthodes de gestion des syndromes manquants sont présentées et évaluées pour quatre systèmes de raisonnement par cas utilisant respectivement les machines à vecteur de support, les réseaux de neurones, un classificateur Bayesien simple et un arbre de décision informatisé. Les résultats expérimentaux montrent que la méthode de la présélection des caractéristiques avec imputation des étiquettes est la plus performante en terme de précision du diagnostic et de temps d'entraînement pour les méthodes d'apprentissage machine étudiées.

(Liu et al., 2022) s'intéresse au problème du transfert de connaissance dans les systèmes utilisant l'apprentissage machine. Un algorithme et une méthodologie utilisant l'adaptation du domaine (*domain adaptation*) sont proposés et testés afin d'améliorer la performance d'un système de diagnostic automatisé sur un nouveau produit, dont la base de cas de défaillance et de réparation est très limitée, à l'aide des données acquises sur des produits matures. La technique proposée permet d'améliorer le taux de succès du diagnostic de façon marquée pour les tout premiers cas (quelques dizaines jusqu'à une centaine). L'écart se réduit cependant à mesure que le volume augmente.

1.11 Applicabilité des différentes approches à la production LVHM

Suite à la description des méthodes décrites à la section précédente, il s'avère important de les confronter aux contraintes spécifiques à l'environnement de production visé. Voici quelques constats issus de cette analyse :

- Le nombre d'unités défaillantes est généralement insuffisant pour alimenter un système de raisonnement par cas ou un réseau de neurones artificiels directement. L'apprentissage machine pourrait cependant être applicable au niveau des défaillances causées par le procédé de fabrication partagé par tous les produits.
- L'effet du GEAC est très important, car l'investissement d'expertise initial ne sera recoupé que sur un petit nombre d'unités. Les systèmes à base de règles, les arbres de décision et la modélisation causale sont particulièrement affectés.
- Il est possible de générer automatiquement les modèles structuraux et comportementaux à partir des données de conception assistée par ordinateur (CAO) utilisées lors du développement, mais l'absence de prise en compte des défectuosités physiques limite leur utilité.
- L'utilisation de systèmes hybrides s'impose afin de mitiger la sensibilité particulière des différentes approches aux particularités de l'environnement.

1.12 Positionnement et originalité de la recherche

Les équipes de Duke University (Jin, Liu, Ye, Chakrabarty, Z. Zhang) et de Politecnico di Milan (Amati, Bolchini) ont en commun de s'être positionnés au niveau du test fonctionnel, alors que nous travaillons en amont, au moment où toutes les inspections et les tests électriques sont complétés, mais sans tenir compte des tests fonctionnels. Notre approche est complémentaire et permet de s'assurer que toutes les défaillances causées par des problèmes structurels sont prises en compte.

Les approches de ces deux groupes ont également en commun d'être orientées vers le concepteur du produit et de se concentrer vers l'identification d'erreur de conception. À l'opposé, nous sommes orientés vers les besoins du manufacturier et nous mettons l'accent sur

l'identification de défectuosités provoquées par le procédé de fabrication. En cela aussi notre approche est complémentaire, mais traite d'aspects négligés par les autres groupes.

Finalement, les approches proposées ne tiennent pas compte des défis particuliers de la production à bas volume et haute complexité et seraient difficilement adaptables à cet environnement. À l'opposé, bien qu'elle puisse être généralisée, notre approche se concentre principalement sur ces particularités.

La recherche proposée tire donc son originalité du fait qu'elle cible l'environnement de production à bas volume et haute complexité. La plupart des articles récents répertoriés jusqu'ici s'intéressent plutôt aux environnements de production à grand volume de cartes électroniques, de manière implicite ou explicite. Cibler l'environnement de production à bas volume et haute complexité ouvre la voie à l'innovation quant aux moyens permettant de pallier la faible quantité d'unités défaillantes. L'utilisation de l'émulation sur FPGA pour l'entraînement d'un système de diagnostic est à notre avis clairement novateur. Sans surprise, il n'existe pas actuellement de publication sur le sujet.

De plus, nous proposons un système de diagnostic complet, capable de s'acquitter des quatres tâches attendues d'un tel système à notre époque, soit la détection, la localisation et la classification des défaillances suivies d'une analyse des causes racines permettant d'émettre des suggestions de réparation. Nous proposons également au travers des différents modules du système, des solutions à chacun des six sous-problèmes du diagnostic automatisé identifiés par Lirov et décrits à la section 1.2.

Le tableau 1.1 montre lesquels des six sous-problèmes identifiés par (Lirov, 1989) sont couverts par les articles répertoriés. La couverture des quatre tâches des systèmes de diagnostic automatisés tels que perçues par l'utilisateur est montrée au tableau 1.2.

Tableau 1.1 Couverture des sous-problèmes du diagnostic automatisé dans la littérature

Article	Génération d'hypothèses	Planification	Raisonnement	Représentation	Connaissances	Interface utilisateur
Nigh et al., 2021	X	X				
Amati et al., 2009	X	X				
X. Zhang et al., 2012	X	X		X		
Bolchini et al., 2013	X	X	X	X	X	
Z. Zhang et al, 2011			X	X	X	
Ye et al., 2012	X		X	X	X	
Dendani et al., 2012					X	X
Ye et al., 2013	X	X				
Ye et al., 2014			X		X	
Bolchini et al., 2014		X	X			
Jin et al., 2016			X	X	X	
Liu et al., 2021				X	X	

Tableau 1.2 Couverture des quatre tâches d'un système diagnostic dans la littérature

Article	Détection	Localisation	Classification	Recommandation
Nigh et al., 2021	X		X	
Amati et al., 2009		X		
X. Zhang et al., 2012			X	
Bolchini et al., 2013	X	X		
Z. Zhang et al, 2011			X	
Ye et al., 2012	X	X	X	X
Dendani et al., 2012			X	
Ye et al., 2013				X
Ye et al., 2014			X	X
Bolchini et al., 2014			X	
Jin et al., 2016	X	X	X	
Liu et al., 2021	X	X	X	

1.13 Conclusion

Dans ce chapitre, nous avons présenté une définition détaillée du problème du diagnostic automatisé ainsi que de ses composants. Nous avons ensuite présenté les différentes méthodes utilisées dans les dernières décennies pour résoudre ce problème, en identifiant leurs forces et leurs faiblesses respectives. Nous avons finalement présenté l'état de l'art dans le domaine du diagnostic des circuits électroniques au niveau cartes puis nous avons positionné notre recherche par rapport aux travaux des autres groupes et selon les particularités de notre environnement industriel.

Dans le prochain chapitre, nous présenterons une vue d'ensemble à haut niveau du système que nous proposons, afin de clarifier le rôle et le positionnement des grands modules qui seront ensuite présentés de façon détaillée dans les chapitres 3 à 6.

CHAPITRE 2

DIAGNOSTIC CENTRÉ SUR LES CONNAISSANCES AVEC RAISONNEMENT PAR CAS ET GÉNÉRATION DE CAS SYNTHÉTIQUES

2.1 Introduction

Le système que nous proposons combine plusieurs approches pour former un ensemble unique, capable de minimiser l'impact du goulot d'étranglement de l'acquisition de connaissances (GEAC) et permettant le diagnostic automatisé avec apprentissage machine en environnement de production à bas volume et haute complexité. Pour débuter, nous effectuerons un survol du système entier. Nous présenterons ensuite les éléments logiciels et matériels constituant notre preuve de concept. Finalement, nous présenterons les cartes de circuits imprimés que nous avons conçues pour nos essais. Les chapitres suivants s'intéresseront plus en détail aux différents composants du système de diagnostic.

2.2 Classification du système

Puisque le système est appelé à évoluer dans un environnement de production à faible volume, il ne peut se baser que sur les données récoltées pendant la production, dont la quantité sera insuffisante. Il devra donc compenser en intégrant des connaissances qui lui permettront de poser un diagnostic éclairé avec un minimum d'information. Dans la littérature sur l'apprentissage machine, une telle approche est qualifiée de pauvre en données (*data-poor*) et de centrée sur les connaissances (*knowledge-intensive*) (Dendani et al., 2012).

2.3 Sélection des sources de connaissances à modéliser

Pour déterminer la nature des connaissances à modéliser, nous avons choisi de nous inspirer du travail des techniciens qui font le diagnostic sur la chaîne de production. Les connaissances du technicien sont bien entendu attribuables à une multitude de sources, mais nous considérons que les principales sont les suivantes :

1. Connaissances générales du domaine, acquises principalement au cours de la formation du technicien.
2. Expérience générale de diagnostic accumulée sur tous les produits, après la formation.
3. Connaissances spécifiques du produit à diagnostiquer, provenant de l'observation directe, mais aussi des plans, listes de pièces et autres données manufacturières.

De la même façon, notre système comportera 3 sources principales d'information :

1. Un système à base de règles, qui modélisera les connaissances du domaine du diagnostic en électronique numérique, et qui sera utilisé principalement pour la **détection** et la **localisation** des défectuosités.
2. Un système de raisonnement par cas dont la base de cas contiendra tous les résultats de diagnostics précédents. Ce composant sera utilisé principalement pour la **classification** des défectuosités.
3. Les données de disposition physique (*board layout*) des traces et composants montés sur les cartes. Ces données seront utilisées pour la détermination des **causes racines** des pannes ainsi que pour l'identification des nœuds partenaires dans le cas des pannes de type court-circuit ou pont directionnels.

2.4 Démarrage accéléré par la génération de cas synthétiques

L'une des innovations proposées dans cette thèse est d'utiliser un système d'émulation de carte au niveau JTAG pour générer des cas synthétiques en utilisant le même équipement de test que celui qui sera connecté à la carte physique lors de la production. Ces cas synthétiques permettront d'accélérer le démarrage du système et d'obtenir une classification correcte des pannes beaucoup plus rapidement.

2.5 Architecture du système proposé

L'architecture du système que nous proposons est inspirée en partie de l'architecture générale du système de KI-CBR (*Knowledge-Intensive Case-Based Reasoning*) présentée dans (Dendani et al., 2012). Il est composé d'un système de raisonnement par cas interagissant avec

un dépôt de connaissances du domaine par l'entremise d'un médiateur. Dans le cas de notre application, le médiateur constitue l'application principale et est également chargé des interactions avec l'utilisateur à l'aide d'une interface graphique. Nous avons choisi de modéliser les connaissances d'une manière différente, soit en utilisant un ensemble de règles pour la détection et la localisation des pannes sur un circuit électronique ainsi que des modélisations de la carte aux niveaux logique et physique plutôt qu'une ontologie du domaine. Un autre élément distinctif de notre architecture est le générateur de cas synthétiques, qui alimente directement la base de cas. Un schéma-bloc du système proposé est présenté sur la figure 2.1. Notre système est donc un hybride de quatre techniques, présentant des éléments (1) des systèmes à base de règles, (2) des dictionnaires de pannes, (3) des modèles structuraux et comportementaux et (4) de l'apprentissage machine basée sur le raisonnement par cas.

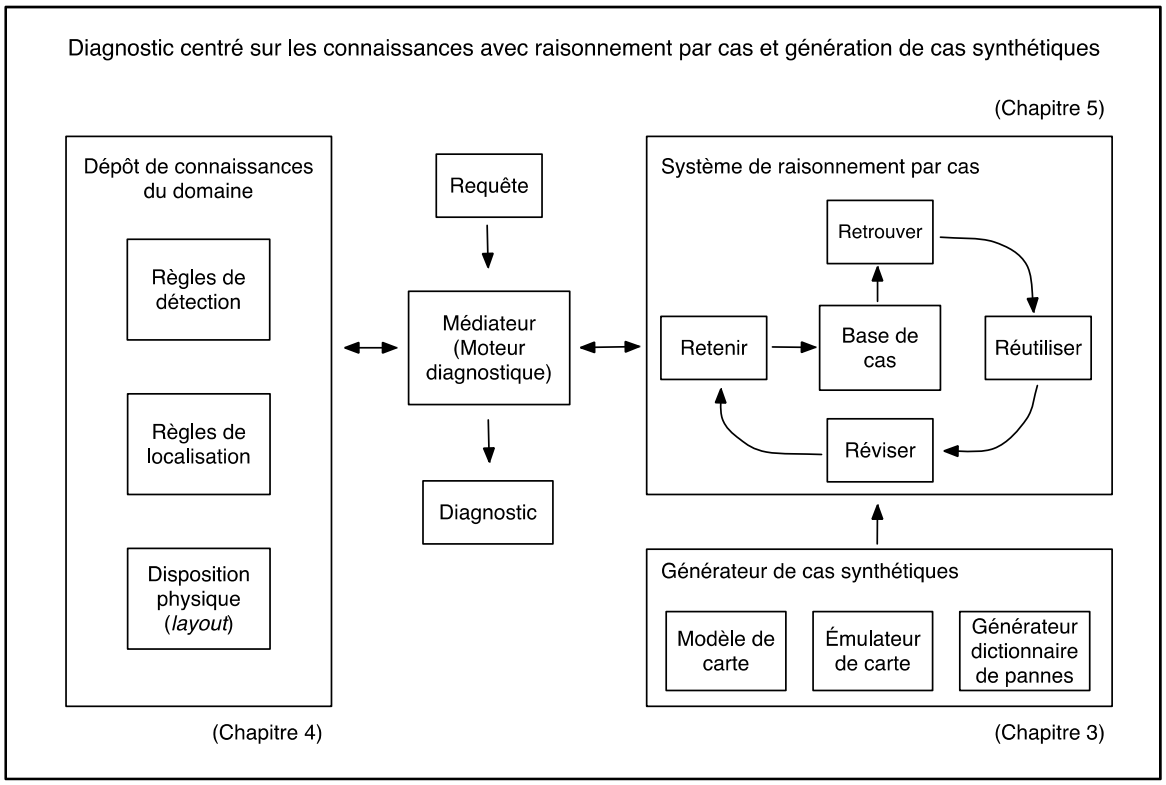


Figure 2.1 Schéma bloc du système de diagnostic automatisé

2.6 Preuve de concept

Afin de démontrer la viabilité du concept et de mesurer son efficacité, nous avons conçu et implémenté un ensemble d'outils matériels et logiciels que nous appelons la preuve de concept. Le tableau 2.1 présente quelques détails sur ces outils.

Tableau 2.1 Composants de la preuve de concept

Nom	Langage et envergure	Module de la figure 2.1	Description
Émulateur de carte	VHDL 7950 lignes Assembleur 1322 lignes	Générateur de cas synthétiques	Système d'émulation de pannes au niveau JTAG basé sur un modèle de la carte et un ensemble de saboteurs reprogrammables. Contrôle des saboteurs et communication avec le PC.
Emucontrol	C++ 1274 lignes	Générateur de cas synthétiques	Contrôle interactif de l'émulateur de pannes à partir d'un PC à l'aide d'une interface graphique.
Siggen	C++ 1241 lignes	Générateur de cas synthétiques	Génération automatique d'un dictionnaire de pannes.
Prototype	Java 4512 lignes	Système de raisonnement par cas. Dépôt de connaissances du domaine. Médiateur	Prototype du système diagnostic. Alimenté par les requêtes soumises et par le générateur de cas synthétiques. Interfaces graphiques pour la gestion de la base de cas, la rétroaction post-réparation, le chargement d'un dictionnaire de pannes et la visualisation du modèle de disposition physique.
Simulateur	Java 653 lignes	Non montré, validation du prototype	Génération aléatoire paramétrable d'informations de production, de défectuosités et de causes physiques.

2.7 Cartes d'essai

Afin de faciliter le développement et la validation de l'ensemble des éléments de la preuve de concept, nous avons conçu deux cartes de circuit imprimé à l'aide d'outils de conception assistée par ordinateur (CAO).

2.7.1 Carte A

Nous avons conçu la carte A au tout début du processus, afin de valider les éléments de base de la modélisation de carte au niveau *boundary-scan* et la génération automatisée d'un dictionnaire de pannes. Nous avions besoin d'un dispositif simple, compatible JTAG et pour lequel un fichier de *Boundary Scan Description Language* (BSDL, voir section 3.4.3) complet était disponible. Nous avons retenu le Texas Instruments SN74BCT8244ADW en format SOIC. Il s'agit d'un dispositif dédié au test qui intègre une chaîne JTAG complète à un tampon octal 74x244. La figure 2.2 montre une représentation de la carte assemblée alors que la figure 2.3 permet de visualiser le routage des deux couches de signal. Les dimensions réelles de la carte sont de 60 mm par 30 mm. La carte A est conçue comme un véhicule de test et déverminage uniquement et n'est pas représentative de l'environnement de production ciblé.

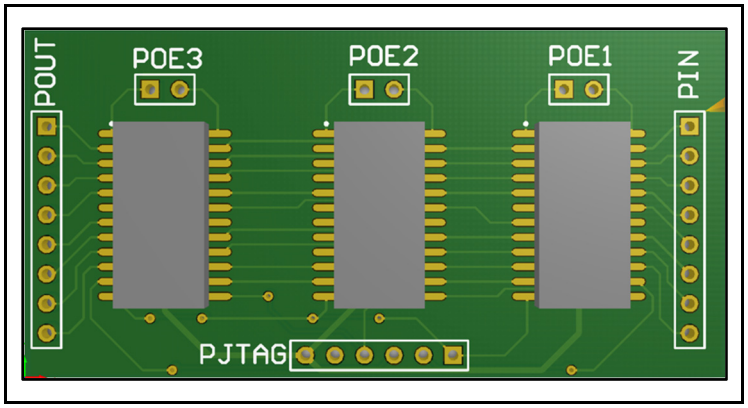


Figure 2.2 Représentation 3D de la carte A assemblée

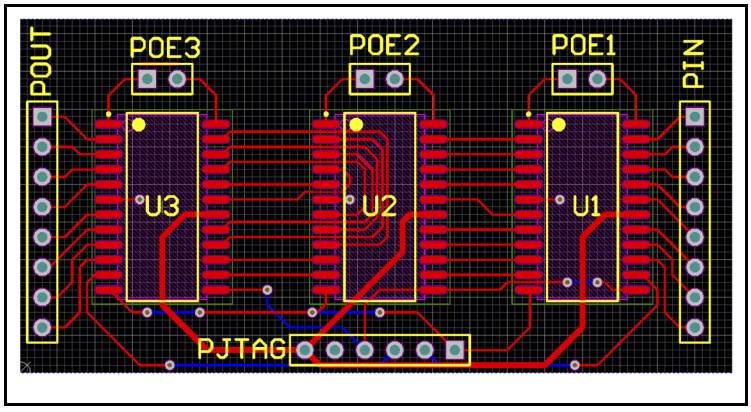


Figure 2.3 Couches de signal de la carte A

2.7.2 Carte B

Une fois les éléments de base du système d'émulation mis en place, nous désirions une carte qui soit beaucoup plus réaliste tout en restant suffisamment simple pour favoriser le déverminage. Nous avons choisi de réaliser une carte générique utilisant des éléments programmables, une mémoire dynamique et des convertisseurs numériques analogiques et analogiques numériques. Un microcontrôleur 32-bit ayant la capacité de communiquer avec un ordinateur hôte est connecté par un bus parallèle à un FPGA lui-même relié à une mémoire vive et aux convertisseurs. Une carte de ce type pourrait être utilisée dans un grand nombre d'applications de traitement de signal, de mesure, de surveillance de procédé, de génération de signal, etc. Elle pourrait être déployée de façon indépendante ou comme périphérique pour un ordinateur personnel. Le tableau 2.2 présente un résumé des principaux dispositifs présents sur la carte. La figure 2.4 montre une représentation de la carte assemblée alors que la figure 2.5 permet de visualiser le routage des deux couches de signal. Des couches internes (non illustrées) sont dédiées aux plans de puissance. Les dimensions réelles de la carte sont de 80 mm x 88 mm.

Tableau 2.2 Principaux dispositifs présents sur la carte B

Type	Modèle	Désignateur	Caractéristiques
Micro-contrôleur	PIC32MX 460F512L	U1	MIPS M4K 80 MHz, 512K Flash, 128K RAM,USB, UART
FPGA	MAX10 10M08SC	U2	8000 LE, 24 multiplicateurs, 42 BRAM, config interne (flash)
ADC	AD9215	U5	10 bit, 105 MSPS
DAC	DAC900	U4	10 bit, 165 MSPS
Mémoire	AS4C16M16SA	U3	SDRAM 16Mx16bit, 166 MHz

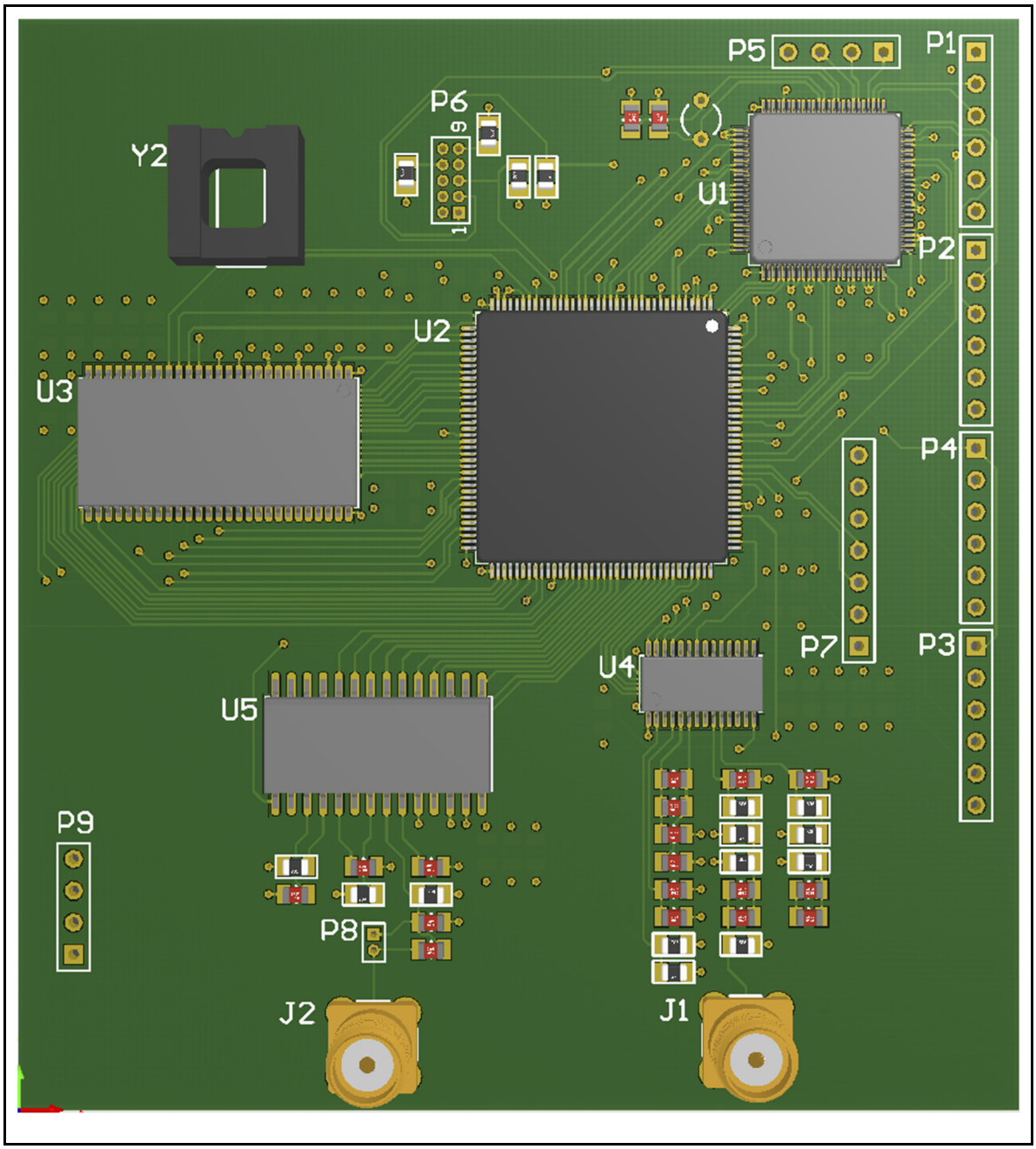


Figure 2.4 Représentation 3D de la carte B assemblée

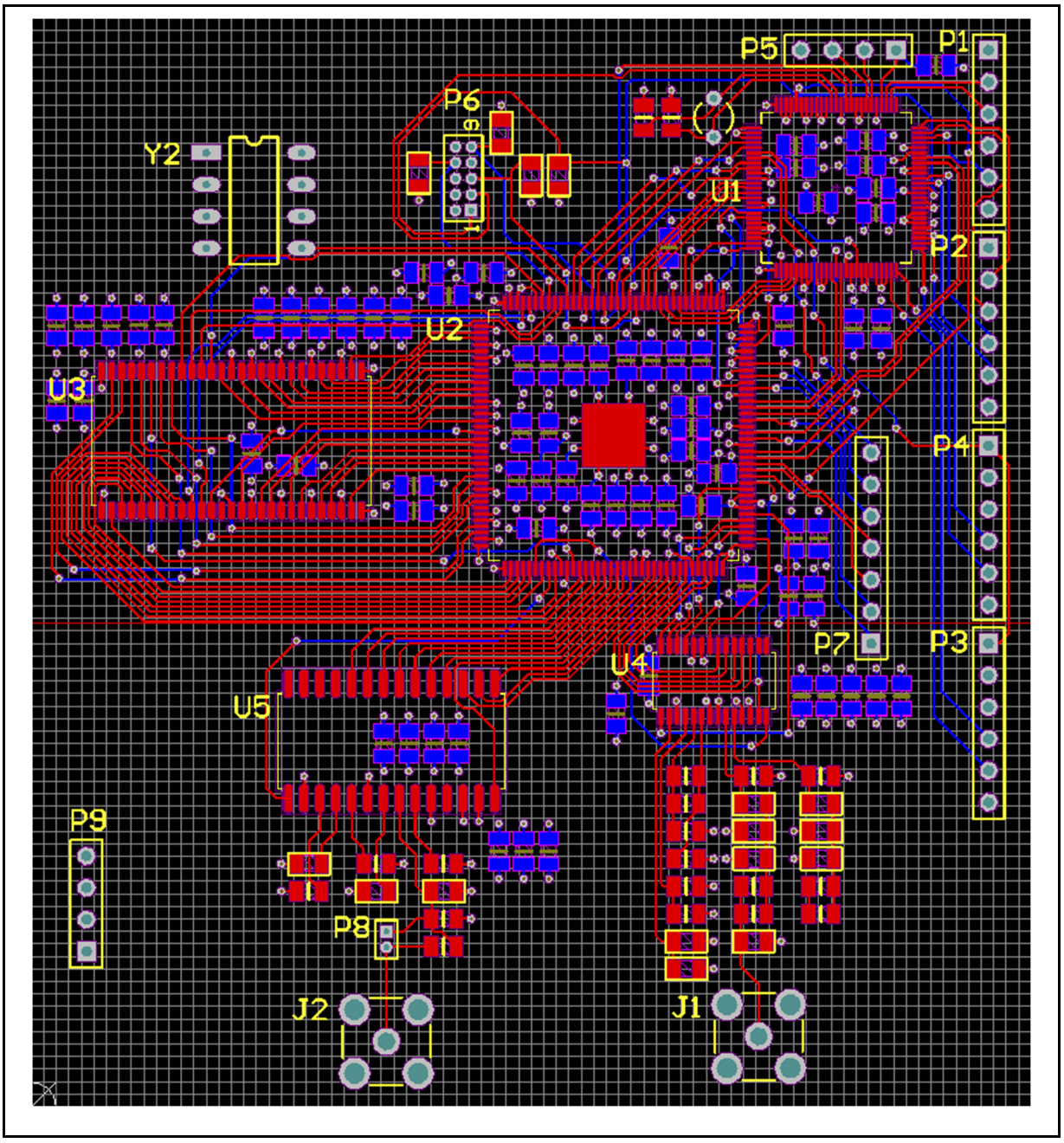


Figure 2.5 Schéma électrique des couches de signal de la carte B

2.8 Conclusion

Ce chapitre nous a permis de présenter une vue d'ensemble du système de diagnostic proposé ainsi qu'un survol des éléments de la preuve de concept. Nous avons également présenté les cartes de circuits imprimés utilisées pour nos essais. Dans les chapitres suivants, nous présenterons plus en détail les éléments du système de diagnostic. Le générateur de cas synthétique est présenté au chapitre 3. Le dépôt de connaissances du domaine constitue de son côté l'objet du chapitre 4. Le système de raisonnement par cas ainsi que le moteur diagnostique sont présentés au chapitre 5, qui contient également les résultats obtenus par le système dans son ensemble et propose une comparaison avec un outil de diagnostic commercial. Le chapitre 6 se concentre sur l'exploitation des données obtenues par le système associées à une rétroaction de l'utilisateur pour la création d'un système recommandeur capable d'émettre des suggestions de réparation. C'est également au chapitre 6 que l'on retrouve la description du simulateur de production utilisé pour évaluer la capacité du système de diagnostic à émettre des recommandations justes et ainsi s'acquitter correctement des quatre tâches qui lui incombent soit la détection, la localisation et la classification des défectuosités suivies de l'analyse des causes racines pour la suggestion de réparation.

CHAPITRE 3

ÉMULATION DE CIRCUIT AU NIVEAU JTAG POUR LA GÉNÉRATION AUTOMATIQUE D'UN DICTIONNAIRE DE PANNES AVEC L'ÉQUIPEMENT DE TEST RÉEL

3.1 Introduction

Afin d'accélérer le démarrage du système de raisonnement par cas dans un environnement de production à faible volume, donc générant peu de données, nous avons décidé d'utiliser des cas obtenus par émulation de pannes. Pour augmenter le réalisme et l'utilité de ces cas, nous avons choisi de construire un émulateur matériel compatible avec l'équipement de test utilisé sur la ligne de production. Enfin, pour maximiser la couverture de ces cas synthétiques, nous avons développé des outils logiciels permettant d'obtenir automatiquement les résultats de l'équipement de test pour toutes les pannes pouvant être émulées touchant un seul nœud du circuit. Nous appelons cet ensemble de résultats de test le « dictionnaire de pannes ».

Dans ce chapitre, nous ferons d'abord un survol de la technique de la chaîne de balayage et des éléments matériels qui composent la chaîne JTAG. Nous introduirons la suite logicielle commerciale utilisée pour le test, puis nous présenterons les détails de réalisation de notre émulateur, en utilisant une approche ascendante.

3.2 Chaîne de balayage et JTAG

L'augmentation du niveau d'intégration des circuits intégrés et la densification des assemblages de circuits imprimés augmentent la difficulté et les coûts reliés au test (et au diagnostic) des produits assemblés. En effet, les méthodes d'encapsulation modernes comme les *ball-grid arrays* (BGA) et les *quad-flat no-leads* (QFN) n'offrent pas de broches accessibles et nécessitent l'ajout de points de test sur la carte. Ces points de test occupent de l'espace sur la carte, ce qui entre en compétition avec l'objectif de densification. Une méthode de test

alternative est donc nécessaire pour atteindre l'objectif de miniaturisation tout en favorisant la testabilité.

La solution réside dans l'intégration des éléments nécessaires au test à l'intérieur des circuits intégrés. Un contrôleur simple permet l'exécution de commandes dédiées aux tests et des cellules spécialisées sont interposées aux frontières entre la puce elle-même et ses broches d'entrées-sorties. Les cellules frontières ont la capacité d'isoler la puce de son environnement, de forcer des entrées ou encore de mesurer des sorties. Elles sont ensuite reliées pour former une chaîne. Cette structure sérielle permet d'accommoder un nombre arbitraire de circuits intégrés avec un nombre limité et fixe de points de connexion sur la carte à tester tout en limitant l'emprise sur les paquetages. Cette méthode de test est nommée le « balayage des frontières » (*boundary-scan*).

Au milieu des années 1980, un groupe d'experts est formé pour normaliser les éléments matériels et les méthodes utilisées pour le test *boundary scan*. Il s'agit du *Joint Test Action Group* (JTAG), qui produira la norme IEEE 1149.1 en 1990. Dans la pratique, les termes *boundary scan*, standard JTAG et IEEE 1149.1 sont généralement utilisés de façon interchangeable.

L'implémentation de la norme JTAG demande l'utilisation de quatre broches (plus une cinquième optionnelle) sur le circuit intégré. Dans une puce implémentant le standard IEEE 1149.1, la chaîne de balayage est formée d'un ensemble de registres à décalage et d'un contrôleur appelé le *Test Access Point Controller* ou *TAP Controller*. L'activation des différents registres est orchestrée par les signaux de contrôle du *TAP Controller*. Les registres JTAG sont décrits dans le tableau 3.1.

Tableau 3.1 Registres JTAG

Nom du registre	Description
Bypass	Une seule cellule, permet de réduire la latence lorsqu'un message n'est pas destiné à une puce.
Device ID	Contient un code permettant d'identifier le manufacturier et le modèle d'une puce.
Instruction Register (IR)	C'est dans ce registre que sont envoyées les commandes destinées à une puce. Le jeu de commandes et son encodage ne sont que partiellement standardisés.
Boundary Register (BR)	Ce registre est formé d'une chaîne de cellules qui permettent entre autres de déconnecter une puce de ses broches et de contrôler directement les entrées/sorties du paquetage sans affecter l'état interne du circuit intégré. Les cellules peuvent ensuite être utilisées pour forcer des sorties et mesurer les entrées, ce qui rend possible le test de la carte.

Tous les nœuds logiques sur le circuit imprimé dont les deux extrémités se situent sur des puces compatibles JTAG peuvent être testés automatiquement en mode stimuli-réponse. Les puces non-JTAG dont l'interface est standardisée comme les mémoires SDRAM ou Flash peuvent également être testées automatiquement à l'aide de patrons en mode stimuli-réponse si elles sont connectées à une puce JTAG.

Sur la chaîne de production, il est rare que le test *boundary scan* soit utilisé seul. Mais lorsque combinée aux méthodes de test électrique traditionnelles ainsi qu'à l'inspection optique ou par rayons X, la technique du *boundary scan* apporte une contribution importante qui permet de prolonger la vie utile des techniques de test en circuits en réduisant le nombre de points de test nécessaires pour obtenir une couverture complète.

3.3 Systèmes commerciaux

Pour les besoins de notre expérimentation, nous avons retenu l'environnement de test ProVision du manufacturier JTAG Technologies inc. puisqu'il était déjà utilisé en production par le partenaire industriel. Nous utilisons la version 2.2.0. Afin de connecter l'application de test au circuit à tester, nous utilisons l'interface JT3705/USB Explorer, qui comporte deux ports de tests, ou TAP (*Test Access Ports*). Un seul port est utilisé pour notre application.

3.3.1 Génération d'une suite de test avec l'outil commercial

Afin de générer une suite de test avec ProVision, il est d'abord nécessaire de lui fournir une description de la structure de la carte de circuit imprimé à tester, sous forme d'une liste d'interconnexions ou *netlist*, produite par l'outil de CAO utilisé pour la conception de la carte. Après analyse de la *netlist*, ProVision affecte chacun des dispositifs présents sur la carte à un modèle. Les dispositifs non-JTAG dont l'interface est standardisée, comme les mémoires, bénéficient de modèles génériques. Dans le cas des puces compatibles JTAG, il est nécessaire de fournir à ProVision des descriptions BSDL qui contiennent les détails de l'implémentation JTAG du dispositif (décrits plus en détail à la section 3.4.3).

Ces informations sont utilisées par ProVision pour élaborer une suite de vecteurs de test permettant une couverture maximale pour les nœuds accessibles par la chaîne de balayage. Une fois cette étape complétée, ProVision génère la séquence de commandes JTAG qui sera transmise via le TAP de l'interface de test. Cette séquence est appelée une *application* dans la terminologie de ProVision. Une fois l'application générée, elle peut être lancée à partir de l'interface graphique de ProVision, mais aussi en ligne de commande, ce qui sera fort utile pour l'automatisation. La procédure de génération d'une application de test sera revue plus en détail à la section 3.4.6.

Le format de sortie des résultats de test de ProVision sera décrit au chapitre 4. La suite ProVision inclut également un outil de diagnostic automatisé qui utilise les résultats de test produits. Cet outil, *Boundary Scan Diagnostics* (BSD) sera utilisé au chapitre 5 pour évaluer la performance de notre système.

3.4 Modélisation de carte de circuit imprimé au niveau JTAG

Afin de pouvoir obtenir des résultats de test sur l'équipement commercial utilisé en production, il est nécessaire de produire un émulateur pouvant se substituer à la carte. Cet émulateur sera formé d'une reproduction de la partie JTAG de chacune des puces faisant partie de la chaîne. Le fonctionnement interne des puces n'est pas émulé. Les puces ainsi émulées seront donc

toujours considérées comme étant en mode de test, dit « pin-permission », et défini dans le standard IEEE 1149.1.

En premier lieu, il est nécessaire de reproduire les pièces les plus élémentaires de la chaîne JTAG, soit les cellules des registres à décalage et le TAP controller. Ces éléments de base seront ensuite réutilisés pour toutes les puces, en suivant l'organisation décrite dans la description BSDL de chacune d'entre elles, produite par le manufacturier.

Les modèles de puces seront interconnectés pour former un modèle de plus haut niveau reproduisant l'organisation logique de la carte de circuit imprimée, toujours en considérant uniquement les éléments pertinents pour la chaîne JTAG. À cette étape, le modèle peut être inséré dans un dispositif programmable de type FPGA et connecté à l'équipement de test. La validation consiste à exécuter les tests JTAG d'infrastructure et fonctionnels dans l'environnement de test commercial JTAG ProVision.

3.4.1 TAP Controller et instructions JTAG

La chaîne JTAG utilise quatre ou cinq broches des circuits intégrés compatibles, décrites dans le tableau 3.2. Ces broches doivent être dédiées au *boundary scan* et ne peuvent être partagées avec aucune autre fonction.

Tableau 3.2 Broches dédiées à la chaîne de balayage

Nom de la broche	Sigle	Description
Test Clock	TCK	Signal d'horloge
Test Mode Select	TMS	Signal de contrôle pour le TAP Controller
Test Data In	TDI	Signal d'entrée du registre à décalage
Test Data Out	TDO	Signal de sortie du registre à décalage
Test Reset	TRST*	Remise-à-zéro asynchrone, actif bas (optionnelle)

La communication sur la chaîne JTAG est contrôlée par une machine à états finis (MEF) intégrée dans chaque puce compatible avec le standard. Ce module est nommé le « *Test Access*

Point Controller » ou *TAP Controller*. Le diagramme d'état de la MEF est présenté à la figure 3.1. Les valeurs associées aux flèches entre les états sont celles de la ligne TMS. On y distingue deux colonnes principales, l'une associée aux données, l'autre aux instructions.

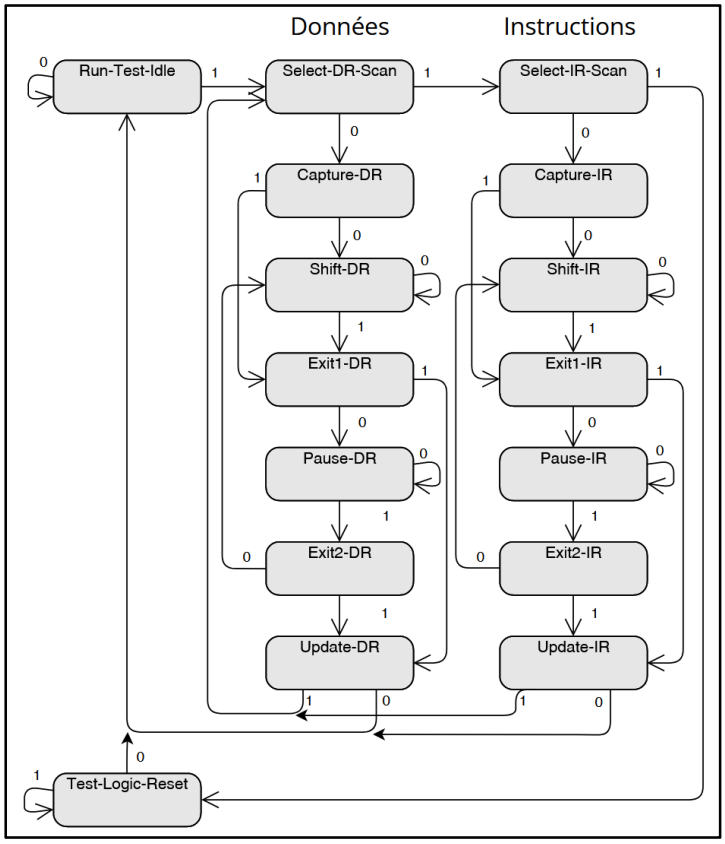


Figure 3.1 Transitions d'états de la MEF du *TAP Controller*
Inspirée de Parker (2016, p.13)

Une des propriétés intéressantes du diagramme d'état est que peu importe l'état actuel, si la ligne TMS est activée pendant 5 fronts montants de TCK, la machine est remise à l'état *Test-Logic-Reset*. Cette séquence est donc largement utilisée pour remettre le *TAP Controller* dans un état connu. Les transitions asynchrones causées par l'activation du signal optionnel TRST*, lorsque présent, ne sont pas illustrées sur la figure, mais mènent toutes directement à l'état *Test-Logic-Reset*.

Pour notre émulateur, nous avons écrit un module VHDL *TAP_Controller* qui reproduit le comportement de la machine à état illustrée à la figure 3.1. Le code VHDL du contrôleur ainsi que le résultat d'analyse RTL de notre implémentation par l'outil de synthèse sont présentés à l'annexe I.

3.4.2 Structure des cellules et registres JTAG

Comme indiqué précédemment, la chaîne JTAG est formée d'un ensemble de registres, pouvant être de quatre types différents (*bypass*, *device ID*, *instruction*, *boundary*). Ces registres sont eux-mêmes constitués de cellules, qui varient selon le type de registre. Ainsi, les registres *bypass* et *device ID* sont des registres à décalage conventionnels, où les cellules sont simplement des bascules. Pour les autres registres, le standard 1149.1 propose différents modèles de cellules adaptées à certaines configurations particulières. Chaque puce indique le modèle de chacune des cellules dans sa description BSDL. Nous avons basé notre implémentation VHDL sur les schémas logiques de ces modèles standard, tels qu'illustrés dans (Parker, 2016).

La structure générale des registres d'instruction et *boundary scan* est illustrée à la figure 3.2. Les registres doivent pouvoir recevoir les données de façon sérielle ou parallèle selon le contexte. Ils doivent également être en mesure de conserver les données présentes lorsque le *TAP Controller* entre dans les états *Capture_IR* ou *Capture_DR*.

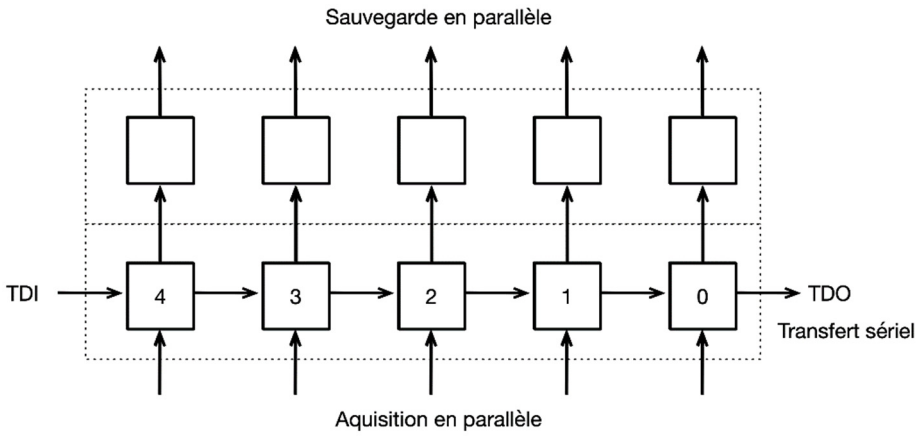


Figure 3.2 Structure générale des registres JTAG
Inspirée de Parker (2016, p.19)

Le fonctionnement du registre d'instruction, le code source VHDL et le résultat d'analyse RTL est présenté à l'annexe II. L'annexe III présente les mêmes éléments pour le *boundary register*.

3.4.3 Description BSDL des circuits intégrés

Le *Boundary-Scan Description Language* (BSDL), introduit lors de la révision 1994 du standard IEEE1149.1, est utilisé pour décrire les éléments JTAG d'un circuit intégré conforme. Originalement un sous-ensemble strict du VHDL, une refonte lors de la révision 2013 du standard l'en a fait dévié. Le langage est maintenant décrit comme « basé » sur le VHDL.

BSDL est un langage purement descriptif. Les éléments décrits par le BSDL sont :

- Le nom, la position et la direction des broches du boîtier
- La longueur des registres d'instruction et *boundary*
- Les instructions supportées par le contrôleur et leur encodage
- L'identifiant contenu dans le *ID Register*
- Le type de cellule, le port associé ainsi que le rôle de chacune des cellules du registre *boundary*.

BSDL ne peut être exécuté et ne constitue pas un « modèle » de l'implémentation JTAG d'une puce. BSDL ne fournit pas d'information sur la conception architecturale ou structurelle d'une

implémentation IEEE1149.1 (Parker, 2016). Une description BSDL ne contient en outre aucune information sur les éléments obligatoires du standard IEEE1149.1. Par exemple, le registre *bypass* n'est pas décrit dans un fichier BSDL puisqu'il est obligatoire et qu'aucune option ou configuration n'est possible.

3.4.4 Génération de modèles VHDL à partir du BSDL

Bien que la description BSDL d'une puce ne constitue pas en elle-même un modèle, elle contient tous les éléments de configuration permettant l'élaboration d'un modèle de l'implémentation JTAG d'une puce. Nous avons utilisé un procédé partiellement automatique pour la génération des modèles utilisés dans ce projet. Tout d'abord, nous avons développé un squelette VHDL comportant tous les éléments communs d'une implémentation JTAG, qui ne sont pas décrits dans le BSDL. La figure 3.3 permet d'observer l'architecture générale de l'implémentation JTAG d'une puce.

Le squelette comporte les éléments suivants :

- Le nom de l'entité et les ports d'entrée-sortie;
- Une série de signaux internes qui correspondent aux liens du schéma figure 3.5;
- La distribution du signal TDI;
- La logique de sélection et les registres pour le signal TDO;
- La logique du registre *bypass;*
- La logique du registre *device ID;*
- La logique du *User Code Register;*
- La logique du registre d'instructions;
- Le décodeur d'instruction;
- Un générateur pour le registre *boundary;*
- Les connexions des entrées-sorties au registre *boundary;*
- Une instance du *TAP Controller.*

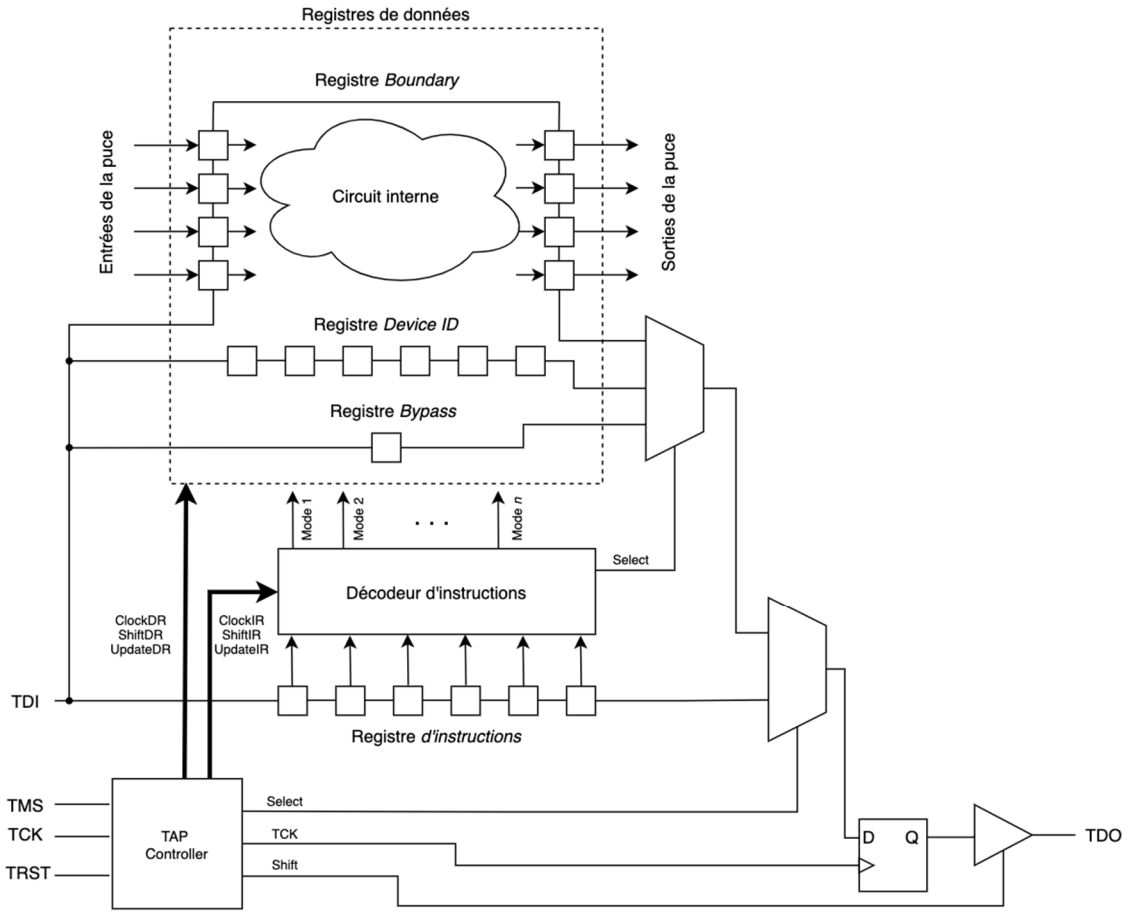


Figure 3.3 Architecture générale d'une implémentation JTAG
Inspirée de Parker (2016, p.30)

Le squelette doit ensuite être adapté manuellement pour intégrer l'information du BSDL. Cependant, une paire de petits scripts utilitaires a été développé pour automatiser les tâches répétitives. Ils sont présentés en Annexe IV. Le tableau 3.3 décrit ces outils. L'annexe V montre un fichier de description BSDL pour une puce JTAG simple ainsi que le modèle VHDL résultant.

Tableau 3.3 Scripts utilisés pour la génération de modèles VHDL

Nom	Description
splitio.pl	Convertit les ports bsdl de type *bit_vector* en *std_logic_vector* et élimine les ports de types *linkage*, qui ne sont pas utilisés dans le modèle. Les ports bidirectionnels sont séparés en un port d'entrée et un port de sortie distincts, puisqu'il ne peuvent être implémentés directement dans les couches internes d'un dispositif FPGA.
map_bs.pl	Analyse la description BSDL du *boundary register* pour générer les assignations entre les ports d'entrée et sortie et le *boundary register* du modèle.

3.4.5 Modélisation de la chaîne JTAG

Une fois que toutes les pièces utilisées par la chaîne JTAG d'un circuit imprimé ont été modélisées, il est nécessaire de créer un modèle pour la chaîne elle-même. C'est ce modèle qui sera relié directement aux broches de la carte d'émulation FPGA puis ultimement à l'équipement de test JTAG. La relation entre les modules VHDL présentés jusqu'à maintenant et les données de conception est représentée à la figure 3.4.

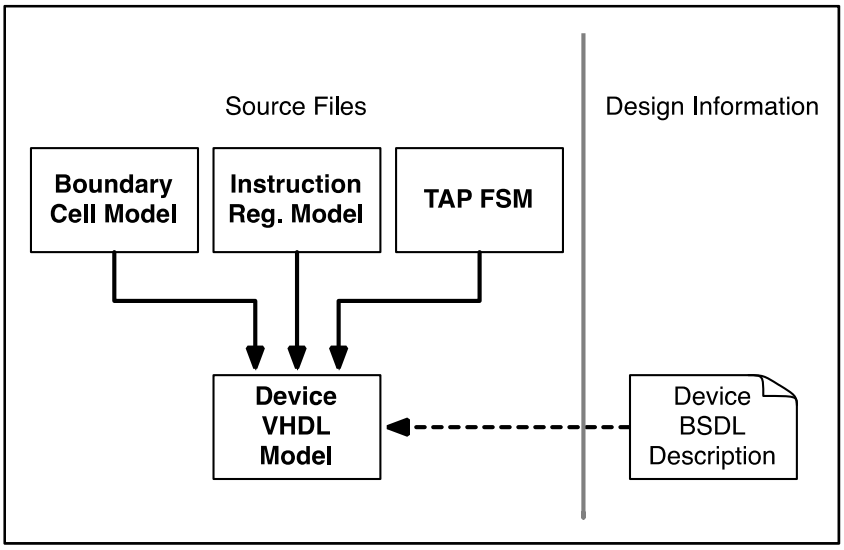


Figure 3.4 Entités VHDL et source d'information nécessaires à l'élaboration du modèle VHDL d'une puce

Pour élaborer le modèle de carte, il suffit d'instancier chacun des composants de la chaîne et de relier correctement leurs entrées et sorties. Le modèle de carte est assemblé à partir des modèles de puces créés précédemment, conformément à la description du réseau contenue dans la *netlist* (figure 3.5).

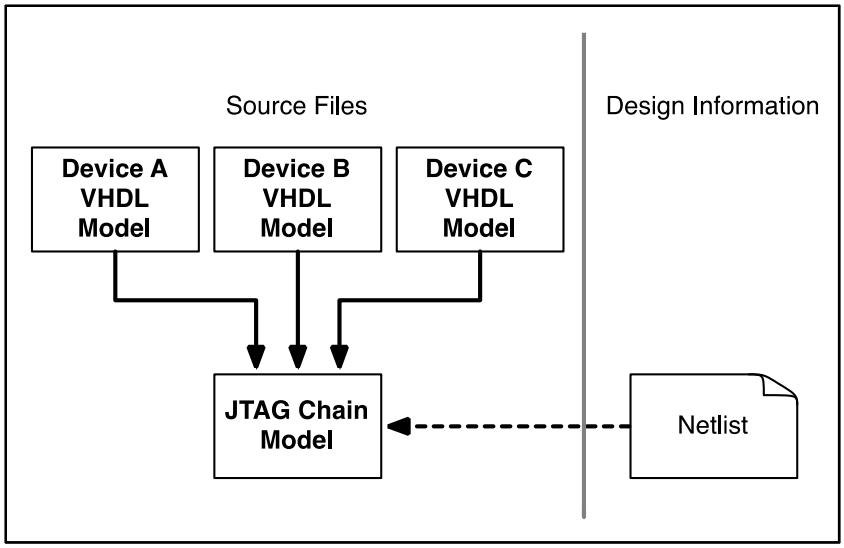


Figure 3.5 Entités VHDL et source d'information nécessaires à l'élaboration du modèle VHDL d'une carte

3.4.6 Vérification du modèle de la chaîne JTAG

Une fois assemblé et synthétisé, le modèle de la carte de circuit imprimé doit être validé à l'aide de l'équipement de test commercial retenu. Avant de procéder à cette vérification, il est nécessaire de générer une suite de tests appropriée à la carte dans l'environnement de développement ProVision. Cette procédure est presque entièrement automatisée. Lors de la création d'un projet, ProVision demande la *netlist* produite par l'outil de CAO. Cette *netlist* est analysée automatiquement et ProVision en tire un modèle interne de la carte à tester au niveau JTAG. Il est ensuite nécessaire de fournir à ProVision les fichiers de description BSDL pour les puces compatibles JTAG qui auront été identifiées lors de l'analyse. ProVision pourra ensuite générer automatiquement des patrons de test pour la vérification.

Deux tests principaux sont offerts par ProVision : le test d'infrastructure et le test d'interconnexions. Le test d'infrastructure vérifie l'intégrité de la chaîne JTAG. Des patrons de test sont générés qui permettent de s'assurer du bon fonctionnement et de la longueur des registres d'instructions et de *boundary scan* et valident l'identité des puces présentes. C'est un test particulièrement important pour nous puisqu'il permet de vérifier que nous avons reproduit sans erreur la chaîne JTAG de chacune des puces modélisées.

Le second test effectué par ProVision est le test d'interconnexion. Ce test vise à vérifier l'intégrité des connexions reliant les broches d'entrées-sorties des puces entre elles. ProVision génère automatiquement un ensemble de vecteurs de tests qui seront utilisés dans un mode stimuli-réponse entre les puces pour détecter la présence de circuits ouverts, de courts-circuits ou de valeurs fixées (« *stuck_at* »). À ce stade, dans notre application, ce test permet de vérifier la modélisation faite au niveau de la carte. Si le test s'exécute sans erreur, c'est que nous avons correctement reproduit les connexions entre les broches des différentes puces. Une fois que les deux tests sont réussis, le modèle de carte est complet. Il est maintenant prêt à être modifié pour permettre l'injection de pannes. Le test d'interconnexion sera utilisé plus tard pour l'émulation de pannes. Les résultats du test d'interconnexion formeront les signatures utilisées pour définir chaque élément du dictionnaire de pannes.

3.5 Émulation de pannes

La méthode retenue pour la génération du dictionnaire de pannes est très simple. Il s'agit de provoquer une à une toutes les défectuosités uniques correspondant à un ensemble de scénarios de pannes déterminés (décrits à la section 3.5.1) afin d'enregistrer la réponse du système de test en présence de cette panne. Vu le grand nombre de défectuosités possibles et son augmentation rapide avec l'accroissement de la complexité des cartes, il est nécessaire d'automatiser l'ensemble du processus.

Nous avons donc utilisé des éléments reprogrammables à chaud du FPGA pour construire des structures appelées saboteurs (décrits à la section 3.5.2), qui sont à la base de notre stratégie

d'émulation. La reconfiguration des saboteurs est effectuée par un programme s'exécutant sur un minuscule processeur embarqué, qui est à son tour sous la gouverne d'un programme sur PC. L'application de contrôle sur PC est également chargée du contrôle des opérations du logiciel de test commercial, ProVision.

L'émulateur est assemblé à partir du modèle de carte instrumenté (*JTAG Chain with Saboteurs*) auxquels des éléments de contrôle (*Saboteurs control*) et de communication (*UART & Interpreter*) sont ajoutés (Figure 3.6).

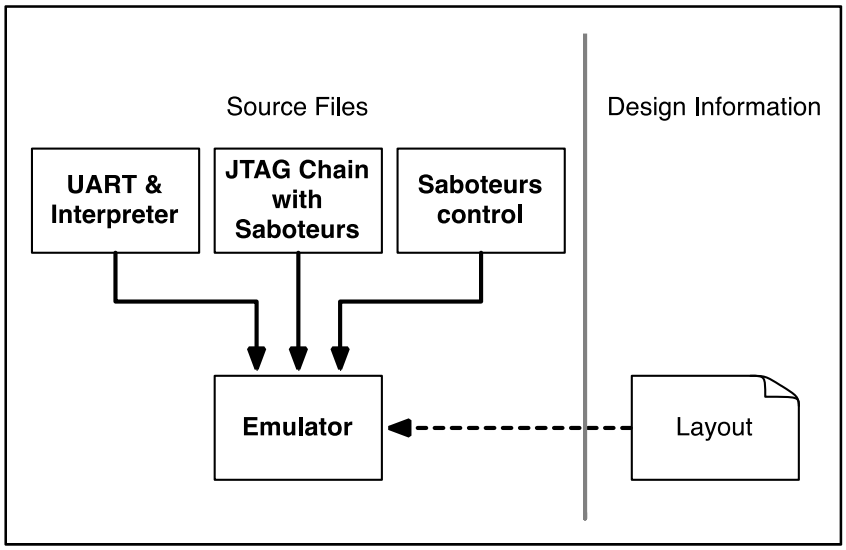


Figure 3.6 Entités VHDL et source d'information nécessaires à l'élaboration de l'émulateur

L'architecture du système d'émulation de pannes complet est présentée à la figure 3.7. Pour ajuster le système à un nouveau produit, seul le modèle de carte et potentiellement les modèles de puces (*Device VHDL Model*) doivent être remplacés. Les autres éléments ne nécessitent qu'un ajustement des constantes de configuration.

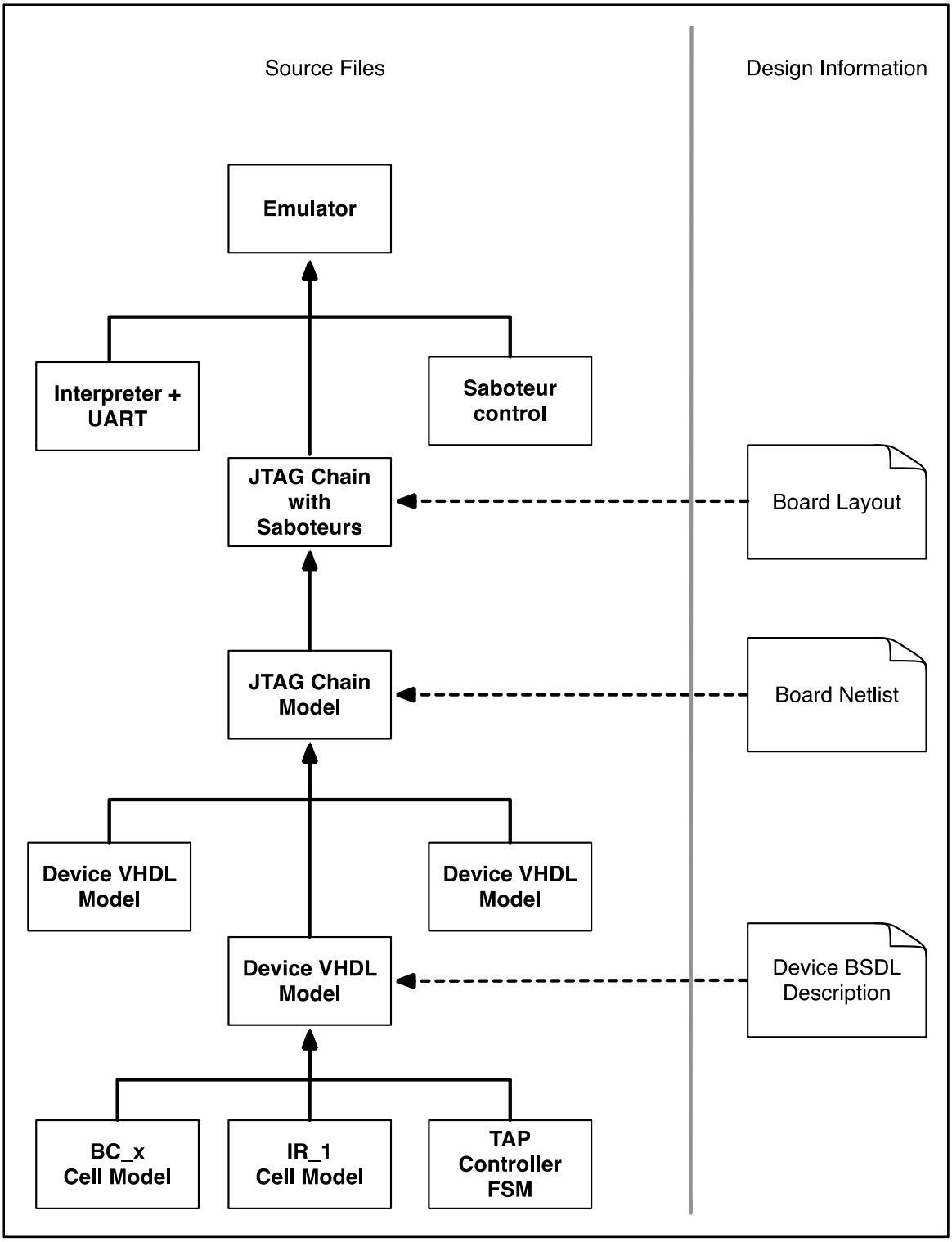


Figure 3.7 Entités VHDL et sources d'informations nécessaires à l'élaboration du système d'émulation de pannes

3.5.1 Scénarios de pannes

L'émulation de pannes est effectuée au niveau logique, puisque c'est le niveau testé par l'équipement commercial. Les types de pannes pouvant être émulées sont : 1) les valeurs fixés (*stuck_at*) à zéro ou à un, 2) les courts-circuits inconditionnels entre deux nœuds, 3) les ponts dominés par les 1, aussi appelés OU câblés (*wired-OR*), et 4) les ponts dominés par les 0, aussi appelés ET câblés (*wired-AND*).

Pour chacun des types de courts-circuits et ponts, une analyse des fichiers de FAO décrite au chapitre 4 produit une liste des nœuds dont les traces ou les broches sont adjacentes sur la carte physique. Puisque la structure reprogrammable à chaud retenue pour les saboteurs comporte cinq entrées, les courts-circuits et ponts seront émulés en utilisant chacun des quatre voisins les plus proches, s'ils existent.

3.5.2 Saboteurs

Le rôle des saboteurs est de simuler une panne sur la carte. Ce sont des dispositifs programmables insérés entre les ports de sortie et d'entrée sur le modèle de carte. Nous avons retenu le composant CFGLUT5 de Xilinx pour accomplir cette tâche, car son interface de reprogrammation sérielle était bien adaptée au schéma de contrôle envisagé et parce qu'il était non seulement disponible sur le Virtex-5 utilisé, mais aussi sur les familles ultérieures 7-Series et Ultrascale. La figure 3.8 montre l'interface de la cellule CFGLUT5.

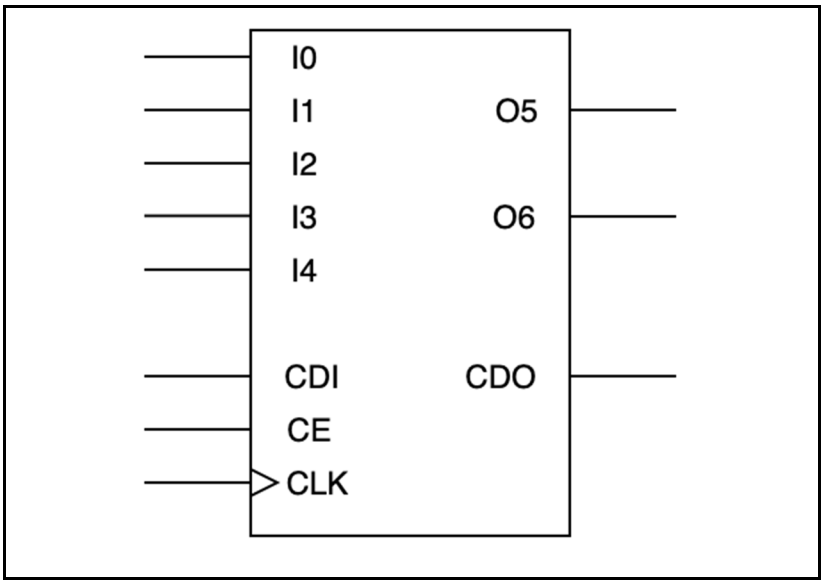


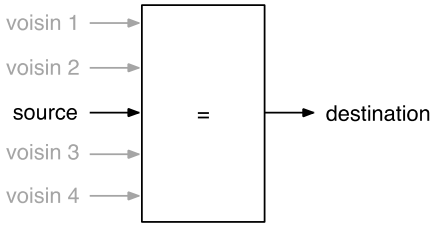
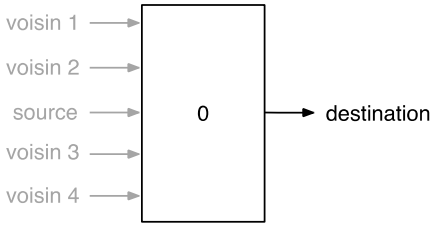
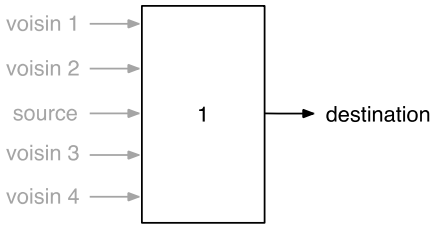
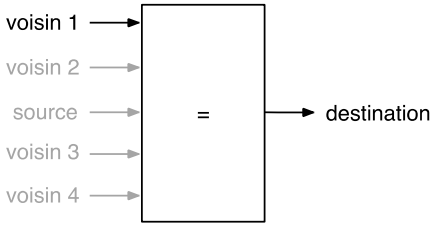
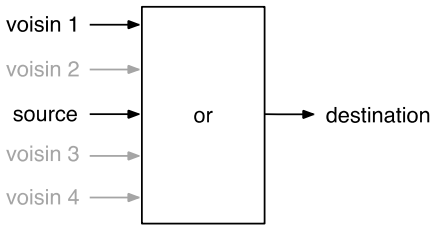
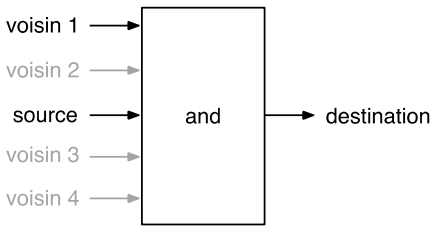
Figure 3.8 Interface de la cellule reprogrammable CFGLUT5
Inspirée de Xilinx UG974 (2014, p.51)

La cellule comporte cinq entrées et deux sorties et permet d'implémenter n'importe quelle fonction logique à cinq variables. Les cellules sont reliées entre elles par leurs broches CDO et CDI pour former une chaîne et sont reprogrammées de façon sérielle. Le signal CE est utilisé pour activer le transfert.

Lorsqu'utilisées comme saboteurs, les cellules interceptent les sorties de leurs nœuds sources et appliquent une opération simple correspondant au scénario de panne désiré pour générer la sortie. Le tableau 3.4 montre un exemple de configuration pour chaque scénario.

Le patron de configuration d'une cellule comporte 32 bits et est obtenu en faisant la table de vérité de l'opération désirée (Xilinx, 2014 page 51). La figure 3.9 est extraite du code de contrôle des saboteurs. Les patrons utilisés y sont montrés en notation hexadécimale.

Tableau 3.4 Exemples de configuration des saboteurs

Scénario	Configuration du saboteur
1: Bypass (aucune panne) Le saboteur se comporte de façon transparente et n'a pas d'effet sur le circuit. C'est la configuration par défaut.	
2: Stuck-at-0 Le saboteur ignore la valeur de la source et force la sortie à 0 en tout temps.	
3: Stuck-at-1 Le saboteur ignore la valeur de la source et force la sortie à 1 en tout temps.	
4: Short Court-circuit. Le saboteur remplace la sortie de la source par celle du voisin sélectionné.	
5: 1-dominant bridge Pont dominé par les 1 ou *Wired-OR*. Le saboteur remplace la sortie de la source par le résultat de l'opération OU-logique effectué entre la source et le voisin sélectionné.	
6: 0-dominant bridge Pont dominé par les 0 ou *Wired-AND*. Le saboteur remplace la sortie de la source par le résultat de l'opération ET-logique effectué entre la source et le voisin sélectionné.	

```
-- Saboteur wiring
--
--                CFGLUT5
--                --------
--      neighbor 1--|i0      |
--      neighbor 2--|i1    o5|--NC
--      source pin--|i2      |
--      neighbor 3--|i3    o6|--target pin
--      neighbor 4--|i4      |
--                --------
--
-- FAULT SCENARIO          OUTPUT FORMULA      CFGLUT BIT PATTERN
--  1   bypass              o6 = i2            F0F0F0F0
--  2   Stuckat0            o6 = 0             00000000
--  3   Stuckat1            o6 = 1             FFFFFFFF
--  4   Short neighbor1     o6 = i0            AAAAAAAA
--  5   Short neighbor2     o6 = i1            CCCCCCCC
--  6   Short neighbor3     o6 = i3            FF00FF00
--  7   Short neighbor4     o6 = i4            FFFF0000
--  8   Strong '1' neighbor1  o6 = i0 or  i2     FAFAFAFA
--  9   Strong '1' neighbor2  o6 = i1 or  i2     FCFCFCFC
-- 10   Strong '1' neighbor3  o6 = i3 or  i2     FFF0FFF0
-- 11   Strong '1' neighbor4  o6 = i4 or  i2     FFFFF0F0
-- 12   Strong '0' neighbor1  o6 = i0 and i2     A0A0A0A0
-- 13   Strong '0' neighbor2  o6 = i1 and i2     C0C0C0C0
-- 14   Strong '0' neighbor3  o6 = i3 and i2     F000F000
-- 15   Strong '0' neighbor4  o6 = i4 and i2     F0F00000
```

Figure 3.9 Patrons de configuration des CFGLUT5

Comme les CFGLUT5 opèrent à la fréquence du circuit (100MHz sur notre carte d'essai) et peuvent décaler un bit par coup d'horloge, la reprogrammation d'une chaîne type de quelques dizaines de saboteurs est très rapide, à raison de 320 nanosecondes par cellule. Ce temps de reprogrammation sera comparativement négligeable lors de la génération du dictionnaire de pannes.

3.5.3 Contrôle des saboteurs

Le contrôle de la reprogrammation des saboteurs est partagé entre trois éléments. La figure 3.10 montre l'architecture retenue. Les éléments principaux seront décrits selon une approche descendante.

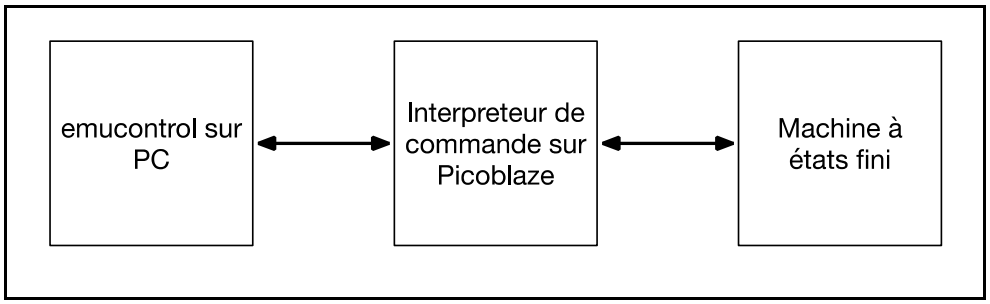


Figure 3.10 Architecture du système de contrôle des saboteurs

Au niveau d'abstraction supérieur, le contrôle de l'injection de panne est réalisé en temps réel par l'entremise d'un logiciel à interface graphique que nous avons conçu et qui permet à l'utilisateur de sélectionner un type de panne et une cible et de provoquer la reprogrammation des saboteurs. Le logiciel permet l'insertion de défectuosités multiples. Il permet également de lancer le test sur l'outil commercial, ce qui résulte en un fichier de signature pour la combinaison de défectuosités sélectionnées. Une capture d'écran de l'interface utilisateur est présentée à la figure 3.11.

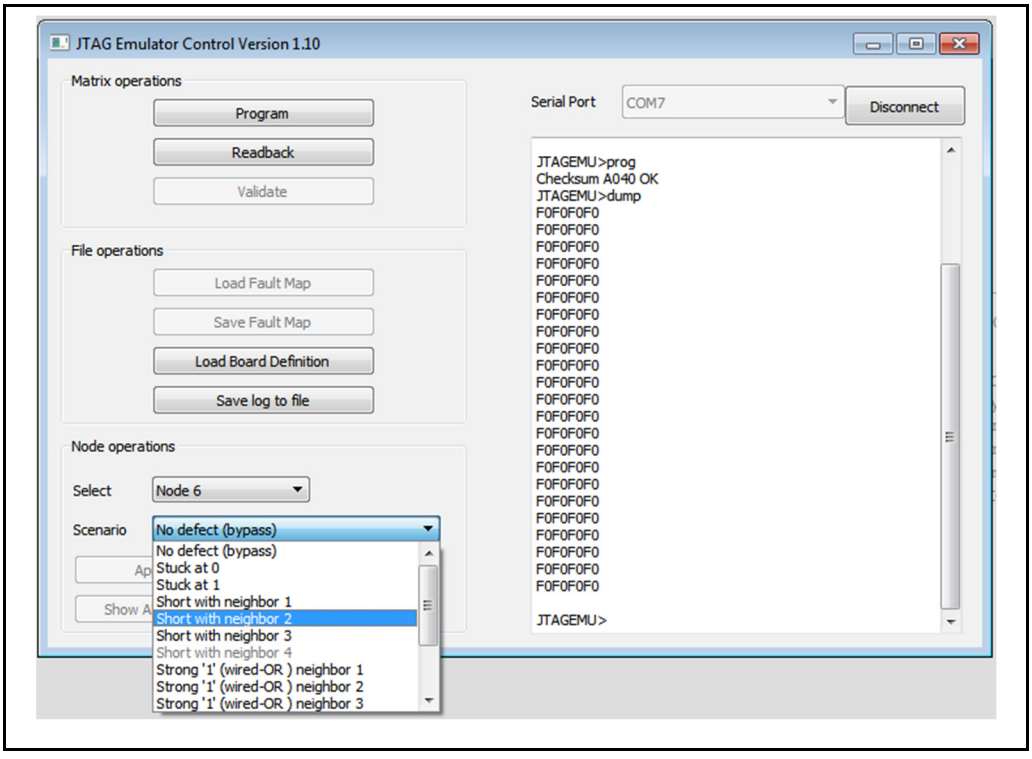


Figure 3.11 Exemple d'utilisation du logiciel Emucontrol sur PC

Pour déterminer les scénarios de pannes qui doivent être offerts à l'utilisateur, le logiciel de contrôle utilise une matrice d'adjacence produite lors de l'analyse des fichiers de conception manufacturière (CAM) détaillée au chapitre 4.

Le logiciel de contrôle est relié à la carte d'essai par un câble USB à série branché sur le connecteur DE9 de la carte. Du côté du FPGA, un système embarqué basé sur le processeur Xilinx Picoblaze et un module de communication sérielle asynchrone (UART) exécute un interpréteur de commande que nous avons écrit en langage assembleur.

Ces commandes de bas niveau permettent d'obtenir la configuration actuelle des saboteurs ou de transmettre une nouvelle configuration. Toutes les transmissions entre le système embarqué et le PC sont validées par une somme de contrôle utilisant l'algorithme de Fletcher (Stone et al, 1998). Dans le cas où une erreur de transmission est détectée, le transfert est recommencé. Puisqu'une erreur d'un seul bit dans la transmission est suffisante pour modifier le comportement des saboteurs (et donc la panne émulée), cette validation est nécessaire.

Un port d'entrée et un port de sortie à 8 bits du système Picoblaze sont reliés au module de contrôle des saboteurs par l'intermédiaire de tampons FIFO de 16 octets chacun. Une machine à états finis est enfin responsable de transférer la configuration vers les cellules CFGLUT de façon sérielle.

3.6 Génération automatique d'un dictionnaire de pannes

Afin de produire des signatures en nombre suffisant pour alimenter un système de raisonnement par cas en démarrage, nous avons décidé que notre émulateur devrait avoir la capacité de produire de façon autonome des fichiers de signatures pour toutes les pannes simples valides pour une configuration de carte donnée.

Pour ce faire, nous avons mis à profit les éléments présentés dans la section 3.5.3 à l'exception de l'application *Emucontrol*, qui a été remplacée par une version dédiée à cette tâche et

nommée *Siggen*. *Siggen* reprend les éléments de communication, d'affichage et de validation d'*Emucontrol*, mais ne permet pas la sélection et l'envoi de pannes par l'utilisateur. *Siggen* a pour rôle principal d'automatiser l'outil de test Provision et de faire la gestion de la chaine de programmation des saboteurs pour s'assurer que chaque mesure effectuée par ProVision est indépendante.

3.6.1 Automatisation de l'outil de test commercial

JTAG Provision est un environnement de développement intégré complet pour le test *JTAG*. La procédure de génération d'un modèle de carte pour ProVision a été décrite précédemment à la section 3.4.6.

Une fois le modèle créé, ProVision génère automatiquement une suite de vecteurs de test lui permettant d'obtenir une couverture optimale. Une fois cette suite générée, l'ensemble des commandes JTAG nécessaires pour l'exécution du test est compilé sous forme d'un fichier d'application ProVision.

ProVision inclut avec sa suite logicielle une série de programmes exécutables en ligne de commande pour Windows dédiés à chacun de ses contrôleurs. Nous utilisons le programme bec3705.exe qui correspond à notre contrôleur.

Afin d'automatiser la génération du dictionnaire de panne, notre *Siggen* exécute donc la séquence suivante en boucle :

1. Programmation de la chaîne de saboteurs avec une seule défectuosité.
2. Exécution de bec3705.exe avec les paramètres appropriés pour l'application correspondante à la carte de test émulée.
3. Déplacement et renommage des fichiers de sortie produits par bec3705.exe afin d'élaborer le dictionnaire de panne. Les noms de fichiers encodent le numéro de

nœud et le scénario de panne afin de faciliter leur chargement automatique par le prototype du système diagnostic décrit au chapitre 5.

4. Reprogrammation de la chaîne de saboteur pour éliminer la défectuosité.

L'exécution d'une itération de cette séquence prend un peu moins de 5 secondes, largement dominée par le temps d'exécution de l'application bec3705.exe. Un dictionnaire de panne comportant 250 signatures peut donc être généré en 20 minutes environ.

3.7 Conclusion

Ce chapitre nous a permis de présenter la technique du *boundary scan* ainsi que la structure et le fonctionnement de notre système d'émulation matérielle de carte de circuit imprimé au niveau JTAG. Cet émulateur, une fois branché sur l'équipement de test JTAG commercial, peut être mis à profit pour le développement de la suite de test utilisée en production avec les cartes réelles. Il permet également l'injection de pannes grâce à l'utilisation de saboteurs. Ce système d'injection de pannes peut non seulement être utilisé à son tour pour la validation du système de test des cartes physiques, mais également pour la génération de fichiers de signatures correspondant à l'ensemble des pannes simples détectables par *boundary scan*. Une collection de ces signatures forme un dictionnaire de pannes, que nous allons exploiter pour accélérer le démarrage d'un moteur diagnostic utilisant la technique du raisonnement par cas, présenté au chapitre 5.

Le prochain chapitre, de son côté, présentera les éléments qui constituent le dépôt de connaissances du domaine utilisé par le prototype, soit les règles de détection et de localisation des pannes ainsi que la modélisation de la disposition physique (*layout*) des cartes.

CHAPITRE 4

REPRÉSENTATION DES CONNAISSANCES POUR LE DOMAINE DU DIAGNOSTIC EN ÉLECTRONIQUE NUMÉRIQUE

4.1 Introduction

Afin de permettre au système proposé d'offrir rapidement un diagnostic de qualité, il est nécessaire de lui fournir le plus d'information possible. N'ayant pas accès aux grands volumes de données nécessaires pour inférer par lui-même les règles de détection et de classification des défectuosités, le système devra s'appuyer sur une représentation logicielle des connaissances nécessaires à sa fonction qui sera présentée dans ce chapitre.

Nous présenterons d'abord la représentation logique du circuit imprimé, c'est-à-dire sa représentation abstraite, indépendante de la technologie d'implémentation. Cette représentation sera utilisée dans la modélisation des tests, de leurs résultats et des défectuosités trouvées. Les outils de test JTAG commerciaux utilisés dans ce projet opèrent exclusivement à ce niveau. De plus, afin d'offrir un diagnostic de meilleure qualité que les outils commerciaux existants, nous proposons d'intégrer certains éléments de la disposition physique du circuit imprimé. Nous détaillerons donc ensuite la méthode que nous avons utilisée pour faire l'analyse des données de fabrication assistée par ordinateur (FAO) et en extraire une représentation du réseau physique, tel qu'implémenté sur la carte de circuit imprimé. Ce modèle physique sera à son tour utilisé pour produire une liste de nœuds voisins pour chacun des éléments de la représentation logique. Cette information sera utilisée lors du processus d'identification des nœuds partenaires pour les différents types de courts-circuits et ponts que le système sera en mesure d'identifier.

La représentation logicielle des connaissances proposée, qui est le résultat d'un processus d'analyse et de conception orientée-objet, consiste en une hiérarchie de classes élaborée en suivant les principes de la conception dirigée par le domaine, décrits dans (Evans, 2004). Cette

méthode permet de maintenir une correspondance robuste entre la terminologie du domaine représenté et l'implémentation du système logiciel.

4.2 Diagrammes de classe UML

Plusieurs illustrations du chapitre courant et du suivant montrent les relations entre différentes classes d'objets dans notre représentation du domaine. Ces figures utilisent un sous-ensemble de la notation du diagramme de classe du « *Unified Modeling Language »* (UML), définit dans la norme ISO/IEC 19501 :2012. Les éléments utilisés sont présentés ici.

4.2.1 Classes

La figure 4.1 représente les 3 niveaux de détails utilisés pour les classes d'objets, soit (1) uniquement le nom de l'entité, (2) avec les données membres pertinentes et (3) avec données membres et principales opérations. La forme à 2 sections (au centre) sera la plus utilisée.

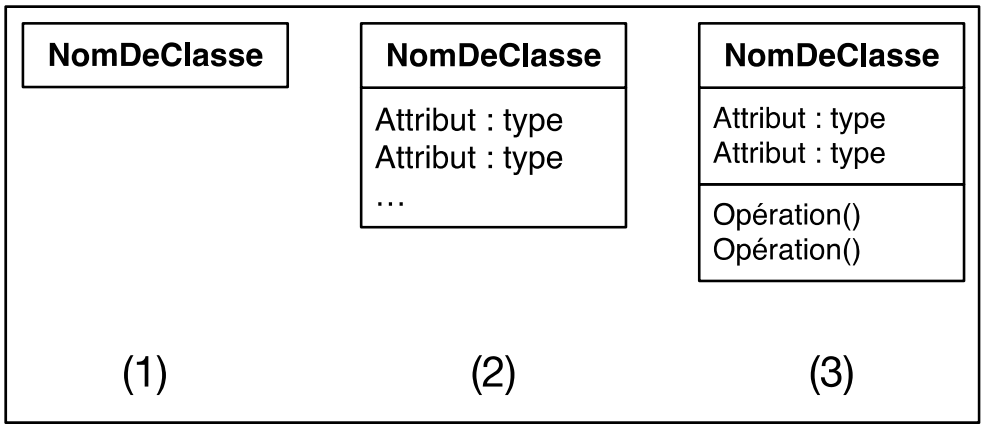
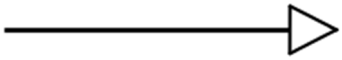
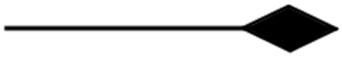
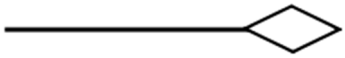


Figure 4.1 Trois niveaux de représentation des classes

4.2.2 Relations

Les relations utilisées pour relier les entités sont présentées au tableau 4.1.

Tableau 4.1 Sous-ensemble des relations du diagramme de classes UML

Relation	Symbole	Définition
Héritage		La classe dite dérivée est une spécialisation de la classe dite parent. La flèche pointe vers le parent.
Composition		La classe indiquée par la flèche regroupe des instances de la classe d'origine.
Agrégation		La classe indiquée par la flèche maintient une référence vers une instance de la classe d'origine, mais leurs cycles de vie sont indépendants. C'est une relation moins forte que la composition.

4.2.3 Cardinalité

La cardinalité des relations indique le nombre d'instances impliquées dans chaque direction. Un astérisque utilisé seul indique 0 ou plus. Une forme comme 1..* indique 1 ou plus. À titre d'exemple, la figure 4.2 présente la relation de composition entre une classe *Document* et une classe *Paragraphe* dans un logiciel de traitement de texte. Un document contient zéro paragraphe ou plus. Chaque paragraphe est relié à un seul document.

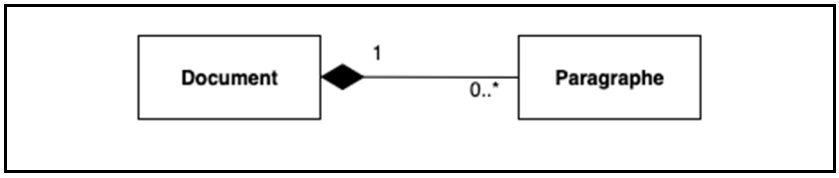


Figure 4.2 Exemple de relation avec cardinalité

4.3 Représentation logique du réseau

Au niveau logique, le modèle de carte est représenté par un objet de type *Configuration*. La *Configuration* est nommée (*name*) et contient une liste d'objets (*netList*) de type *LogicalNet*. Ces nœuds logiques représentent les liens entre les broches d'entrée-sortie des différentes puces qui composent le modèle de carte. Seuls les nœuds faisant partie de la chaîne JTAG sont représentés dans la représentation logique. Ils sont nommés d'après le nom de la broche où la

valeur du signal est mesurée. L'objet contient aussi une liste des nœuds voisins (*neighbors*), qui seront identifiés grâce à l'analyse de la disposition physique (voir section 4.7). En dernier lieu, un identifiant (*proVID*) permettant de relier le *LogicalNet* au nœud correspondant du modèle utilisé par l'outil commercial est conservé. En plus de la liste des *LogicalNet*, la *Configuration* contient également une référence au modèle physique de la carte (*layoutData*), décrit à la section 4.5. La représentation logique est illustrée en format UML à la figure 4.3.

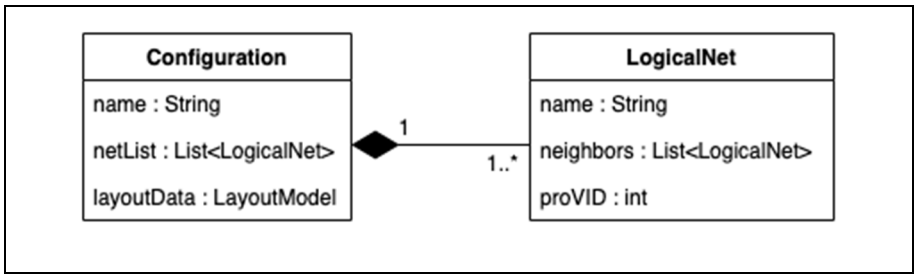


Figure 4.3 Représentation logique du réseau

4.4 Représentation des mesures et des défectuosités

La figure 4.4 présente un exemple de sortie de l'outil de test commercial ProVision tel qu'utilisé en entrée par notre système pour générer un *Syndrome*. La sortie est divisée en deux sections. La section du haut contient les noms des nœuds mesurés par ProVision, affichés à la verticale. La section du bas contient les résultats de test, organisés sous forme de matrice. Les lignes de la matrice sont numérotées et représentent les vecteurs de test. Les colonnes représentent donc les résultats d'un seul nœud pour l'ensemble des vecteurs. Des couples formés d'un chiffre et d'une lettre représentent les résultats mesurés (1 ou 0) et attendus (H ou L). Le symbole « ^ » est placé sous les résultats aberrants pour les mettre en évidence. Dans notre système, les couples chiffre-lettre deviendront des objets *TestResult*.

L'unité de base du modèle est un objet de type *TestResult*. Le *TestResult* contient les résultats attendus (*expected*) et obtenus (*measured*) pour une seule mesure, ainsi que le numéro du vecteur de test (*vector*) contenant cette mesure et une référence (*net*) au *LogicalNet* sur lequel cette mesure a été effectuée. Les *TestResult* sont regroupés simultanément de deux façons :

```
  N N N N N N N N N N N N N N N N
  e e e e e e e e e e e e e e e e
  t t t t t t t t t t t t t t t t
  U U U U U U U U U U U U U U U U
  1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2

  2 3 4 5 7 8 9 1 2 3 4 5 7 8 9 1
                0               0
-------------------------------------------------------
 1 1H 1H 0H 1H 1H 1H 1H 1H 1H 1H 1H 1H 1H 1H 1H 1H
       ^
 2 0L 0L 0L 0L 0L 0L 0L 0L 0L 0L 0L 0L 0L 0L 0L 0L
 3 0L 0L 0L 0L 0L 0L 0L 0L 0L 0L 0L 1H 1H 1H 1H 1H
 4 0L 0L 0L 0L 0L 1H 1H 1H 1H 1H 1H 0L 0L 0L 0L 0L
 5 0L 0L 0L 1H 1H 0L 0L 0L 1H 1H 1H 0L 0L 1H 1H 1H
 6 0L 1H 0H 0L 1H 0L 0L 1H 0L 1H 1H 0L 1H 0L 0L 1H
       ^
 7 0L 0L 0H 1H 0L 0L 1H 1H 0L 0L 1H 1H 0L 0L 1H 1H
       ^
 8 1H 0L 0H 0L 1H 0L 1H 0L 1H 0L 1H 0L 1H 0L 1H 0L
       ^
 9 0L 1H 0L 1H 0L 1H 0L 1H 0L 1H 0L 1H 0L 1H 0L 1H
10 1H 1H 0L 0L 1H 1H 0L 0L 1H 1H 0L 0L 1H 1H 0L 0L
11 1H 0L 0L 1H 0L 1H 1H 0L 1H 0L 0L 1H 0L 1H 1H 0L
12 1H 1H 0H 0L 0L 1H 1H 1H 0L 0L 0L 1H 1H 0L 0L 0L
       ^
13 1H 1H 0H 1H 1H 0L 0L 0L 0L 0L 0L 1H 1H 1H 1H 1H
       ^
14 1H 1H 0H 1H 1H 1H 1H 1H 1H 1H 1H 0L 0L 0L 0L 0L
       ^
```

Figure 4.4 Sortie de ProVision utilisée pour générer un Syndrome

1. Les résultats d'un vecteur pour tous les nœuds sont regroupés dans un *TestVector*;
2. Les résultats d'un nœud pour tous les vecteurs sont regroupés dans un *NodeResult*.

La représentation obtenue est illustrée à la figure 4.5.

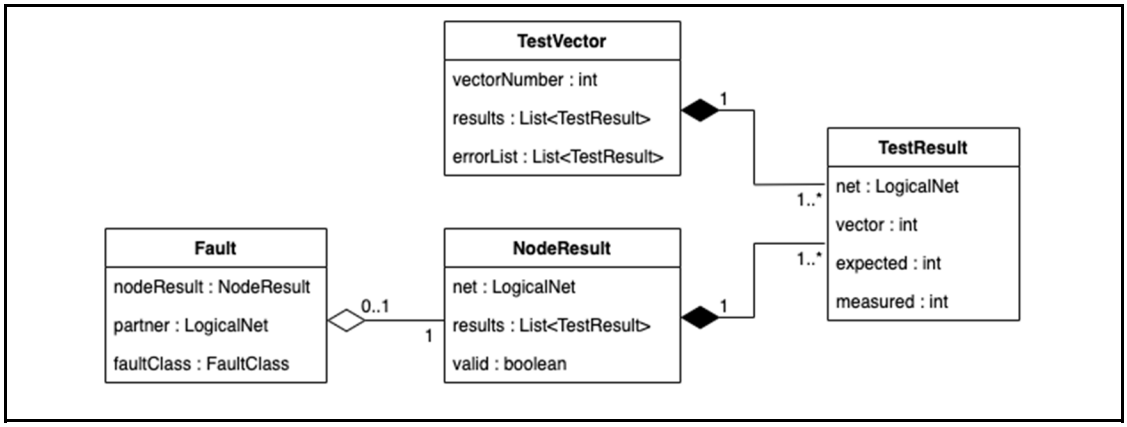


Figure 4.5 Représentation des résultats de test en deux collections

Les défectuosités détectées sont représentées par des objets de type *Fault*, qui permettent d'associer une classe de défectuosité à un *NodeResult*. Les classes modélisées sont décrites dans le tableau 4.2.

Tableau 4.2 Classes de défectuosités associées à un NodeResult

No.	**Classe de défaut**	**Définition**
0	Aucune défectuosité	Tous les résultats mesurés du nœud sont conformes aux résultats attendus.
1	Stuck-at 0	Le nœud comporte des erreurs et tous les résultats mesurés sont au niveau logique bas.
2	Stuck-at 1	Le nœud comporte des erreurs et tous les résultats mesurés sont au niveau logique haut.
3	Court-circuit	Le nœud comporte des erreurs et tous les résultats mesurés sont identiques à ceux d'un autre nœud.
4	Pont dominé par les 1	Le nœud comporte des erreurs et tous les résultats mesurés sont identiques à ceux d'un autre nœud lorsque ce dernier est au niveau logique haut. Les deux nœuds forment une porte logique OU-cablé (*Wired-OR*).
5	Pont dominé par les 0	Le nœud comporte des erreurs et tous les résultats mesurés sont identiques à ceux d'un autre nœud lorsque ce dernier est au niveau logique bas. Les deux nœuds forment une porte logique ET-cablé (Wired-AND).

Dans le cas des courts-circuits et des ponts, une référence vers le nœud partenaire (*partner*) est également conservée. L'ensemble des éléments décrits ci-haut est regroupé dans un *Syndrome*, qui encapsule la *Configuration*, les mesures et les défectuosités pour une séance d'essais sur une carte donnée. La hiérarchie de classe constituant la modélisation logique complète est représentée à la figure 4.6. Elle intègre les figures 4.3 et 4.5.

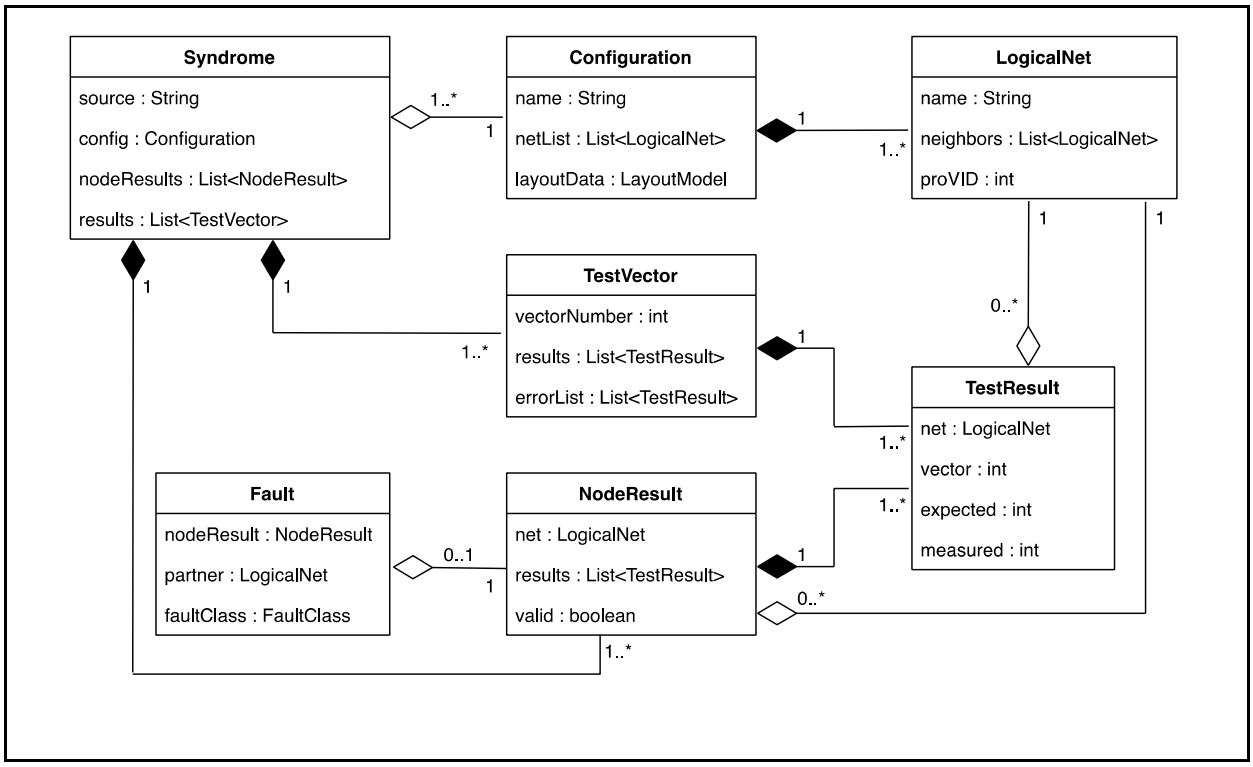


Figure 4.6 Modélisation complète d'une séance d'essais sur une carte au niveau logique

4.5 Intégration des données de fabrication assistée par ordinateur (FAO)

L'agencement physique des traces et des composants sur les différentes couches de la carte de circuit imprimée est une source d'information potentiellement très utile pour l'opération de diagnostic des pannes. Cependant cette information n'est pas prise en compte par l'outil commercial de référence. Dans le but d'améliorer la qualité du diagnostic de notre système, nous avons décidé d'intégrer cette information à notre modèle de carte.

La source primaire des données d'agencement physique réside dans les fichiers de conception assistée par ordinateur (CAO) produits par l'ingénieur.e ayant conçu la carte. Deux facteurs importants, respectivement de nature technique et juridique, nous empêchent en pratique d'utiliser ces fichiers. En effet, puisque les différents outils de CAO utilisent des formats de fichiers incompatibles et souvent propriétaires, il serait très difficile de tous les prendre en charge dans notre système. De plus, comme les fichiers de CAO constituent la propriété

intellectuelle de l'entreprise, ils ne sont généralement pas disponibles au niveau du test en production, surtout si la production est sous-contractée comme c'est généralement le cas en environnement de production *low-volume high-mix*, auquel notre système est destiné. Les fichiers de CAO n'étant pas disponibles, nous avons décidé de nous appuyer uniquement sur les données qui doivent obligatoirement être transmises au manufacturier pour permettre la fabrication. Ces données sont dites de fabrication assistée par ordinateur (FAO), ou en anglais : « computer-assisted manufacturing » (CAM).

Il existe deux standards *de facto* utilisés dans l'industrie, soit les fichiers Gerber et les bases de données ODB++. Puisque la fabrication des cartes de circuits imprimés est un procédé photolithographique, les deux standards ont en commun d'être basés sur une représentation vectorielle des éléments géométriques présents sur les différentes couches du circuit imprimé. Ils diffèrent principalement dans la quantité d'information supplémentaire qu'ils intègrent à propos du produit. Des outils de visualisation gratuits sont disponibles dans les deux cas.

4.5.1 Extended Gerber (Ucamco NV, 2021)

Aussi appelé RS-274X ou X-Gerber, ce format est un successeur du format développé par Gerber Systems Corporation pour piloter ses phototraceurs dans les années 1970. Le format X-Gerber original ou X1 est disponible depuis 1998 et contient uniquement les séquences de commandes nécessaires à la génération d'une image vectorielle représentant une couche du circuit imprimé. C'est un format propriétaire, mais l'entreprise Ucamco, qui le contrôle, rend disponible la documentation sans restrictions. Un format étendu, X2, lancé en 2014, permet l'ajout de métadonnées aux fichiers de commandes. Ces métadonnées fournissent de l'information supplémentaire comme la fonction d'une couche ou d'une pastille (*pad*). Le format X2 permet l'intégration du réseau logique (*netlist)*, mais ce n'est pas une pratique répandue. Certains outils de CAO comme KiCAD supportent cette fonctionnalité, mais d'autres comme Altium Designer ou Circuit Studio ne l'implémentent pas. Dans un procédé Gerber, la *netlist* est généralement décrite dans un fichier séparé de type IPC-D-356A, un format utilisé pour le test électrique. De façon similaire, les données de perçage sont

généralement fournies dans un autre format propriétaire appelé Excellon. La description des composants n'est pas standardisée et varie d'un environnement de CAO à l'autre. L'ensemble des fichiers est généralement regroupé dans une archive de type .zip avant la transmission au manufacturier.

4.5.2 ODB++ (Siemens, 2021)

Originalement lancé en 1997, Open DataBase++ (ODB++), est aussi un format propriétaire, développé par l'entreprise Valor, spécialisée dans la FAO. Suite à une acquisition en 2010, le format est maintenant contrôlé par le géant de la CAO Mentor, anciennement Mentor Graphics, elle-même une constituante du conglomérat Siemens depuis 2017. Une inscription est nécessaire pour accéder au visualisateur officiel, mais la documentation du format est maintenant en accès libre, comme c'est le cas pour X-Gerber.

L'intérêt principal d'ODB++ réside dans le fait que c'est un format conçu directement pour la FAO. Il est formé d'une hiérarchie standardisée de fichiers qui détaillent tous les éléments nécessaires à la fabrication et à l'assemblage d'une carte de circuit imprimé. En plus du dessin des couches, le format inclut la *netlist*, les données de perçage et une description mécanique des composants à placer. La hiérarchie de fichiers est généralement compressée dans une archive .zip que les outils compatibles peuvent utiliser directement. Un exemple typique de hiérarchie ODB++ pour une carte de circuit imprimé 4 couches est montré sur la figure 4.7.

4.5.3 Sélection d'un format et intégration

Pour que les données de FAO soient intégrables à notre système, il est primordial d'avoir la capacité d'établir un lien entre un nœud logique, élément de la *netlist*, et l'ensemble des points, lignes, vias et pastilles, potentiellement répartis sur plusieurs couches, qui en constitue l'implémentation. Puisque le format X-Gerber n'intègre généralement pas les données de la *netlist*, nous avons choisi d'utiliser le format ODB++. Ce format est disponible entre autres dans les environnement de CAO des éditeurs Altium, Mentor et Cadence.

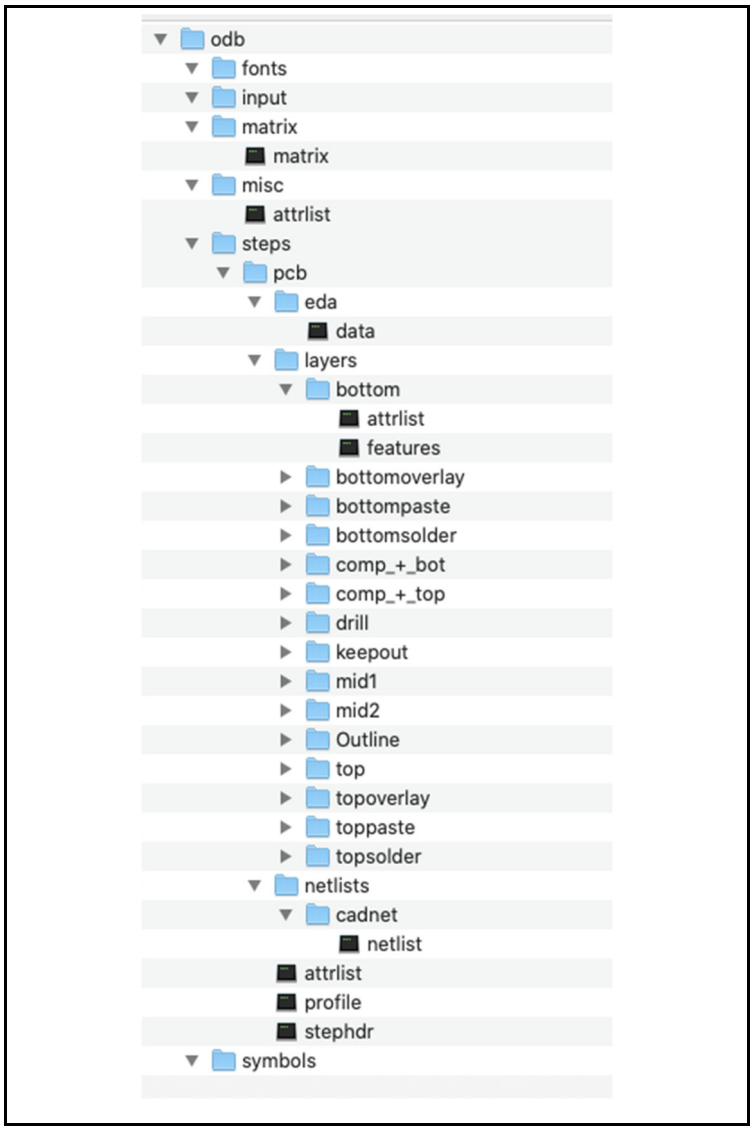


Figure 4.7 Hiérarchie ODB++ pour une carte 4 couches

Pour réaliser l'intégration, nous avons besoin de quatre types de fichiers contenus dans la hiérarchie ODB++ :

1. La *netlist* ODB++ (odb/steps/pcb/netlists/cadnet/netlist) qui contient une liste de noms de nœuds dérivés des fichiers de CAO, suivis d'une liste de coordonnées qui représentent des points intermédiaires sur le ou les chemins constituant le nœud. Ces points correspondent au point central de vias ou de pastilles reliés au nœud.
2. Les fichiers de caractéristiques (*features*) (odb/steps/pcb/layers/bottom/features) correspondant à chacune des couches électriques. Ces fichiers contiennent une liste de

symboles qui décrivent par exemple la forme et la dimension des via ou des pastilles, suivie d'une liste de vecteurs qui indiquent la position des éléments sur la couche.

3. Le fichier de paquetages (*packages*) (odb/steps/pcb/eda/data) décrit l'empreinte (*outline*) des dispositifs montés sur la carte.
4. Le fichier de composants (*components*) (odb/steps/pcb/layers/comp_+_top/components) qui décrit chaque instance de tous les dispositifs présents sur la carte : puces, éléments passifs et connecteurs. Le fichier décrit la position, l'orientation et le paquetage associé à chaque dispositif, ainsi qu'une liste de pastilles associées, avec leurs numéros de broche et leurs noms dans la *netlist*.

Les quatre types de fichiers, qui vont servir à définir le *LayoutModel* qu'on retrouve dans l'objet *Configuration* (Figure 4.3), sont représentés dans notre système de la façon suivante :

1. Les nœuds physiques de la *netlist* sont modélisés par des objets de type *LayoutNet*. Les points associés aux nœuds sont modélisés par des objets *Waypoint*. Les *Waypoint* constituent la clé permettant de faire l'association entre l'ensemble des lignes, pastilles et symboles du dessin des couches électriques et les nœuds logiques.
2. Les éléments des fichiers de caractéristiques sont modélisés en trois listes :
    a. Les *Symbols*, qui emmagasinent la forme et la dimension des symboles;
    b. Les *Segments*, qui représentent un segment de droite d'un chemin;
    c. Les *Pads*, qui représentent à la fois les pastilles et les vias.
3. L'empreinte des dispositifs présents dans le fichier de paquetages est emmagasinée dans une liste d'objets de type *Package*.
4. Une liste d'objets de type *Component* représente la position, l'orientation et le *Package* associé à chaque dispositif du fichier de composants.

La figure 4.8 montre les classes utilisées pour modéliser la disposition physique de la carte dans notre système.

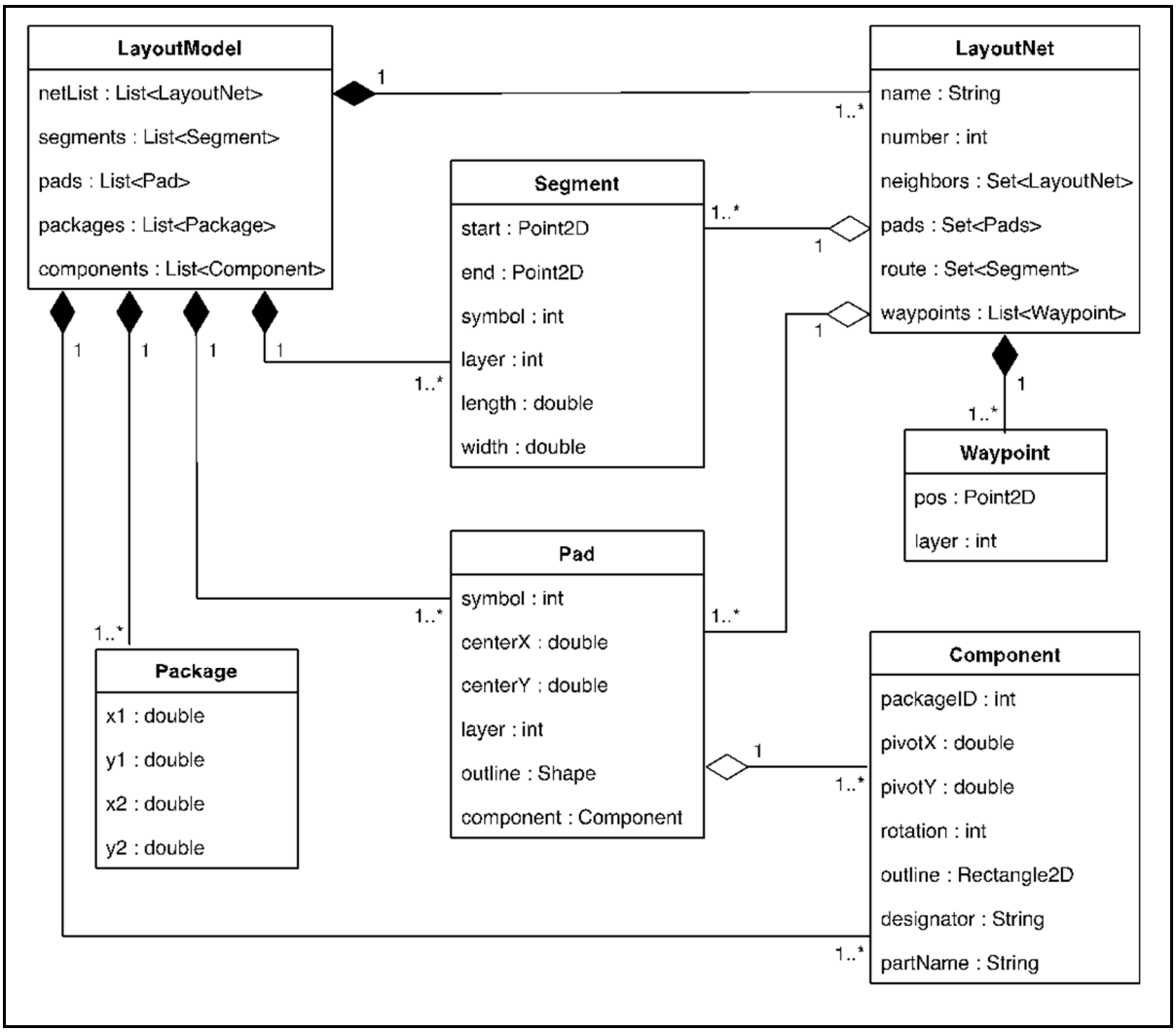


Figure 4.8 Modélisation de la disposition physique des cartes

Une représentation combinant la disposition physique de la carte et les modèles logiques sera présentée à la section 4.6.

4.5.4 Reconstruction des chemins

Les fichiers de *features* ne contiennent aucune indication à propos de l'appartenance des éléments qu'ils décrivent aux nœuds logiques. Nous avons donc développé l'algorithme 4.1 pour reconstruire l'ensemble des chemins associés à un nœud, que nous avons appelé une « route ». Pour chacun des *LayoutNet*, on dispose au départ uniquement des coordonnées de

waypoints, qui indiquent la position de pastilles ou vias connectés au *LayoutNet*. L'algorithme 1 parcours d'abord la liste des segments afin de trouver un segment dont le point de départ ou d'arrivée correspond au *waypoint*. Une fois ce segment trouvé, il est ajouté à la route et l'algorithme cherche de façon gloutonne tous les autres segments connectés, puis recommence pour chacun des *waypoints*. Comme chaque segment ne peut faire partie que d'un seul *LayoutNet*, la recherche s'accélère à mesure que l'algorithme progresse. À la fin, tous les segments ont été associés au *LayoutNet* approprié.

Algorithme 4.1 Reconstruction des chemins pour un LayoutNet

Algorithm 1 Route discovery for a LayoutNet using Waypoints to link Segments

```
Data:
    Let S be the set of all Segments not yet associated with a LayoutNet.
    Let X be the current LayoutNet.
    Let X.route be the set of all Segments associated with LayoutNet X.
    Let X.waypoints be the set of all Waypoints associated with LayoutNet X.

 1: function INITROUTE(LayoutNet X)
 2:     for each Waypoint wp in X.waypoints do
 3:         buildRoute(wp)
 4:     end for
 5: end function

 6: function BUILDROUTE(Waypoint wp)
 7:     Let LS be a list of Segments, initially empty
 8:     for each Segment s in S on the same signal layer as wp do
 9:         if s.start = wp.pos or s.end = wp.pos then
10:             add s to LS
11:         end if
12:     end for
13:     findConnected(LS)
14: end function

15: function FINDCONNECTED(List L)
16:     if L is not empty then
17:         for each Segment l in L do
18:             Let LC be a list of connected Segments, initially empty.
19:             for each Segment s in S do
20:                 if s.start = l.start or s.end = l.start or s.start = l.end or s.end = l.end then
21:                     add s to X.route
22:                     add s to LC
23:                     remove s from S        ▷ This Segment has been associated with a LayoutNet
24:                 end if
25:             end for
26:             findConnected(LC)                          ▷ Recursive call for a depth-first search
27:         end for
28:     end if
29: end function
```

4.5.5 Visualisation et validation du *LayoutModel*

Afin d'aider au débogage et à la validation de notre modélisation, nous avons développé un module permettant de visualiser le *LayoutModel*. Les figures 4.9, 4.10 et 4.11 montrent respectivement les visualisations présentées par l'outil de CAO, le visualisateur ODB++ officiel utilisant les données de FAO et notre visualisateur de *LayoutModel* utilisant les chemins reconstruits à l'aide de l'algorithme 4.1. La différence principale est que notre modèle n'inclut pas les données de perçage ni les plans de puissance. La mise en évidence de certains nœuds sur la figure 4.11 est liée au système d'identification des nœuds voisins présenté à la section 4.6.

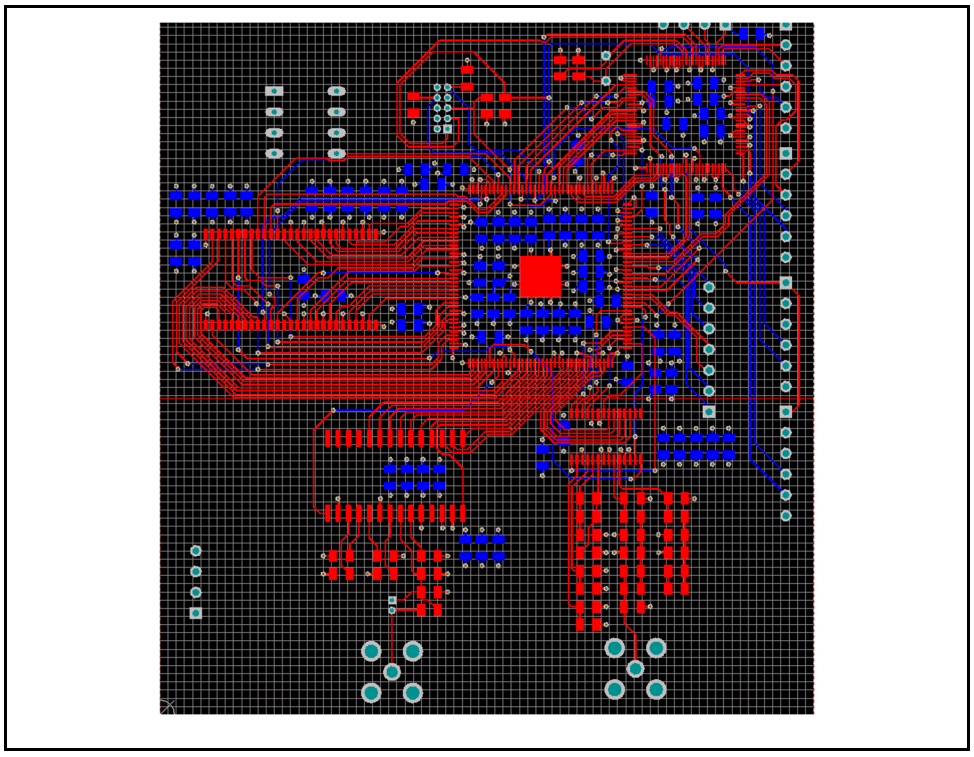


Figure 4.9 Visualisation offerte par l'outil de CAO Altium Circuit Studio

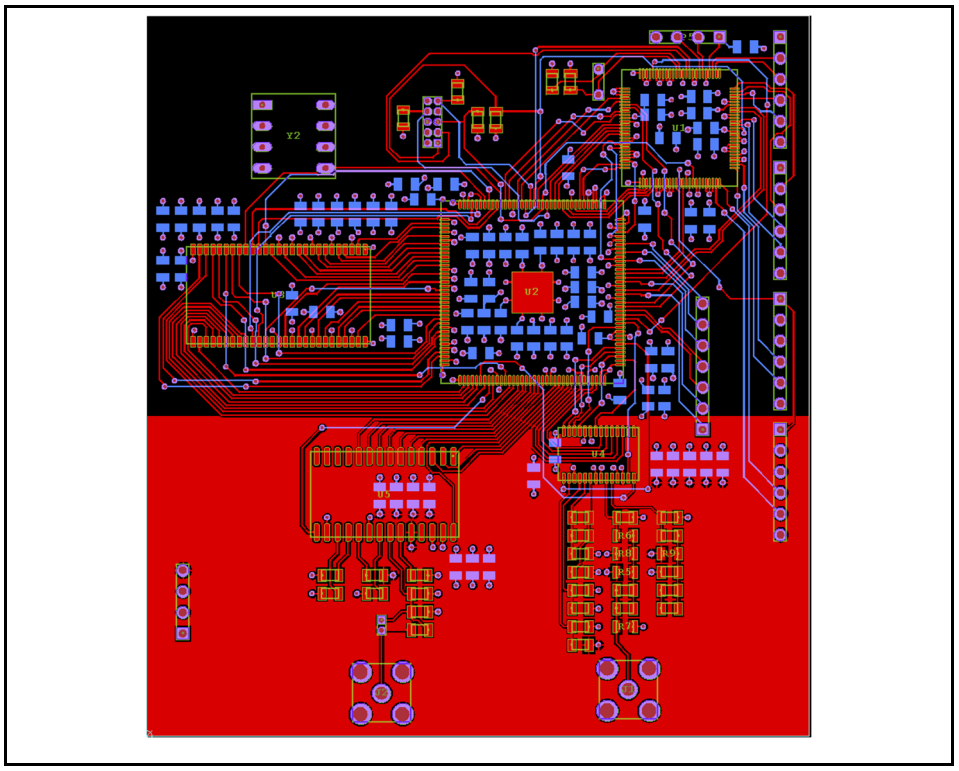


Figure 4.10 Visualisateur officiel ODB++ (données de FAO)

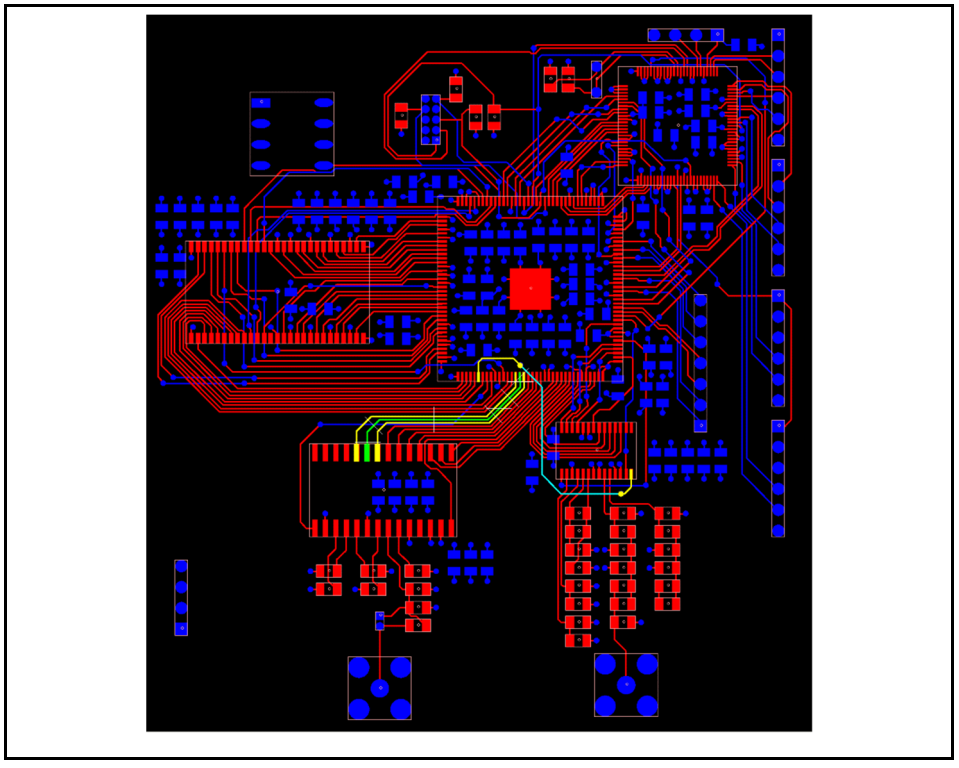


Figure 4.11 Notre visualisateur de LayoutModel

Une fois le *LayoutModel* généré, il est ajouté à l'objet *Configuration* correspondant à cette révision de la carte et est donc relié au modèle logique établi précédemment, comme il a été mentionné plus tôt. La figure 4.12 présente le modèle complet du domaine. La *Configuration* ainsi que l'ensemble des résultats de tests sont regroupés dans un objet de type *Syndrome*. C'est le *Syndrome* qui constituera la requête soumise au système de raisonnement par cas décrit au prochain chapitre en vue d'obtenir un diagnostic.

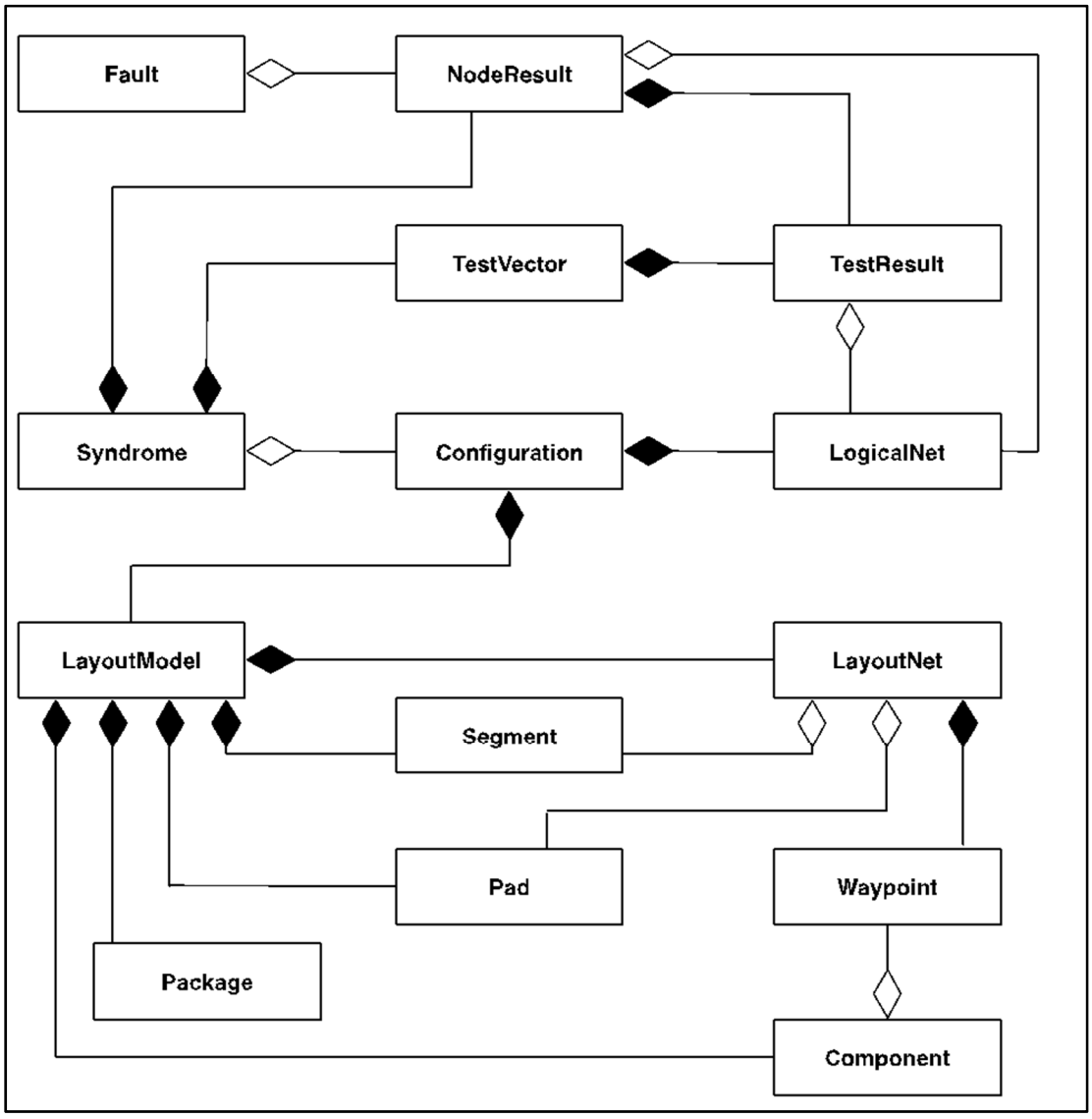


Figure 4.12 Modèle du domaine

4.6 Règles de détection et localisation des pannes

Puisque le système de mesure utilisé est entièrement numérique et basé sur un modèle logique de la carte, les résultats de test sont très simples. Pour chacun des vecteurs de test, puis pour chacun des nœuds mesurés, une valeur logique haute ou basse est attendue. Puisque seules deux valeurs sont possibles, si la valeur mesurée est différente de la valeur attendue, il y a présence d'une défectuosité. Pour adapter le moteur diagnostic à un système de mesure plus complexe, comme le testeur en circuit (ICT), il faudrait par exemple ajouter à l'objet *TestResult* la notion de seuil de tolérance.

Puisque les résultats de tests sont déjà organisés par nœud, la localisation des erreurs est effectuée au même moment que la détection. C'est le système de raisonnement par cas qui sera chargé de l'analyse de l'ensemble des résultats de test afin de faire la classification des défectuosités selon les différents scénarios de pannes présentés à la section 4.4. Cependant, dans le cas des courts-circuits et des ponts, le système devra localiser l'autre nœud impliqué dans la panne.

4.7 Identification des nœuds voisins et partenaires

L'intégration des données de disposition physique de la carte nous permet d'introduire dans notre représentation la notion de nœud voisin. Deux nœuds sont considérés comme étant voisins si leurs chemins respectifs sont adjacents en un point quelconque de leurs parcours, l'adjacence étant définie comme étant le segment le plus près d'un autre dans une direction donnée à l'intérieur d'un seuil de distance maximale. La figure 4.13 illustre ces notions à l'aide du visualisateur de *LayoutModel*. Le nœud sélectionné est représenté en vert. Les chemins des nœuds voisins détectés sont représentés en jaune sur la couche du dessus et en cyan sur la couche du dessous de la carte. Les segments ne sont pas tous adjacents. Le chemin entier est mis en surbrillance dès qu'un nœud est considéré comme un voisin. La visualisation montre les points de détection par des traits blancs dont la longueur représente le seuil de distance. L'algorithme 4.2 a été développé pour la détection des voisins. Deux nœuds peuvent également

être voisins même si aucun segment n'est adjacent. Ce sera alors l'adjacence des pastilles qui sera le facteur déterminant. La figure 4.14 montre une occurrence de cette situation.

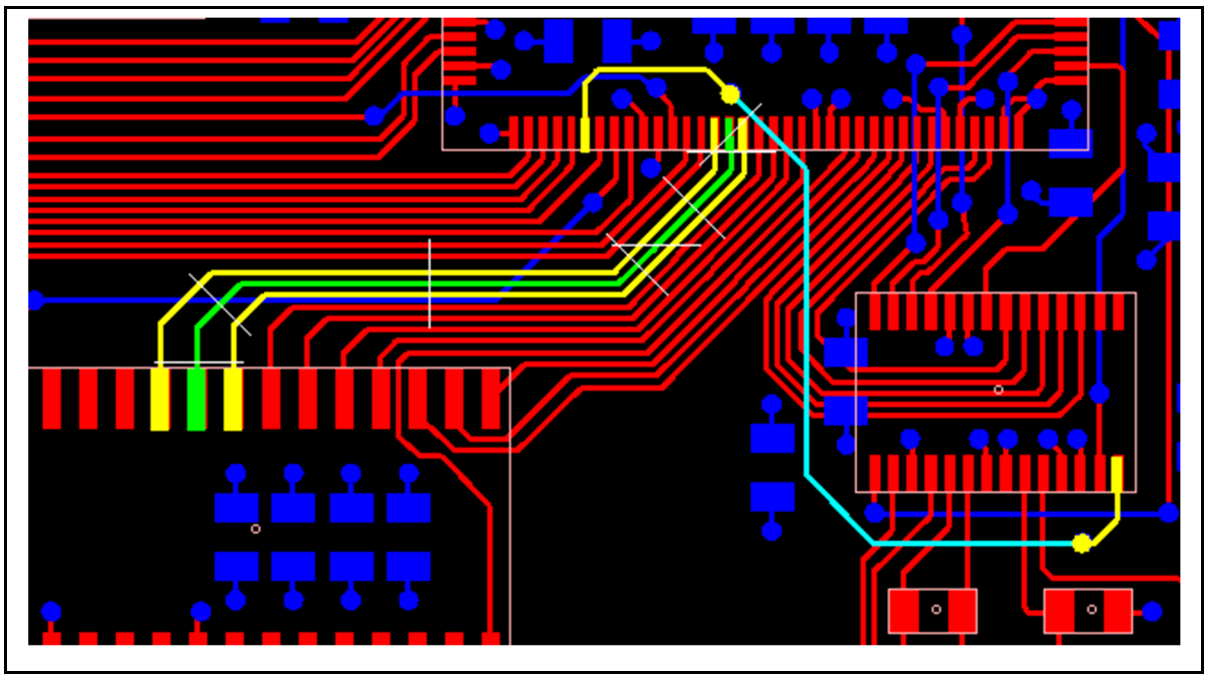


Figure 4.13 Visualisation de la détection des voisins

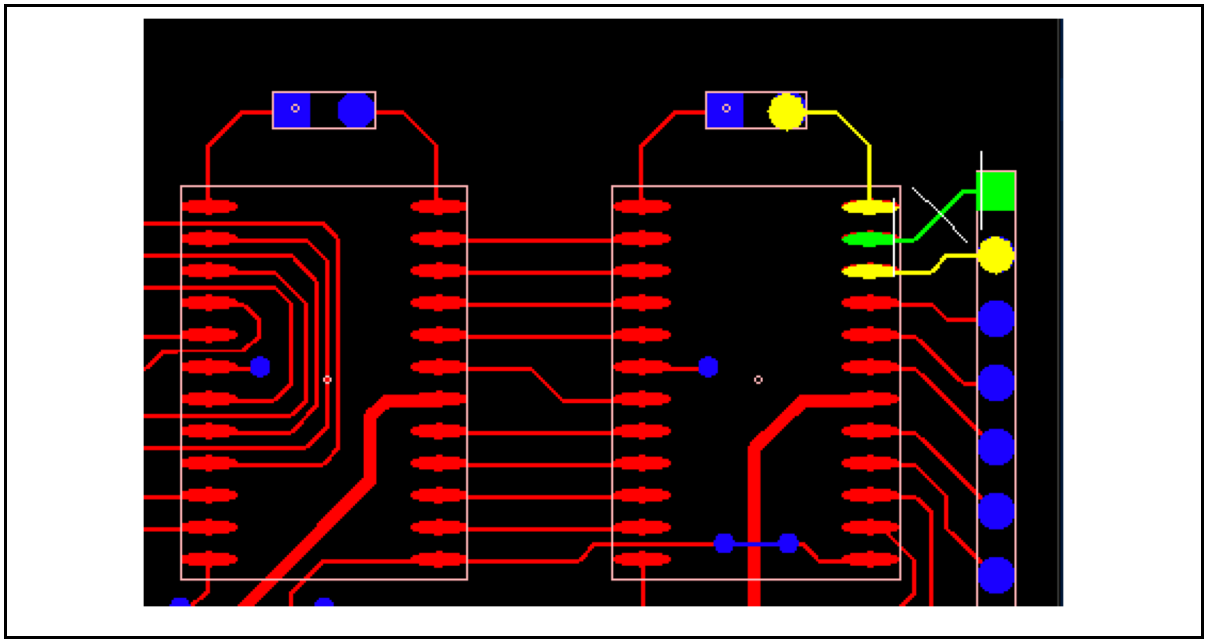


Figure 4.14 Nœuds voisins par adjacence des pastilles

Algorithme 4.2 Identification des noeuds voisins

Algorithm 2 Neighbors identification

Data:
Let *M* be the current LayoutModel.
Let *M.netlist* be the set of all LayoutNet associated with a LayoutModel.
Let *X* be the current LayoutNet.
Let *X.route* be the Route, the set of all Segments associated with LayoutNet X
Let *X.neighbors* be the set of LayoutNets that are considered neighbors of X

```
1:  function FINDNEIGHBORS(LayoutModel M, double distanceMax)
2:      for each LayoutNet X in M.netlist do
3:          for each Segment s in X.route do
4:              Let l be a normal line to s, of length distanceMax, pointing left
5:                                                  ▷ from the point of view of s
6:              Let r be a normal line to s, of length distanceMax, pointing right
7:              searchForNeighbors(M, X, s, l)
8:              searchForNeighbors(M, X, s, r)
9:          end for
10:     end for
11: end function

12: function SEARCHFORNEIGHBORS(LayoutModel M, LayoutNet X, Segment s, Line2D normal)
13:     Let SN be a set of Segments, initially empty
14:     for each LayoutNet Y of M.netlist, different from X do
15:         for each Segment s2 of Y.route on the same layer as s do
16:             if normal intersects s2 then
17:                 add s2 to SN
18:                 exit inner loop and proceed to next Y
19:             end if
20:         end for
21:     end for
22:     if SN is not empty then
23:         compute distance between s and every member of SN
24:         add the member of SN with the shortest distance to X.neighbors
25:                                     ▷ Duplicate insertions into a set are ignored.
26:     end if
27: end function
```

Cette information à propos du *layout* sera utilisée lorsque le système de raisonnement par cas classe une défectuosité comme court-circuit ou pont directionnel. Il demandera alors au système de modélisation des connaissances d'identifier parmi les nœuds voisins du nœud défectueux celui dont les résultats correspondent au profil attendu. Les règles d'identification sont présentées au tableau 4.3.

Tableau 4.3 Règles d'identification des nœuds partenaires

Scénario de panne	**Règles d'identification du partenaire**
Court-circuit	Tous les résultats sont identiques
Pont dominé par les 1	Tous les résultats erronés sont des 1 chez les deux partenaires
Pont dominé par les 0	Tous les résultats erronés sont des 0 chez les deux partenaires

4.8 Conclusion

Dans ce chapitre, nous avons présenté les éléments qui constituent le dépôt de connaissances du domaine, tel que montré sur la figure 1 du chapitre 2. Ce réservoir d'informations, à la fois générales dans le cas des règles de détection et localisation des pannes, et spécifiques dans le cas du réseau logique, des résultats de test ou encore des données de FAO, constituera le fonds de commerce du moteur diagnostique. C'est en comparant les données spécifiques et en appliquant les règles générales que l'outil parviendra à proposer un diagnostic pour un syndrome donné. Le mécanisme utilisé pour retrouver les cas comparables, les réutiliser, les réviser pour qu'ils soient applicables à un nouveau syndrome et retenir le résultat de sa démarche fera l'objet du prochain chapitre, qui présentera en détail le système de raisonnement par cas. C'est ce sous-système qui constituera la mémoire de l'engin diagnostic et qui permettra l'amélioration progressive des résultats par l'apprentissage machine.

CHAPITRE 5

DIAGNOSTIC AUTOMATISÉ PAR LA MÉTHODE DU RAISONNEMENT PAR CAS

5.1 Introduction

Le module de raisonnement par cas est l'élément de notre système dédié à l'apprentissage machine. La base de cas constitue la mémoire du système et contient l'ensemble des requêtes, ici appelées « syndromes », préalablement évaluées ainsi que le diagnostic correspondant.

Dans notre système hybride de diagnostic automatisé, le module à base de règles présenté au chapitre 4 est responsable des tâches de détection et de localisation des défectuosités. Le système de raisonnement par cas présenté ici est chargé de la classification. Les résultats montrés aux sections 5.9 et suivantes incluent la contribution des deux modules pour obtenir un diagnostic.

Les objectifs spécifiques de ce chapitre sont de :

1. Présenter la technique du raisonnement par cas ainsi que les choix de conception effectués pour le développement du prototype.
2. Montrer que le prototype fonctionne, c'est-à-dire qu'il offre une classification correcte des défectuosités qui lui sont présentées.
3. Montrer qu'il y a bien un apprentissage machine, c'est-à-dire que la performance du système s'améliore avec l'expérience.
4. Évaluer la technique d'accélération du démarrage à l'aide de cas synthétiques obtenus par émulation, une des contributions principales de ce projet.

5.2 Raisonnement par cas

Le raisonnement par cas ou *case-based reasoning* (CBR) est une technique d'intelligence artificielle ou d'apprentissage machine qui repose sur des théories et des modèles du fonctionnement de la mémoire humaine développés par des chercheurs du domaine des sciences cognitives (Richter & Aamodt, 2005). L'idée de départ est intuitive : pourquoi ne pas adapter le processus de résolution de problème utilisé par les humains tous les jours dans leurs tâches quotidiennes à l'ordinateur et à son énorme capacité mémoire?

5.2.1 Modèle humain

Mais quel est-il au juste ce processus utilisé par les humains? La plupart des problèmes rencontrés au quotidien sont en fait des variations de problèmes que nous avons résolus antérieurement. Trouver une place de stationnement, commander au restaurant, choisir les vêtements appropriés à la température ou à l'occasion sont des exemples courants. Dans chacune de ces situations, nous faisons appel, sans en avoir conscience, à notre expérience. Nous cherchons inconsciemment dans notre mémoire le souvenir d'un problème similaire. Nous adaptons ensuite la solution utilisée dans notre souvenir pour les conditions du nouveau problème afin de produire une nouvelle solution. Finalement, après avoir essayé la nouvelle solution, nous mémorisons le résultat (positif ou négatif) qui pourra nous aider lors de la résolution d'un problème similaire dans le futur.

5.2.2 Modèle machine

Tel que présenté à la section 1.8.1, la démarche du raisonnement par cas est divisée en cinq étapes, appelées les cinq « R » (Richter et Weber, 2013).

1. La **représentation** du problème;
2. Le **rappel** d'une expérience similaire;
3. La **réutilisation** de l'expérience trouvée;
4. La **révision** de la solution pour l'adapter au nouveau problème;
5. La **rétention** de la solution et de son résultat (positif ou négatif).

Le CBR est organisé autour d'un dépôt de données partagé entre les étapes appelé la **base de cas**, qui joue le rôle de l'expérience de l'être humain. La relation entre les éléments du CBR est illustrée à la figure 5.1.

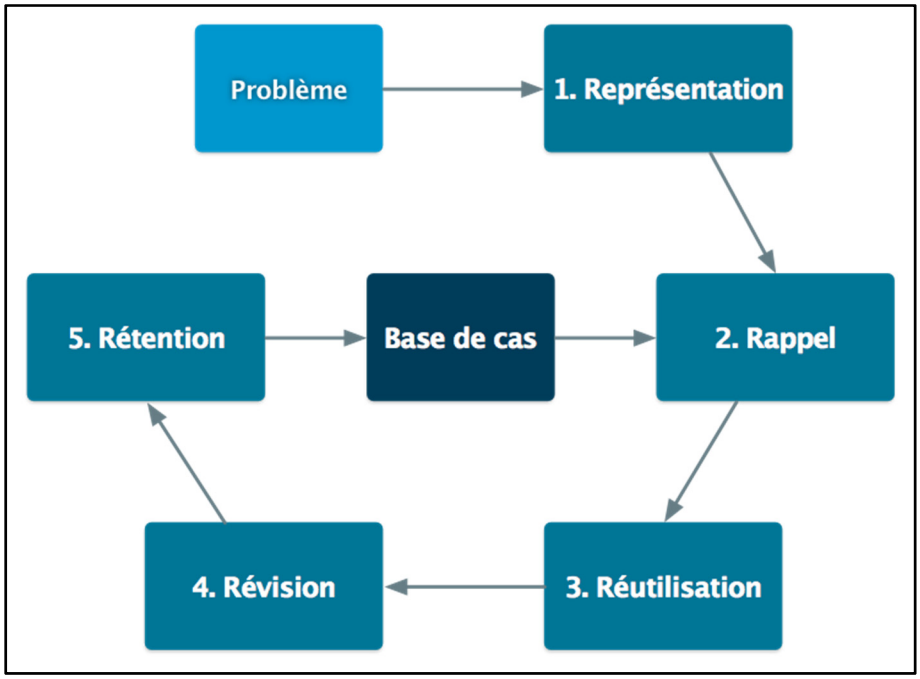


Figure 5.1 Étapes du raisonnement par cas

Les sections suivantes de ce chapitre présentent en détail l'implémentation du CBR dans notre système élaboré comme preuve de concept décrit au chapitre 2.

5.3 Représentation des cas

Chacun des cas de la base est composé de deux éléments :

1. Une requête, de type *Syndrome*, décrite au chapitre 4;
2. Une solution, de type *Diagnosis*, qui contient une liste des défectuosités (*Fault*) trouvées dans un *Syndrome*, chacune d'entre elles étant associée à une classe de pannes.

Les cas (*Case*) sont regroupés dans un objet de type *CaseBase*, qui est également chargé des fonctionnalités reliées à la persistance des données. Les classes d'objets responsables de la représentation des cas sont illustrées à la figure 5.2.

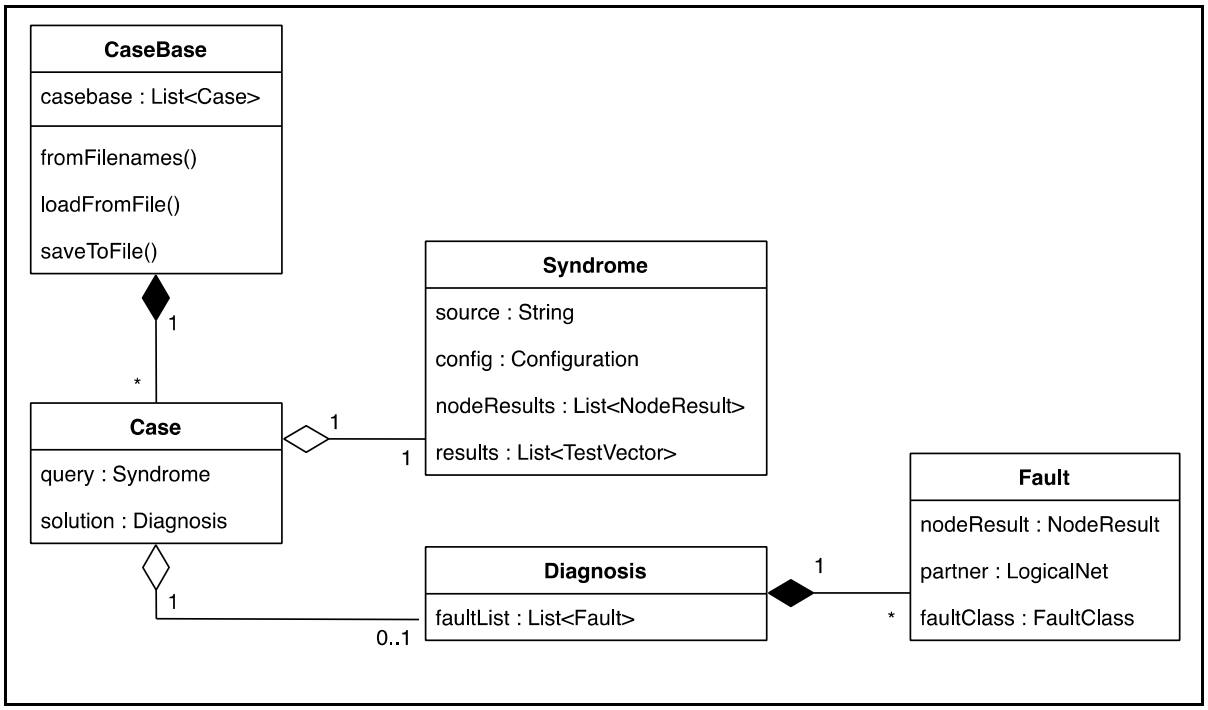


Figure 5.2 Modélisation de la base de cas

5.4 Classes de défectuosités

Les six classes de défectuosités sont les mêmes que celles présentées au chapitre 4, reproduites dans le tableau 5.1 :

Tableau 5.1 Classes de défectuosités

No.	Classe de défectuosités	Définition
0	Aucune défectuosité	Tous les résultats mesurés du nœud sont conformes aux résultats attendus.
1	*Stuck-at* 0	Le nœud comporte des erreurs et tous les résultats mesurés sont au niveau logique bas.
2	*Stuck-at* 1	Le nœud comporte des erreurs et tous les résultats mesurés sont au niveau logique haut.
3	Court-circuit	Le nœud comporte des erreurs et tous les résultats mesurés sont identiques à ceux d'un autre nœud.
4	Pont dominé par les 1	Le nœud comporte des erreurs et tous les résultats mesurés sont identiques à ceux d'un autre nœud lorsque ce dernier est au niveau logique haut. Les deux nœuds forment une porte logique OU-cablé (*Wired-OR*).
5	Pont dominé par les 0	Le nœud comporte des erreurs et tous les résultats mesurés sont identiques à ceux d'un autre nœud lorsque ce dernier est au niveau logique bas. Les deux nœuds forment une porte logique ET-cablé (Wired-AND).

5.4.1 Défectuosités multiples

Ces classes de défectuosités représentent des pannes simples, touchant un ou deux nœuds du circuit. Il est possible qu'un *Syndrome* soumis au système pour diagnostic contienne plusieurs pannes, sur des nœuds différents. C'est pourquoi l'objet de type *Diagnosis* présent dans un *Case* contient une liste de défectuosités plutôt qu'une seule.

Cette situation a cependant un impact sur la conception du rappel. Si, pour les besoins du rappel, on considère comme une « expérience » seulement les objets de type *Case* au complet, dans le cas d'une nouvelle requête avec défectuosités multiples, il faudra trouver un autre cas similaire avec la même combinaison de défectuosités pour obtenir un diagnostic satisfaisant. Ceci réduit artificiellement la qualité du diagnostic atteignable pour une base de cas d'une taille donnée et ne correspond pas à l'idée de reproduire la démarche d'un technicien humain, capable de combiner ses expériences.

Pour cette raison, nous avons décidé d'affiner la granularité de ce qui constitue une « expérience » pour les besoins du rappel. Plutôt que de tenter de retrouver un cas entier

correspondant à la requête reçue, chacun des objets de type *Fault* présent dans le *Diagnosis* sera considéré de façon indépendante comme candidat pour la classification de chacune des défectuosités présentes dans la requête.

Il y a donc deux façons de visualiser la base de cas (Figure 5.3):

1. L'ensemble des objets de type *Case* présents dans la *CaseBase* courante;
2. L'ensemble des objets de type *Fault* présents dans la *CaseBase.*

Dans le cas de *Syndromes* avec une seule défectuosité, les 2 représentations sont équivalentes.

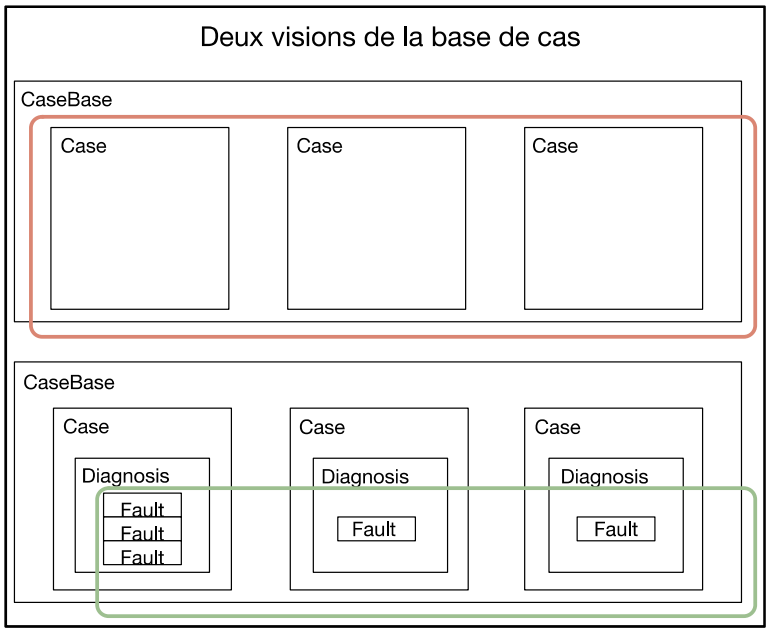


Figure 5.3 Visualisations de la base de cas

5.5 Notion de similarité (Richter et Weber, 2013)

La notion de similarité est centrale à la technique du raisonnement par cas. Elle permet de quantifier le degré de ressemblance entre deux situations ou expériences présentées au système. La mesure de similarité pour un espace de problème *P* est une fonction

$$sim: P \times P \rightarrow [0,1] \tag{5.1}$$

Une plus grande valeur de *sim* indique une plus grande similarité et une valeur de 0 indique qu'il n'y a aucune similarité entre les éléments comparés. Les axiomes suivants sont définis pour la fonction de similarité pour tous *x*, *y*, *z* :

$$sim(x,x) = 1\ (réflexivité) \tag{5.2}$$
$$sim(x,x) = sim(y,y)\ (constance\ ou\ autosimilarité)$$
$$sim(x,y) = sim(y,x)\ (symmétrie)$$

Pour tout problème *x*, un **plus proche voisin** est un problème *x'* qui possède la similarité **maximale** parmi tous les problèmes présents dans l'ensemble *P*.

5.5.1 Similarité et distance (Richter et Weber, 2013)

Comme la notion de similarité, contrairement à l'égalité par exemple, a tendance à rester assez floue, la mesure de distance, mieux définie, est souvent utilisée comme substitut. Une mesure de distance sur un ensemble U est une fonction

$$d: U \times U \rightarrow [0,\infty] \tag{5.3}$$

On remarque que la plage de valeurs possibles pour la distance n'est pas bornée, mais dans la pratique elle sera souvent restreinte. La mesure de distance possède les propriétés suivantes :

$$\forall x: d(x,x) = 0\ (réflexivité) \tag{5.4}$$
$$\forall x,y: d(x,y) = 0 \iff x = y\ (réflexivité\ forte)$$
$$\forall x,y: d(x,y) = d(y,x)\ (symmétrie)$$
$$\forall x,y,z: d(x,y) + d(y,z) \geq d(x,z)\ (Inégalité\ triangulaire)$$

Une mesure de distance peut être utilisée comme substitut pour la similarité si :

$$sim(x,y) \geq sim(x,z) \Leftrightarrow d(x,y) \leq d(x,z) \tag{5.5}$$

Dans ce cas, pour tout problème *x*, un plus proche voisin est un problème *x'* qui possède la distance **minimale** parmi tous les problèmes présents dans l'ensemble P.

5.5.2 Choix des critères d'évaluation

Pour déterminer le score de similarité, il faut d'abord sélectionner les critères à considérer pour le calcul. Ici encore, nous tenterons de reproduire le processus utilisé par les humains pour notre système raisonnant. La figure 5.4 présente un exemple de sortie de l'outil de test commercial ProVision tel qu'utilisé en entrée par notre système pour générer un *Syndrome*.

```
    N  N  N  N  N  N  N  N  N  N  N  N  N  N  N  N
    e  e  e  e  e  e  e  e  e  e  e  e  e  e  e  e
    t  t  t  t  t  t  t  t  t  t  t  t  t  t  t  t
    U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U
    1  1  1  1  1  1  1  1  2  2  2  2  2  2  2  2

    _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
    2  3  4  5  7  8  9  1  2  3  4  5  7  8  9  1
                        0                       0
-------------------------------------------------
  1 1H 1H 0H 1H 1H 1H 1H 1H 1H 1H 1H 1H 1H 1H 1H
           ^
  2 0L 0L 0L 0L 0L 0L 0L 0L 0L 0L 0L 0L 0L 0L 0L
  3 0L 0L 0L 0L 0L 0L 0L 0L 0L 0L 0L 1H 1H 1H 1H
  4 0L 0L 0L 0L 0L 1H 1H 1H 1H 1H 1H 0L 0L 0L 0L
  5 0L 0L 0L 1H 1H 0L 0L 0L 1H 1H 1H 0L 0L 1H 1H
  6 0L 1H 0H 0L 1H 0L 0L 1H 0L 1H 1H 0L 1H 0L 0L
           ^
  7 0L 0L 0H 1H 0L 0L 1H 1H 0L 0L 1H 1H 0L 0L 1H
           ^
  8 1H 0L 0H 0L 1H 0L 1H 0L 1H 0L 1H 0L 1H 0L 1H
           ^
  9 0L 1H 0L 1H 0L 1H 0L 1H 0L 1H 0L 1H 0L 1H 0L
 10 1H 1H 0L 0L 1H 1H 0L 0L 1H 1H 0L 0L 1H 1H 0L
 11 1H 0L 0L 1H 0L 1H 1H 0L 1H 0L 0L 1H 0L 1H 1H
 12 1H 1H 0H 0L 0L 1H 1H 1H 0L 0L 0L 1H 1H 0L 0L
           ^
 13 1H 1H 0H 1H 1H 0L 0L 0L 0L 0L 0L 1H 1H 1H 1H
           ^
 14 1H 1H 0H 1H 1H 1H 1H 1H 1H 1H 1H 0L 0L 0L 0L
           ^
```

Figure 5.4 Exemple de sortie de ProVision utilisée pour générer un Syndrome

En regardant la figure 5.4, l'opérateur humain expérimenté reconnaît très rapidement que le nœud *NetU1_4* est défectueux et qu'il s'agit fort probablement d'une panne de type *stuck-at-0* puisque toutes les valeurs mesurées sont à 0, sans exception. L'opérateur qui a déjà vu ce

type de panne saura le reconnaître sur tous les produits, peu importe le nombre de nœuds ou de vecteurs de test présentés.

Bien sûr, les pannes de type *stuck-at-0* ou *stuck-at-1* ont une signature très particulière et peuvent être identifiées par des règles très simples. Mais qu'en est-il des pannes plus complexes, comme le pont dominé par les zéros (ET-câblé) par exemple? La figure 5.5 montre un autre exemple, cette fois tiré du module de visualisation de la base de cas intégré à notre système.

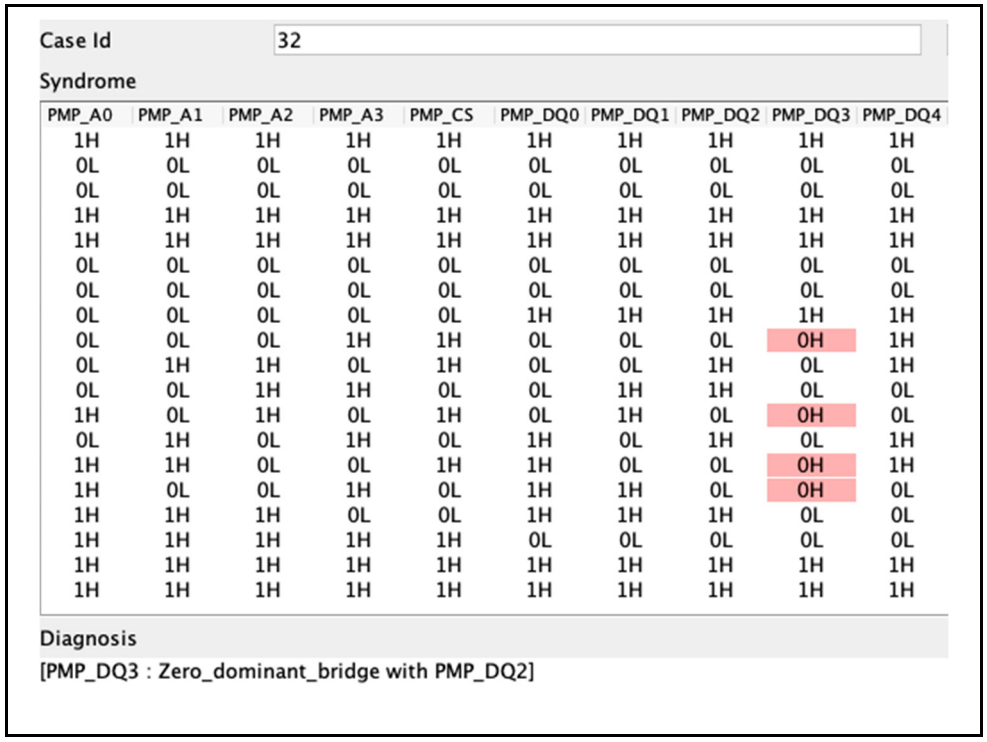


Figure 5.5 Exemple de Syndrome montrant un pont dominé par les zéros

En nous basant sur notre propre expérience, nous émettons l'hypothèse que le processus de classification de cette défectuosité par un opérateur humain ressemble au suivant:

1. Il remarque que 4 résultats du nœud PMP_DQ3 sont erronés;
2. Tous les résultats aberrants sont des 0 alors qu'on attendait des 1;
3. Il ne s'agit pas d'un *stuck-at* puisqu'il y a des résultats corrects pour les 2 valeurs;
4. Il ne s'agit pas d'un court-circuit puisque toutes les erreurs sont dans la même direction;

5. Il s'agit probablement d'un pont dominé par les zéros (*Wired-AND*).

Lorsqu'on répète l'exercice sur plusieurs *Syndrome* avec des produits différents (dont le nombre de nœuds et de vecteurs diffère), on constate qu'avec l'expérience on développe une intuition sur la nature des pannes basée sur la reconnaissance de patrons. En effet, tous les *stuck-at* ont une certaine apparence, de même que les courts-circuits et les ponts directionnels.

Suite à l'analyse de ce processus hypothétique, nous avons retenu 2 critères principaux pour établir la distance (et par le fait même le degré de similarité) entre 2 cas :

1. La proportion de résultats erronés par rapport au total;
2. La proportion d'erreurs de type 1 vers 0 par rapport aux 0 vers 1.

Puisque nous avons retenu deux critères, il devient possible de visualiser la base de cas comme un espace euclidien à deux dimensions. Le syndrome reçu comme requête, ainsi que chacune des expériences contenues dans la base de cas possède une position dans l'espace déterminée par la valeur pondérée des deux critères.

La figure 5.6 présente une illustration d'une petite base de cas (20 cas, sans doublons) où la distance entre le syndrome K (en rouge) et les quatre plus proches voisins (B, I, J, T, en bleu) est indiquée.

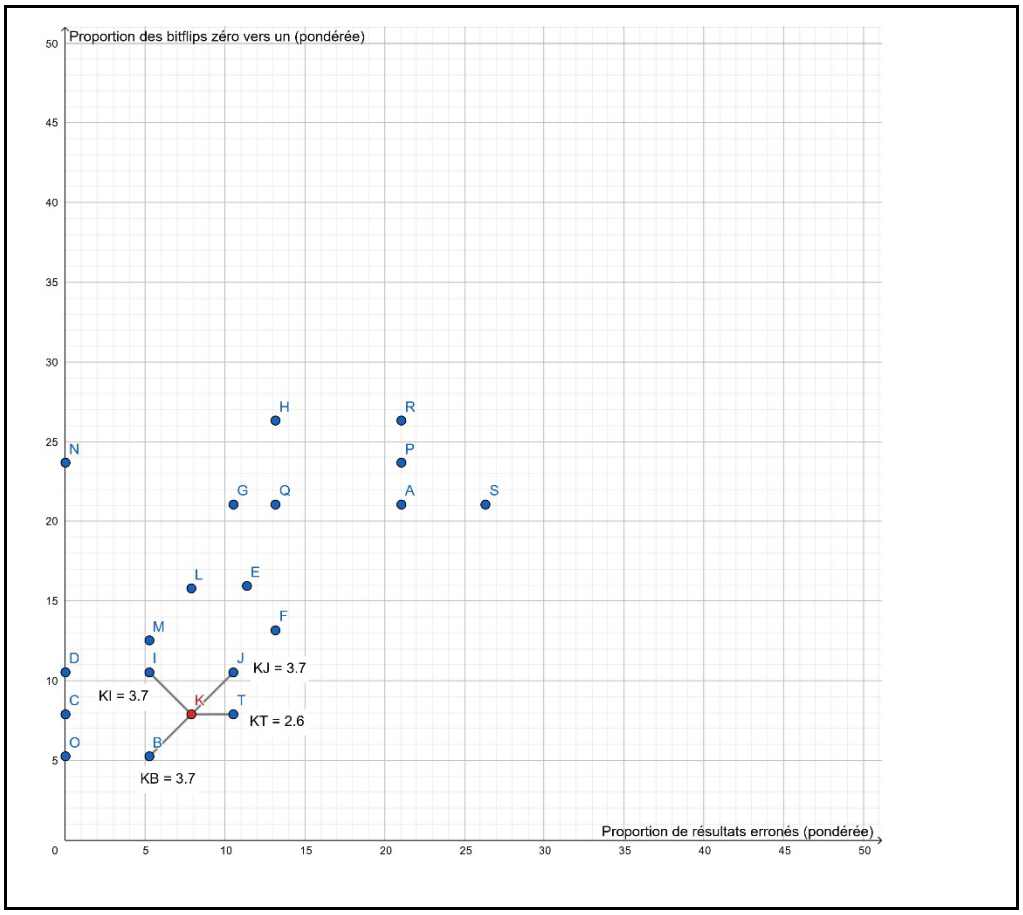


Figure 5.6 Représentation des cas d'une base selon leur distance

Le calcul de la distance entre deux cas *A* et *B* est basé sur la distance euclidienne entre les résultats de calcul des 2 proportions retenues, selon la formule 5.6 :

$$x_1 = proportionErronésA * poids1 \quad (5.6)$$
$$x_2 = proportionErronésB * poids1$$
$$y_1 = proportionZeroVersUnA * poids2$$
$$y_2 = proportionZeroVersUnB * poids2$$
$$\mathrm{d} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$
$$sim = (poids1 + poids2) - d$$

L'utilisation de proportions pour la reconnaissance de patrons est la clé pour permettre la réutilisation de l'expérience acquise sur un produit vers un produit différent. Le niveau

d'efficacité d'une telle réutilisation sera évalué à la section 5.9.1, mais les bénéfices potentiels dans un environnement à faible volume sont considérables.

5.5.3 Attribution des poids relatifs

À première vue, rien ne permet de déterminer lequel des deux critères retenus devrait être dominant. Nous avons donc utilisé une répartition 50-50 comme point de départ. Cette répartition permet déjà d'obtenir de bons résultats. Nous avons ensuite tenté de déterminer des proportions optimales de façon empirique. Les résultats présentés aux figures 5.7 et 5.8 sont cependant peu concluants. En faisant varier les proportions, on obtient une légère amélioration du taux de succès pour certains types de pannes, mais une diminution pour d'autres. Les taux de succès divergent aussi selon le produit testé. Il est important de noter l'échelle verticale des figures 5.7 et 5.8 (de 96 à 99 %) puisque même si les courbes semblent dramatiques, l'effet réel est mineur.

Sur les deux cartes testées, le taux de succès semble quasi optimal avec une répartition 50-50. Nous avons donc décidé de conserver ces proportions, en attendant de pouvoir procéder à des essais similaires sur un échantillon de produits plus grand.

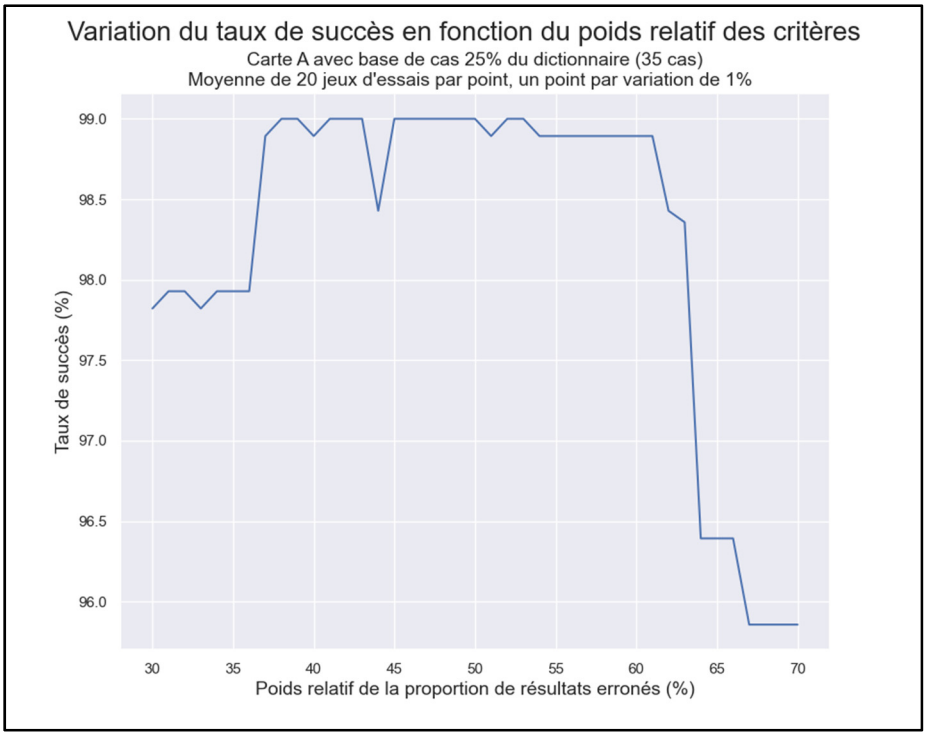


Figure 5.7 Taux de succès selon le poids relatif des critères, carte A

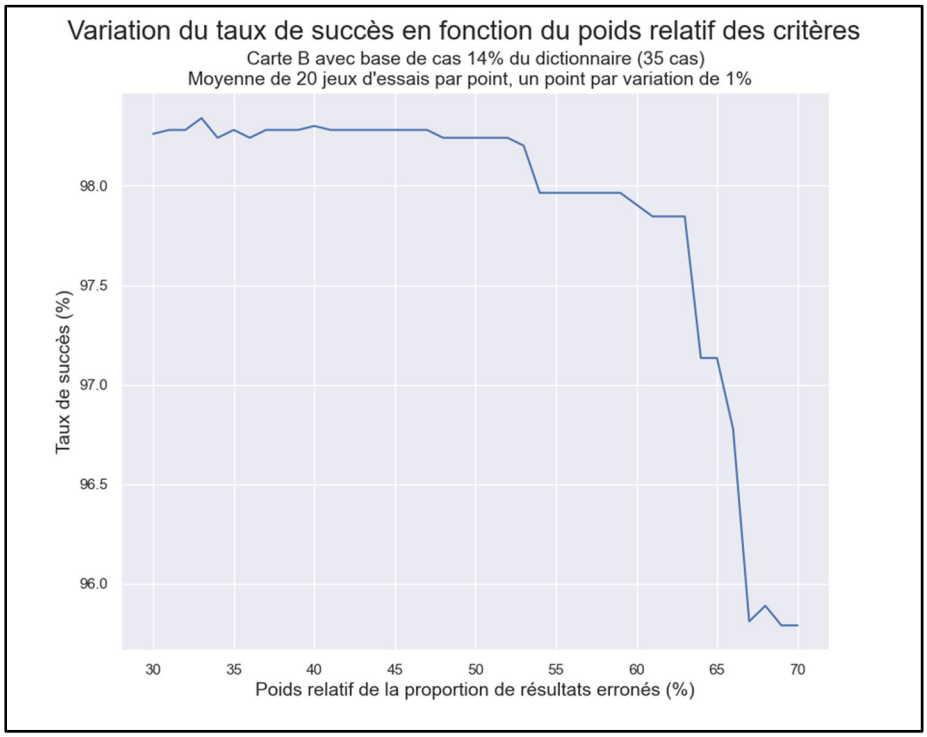


Figure 5.8 Taux de succès selon le poids relatif des critères, carte B

5.6 Rappel de cas

Lors de la réception d'une nouvelle requête, le système CBR doit trouver dans la base de cas les cas précédents qui se rapprochent le plus de la nouvelle requête. Cette étape est cruciale, car le diagnostic proposé pour la nouvelle requête sera basé sur le diagnostic offert pour le ou les cas précédents ayant la plus forte similarité (i.e. la plus faible distance). Lors de chaque requête, un objet appelé *SimilarityService* sera donc chargé d'établir un score de similarité entre chacun des cas présents dans la base et la nouvelle requête. Afin de faciliter les manipulations ultérieures, ce score sera encapsulé avec une référence vers le cas correspondant dans un objet de type *Candidate*. La figure 5.9 illustre les classes impliquées dans le rappel de cas.

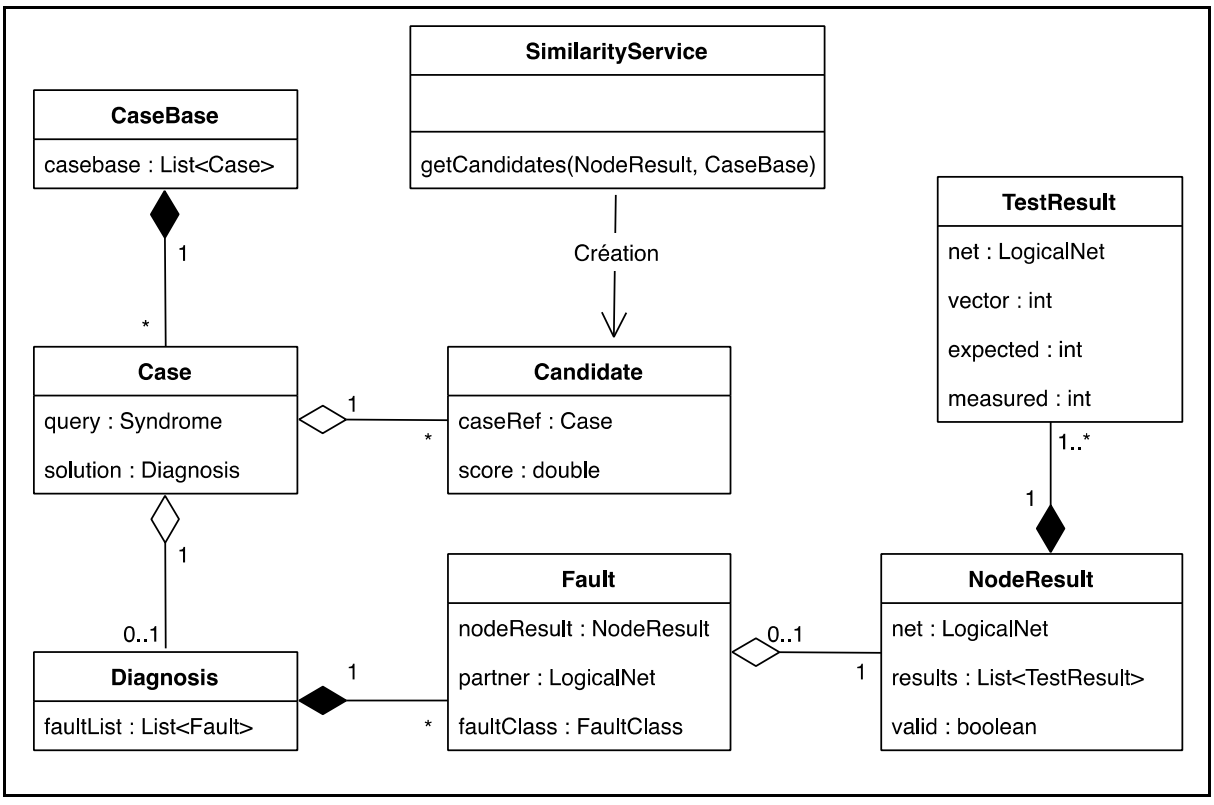


Figure 5.9 Classes impliquées dans le rappel de cas

La procédure de rappel consistera donc en la production par l'objet *SimilarityService* d'une liste de *Candidate* pour chacun des *NodeResult* indiquant un nœud défectueux dans le

Syndrome reçu. Pour créer une liste de *Candidate*, *SimilarityService* devra comparer un *NodeResult* défectueux du *Syndrome* avec tous les *NodeResult* contenus dans les objets *Fault* de tous les *Diagnosis*, eux-mêmes contenus dans les objets *Case* de la *CaseBase*. Pour chacun de ces *NodeResult*, un score de similarité est calculé et un objet *Candidate* est créé. La liste est ensuite triée par score décroissant pour obtenir les meilleurs candidats.

5.6.1 Rappel séquentiel et méthode des *k* plus proches voisins

La méthode utilisée pour le rappel dans notre système consiste à calculer le degré de similarité entre un *NodeResult* défectueux du *Syndrome* constituant la requête et l'ensemble des *NodeResult* ayant déjà fait l'objet d'un diagnostic. Dans la terminologie du raisonnement par cas, cette méthode de rappel de base est appelée le rappel séquentiel.

La complexité algorithmique de cette méthode croît de façon linéaire selon la taille de la base de cas. Lors de nos essais portant sur plus de 5 millions de requêtes, nous avons observé des rythmes d'évaluation variant de 700 à plus de 4000 requêtes par seconde, selon la carte concernée et la taille de la base de cas. Puisque l'application cible les environnements à faible volume, la puissance de calcul disponible devrait être suffisante pour assurer un temps de réponse adéquat avec les tailles de base de cas envisageables en pratique.

Notre méthode de rappel générant une liste des cas triés selon la distance la plus courte (définie à la section 5.4.2) est tout à fait appropriée pour l'implémentation de la méthode de classification dite des *k* plus proches voisins (*k-nearest neighbors* ou *k*-NN) (Richter et Weber, 2013).

Dans l'implémentation actuelle du système, la classe de panne proposée pour une nouvelle requête est identique à celle du voisin identifié comme le plus proche, donc *k=1*. Cependant, l'interface utilisateur montre la classification et le score de similarité des 10 plus proches voisins. Une discussion des implications de ce choix et une exploration des taux de succès avec des valeurs de k différentes seront présentées à la section 5.7.2, à la suite des résultats généraux.

5.7 Réutilisation, révision et rétention

Selon l'implémentation actuelle, la classification de pannes du cas le plus proche retrouvé est réutilisée directement comme solution proposée pour la nouvelle requête. La révision consiste en l'ajustement de l'identification du nœud défectueux et du nœud partenaire dans le cas des courts-circuits et ponts. Le diagnostic est ensuite proposé à l'utilisateur. Si l'utilisateur confirme que le diagnostic est approprié, la proposition est ajoutée dans la base de cas. Dans le cas où le diagnostic est erroné, l'utilisateur doit procéder à une entrée manuelle de la solution, à l'aide d'un formulaire distinct. Ce système d'entrée manuelle est également utilisé lors de la phase de démarrage alors que la base de cas est vide ou trop petite pour effectuer un diagnostic. La figure 5.10 montre une capture d'écran du module de saisie.

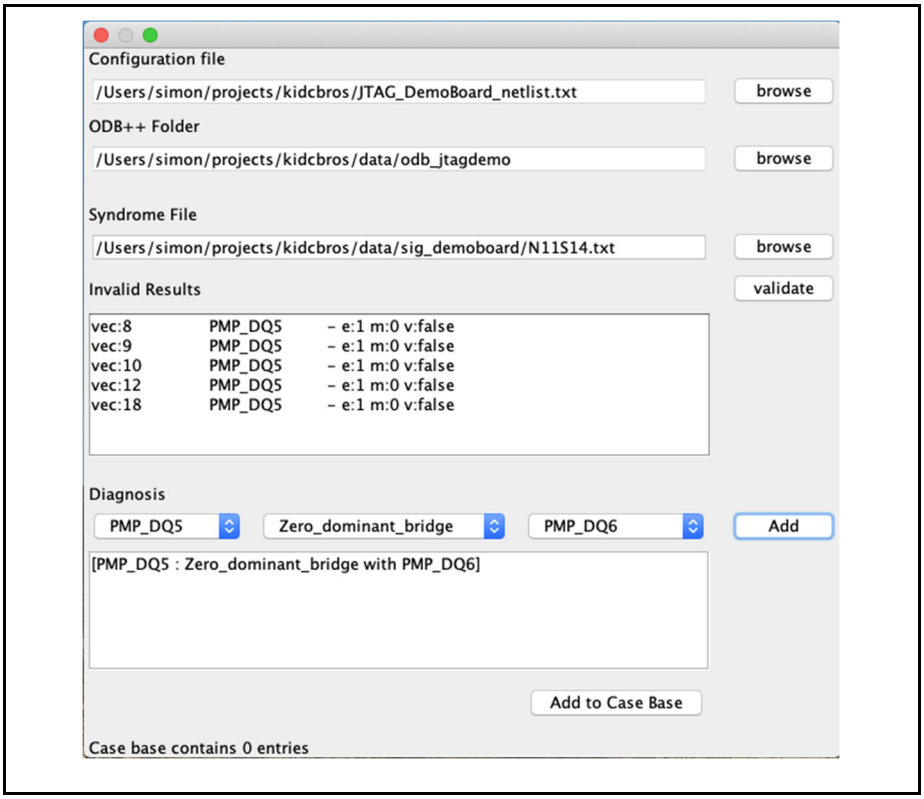


Figure 5.10 Interface utilisateur du module de saisie manuelle des cas

5.8 Expérimentation

Les résultats présentés dans la prochaine section ont été obtenus en effectuant une série de requêtes indépendantes sur des bases de cas de différentes tailles, constituées elles-mêmes de cas choisis aléatoirement, de façon indépendante pour chaque taille. Par exemple, la base de cas à 50 échantillons n'est pas un surensemble de la base de cas à 25 échantillons.

Les échantillons constituant la base de cas ont été sélectionnés de façon aléatoire à partir des répertoires contenant l'ensemble des pannes simples possibles.

Les échantillons utilisés sont des fichiers de sortie de ProVision similaires à celui montré à la figure 5.4. Ils font partie des dictionnaires de pannes générés automatiquement par les outils présentés au chapitre 3. Pour chacune des tailles de base de cas testée, nous avons généré 20 jeux d'essais.

Pour chacune des bases de cas générées, on soumet une à une toutes les pannes simples possibles pour cette carte comme requête. On mesure le taux de succès du diagnostic, c'est-à-dire la proportion des cas où la classe de panne du plus proche voisin trouvé correspond effectivement à la défectuosité présente dans la requête. Les essais sont effectués séparément pour chacune des deux cartes présentées au chapitre 2. Tous les essais sont indépendants, les résultats n'étant pas ajoutés aux bases de cas générées au préalable.

Les essais sont effectués de façon automatisée et répétable, un paramètre en ligne de commande permettant de faire basculer l'application de CBR du mode interactif avec interface graphique au mode de traitement par lot piloté par des fichiers de configuration utilisés pour les essais.

Nous sommes principalement intéressés par le comportement de l'outil de diagnostic pour les bases de cas de très petites tailles. Nous avons testé toutes les tailles de base de cas à partir de 10 cas jusqu'à 50% de la taille du dictionnaire de panne pour chaque carte.

Pour la carte A, nous avons donc procédé à 20 jeux × 61 tailles de base de cas × 140 pannes simples possibles, soit 168 140 requêtes pour cette première partie des essais. La carte B est plus complexe et a nécessité 20 jeux × 117 tailles × 253 pannes possibles, soit 587 213 requêtes. Les graphiques présentés à la section suivante montrent l'évolution de la moyenne du taux de succès des 20 jeux d'essais pour chacune des tailles ainsi que les valeurs minimales et maximales obtenues. Les courbes de taux de succès montrées sur les figures 5.11 à 5.14 présentent parfois un comportement non-monotonique de l'évolution du taux de succès. Ceci est attribuable à la formation aléatoire des bases de cas utilisées. En effet, certaines bases de cas aléatoires présentent peu de diversité et obtiennent des taux de succès très inférieurs à la moyenne. Les cas extrêmes sont représentés par la courbe « *min* » montrant la pire base de cas parmis les 20 générées pour chaque taille.

5.9 Résultats

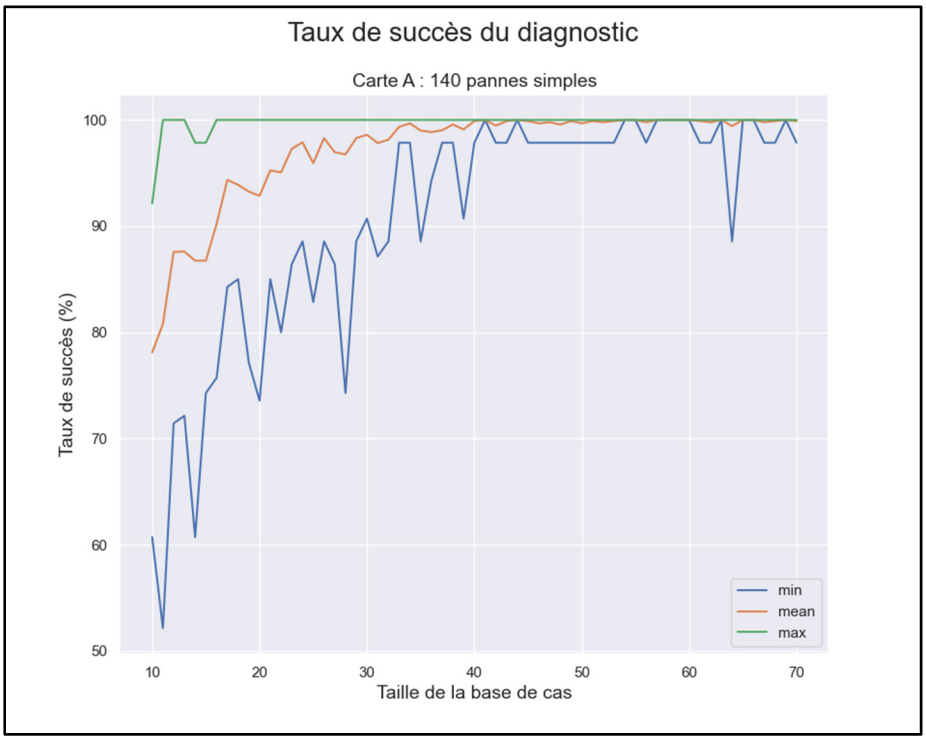


Figure 5.11 Taux de succès minimum, maximum et moyen du diagnostic selon la taille de la base de cas, carte A

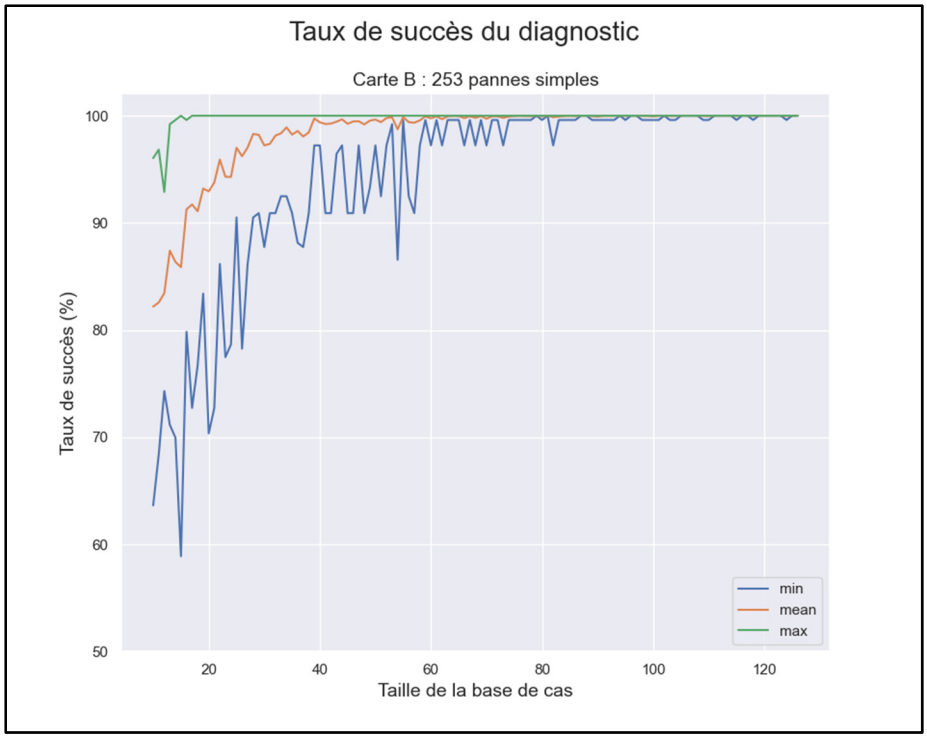


Figure 5.12 Taux de succès minimum, maximum et moyen du diagnostic selon la taille de la base de cas, carte B

Les premiers résultats de l'expérimentation décrite à la section précédente sont présentés aux figures 5.11 et 5.12, pour les cartes A et B, respectivement. Dans chacune de ces deux figures on retrouve trois courbes : les taux de succès maximum (*max*), moyen (*mean*) et minimum (*min*) obtenus pour chaque taille de base de cas et son jeu de 20 essais. On constate que le taux de succès moyen pour les deux cartes est constamment supérieur à 90% dès que la base de cas atteint une taille de 20 cas seulement. L'augmentation de la taille de la base de cas permet à la moyenne de tendre vers 100%. On constate également que pour les deux cartes, il existe au moins une configuration de base de cas capable de diagnostiquer correctement l'ensemble du dictionnaire de pannes avant même d'avoir atteint une taille de 20 cas.

5.9.1 Applicabilité de l'apprentissage obtenu sur un produit vers un autre

Pour poursuivre l'analogie entre le système de raisonnement par cas et l'approche de résolution de problème utilisée par le technicien chargé du diagnostic, il est nécessaire d'évaluer la

transférabilité des apprentissages acquis sur un produit vers un autre produit. Pour ce faire, nous avons procédé à une expérience similaire à celle décrite à la section 5.6, mais cette fois, nous avons présenté des requêtes concernant la carte B à un système dont la base de cas contient uniquement des cas se rapportant à la carte A, et vice-versa. Les résultats sont présentés aux figures 5.13 et 5.14.

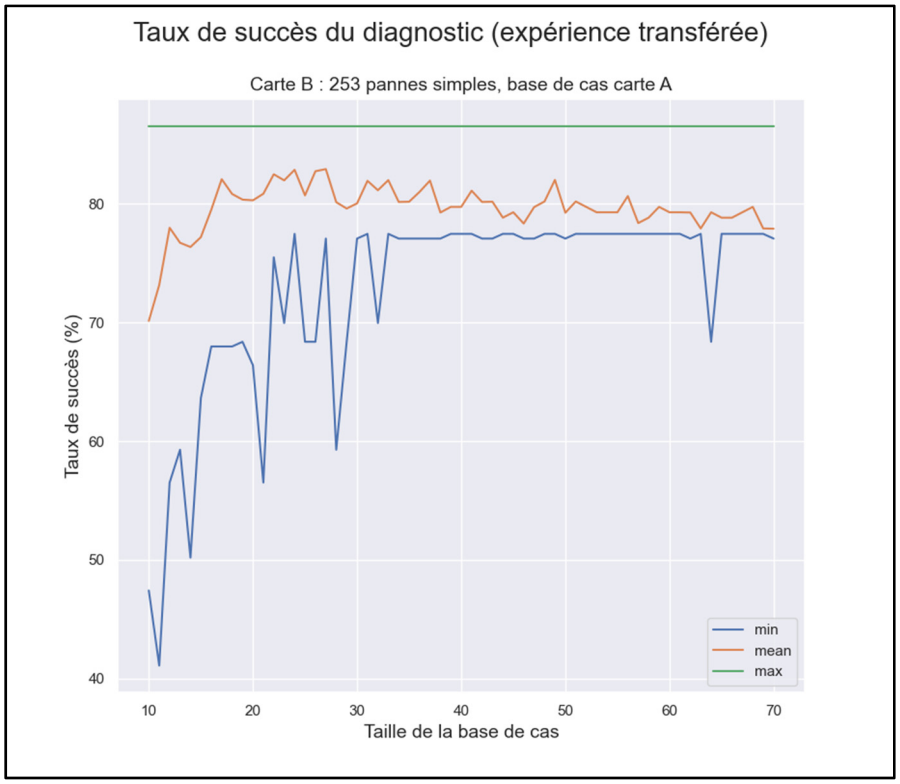


Figure 5.13 Taux de succès minimum, maximum et moyen du diagnostic selon la taille de la base de cas, carte B, avec expérience sur la carte A uniquement

Lorsque des requêtes concernant une carte complexe sont dirigées vers un système dont l'expérience a été acquise sur un produit très simple, on constate un plafonnement de la capacité de diagnostic. Dans ce cas précis, le plafond se situe à 219 pannes simples correctement diagnostiquées sur 253, ou 87%. Vu le manque de complexité des cas présents dans la base, le système est simplement incapable de reconnaître les patrons présents dans certaines requêtes. On constante aussi une reduction du taux de succès moyen pour les tailles

de bases de cas plus grandes alors que la prolifération de cas insuffisamment similaires, mais dont les scores sont très proches, augmente le risque d'erreur de classification.

Comme le niveau de complexité de la carte A constitue pratiquement le minimum possible sur une carte avec une chaîne JTAG fonctionnelle, la situation présentée à la figure 5.13 est un cas extrême. La transférabilité de l'apprentissage devrait donc être meilleure sur une production constituée d'un mélange de produits d'une complexité moyenne à élevée, typique de l'environnement de production *low-volume*, *high-mix* auquel le système est destiné. La figure 5.14 montre la situation inverse, soit des requêtes conçernant uniquement la carte A envoyé à un système dont la base ne contient que des cas conçernant la carte B.

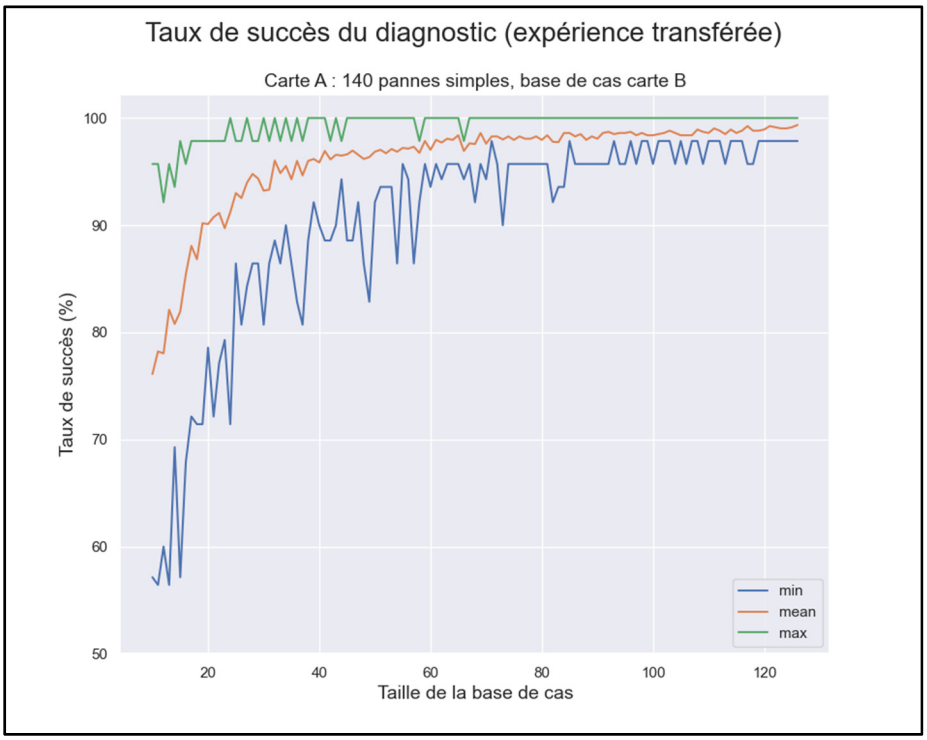


Figure 5.14 Taux de succès minimum, maximum et moyen du diagnostic selon la taille de la base de cas, carte A, avec expérience sur la carte B uniquement

On constate que dans le cas de requêtes concernant un produit très simple effectuées sur un système dont l'expérience a été acquise sur une carte plus complexe, la transférabilité est très

bonne et la courbe d'évolution de la moyenne présente une forme similaire aux courbes de la section 5.9, malgré une légère réduction de l'efficacité.

5.9.2 Taux de succès avec $k > 1$

Afin d'améliorer le taux de succès sur les bases de cas de très petite taille, il pourrait être intéressant de considérer plusieurs candidats plutôt qu'uniquement celui ayant obtenu le meilleur score. Il pourrait par exemple arriver pour certains cas particuliers que le plus proche voisin soit erroné alors que le bon diagnostic se retrouve dans les quatre ou cinq suivants. Afin d'évaluer la pertinence d'une telle approche, nous avons répété l'expérience de la figure 5.14 présentée à la section 5.9.1, en évaluant la présence d'un candidat au diagnostic correct parmi les *k* premiers, pour *k* variant de 1 à 5 (Figure 5.15).

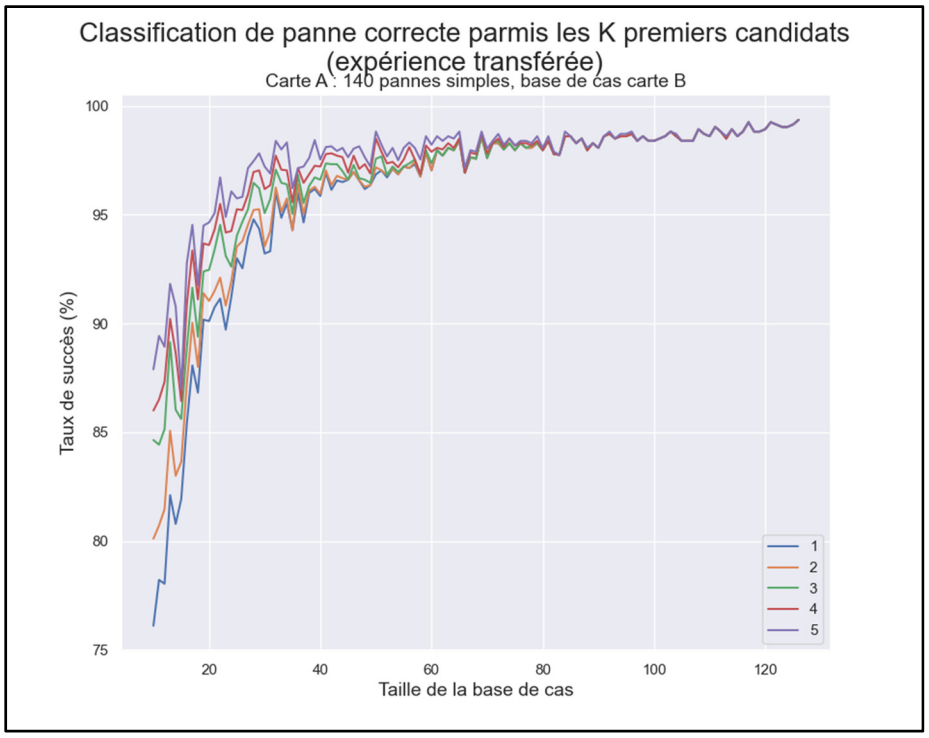


Figure 5.15 Présence du diagnostic correct parmi les *k* plus proches voisins selon la taille de la base de cas, carte A, avec expérience sur la carte B uniquement, pour *k* variant de 1 à 5

Pour les plus petites bases de cas, une majoration importante du taux de succès, pouvant atteindre 10%, est possible en augmentant la valeur de k jusqu'à 5. Cet avantage diminue graduellement à mesure que la taille de la base de cas augmente, jusqu'à disparaître entièrement. Lors de travaux futurs, un système de pointage pourrait être envisagé afin de tirer profit de cette situation, mais une autre innovation, l'utilisation de cas synthétiques pour augmenter la taille de la base de cas dès sa création, présentée en détail à la section 5.10 offre de meilleures perspectives et c'est celle que nous avons retenue.

5.10 Utilisation de cas synthétiques pour l'accélération du démarrage

Une des principales innovations du présent travail consiste à utiliser un système d'émulation matérielle (présenté au chapitre 3) afin de produire automatiquement un dictionnaire de pannes qui sera fourni dès le départ au système de raisonnement par cas. Puisqu'ils ne proviennent pas de défectuosités réelles observées sur le produit, nous qualifions ces cas de « synthétiques ». Ils sont utilisés pour les opérations de diagnostic, mais sont marqués afin d'être ignorés si nécessaire lors des opérations subséquentes d'exploitation des données (présentées au chapitre 6).

L'utilisation des cas synthétiques permet de court-circuiter la courbe de croissance du taux de succès observée à la section 5.9 pour le diagnostic des pannes simples. Puisque nous avons montré expérimentalement à la section 5.9 (figures 5.11 et 5.12) qu'il existe au moins une configuration de base de cas (obtenue sur la carte cible) capable de diagnostiquer correctement toutes les pannes simples dès que la taille dépasse 20 cas, il est acquis qu'une base de cas présentant toutes les pannes simples constitue une telle configuration. Nos essais ont permis de confirmer que le taux de succès du diagnostic des pannes simples est de 100% lorsque la base de cas contient le dictionnaire de panne entier pour la carte testée.

Mais qu'en est-il des pannes plus complexes? Il existe plusieurs scénarios réels où quelques nœuds sont touchés simultanément par une défectuosité dont la cause est commune. Un excès

de soudure connectant plusieurs broches voisines, ou à l'opposé un manque de soudure laissant plusieurs broches voisines non connectées constituent des exemples courants.

Afin d'évaluer la réponse du système proposé à ce genre de scénarios, nous avons construit un générateur qui utilise l'information de disposition physique des cartes décrite au chapitre 4 pour produire aléatoirement des requêtes présentant des défectuosités de même type touchant trois nœuds voisins. Nous avons soumis 1000 requêtes indépendantes touchant la carte B au système de raisonnement par cas équipé du dictionnaire de pannes complet. La figure 5.16 et le tableau 5.2 présentent les résultats obtenus.

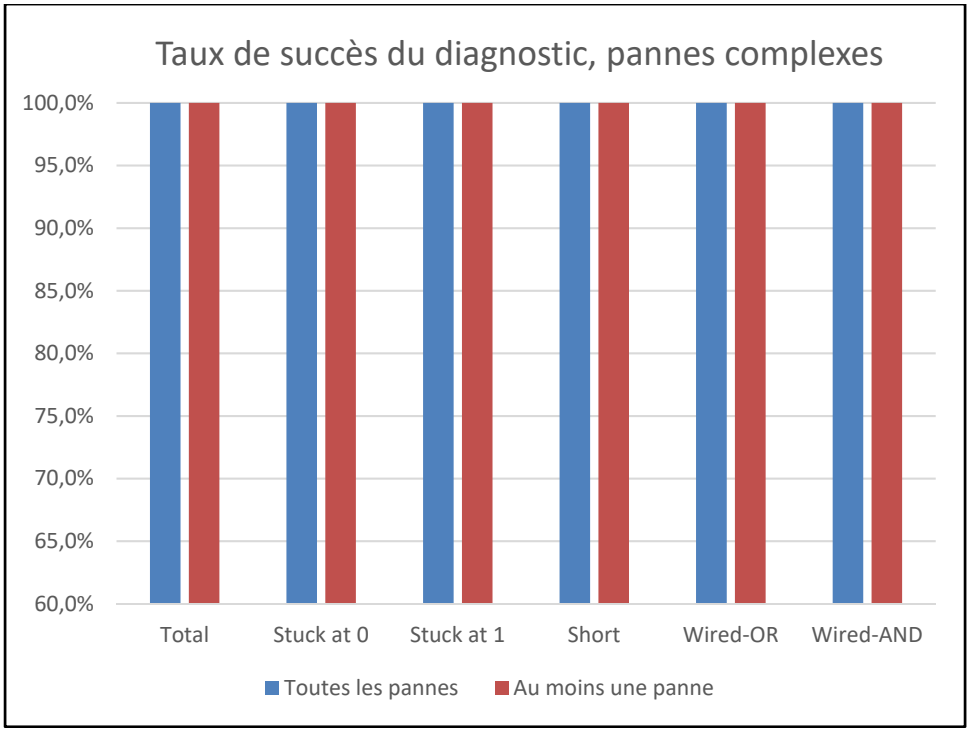


Figure 5.16 Taux de succès du diagnostic pour syndrome comportant une défectuosité touchant trois nœuds voisins, carte B

Tableau 5.2 Résultats pour des défectuosités touchant trois nœuds voisins, carte B

	Total	Stuck-at 0	Stuck-at 1	Court-circuit	Pont dominé par les 1	Pont dominé par les 0
Nombre d'essais	1000	179	202	208	208	203
Toutes les pannes, nombre	1000	179	202	208	208	203
Toutes les pannes, %	100.0	100.0	100.0	100.0	100.0	100.0
Au moins une panne, nombre	1000	179	202	208	208	203
Au moins une panne, %	100.0	100.0	100.0	100.0	100.0	100.0

Pour chacun de ces essais, le système raisonnant se trouve dans l'état où il se trouverait lors de la première soumission d'un cas réel, avec seulement les cas synthétiques en mémoire. Dans le cas des défectuosités de type *stuck-at*, trois nœuds doivent être correctement diagnostiqués. Dans le cas des courts-circuits et des ponts, deux nœuds seulement doivent être diagnostiqués puisque le nœud dominant produit les résultats de test attendus.

Même pour ces scénarios de défaillance plus complexe, la présence d'un dictionnaire de panne complet dans la mémoire du système raisonnant lui a permis de détecter, localiser et classer correctement toutes les pannes présentes, tel que présenté au tableau 5.2 et à la figure 5.16. Au tableau 5.2, on retrouve six colonnes de résultats, une pour l'ensemble des essais (Total), les cinq autres pour chacune des cinq classes de pannes (*stuck-at 0*, *stuck-at 1*, court-circuit, pont dominé par les uns et par les zéros). On y retrouve également cinq lignes de résultats. La première ligne (Nombre d'essais) donne le nombre d'essais effectués au total et pour chacune des cinq classes. La seconde (Toutes les pannes, nombre) et la troisième (Toutes les pannes, %) ligne donnent respectivement le nombre et le pourcentage de cas où toutes les pannes ont été correctement diagnostiquées. Finalement, la quatrième (Au moins une panne, nombre) et la cinquième (Au moins une panne, %) ligne donnent respectivement le nombre et le pourcentage de cas où au moins une panne a été correctement diagnostiquée. La figure 5.16, quant à elle, donne une représentation graphique de la troisième et de la cinquième lignes de résultats.

5.11 Comparaison avec l'outil commercial

Afin de comparer l'efficacité du système proposé avec notre outil commercial de référence, *JTAG Technologies Boundary-Scan Diagnostics* (BSD) présenté au chapitre 3, nous avons conçu la procédure de test suivante :

1. Utiliser le générateur de défectuosités multiples touchant 3 nœuds voisins présenté à la section 5.10 pour produire un ensemble de scénarios pour chacune des cartes;
2. Programmer l'émulateur de la carte A avec saboteurs sur le circuit FPGA;
3. Reproduire un scénario généré en 1 avec l'outil de contrôle des saboteurs;
4. Effectuer le test JTAG sur l'équipement de mesure *JTAG Technologies*;
5. Analyser les résultats du test et obtenir un diagnostic avec BSD;
6. Analyser les résultats du test et obtenir un diagnostic avec notre prototype;
7. Comparer les diagnostics et noter les résultats;
8. Désactiver tous les saboteurs;
9. Répéter les étapes 3 à 8 pour chacun des scénarios de la carte A;
10. Répéter les étapes 2 à 9 pour la carte B.

On considère un diagnostic comme réussi lorsque la détection, la localisation et la classification de la défectuosité sont valides. L'outil commercial BSD ne fait pas la différence entre les ponts directionnels et les courts-circuits. Nous avons considéré comme valide dans son cas un diagnostic de BRIDGE pour chacune de ces trois classes de pannes. Cette classification plus fine constitue un avantage qualitatif de notre méthode. Les résultats obtenus pour la carte A sont présentés à la figure 5.17 et au tableau 5.3. Les résultats obtenus pour la carte B sont présentés à la figure 5.18 et au tableau 5.4.

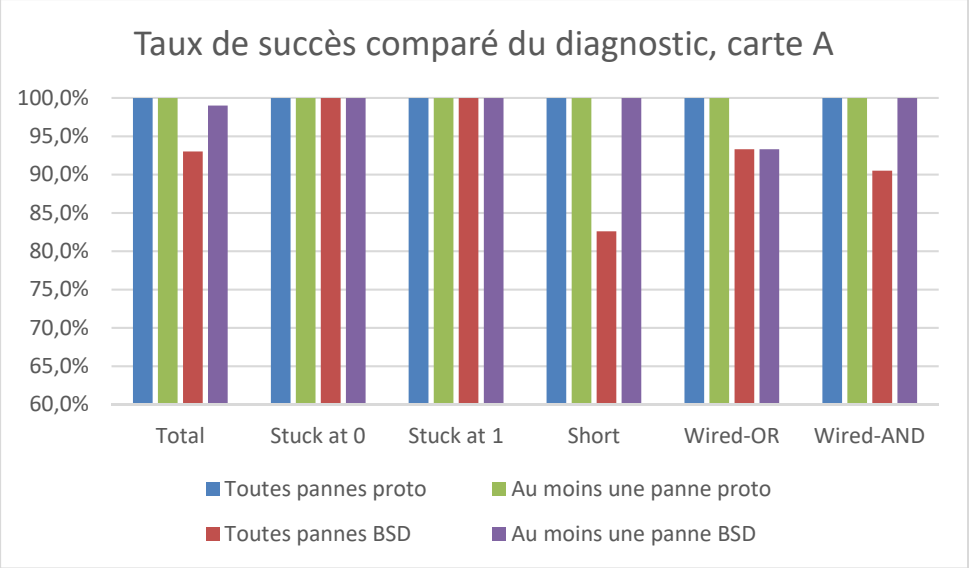


Figure 5.17 Taux de succès comparés du prototype et de BSD pour des défectuosités touchant 3 nœuds voisins, carte A

Tableau 5.3 Résultats comparatifs entre le prototype et BSD, carte A

	Total	Stuck-at 0	Stuck-at 1	Short	Wired-OR	Wired-AND
Nombre d'essais	100	17	24	23	15	21
Toutes les pannes proto, nombre	100	17	24	23	15	21
Toutes les pannes proto, %	100.0	100.0	100.0	100.0	100.0	100.0
Toutes les pannes BSD, nombre	93	17	24	19	14	19
Toutes les pannes BSD, %	93.0	100.0	100.0	82.6	93.3	90.5
Au moins une panne proto, nombre	100	17	24	23	15	21
Au moins une panne proto, %	100.0	100.0	100.0	100.0	100.0	100.0
Au moins une panne BSD, nombre	99	17	24	23	14	21
Au moins une panne BSD, %	100.0	100.0	100.0	100.0	93.3	100.0

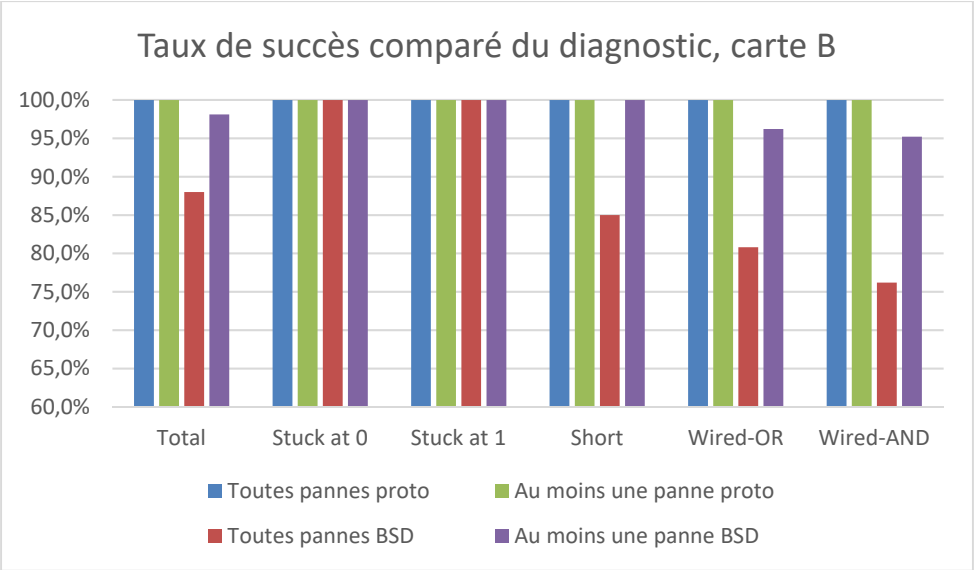


Figure 5.18 Taux de succès comparés du prototype et de BSD pour des défectuosités touchant 3 nœuds voisins, carte B

Tableau 5.4 Résultats comparatifs entre le prototype et BSD, carte B

	Total	Stuck-at 0	Stuck-at 1	Short	Wired-OR	Wired-AND
Nombre d'essais	108	23	18	20	26	21
Toutes les pannes proto, nombre	108	23	18	20	26	21
Toutes les pannes proto, %	100.0	100.0	100.0	100.0	100.0	100.0
Toutes les pannes BSD, nombre	95	23	18	17	21	16
Toutes les pannes BSD, %	88.0	100.0	100.0	85.0	80.8	76.2
Au moins une panne proto, nombre	108	23	18	20	26	21
Au moins une panne proto, %	100.0	100.0	100.0	100.0	100.0	100.0
Au moins une panne BSD, nombre	106	23	18	20	25	20
Au moins une panne BSD, %	98.1	100.0	100.0	100.0	96.2	95.2

Comme au tableau 5.2, on retrouve aux tableaux 5.3 et 5.4 les mêmes six colonnes de résultats. On y retrouve cependant ici neuf lignes de résultats. La première ligne (Nombre d'essais) donne encore le nombre d'essais effectués au total et pour chacune des cinq classes. La seconde (Toutes les pannes proto, nombre) et la troisième (Toutes les pannes proto, %) ligne donnent

respectivement le nombre et le pourcentage de cas où toutes les pannes ont été correctement diagnostiquées à l'aide de notre prototype. La troisième (Toutes les pannes BSD, nombre) et la quatrième (Toutes les pannes BSD, %) ligne donnent respectivement le nombre et le pourcentage de cas où toutes les pannes ont été correctement diagnostiquées à l'aide de l'outil BSD. La sixième (Au moins une panne proto, nombre) et la septième (Au moins une panne proto, %) ligne donnent respectivement le nombre et le pourcentage de cas où au moins une panne a été correctement diagnostiquée à l'aide de notre prototype. Finalement, la huitième (Au moins une panne BSD, nombre) et la neuvième (Au moins une panne BSD, %) ligne donnent respectivement le nombre et le pourcentage de cas où au moins une panne a été correctement diagnostiquée à l'aide de l'outil BSD. La figure 5.17, quant à elle, donne une représentation graphique des lignes de résultats numéro 3, 5, 7, et 9.

Le manuel de l'outil BSD mentionne en page 6 que les « *diagnostics for infrastructure, interconnect, memory clusters and Dot6 nets is based on a built-in algorithm* ». Les résultats montrent les limites de l'algorithme utilisé par BSD. L'outil commercial obtient un score parfait pour les pannes de type *stuck-at* mais a plus de difficulté avec les courts-circuits et les ponts directionnels, avec un taux de succès de (52/59=) 88% pour toutes les pannes de type *bridge* bien diagnostiquées. BSD est capable d'identifier correctement au moins une panne dans la très grande majorité des cas (58/59=) 98% pour les pannes de type *bridge* cependant, ce qui est déjà fort utile pour guider le technicien dans sa recherche. Notons finalement que notre prototype a encore obtenu un score parfait. Des résultats similaires ont été obtenus pour la carte B (tableau 5.4, figure 5.18), avec un score toujours parfait pour notre prototype pour l'ensemble des classes de pannes, un score encore parfait pour l'outil BSD pour les pannes de type *stuck-at*, qui affiche également un taux de succès de (54/67=) 81% pour toutes les pannes de type *bridge* bien diagnostiquées et une capacité d'identifier correctement au moins une panne dans (65/67=) 97% pour les pannes de type *bridge*.

Un outil de classification de nouvelle génération, basé sur la reconnaissance de patrons appuyée par l'apprentissage machine tel que nous le proposons, offre un potentiel supérieur pour le diagnostic des pannes complexes. L'utilisation des cas synthétiques nous permet de

réaliser une grande partie de ce potentiel très rapidement, ce qui rend viable l'utilisation d'un tel système dans un environnement à faible volume et haute complexité.

5.12 Conclusion

Dans ce chapitre, nous avons présenté la technique du raisonnement par cas ainsi que les paramètres retenus pour notre implémentation. Nous avons montré que le prototype développé est capable de produire une classification correcte et que ses performances s'améliorent avec l'expérience. Nous avons finalement montré la pertinence de notre technique d'accélération du démarrage à l'aide de cas synthétiques obtenus par émulation.

Les résultats sont concluants pour les cartes d'essais A et B et montrent que l'utilisation de cas synthétiques pour un système d'apprentissage machine en démarrage dans un environnement pauvre en données représente une avancée réelle par rapport à l'outils commercial de référence et qu'il serait pertinent d'adapter le système à un environnement commercial pour en mesurer les bénéfices réels.

Les outils traditionnels comme BSD se concentrent sur l'analyse d'une seule carte défaillante, mais un système raisonnant comme celui que nous proposons permet l'accumulation d'expérience de diagnostic sur plusieurs cartes dans un environnement de production donné et offre l'opportunité d'acquérir des métadonnées sur le processus manufacturier lui-même. L'exploitation de ces données pour l'émission de recommandations de réparations et la surveillance du processus manufacturier fera l'objet du prochain chapitre.

CHAPITRE 6

SYSTÈME RECOMMANDEUR POUR L'ANTICIPATION DES CAUSES PHYSIQUES DE DÉFECTUOSITÉ

6.1 Introduction

Le système de raisonnement par cas décrit au chapitre précédent est chargé de faire la classification des défectuosités détectées sur la carte testée, puis de présenter un diagnostic à l'opérateur. Ce diagnostic présente la ou les défectuosités trouvées en identifiant le nœud du circuit sur lequel chacun des problèmes est situé, la classe de défectuosités ainsi que le nœud partenaire dans le cas des courts-circuits et des ponts directionnels. L'opérateur doit ensuite procéder à l'identification de la cause physique de la défectuosité, puis procéder à sa remédiation.

Suite à cette étape du procédé, une simple rétroaction de l'opérateur, indiquant le composant exact impliqué ainsi que la nature du problème, permet d'augmenter grandement la portée et l'utilité potentielle du système proposé. Dans ce chapitre, nous présenterons les deux grands bénéfices de cette collecte de données, soit :

1. La suggestion de causes physiques probables lors du diagnostic et ;
2. La détection de défaillances ou de mauvais réglages sur l'équipement de production entraînant une hausse soudaine des défectuosités d'origine systémique.

6.2 Rétroaction et identification des causes physiques

Afin de permettre au système d'apprendre et de s'améliorer avec le temps, il est nécessaire de lui donner une rétroaction sur les réparations effectuées suite à l'opération de diagnostic. La rétroaction est entrée directement dans l'interface graphique du système de raisonnement par cas. L'opérateur sélectionne le cas pour lequel il désire effectuer la rétroaction puis choisit dans des menus déroulants le composant défaillant (qui peut être n'importe quelle pièce

présente sur le *Bill of Material* (BOM) ou encore la carte de circuit imprimé elle-même) puis la classe de cause physique de la défectuosité.

6.2.1 Causes physiques de défaillance

Comme le rapportent les auteurs de (Huang et al, 2019), la littérature scientifique est très pauvre en données à propos des causes de défectuosités découlant de la fabrication et de l'assemblage de circuits imprimés. En effet, ces informations sont généralement considérées comme faisant partie du secret industriel et leur publication n'est pas considérée comme apportant un bénéfice pour le manufacturier. Cependant (Meixner, 2020) rapporte les résultats d'une étude réalisée sur plus de 4 millions de produits, comportant plus de 100 000 défectuosités. Ces défectuosités ont été détectées à la fin de l'assemblage final des produits, malgré le recours accru à des technologies d'inspection sur la ligne de plus en plus sophistiquées. Le tableau 6.1 montre les dix catégories de défectuosités les plus courantes ainsi que leur prévalence parmi les produits finis testés.

Tableau 6.1 Catégories de défectuosités les plus courantes et leur prévalence
Adapté de Meixner (2020, p. 2)

Catégorie de défectuosités	**Prévalence parmi les produits testés**
Problèmes d'adhésifs	0.73%
Mousses manquantes	0.71%
Rubans et étiquettes	0.70%
Problèmes de soudure	0.45%
Problèmes de routage des conducteurs	0.36%
Pièce manquante (à l'exclusion des mousses et vis)	0.35%
Pièces mal alignées	0.18%
Connecteurs mal connectés	0.17%
Couches protectrices manquantes	0.13%
Contamination	0.11%

Puisque les défectuosités mentionnées dans l'étude ont été observées sur des produits complètement assemblés et prêts à livrer, les catégories du tableau 6.1 ne concernent pas toutes le domaine de l'assemblage de circuits imprimés sur lequel nous nous concentrons. Par

exemple, les problèmes de mousses, d'adhésifs et de rubans concernent plutôt l'assemblage du produit entier que le circuit imprimé. Nous avons cependant retenu les quatre catégories les plus pertinentes, soient les problèmes de pièces manquantes, d'alignement, de soudure et de routage. Puisqu'elles se manifestent différemment selon les classes de défectuosités que nous avons définies à la section 5.4, nous avons choisi de subdiviser deux catégories (problèmes de soudure et problèmes de routage des conducteurs) dans l'élaboration de notre liste de causes physiques. Le tableau 6.2 présente une liste des six classes de causes physiques actuellement reconnues par le système.

Tableau 6.2 Classes de causes physiques utilisées par le prototype

Classe de cause physique	**Définition**
Pièce manquante	La pièce en cause n'est pas présente sur la carte assemblée
Pièce mal alignée	La pièce est mal positionnée sur la carte après assemblage
Excès de soudure	Un excès de pâte pour ce composant cause une défaillance
Manque de soudure	Une quantité de pâte insuffisante pour ce composant cause une défaillance
Trace manquante	Une trace sur la carte est manquante ou brisée
Trace court-circuitée	Une trace sur la carte est connectée à une autre trace ou à une pastille

Le tableau 6.3 présente l'applicabilité des classes de cause physique du tableau 6.2, regroupées selon les catégories de l'étude rapportée par (Meixner, 2020) pour chacune des classes de défectuosités définies à la section 5.4. On peut y constater que les deux premières classes (stuck-at-0, stuck-at-1) sont applicables aux catégories « pièce manquante », « alignement », « manque de soudure » et « circuit ouvert de routage », alors que les trois dernières (court-circuit, point dominé par les zéros et par les uns) sont applicables aux catégories « alignement », « excès de soudure » et « circuit ouvert de routage ».

Tableau 6.3 Applicabilité des causes physiques aux classes de défectuosités

Classe de défectuosité	Pièce manquante	Alignement	Soudure		Routage	
			Excès	Manque	Circuit ouvert	Court-circuit
Stuck-at-0	Oui	Oui	Non	Oui	Oui	Non
Stuck-at-1	Oui	Oui	Non	Oui	Oui	Non
Court-circuit	Non	Oui	Oui	Non	Non	Oui
Pont dominé par 0	Non	Oui	Oui	Non	Non	Oui
Pont dominé par 1	Non	Oui	Oui	Non	Non	Oui

6.3 Intégration des données de production et de réparation

Pour réaliser le diagnostic, il n'était pas nécessaire de conserver les informations d'identification exacte du produit défectueux, ces données n'étaient donc pas présentes dans la base de cas. Nous en aurons cependant besoin pour établir des corrélations permettant d'identifier des causes communes de défectuosité entre les produits. C'est pourquoi nous ajouterons au *Case*, une référence vers un objet de type *ProductionInfo*, montré sur la figure 6.1.

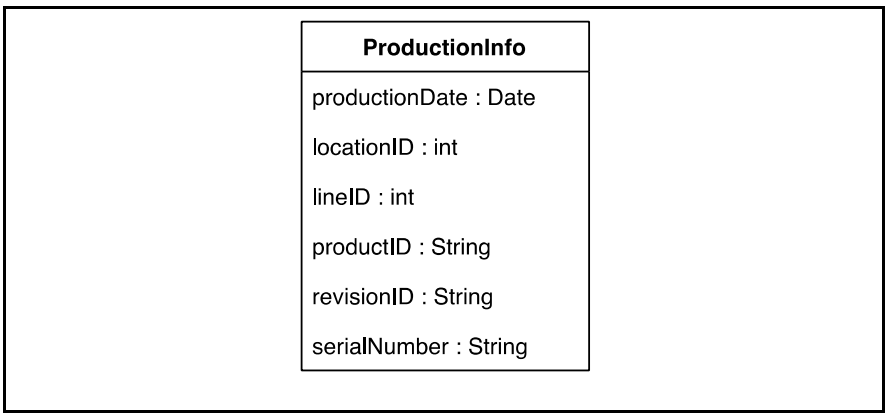


Figure 6.1 Données encapsulées par l'objet de type *ProductionInfo*

Cet objet contient la date de production, le numéro d'usine ou de site (*locationID*), le numéro de ligne de production (*lineID*), l'identifiant du produit et de sa révision exacte ainsi que le numéro de série du produit défectueux.

C’est également dans l’objet *Case* que nous conserverons la rétroaction de l’opérateur concernant la ou les réparations effectuée(s) sur le produit, qui seront encapsulées dans des objets de type *RepairInfo*, montrés sur la figure 6.2 et associant la classe de cause physique choisie dans le tableau 6.2 à l’identificateur (*designator*) du composant fautif.

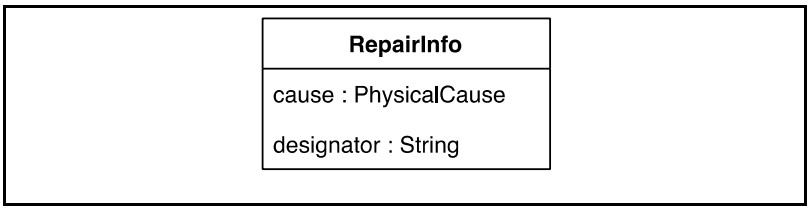


Figure 6.2 Données encapsulées par l'objet de type *RepairInfo*

6.4 Anticipation des causes physiques lors du diagnostic

Jusqu’à maintenant, le système CBR jouait un rôle de classificateur. La méthode des cas synthétiques présentée au chapitre 5 a permis d’accélérer sa phase d’apprentissage jusqu’à obtenir une classification sans faute, même en présence de défectuosité multiples. Les capacités d’apprentissage machine à plus longue échéance y étaient cependant négligées, fautes de données. Mais comme les données de production et de rétroaction font maintenant partie de la base de cas, il est possible de les exploiter directement dans l’engin diagnostic. Nous proposons d’ajouter au rôle de classificateur celui de système recommandeur.

6.4.1 Systèmes recommandeurs

Les systèmes recommandeurs sont une des applications de filtrage de l’information les plus courantes. Ils ont pour mission de proposer à un utilisateur un bien, un service ou un élément d’information qui soit d’intérêt pour lui. Ils sont très visibles dans la vie courante dans le domaine du commerce électronique, mais aussi sur les plateformes de diffusion en continu (Netfllix, Spotify, Youtube, etc.) et sur les réseaux sociaux.

Il existe plusieurs façons de les concevoir selon le volume d’information disponible. La plupart des systèmes commerciaux sont des hybrides. Dans le cas de Netflix par exemple, les

recommandations proviennent d'une combinaison des habitudes de recherche et d'écoute des utilisateurs comparables (une approche appelée filtrage collaboratif) ainsi que des caractéristiques des films évalués positivement par l'utilisateur (approche dite du filtrage basé sur le contenu). (Gomez-Uribe et Hunt, 2015).

6.4.2 Raisonnement par cas et fonction de similarité

Dans notre environnement généralement pauvre en données, nous allons utiliser une forme de filtrage basé sur le contenu, qui est bien adapté au CBR. Nous utiliserons la même base de cas qu'à l'étape de classification, mais avec une fonction de similarité complètement différente.

L'objectif général, cette fois, sera de trouver parmi tous les cas non synthétiques de la base (les cas synthétiques n'ayant pas de cause physique), le cas où le plus grand nombre de caractéristiques d'intérêt sera équivalent. Les caractéristiques d'intérêt seront :

- La classe de défectuosité
- Le nœud défectueux
- La liste des nœuds voisins
- L'identifiant de produit
- L'identifiant d'usine
- L'identifiant de ligne de production
- La date de production

Le cas idéal (100% similaire) étant celui présentant la même défectuosité, sur le même nœud, sur le même produit, fabriqué dans la même usine, sur la même ligne, juste avant le produit en cours de diagnostic.

Puisqu'il sera basé sur l'évaluation des caractéristiques d'intérêt, notre système recommandeur constituera une application du filtrage basé sur le contenu (Gomez-Uribe et Hunt, 2015). De manière plus spécifique, nous l'utiliserons de deux manières différentes : 1) pour suggérer des causes physiques lors du diagnostic, et 2) pour détecter les bris sur une ligne de production.

6.4.3 Application 1 : Suggestion de causes physiques lors du diagnostic

Le système de diagnostic présenté au chapitre 5 permet déjà la détection, la localisation et la classification des défectuosités présentes sur la carte. Afin de maximiser l'utilité de l'outil pour le technicien chargé de la réparation des cartes, nous proposons d'utiliser le système recommandeur pour suggérer les causes physiques les plus probables pour chacune des défectuosités présentes sur une carte. Une rétroaction du technicien après la réparation permet d'évaluer si la cause physique réelle fait effectivement partie des recommandations présentées, ce qui sera considéré comme un succès pour le système recommandeur. L'apprentissage machine se manifeste par une augmentation graduelle du taux de succès des recommandations à mesure que la taille de la base de cas augmente, jusqu'à atteindre un niveau relativement stable. Par analogie, on peut considérer que cette période d'apprentissage représente l'expérience nécessaire pour que le système intègre les particularités du produit et de la ligne de production, ainsi que de leurs interactions.

6.4.4 Application 2 : Détection de bris sur une ligne de production

La présence d'une région de stabilité relative dans la courbe du taux de succès de la suggestion des causes physiques pour un produit et une ligne de production donnée peut être exploitée à son tour pour l'élaboration d'une deuxième application, cette fois destinée à l'équipe de gestion plutôt qu'au technicien. En effet, toute apparition soudaine d'un bris ou d'un mauvais réglage sur la ligne de production ayant pour effet de favoriser grandement un type de cause physique plutôt qu'un autre entraînera un changement rapide du taux de succès du système recommandeur pour cette ligne. Il sera également possible de localiser l'apparition du bris dans le temps par l'observation du point d'inflexion de la courbe du taux de succès.

6.5 Simulateur de cas

Afin d'évaluer la faisabilité et la performance des applications proposées, il est nécessaire de soumettre au système un grand nombre de requêtes correspondant à des produits défaillants. Or, en l'absence de données de production réelles, nous avons dû recourir à la simulation.

L'utilisation d'un simulateur développé à cette fin nous permettra également de faire varier certains paramètres pour modéliser différentes hypothèses quant à la distribution des causes physiques parmi les défectuosités rencontrées. Le simulateur est décrit dans ce qui suit, en fonction des aspects suivants :

- la nature des défectuosités (aléatoires versus systématiques),
- la modélisation des particularités d'un produit,
- la modélisation des particularités d'une ligne de production,
- l'apparition des défectuosités en grappe,
- l'architecture logicielle du simulateur, et
- le fichier de configuration du simulateur.

6.5.1 Défectuosités aléatoires vs systématiques

Bien que toutes les défectuosités observées soient ultimement dues à une cause ou à un ensemble de causes physiques, dites « causes racines » (*root causes*), il n'est pas toujours possible de les déterminer avec certitude.

L'objectif de ce projet étant d'améliorer le rendement dans un environnement de production à faible volume et haute complexité, nous nous intéressons particulièrement aux défectuosités causées par l'interaction entre les caractéristiques du produit et les capacités de l'équipement de production. Dans notre étude, nous qualifierons les défectuosités causées par cette interaction de systématiques (c'est-à-dire engendrées par le système) alors que les autres défectuosités seront considérées comme aléatoires.

Afin de modéliser les défectuosités aléatoires, nous avons établi pour notre simulateur une distribution de base des probabilités d'occurrence des six causes physiques pour chacune des cinq classes de défectuosités identifiées par l'engin diagnostic décrit au chapitre 5. Cette distribution est basée sur les données de l'étude présentée à la section 6.2. Certaines causes physiques ne s'appliquent pas à toutes les classes de défectuosités. Par exemple, une pièce manquante n'engendrera pas de court-circuit détectable par *boundary-scan* mais plutôt un

problème de type *stuck-at*. Afin d'établir la distribution, nous avons réparti les classes de causes physiques proportionnellement selon les données du tableau 6.1 pour les catégories de causes physiques applicables à une classe de défectuosité donnée.

Par exemple, comme mentionné précédemment selon le tableau 6.2, dans le cas des défectuosités de type *stuck-at* (0 ou 1), seules les catégories « Problèmes de soudure », « Problèmes de routage des conducteurs », « Pièce manquante » et « Pièces mal alignées » du tableau 6.1 sont applicables. Elles se manifestent respectivement par les classes de cause physique « Manque de soudure », « Trace manquante », « Pièce manquante » et « Pièce mal alignée » du tableau 6.3. Le total des taux d'occurrence pour ces catégories dans le tableau 6.1 est de 0.0045 + 0.0036 + 0.0035 + 0.0018 = 0.0134. Le poids relatif de la classe de cause physique « Manque de soudure » pour la classe de défectuosité « *stuck-at* » est donc de 0.0045 / 0.0134 ou environ 34%. Le tableau 6.4 présente la distribution de base, modélisant les défectuosités aléatoires.

Tableau 6.4 Distribution de base des causes physiques par classe de défectuosité

Classe de défectuosité	**Pièce manquante**	**Alignement**	**Soudure**		**Routage**	
			Excès	**Manque**	**Circuit ouvert**	**Court-circuit**
Stuck-at-0	26%	13%	0%	34%	27%	0%
Stuck-at-1	26%	13%	0%	34%	27%	0%
Court-circuit	0%	19%	45%	0%	0%	36%
Pont dominé par 0	0%	19%	45%	0%	0%	36%
Pont dominé par 1	0%	19%	45%	0%	0%	36%

Afin de modéliser les défectuosités systématiques, cette distribution de base sera ensuite modifiée pour tenir compte des particularités des produits et des lignes de production.

6.5.2 Modélisation des particularités d'un produit

Afin de représenter les vulnérabilités particulières d'un produit face à certaines classes de défectuosités ou à certaines causes physiques, notre simulateur permet de définir pour chaque produit un ensemble de facteurs qui serviront de multiplicateurs pour les probabilités

d'occurrence de certains évènements. Il s'agit d'une représentation abstraite, basée sur l'observation des caractéristiques du produit. Par exemple, un produit intégrant plusieurs composants comportant un grand nombre de broches très rapprochées sera généralement plus susceptible aux défectuosités de type courts-circuits ou ponts directionnels causées par un excès de soudure.

Bien que nous ayons utilisé une approche empirique pour la détermination des vulnérabilités particulières de nos cartes d'essai, il existe une synergie potentielle entre notre outil de simulation et les outils commerciaux d'évaluation du design et de prévision du rendement manufacturier comme Valor de Siemens. Le rapport d'évaluation de Valor pourrait par exemple être utilisé pour identifier les vulnérabilités d'un produit et faire l'attribution des modificateurs. Dans un contexte manufacturier réel, les vulnérabilités et leurs niveaux de risque pourraient être extraits directement des données de l'historique des réparations.

6.5.3 Modélisation des particularités d'une ligne de production

De la même façon, notre simulateur permet d'attribuer des facteurs multiplicatifs aux probabilités d'occurrence des classes de défectuosité et des causes physiques reliées aux équipements constituant une ligne de production. Cette abstraction représente l'ensemble des tolérances de fabrication et d'ajustement des équipements. Il définit en quelque sorte la « personnalité » d'une ligne de production. Par exemple, une ligne dont certains paramètres de l'applicateur de pâte à souder seront très près des limites maximales acceptables aura tendance à enregistrer une plus grande proportion de défectuosités causées par un excès de soudure. Cette tendance pourra être observée sur tous les produits assemblés sur cette ligne. En présence de données réelles, il serait possible d'extraire un profil pour chacune des lignes de production à partir de l'historique des réparations.

6.5.4 Phénomène d'apparition en grappes des défectuosités

Puisqu'elles reposent sur des propriétés de l'équipement et des produits, les défectuosités systématiques ont tendance à exhiber une certaine localité temporelle et spatiale. À titre

d'exemple, un élément de conception aux limites des tolérances manufacturières sur un produit aura tendance à augmenter les probabilités de défaillance autour de cet élément (localité spatiale). De façon similaire, une dégradation dans les paramètres d'opération d'une ligne de production augmentera le risque d'apparition des défectuosités sensibles à ces paramètres jusqu'à la rectification du problème (localité temporelle).
Dans notre simulateur ce phénomène sera représenté par un « facteur d'apparition en grappe », sous forme de probabilité en pourcentage, pour chaque ligne de production.

6.5.5 Fonctionnement du simulateur

De façon générale, le simulateur de production développé pour la validation du système recommandeur opère en suivant les étapes suivantes :

1. On génère un produit.
2. On sélectionne dans le dictionnaire de panne un syndrome pour ce produit.
3. On sélectionne une cause racine à l'origine du syndrome choisi en 2.
4. On soumet le syndrome comme requête au système de diagnostic.
5. On vérifie que le diagnostic est correct (détection, localisation, classification).
6. On compare les k meilleurs candidats de cause physique du système recommandeur avec la cause racine sélectionnée en 3.
    a. Si la cause sélectionnée en 2 fait partie des candidats, on comptabilise un succès.
    b. Dans le cas contraire on comptabilise un échec.

Un peu plus en détails, pour chacune des trois premières étapes, le simulateur doit déterminer certains facteurs de façon pseudo-aléatoire en intégrant des éléments configurables regroupés dans des fichiers de configuration pour chaque combinaison ligne-produit, décrits à la section 6.5.7.

Nous avons partitionné l'implémentation du simulateur en plusieurs classes, chargées respectivement de la génération des informations de produit, du syndrome et de la cause racine.

Les causes racines seront stockées dans la base de cas comme des données de rétroaction post-réparation telles que décrites dans la section 6.3. Le module chargé de les générer est donc appelé « générateur de réparation ». La section suivante présente de façon plus détaillée l'architecture logicielle du simulateur ainsi que les responsabilités des différentes classes qui le composent.

6.5.6 Architecture logicielle

Le simulateur de produit défectueux est représenté par une instance de la classe *ProductionSimulator*. Le simulateur contient une liste de *CaseGenerators*, chacun capable de générer à la demande des cas complets (information de produit, syndrome, diagnostic et information de réparation) pour un type de produit sur une ligne de production donnée. La figure 6.3 présente les relations entre les principales composantes du simulateur. Les paragraphes suivant décrivent le fonctionnement de chacun des modules.

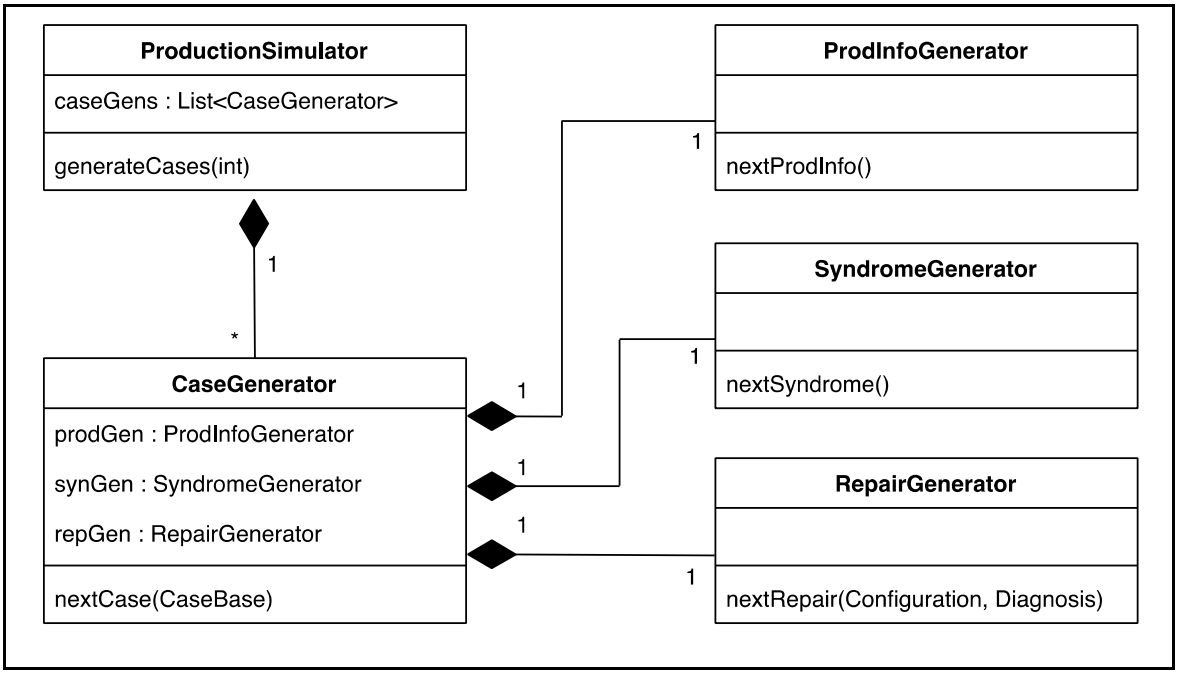


Figure 6.3 Relations entre les classes du simulateur de produit défectueux

ProdInfoGenerator

Cet objet est utilisé pour générer les données de production d'une unité défectueuse d'un seul produit, sur une ligne de production donnée. Ce sont des informations telles que son numéro de série et sa date de production (encapsulés dans un objet de type *ProductionInfo*). Les données générées qui possèdent une composante aléatoire sont contrôlées par des paramètres du fichier de configuration, décrits à la section 6.5.7.

Par exemple, pour générer la date de production, l'objet part de la dernière date de production générée (ou d'une date de départ tirée du fichier de configuration dans le cas du premier produit) puis ajoute un nombre aléatoire de secondes borné par un paramètre de configuration. Une procédure similaire est utilisée pour les numéros de série. De cette façon, les dates et les numéros de série générés seront toujours en ordre croissant même si le temps écoulé et la différence de numéro de série entre deux produits défectueux sont aléatoires.

SyndromeGenerator

Cet objet est chargé de générer un *Syndrome* en sélectionnant un élément du dictionnaire de panne. Le générateur commence par déterminer si la défectuosité à générer fait partie d'une grappe, selon la probabilité d'occurrence établie. Si le *Syndrome* fait partie d'une grappe, la classe de défectuosité sera identique au dernier *Syndrome* généré et le nœud défectueux sera le même que précédemment ou l'un de ses voisins. Si le *Syndrome* à générer ne fait pas partie d'une grappe, l'élément du dictionnaire de panne est sélectionné au hasard en respectant les probabilités d'occurrences des différentes classes de défectuosités, probabilités qui tiennent compte des particularités du produit et de la ligne de production.

RepairGenerator

Cet objet simule la rétroaction du technicien. Il détermine d'abord si l'objet *Diagnosis* reçu concerne une défectuosité faisant partie d'une grappe en consultant une variable d'état. Si la défectuosité fait effectivement partie d'une grappe, un objet *RepairInfo* avec la même cause physique que le précédent est généré. Si la défectuosité ne fait pas partie d'une grappe, une cause physique est choisie aléatoirement en tenant compte des probabilités de base présentées

au tableau 6.4, modifiées par les facteurs représentant les particularités du produit et de la ligne de production.

CaseGenerator

L'objet de type *CaseGenerator* représente une ligne de production pour un produit. Il charge les données de configuration à partir du fichier reçu du *ProductionSimulator*. Une fois les paramètres de configuration chargés, le *CaseGenerator* les utilise pour créer les trois générateurs nécessaires à la production d'un cas complet pour la simulation.

Lors d'un appel de sa méthode *nextCase*(), le *CaseGenerator* obtient d'abord un nouveau *Syndrome* du *SyndromeGenerator* qui lui est attaché, puis utilise l'engin de CBR décrit au chapitre 5 pour produire le *Diagnosis* correspondant. Il obtient ensuite un objet *ProductionInfo* de son *ProdInfoGenerator* puis encapsule les trois objets (*Syndrome*, *Diagnosis* et *ProductionInfo*) dans un objet de type *Case*. Il utilise enfin le générateur de *RepairInfo* pour ajouter la cause physique au *Case*, puis retourne l'objet *Case* complet au *ProductionSimulator*. Le diagramme de séquence UML à la figure 6.4 illustre la séquence d'appel.

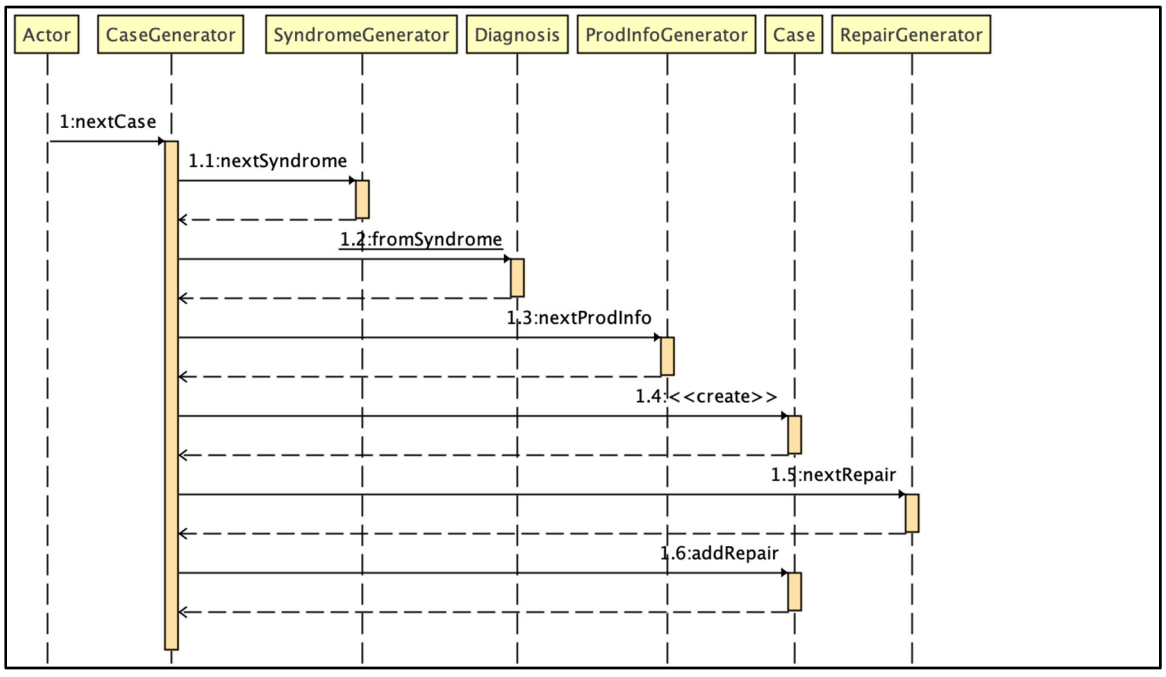


Figure 6.4 Séquence d'appels nécessaires à la création d'un objet *Case* complet

ProductionSimulator

Il s'agit du simulateur lui-même. Il est responsable de créer pour chaque combinaison de ligne de production et de produit, les objets *CaseGenerator* appropriés en leur fournissant un fichier de configuration, puis de lancer la simulation des cas, de recueillir les résultats et d'en faire l'analyse. À noter que seuls les produits défectueux sont simulés.

6.5.7 Fichiers de configuration

Les paramètres de simulation ainsi que les particularités de chacune des combinaisons de ligne de production et produit sont stockés dans des fichiers de configuration au format standard JSON. Il s'agit d'un format textuel simple basé sur des paires « clé : valeur » présentant le nom d'une variable suivi de sa valeur séparée par un symbole « : ». La figure 6.5 montre le contenu d'un fichier de configuration pour la carte B, sur la ligne 2.

```
{
  "sourcePath" : "./data/sig_demoboard/",
  "configFile" : “./data/JTAG_DemoBoard_netlist.txt",
  "odbPath" : "./data/odb_jtagdemo/",
  "prodMaxDeltaSeconds" : 86400,
  "serialStub" : "B22359-G641_",
  "maxSerialDelta" : 20,
  "serialStart" : 1000,
  "lineID" : 2,
  "productID" : "Board_B",
  "lineClassMatrix" : [ 1.0, 1.0, 8.0, 8.0, 8.0 ],
  "boardClassMatrix" : [ 1.0, 1.0, 8.0, 8.0, 8.0 ],
  "linePhysMatrix" : [
   [ 1.0, 1.0, 1.0, 1.0, 1.0, 1.0 ],
   [ 1.0, 1.0, 1.0, 1.0, 1.0, 1.0 ],
   [ 1.0, 1.0, 8.0, 1.0, 1.0, 1.0 ],
   [ 1.0, 1.0, 8.0, 1.0, 1.0, 1.0 ],
   [ 1.0, 1.0, 8.0, 1.0, 1.0, 1.0 ]
                    ],
  "boardPhysMatrix" : [
    [ 1.0, 1.0, 1.0, 1.0, 1.0, 1.0 ],
    [ 1.0, 1.0, 1.0, 1.0, 1.0, 1.0 ],
    [ 1.0, 1.0, 8.0, 1.0, 1.0, 8.0 ],
    [ 1.0, 1.0, 8.0, 1.0, 1.0, 8.0 ],
    [ 1.0, 1.0, 8.0, 1.0, 1.0, 8.0 ]
  ]
}
```

Figure 6.5 Exemple de fichier de configuration pour une combinaison ligne-produit

Les chemins de données *sourcePath*, *configFile* et *odbPath* indiquent respectivement l'emplacement des fichiers du dictionnaire de panne, de la liste des voisins pour chacun des nœuds accessibles par JTAG et des données de disposition physiques ODB++. Ils sont utilisés pour le contrôle du *SyndromeGenerator*.

Les six champs suivants contrôlent le fonctionnement du *RepairInfoGenerator*. Le champ *prodMaxDeltaSeconds* indique le nombre de secondes maximum entre deux produits défectueux (ici équivalent de 24 heures). *serialStub* contient la portion statique du numéro de série pour ce produit alors que *serialStart* contient le point de départ de la portion variable et que *maxSerialDelta* indique la différence maximale entre les numéros de série de deux produits défectueux consécutifs. *lineID* et *productID* sont des constantes utilisées pour identifier la ligne de production et le produit.

Les quatre derniers champs contiennent les vecteurs et matrices qui déterminent les vulnérabilités particulières de la ligne et du produit. Les champs *lineClassMatrix* et *boardClassMatrix* sont utilisés pour influencer les probabilités d'occurrences des différentes classes de défectuosités dans le *SyndromeGenerator* tandis que les champs *linePhysMatrix* et boardPhysMatrix jouent un rôle analogue pour le *RepairGenerator*. Les matrices 2D montrent les facteurs dans le même ordre que le tableau 6.4. Le scénario modélisé par le fichier montré est celui d'une carte comportant plusieurs composants dont les broches sont très rapprochées (et donc particulièrement sensible aux excès de soudure) fabriquée sur une ligne ayant tendance à produire elle-même de tels excès.

6.6 Expérimentation

Afin d'évaluer la performance du système recommandeur, nous avons procédé à plusieurs campagnes d'essais à l'aide du simulateur décrit à la section précédente. Pour toutes les campagnes, nous avons mesuré le taux de succès du système recommandeur pour deux lignes de production et deux produits, représentant quatre scénarios possibles d'interactions entre les particularités d'un produit et d'une ligne. Le tableau 6.5 présente les particularités des lignes

et des produits (cartes de circuits imprimés). Les particularités retenues sont délibérément simples afin de mettre en évidence les interactions et leurs effets sur le système recommandeur dans les résultats.

Tableau 6.5 Particularité des lignes de production et des produits pour la simulation

Élément	Particularité	Effet
Ligne 1	Configurée près des limites inférieures des paramètres d'application de soudure	Tendance accrue aux classes de défectuosité *Stuck-at-0* et *Stuck-at-1* pour cause de manque de soudure
Ligne 2	Configurée près des limites supérieures des paramètres d'application de soudure	Tendance accrue aux classes de défectuosité couvrant les courts-circuits et les ponts causés par un excès de soudure
Carte A	Présence de gros connecteurs	Sensibilité accrue au manque de soudure
Carte B	Présence de petites traces et de composants avec des broches très rapprochées	Sensibilité accrue à la qualité du PCB et aux excès de soudure

Tel que mentionné à la section 6.5.5, nous considérerons une utilisation du système recommandeur comme un succès si la cause physique simulée fait partie des recommandations. Puisque les recommandations sont déterminées par la méthode des K plus proches voisins, nous évaluerons le taux de succès pour K = 1 et pour K = 2, ainsi que l'effet du choix de K pour les deux applications proposées (suggestion au technicien et détection de bris sur la ligne de production).

Afin de montrer l'effet de l'apparition en grappe des défectuosités, nous considérerons le scénario où cet effet n'est pas présent (taux d'apparition de 0%) et celui où une ligne de production fonctionnant correctement est sujette à un taux d'apparition en grappe de 20%. Nous simulerons l'apparition soudaine d'un bris sur une ligne par le passage à 80% de probabilité d'apparition en grappe sur cette ligne au moment du bris.

6.7 Résultats

6.7.1 Conditions de test

Les figures des sous-sections suivantes illustrent l'évolution de la probabilité que le cas le plus similaire présente la bonne cause physique (K = 1) ou que la cause physique recherchée se trouve parmi les deux meilleurs candidats (K = 2), sur une série de 2000 cas simulés. Pour chacun des scénarios, la base de cas contient uniquement les cas synthétiques (utilisés pour l'entraînement du classificateur) au départ. La base de cas est ensuite enrichie par chacun des 2000 cas simulés en séquence. Dans chacun des scénarios, deux figures seront présentées, soit l'évolution du taux de succès pour le système entier et l'évolution du taux de succès pour chacune des combinaisons ligne-produit.

6.7.2 K = 1 et 0% de probabilité d'apparition en grappe

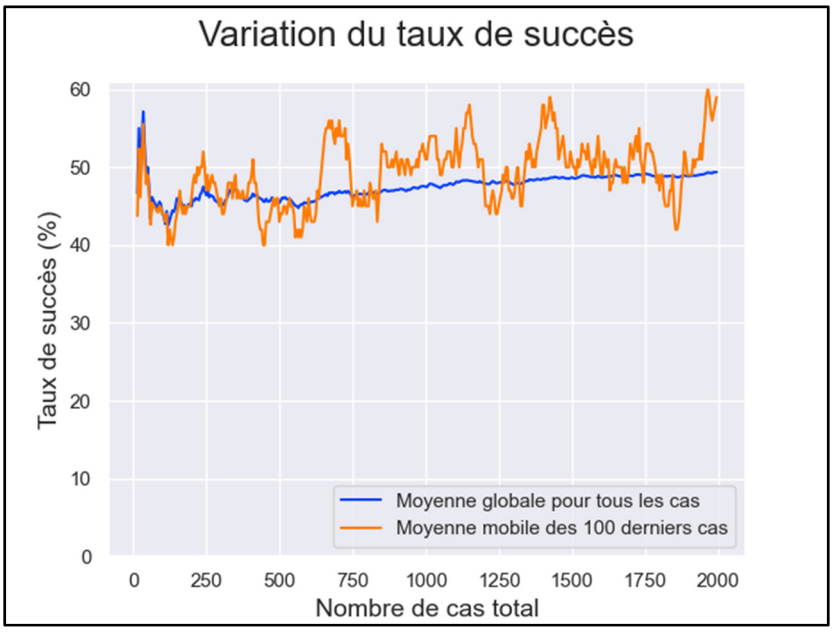


Figure 6.6 Taux de succès avec K = 1 sans apparition en grappe

Puisqu'en plus de reposer sur une recommandation unique, il n'y a aucune simulation du phénomène d'apparition en grappe des défectuosités, ces conditions constituent le scénario le

plus difficile pour notre système. En effet, les défectuosités générées sont complètement indépendantes ce qui rend caduque la composante temporelle du calcul de similarité en plus de diminuer l'importance des composantes spatiales (numéro de ligne, nœud affecté, nœuds voisins).

La figure 6.6 montre l'évolution du taux de succès sur 2000 cas. Au départ, lorsque la base de cas est presque vide, le taux de succès est surtout affaire de chance et varie beaucoup. Cette courte phase de démarrage est suivie d'une phase d'apprentissage où le taux de succès augmente progressivement jusqu'à atteindre une zone de relative stabilité.

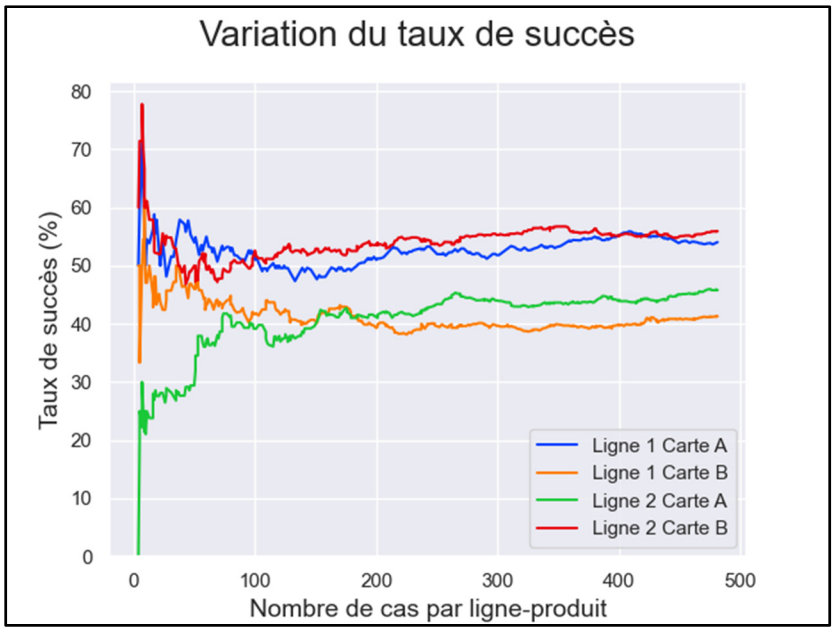


Figure 6.7 Taux de succès par ligne-produit avec
K = 1 sans apparition en grappe

La figure 6.7 montre le détail des résultats pour chacune des combinaisons ligne-produit. L'effet aléatoire des premiers cas est plus marqué et détermine la direction originale des courbes, mais on peut remarquer qu'à partir de 100 cas par ligne-produit environ, la tendance générale des courbes est à la hausse, alors que le système recommandeur intègre les particularités de chaque combinaison ligne-produit.

Puisqu'il y a une seule base de cas, l'apprentissage est effectué globalement. Il est donc normal de retrouver des régions de taux de succès à la baisse sur certaines combinaisons ligne-produit à mesure que le système apprend à les discriminer.

Bien que les taux de succès observés pour cette configuration soient relativement bas, ils sont tout de même supérieurs à une sélection aléatoire parmi les 6 causes physiques, dont le taux de succès moyen tendrait vers 16,67%.

La configuration suivante tient compte du phénomène d'apparition en grappe des défectuosités pour en montrer l'effet sur le système recommandeur.

6.7.3 K = 1 et 20% de probabilité d'apparition en grappe

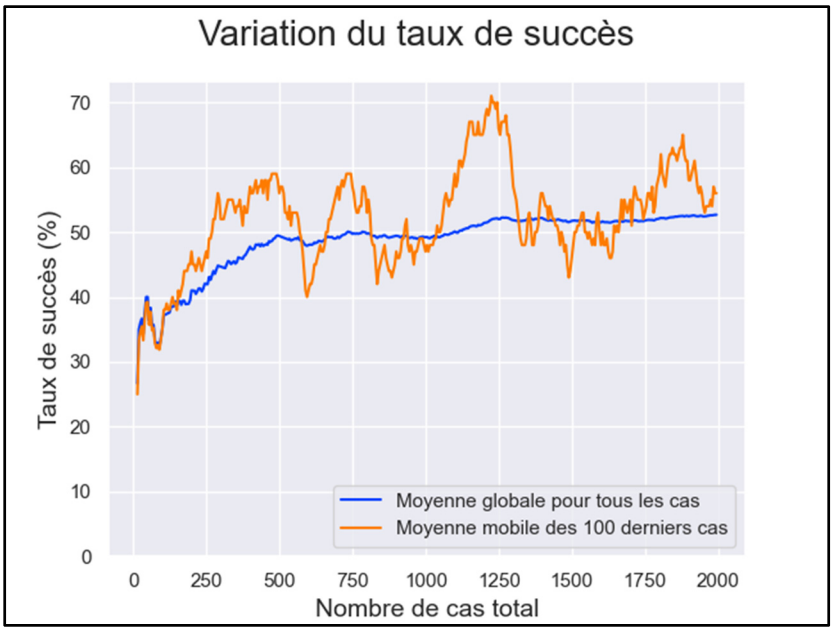


Figure 6.8 Taux de succès avec K = 1 et 20% de probabilité d'apparition en grappe

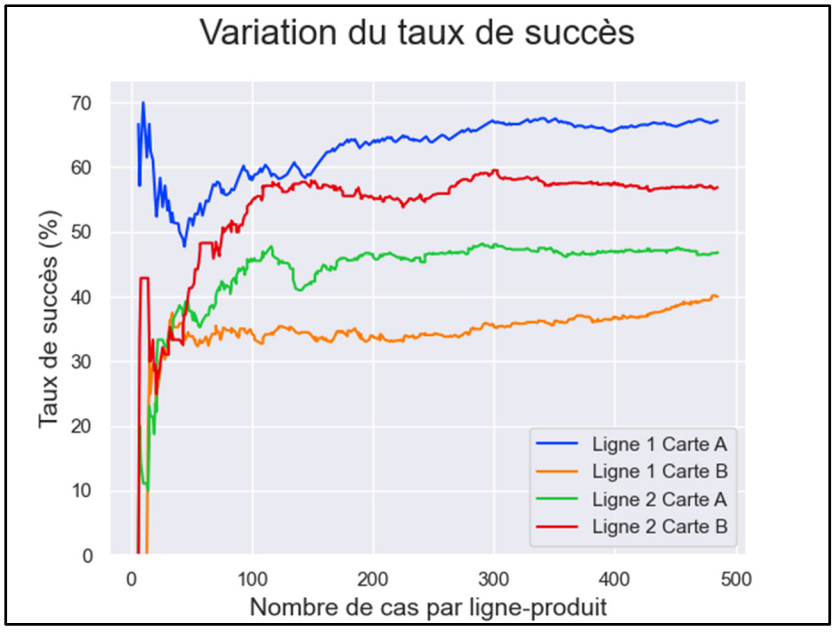


Figure 6.9 Taux de succès par ligne-produit avec K = 1 et 20% de probabilité d'apparition en grappe

Les figures 6.8 et 6.9 illustrent un scénario où les effets de localité spatiale et temporelle des défectuosités sont modélisés par la probabilité d'apparition en grappe. La figure 6.9 montre les taux de succès des quatre combinaisons ligne-produit plus espacés. Cette distribution est établie en fonction des interactions entre les particularités des lignes et des produits. En effet, les combinaisons ligne 1 - carte A et ligne 2 – carte B ont des faiblesses communes qui augmentent la probabilité de certaines défectuosités et causes physiques. Le système peut atteindre un taux de succès plus élevé une fois qu'il a intégré ces particularités. Ce phénomène de gradation, bien que toujours présent, n'est pas toujours aussi marqué sur les figures, puisque la durée nécessaire à son établissement dépend beaucoup des taux de succès initiaux, très aléatoires.

La configuration suivante illustre l'effet sur cette distribution de l'apparition soudaine d'un bris sur une ligne de production.

6.7.4 K = 1 et 20% de probabilité d'apparition en grappe suivie d'un bris

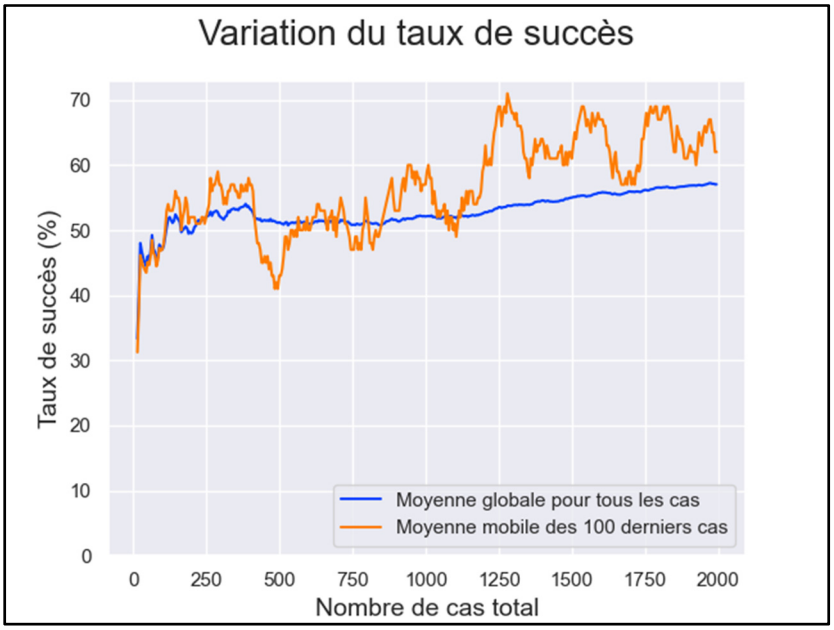


Figure 6.10 Taux de succès avec K = 1, 20% de probabilité d'apparition en grappe et apparition d'un bris

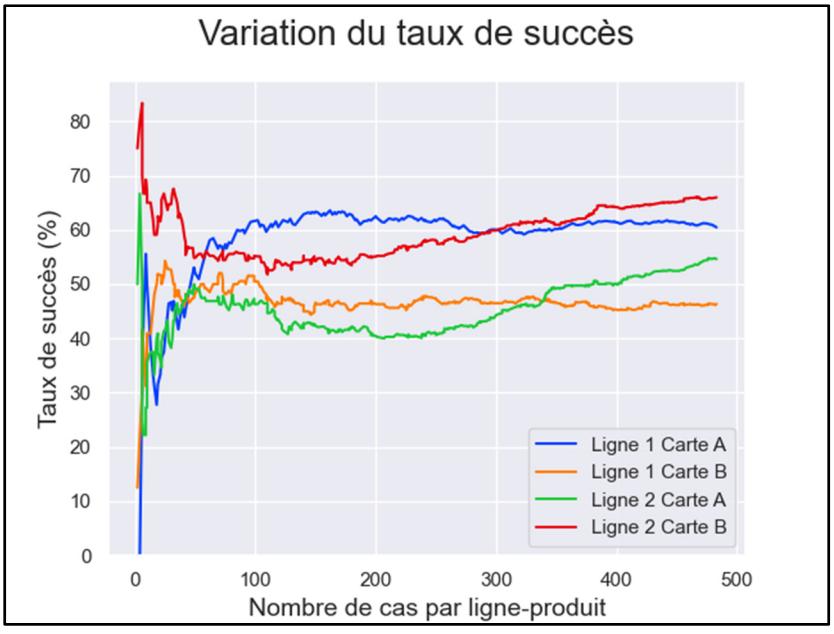


Figure 6.11 Taux de succès par ligne-produit avec K = 1, 20% de probabilité d'apparition en grappe et apparition d'un bris

Les figures 6.10 et 6.11 montrent un scénario similaire au précédent, mais où un bris apparaît sur la ligne 2 au cas 1200 au total (figure 6.10), ce qui correspond au cas 300 environ sur la figure 6.11. Le bris est modélisé par le passage instantané des probabilités d'apparition en grappe des défectuosités de 20 à 80% sur la ligne 2.

La figure 6.10 montre une reprise de la croissance du taux de succès global au moment du bris, mais aussi un déplacement marqué du plancher de la courbe de la moyenne mobile qui se positionne durablement au-dessus du niveau de la moyenne globale atteinte dans la région antérieure au bris, ce qui n'est pas présent dans les autres scénarios.

La figure 6.11 montre l'augmentation rapide du taux de succès des recommandations pour la ligne affectée (ligne 2) après l'évènement et présente un aspect inhabituel avec un changement marqué des positions relatives des courbes à un moment où une certaine stabilité devrait normalement s'être installée. Ce changement d'aspect est typique du scénario de bris et pourrait être utilisé pour les détecter. Une fois que la présence d'un bris est détectée de cette façon, il est possible de déterminer approximativement le moment de son apparition par l'analyse du point d'inflexion des courbes. Ceci permettrait d'identifier les produits potentiellement affectés.

Les trois dernières configurations reprennent les mêmes scénarios, mais cette fois en considérant la présence de la cause physique simulée parmi les deux cas les plus similaires trouvés par le système recommandeur comme un succès (K=2).

6.7.5 K = 2 et 0% de probabilité d'apparition en grappe

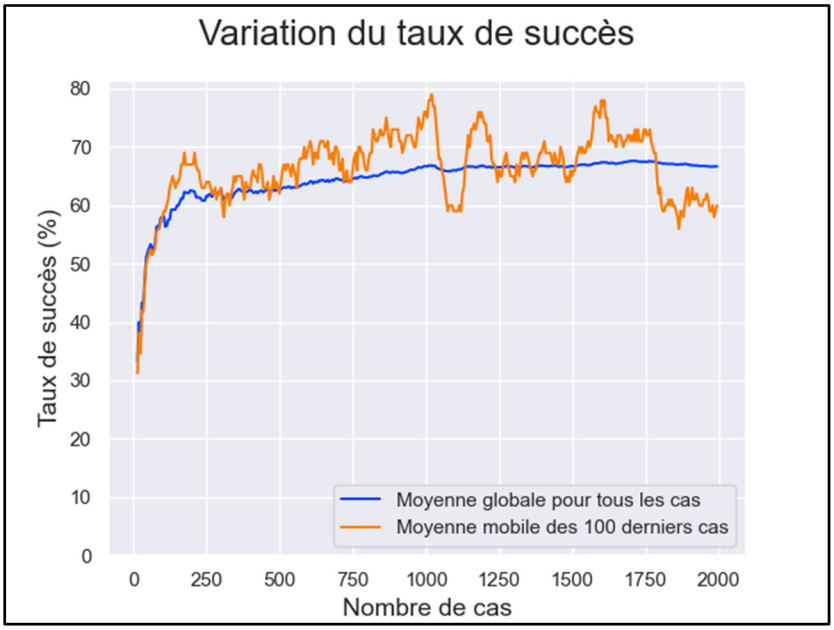


Figure 6.12 Taux de succès avec K = 2 sans apparition en grappe

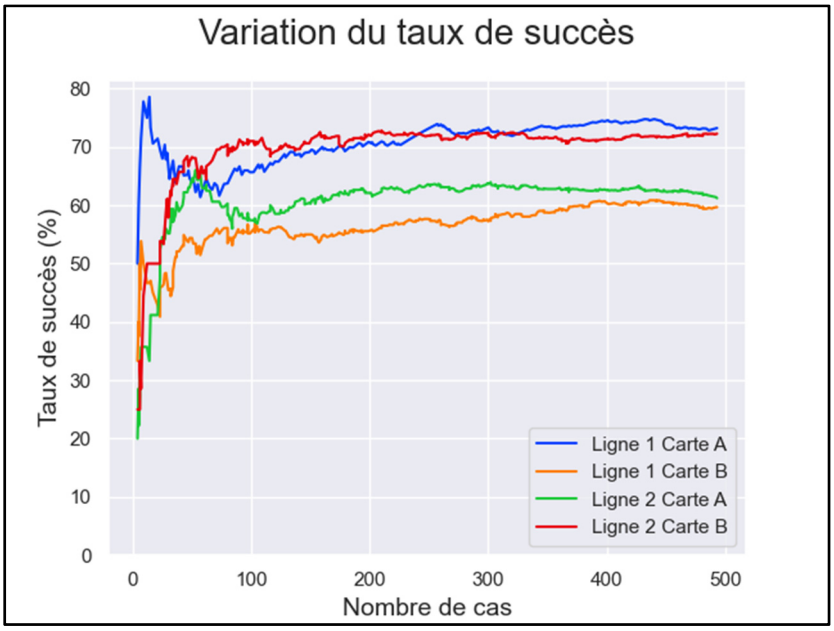


Figure 6.13 Taux de succès par ligne-produit avec K = 2
sans apparition en grappe

Les figures 6.12 et 6.13 permettent de constater immédiatement que le passage à K = 2 augmente considérablement le taux de succès de la recommandation. De plus, cette augmentation est plus marquée pour les combinaisons ligne-produit les moins performantes. Les taux de succès enregistrés sont également largement supérieurs à la sélection de deux causes physiques aléatoires sans remise, dont le taux de succès moyen tendrait vers 33.33%.

6.7.6 K = 2 et 20% de probabilité d'apparition en grappe

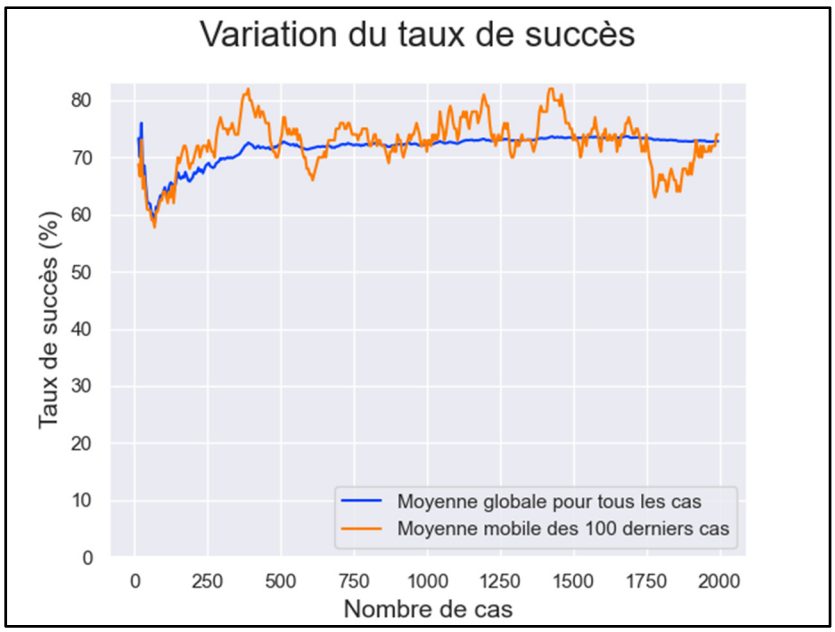


Figure 6.14 Taux de succès avec K = 2 et 20% de probabilité d'apparition en grappe

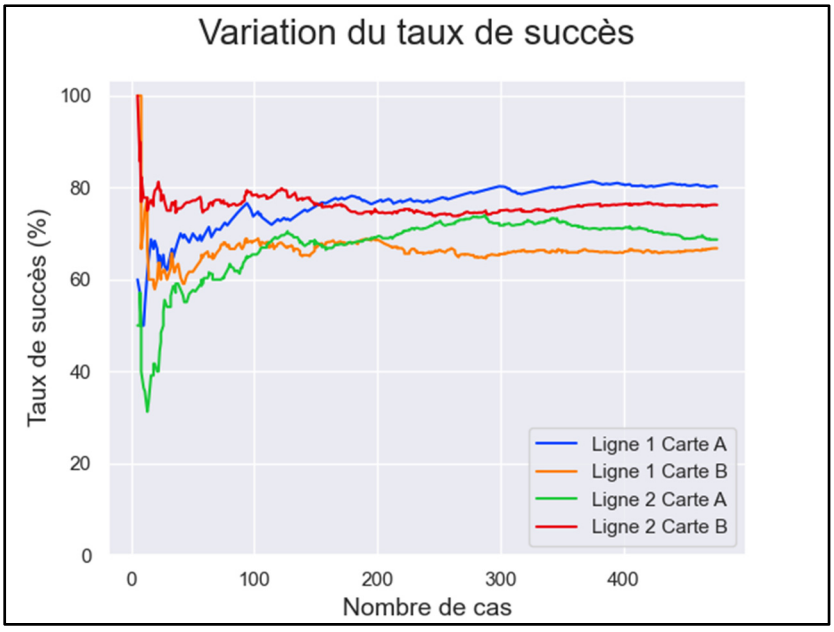


Figure 6.15 Taux de succès par ligne-produit avec K = 2 et 20% de probabilité d'apparition en grappe

Les figures 6.14 et 6.15 montrent qu'en tenant compte des effets de localité temporelle et spatiale, le système recommandeur peut atteindre un taux de succès suffisant pour constituer un ajout appréciable à un engin diagnostic permettant déjà la détection, la localisation et la classification correcte des défectuosités. En plus d'augmenter le taux de succès global, l'utilisation des deux meilleurs cas pour la recommandation permet de réduire considérablement l'écart entre les différentes combinaisons ligne-produit par rapport aux résultats montrés à la section 6.7.3. Cette réduction de la disparité des taux de succès constitue un facteur important pour l'acceptabilité du système par les utilisateurs ciblés.

6.7.7 K = 2 et 20% de probabilité d'apparition en grappe suivie d'un bris

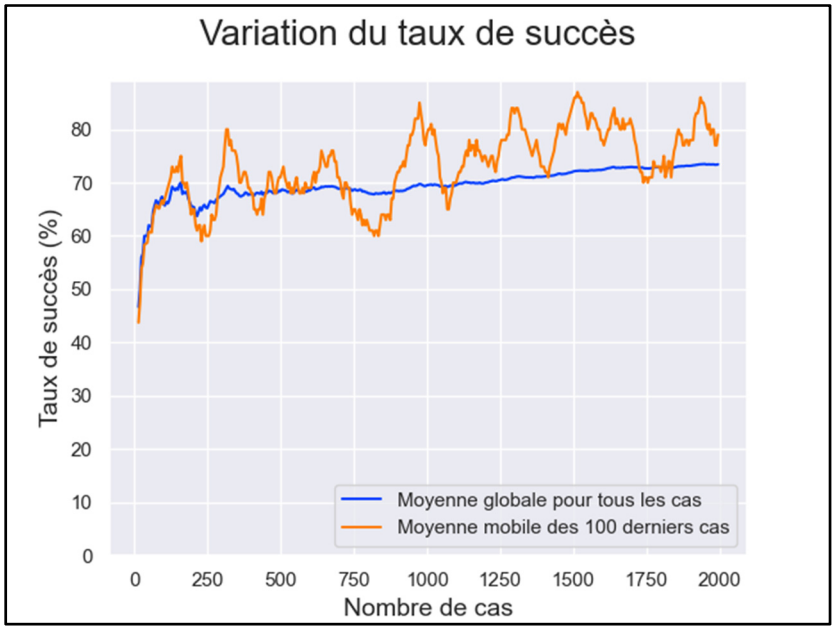


Figure 6.16 Taux de succès avec K = 2, 20% de probabilité d'apparition en grappe et apparition d'un bris

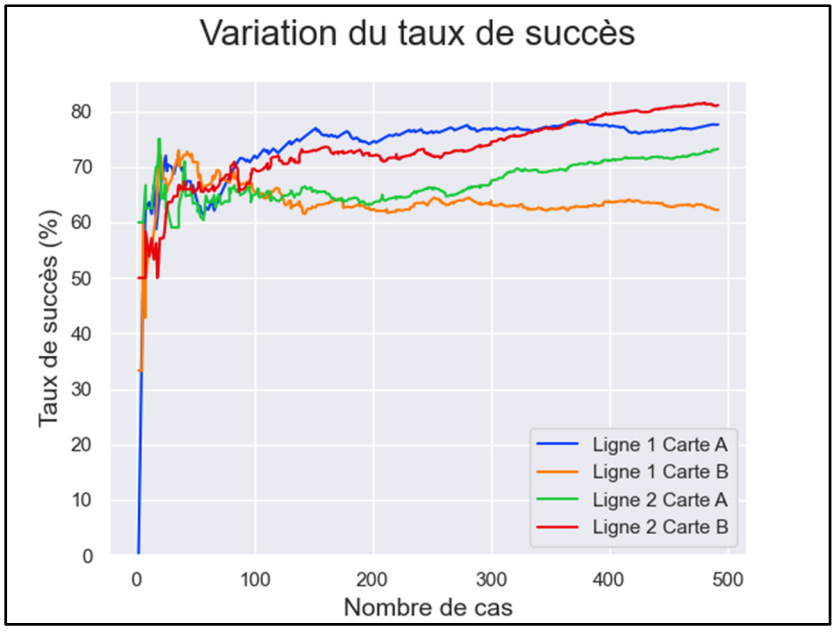


Figure 6.17 Taux de succès par ligne-produit avec K = 2, 20% de probabilité d'apparition en grappe et apparition d'un bris

Les figures 6.16 et 6.17 montrent que malgré des taux de succès de départ beaucoup plus élevés, les effets montrés à la section 6.7.4 sont tous encore visibles, bien que de façon un peu moins marquée. Rappelons que ces effets sont:

1. La reprise de la croissance du taux de succès global (en bleu, figure 6.16);
2. Le déplacement durable de la courbe de la moyenne mobile au-dessus du niveau de la moyenne globale antérieure (en brun, figure 6.16) et;
3. La croissance soudaine et rapide des taux de succès des combinaisons ligne-produit affectées (en rouge et en vert, figure 6.17).

Il pourrait donc être avantageux pour l'application de détection des bris de considérer uniquement le cas le plus similaire pour l'établissement des courbes (K = 1).

6.8 Conclusion

Nous croyons que les résultats obtenus avec notre système recommandeur prototype et le simulateur de cas développés comme preuve de concept montrent la faisabilité, la pertinence et le potentiel de l'utilisation d'un système recommandeur pour faire l'anticipation des causes physiques des défectuosités identifiées, classifiées et localisées par notre engin diagnostic.

Notre système montre aussi qu'une telle anticipation des causes physiques peut être utilisée par des intervenants différents du processus manufacturier à bas volume et haute complexité, soient le technicien chargé du diagnostic et de la réparation des produits défectueux et le gestionnaire chargé de la surveillance du procédé de fabrication.

La modélisation par le simulateur de l'effet des particularités des produits et des lignes de production sur leurs susceptibilités à différentes classes de défectuosités et de causes physiques pourrait être améliorée par l'exploitation des données de réparation existantes ainsi que par une utilisation complémentaire des logiciels avancés de fabrication assistée par ordinateur pour établir des prévisions ou projections de rendement des produits futurs sur l'équipement existant.

CONCLUSION

Dans cette thèse, nous avons proposé une nouvelle approche, basée sur la modélisation des connaissances et le raisonnement par cas, pour le diagnostic automatisé de cartes de circuits imprimés adapté à l'environnement de production à faible volume et haute complexité. Notre approche hybride, dont l'efficacité a été démontrée à l'aide d'un prototype de système de diagnostic que nous avons développé, permet de surmonter le goulot d'étranglement de l'acquisition des connaissances (*knowledge-acquisition bottleneck*) malgré le fait qu'un environnement pauvre en données soit ciblé. L'approche proposée ne nécessite pas la contribution des concepteurs ou d'experts du produit et est conçue pour opérer uniquement avec l'information disponible lors de la fabrication du produit. De plus, bien qu'elle repose sur l'apprentissage machine, cette approche n'utilise pas les réseaux de neurones artificiels et n'est pas affectée par le phénomène d'opacité des résultats inhérent à ces réseaux, en particulier pour la méthode de l'apprentissage profond. Les résultats du diagnostic sont déterministes, explicables et auditables.

Le prototype de système de diagnostic résultant est capable de détecter, localiser et faire la classification de toutes les défectuosités simples ou touchant jusqu'à 3 nœuds voisins présentes dans les fichiers de sortie d'un outil de test JTAG commercial. Il est de plus capable de proposer à l'utilisateur des causes physiques probables des défectuosités observées et de permettre la détection d'un changement soudain de la configuration ou un bris d'équipement pendant la production. Ainsi, l'architecture du système, basé sur l'approche que nous proposons, permet de couvrir toutes les étapes du diagnostic, soient la détection, la localisation, la classification et l'analyse des causes des défectuosités présentes sur les produits.

De manière plus détaillée, cette thèse a été structurée comme suit. Après la mise en contexte et la revue de littérature, nous avons présenté une vue d'ensemble du système de diagnostic résultant, qui représente le squelette de l'approche proposée, ainsi qu'une description de ses composants principaux. Nous avons également introduit les cartes de circuits imprimées conçues pour le test et la validation du prototype de ce système. Nous avons ensuite présenté

la technique du *boundary scan* ainsi que la structure et le fonctionnement de notre système d'émulation matérielle sur FPGA des circuits imprimés au niveau JTAG, utilisé pour la génération automatique de dictionnaires de pannes. L'émulateur, une fois branché à l'équipement de test JTAG commercial, permet d'obtenir la réponse de cet équipement en présence de toutes les défectuosités simples observables par *boundary scan* en utilisant des saboteurs intégrés au modèle et contrôlables par logiciel. Les dictionnaires de pannes produits par l'émulateur sont utilisés pour accélérer l'apprentissage du système de raisonnement par cas responsable de la classification des défectuosités.

Nous avons poursuivi par la présentation des éléments constitutifs du dépôt de connaissances du domaine : un réservoir d'information constitué non seulement des règles générales de détection et de localisation des défectuosités, mais également des éléments propres à un produit que sont la structure du réseau logique, les résultats des mesures effectuées sur l'équipement de test JTAG et les données de fabrication assistée par ordinateur, comme la liste des composants utilisés et la disposition physique des cartes de circuits imprimés.

Ensuite, nous avons décrit le cœur de notre approche, à savoir la technique du raisonnement par cas et son application au problème du diagnostic. Nous avons présenté un prototype de notre engin diagnostic que nous avons testé afin de démontrer qu'il offre une classification correcte des défectuosités qui lui sont présentées. Nous avons montré que la phase d'apprentissage du système peut-être fortement raccourcie par l'utilisation de cas synthétiques obtenus par émulation en utilisant le générateur de dictionnaire de panne pour alimenter l'engin diagnostic en démarrage. Cette innovation est importante, puisqu'elle rend envisageable l'utilisation d'un système de diagnostic automatisé basé sur l'apprentissage machine dans un environnement à faible volume, pauvre en données.

Nous avons ensuite comparé notre prototype à un outil de diagnostic JTAG commercial, *Boundary Scan Diagnostic* (BSD) de la compagnie *JTAG Technologies*, chef de file dans le domaine du test *boundary scan*. Les résultats obtenus montrent que notre prototype offre une meilleure classification que BSD, surtout en présence de défectuosités multiples.

Finalement, nous avons proposé d'ajouter les données de production ainsi qu'une rétroaction de l'utilisateur indiquant les composants fautifs à la base de cas afin d'utiliser à nouveau le raisonnement par cas pour faire l'analyse des causes physiques probables. Nous avons présenté deux applications utilisant ces nouvelles données, soit la recommandation des causes physiques les plus probables lors du diagnostic et la détection de l'apparition de bris sur la ligne de production par l'analyse de l'évolution du taux de succès des recommandations pour chaque ligne et produit.

Afin de valider la portée de l'approche proposée et de démontrer le fonctionnement de l'ensemble du système résultant, nous avons conçu et réalisé un simulateur capable de générer toutes les données d'une requête soumise à l'engin diagnostic (résultats de test JTAG, données de production et rétroaction indiquant le composant fautif) en tenant compte des probabilités d'apparition des différentes classes de causes physiques trouvées dans la littérature, d'une modélisation des caractéristiques propres à un produit et à une ligne de production ainsi que du phénomène d'apparition en grappe des défectuosités. Les résultats de simulation obtenus montrent que le système est suffisamment performant pour envisager son utilisation dans l'environnement de production à faible volume et haute complexité pour lequel il a été conçu et où il pourrait constituer une solution complète de diagnostic automatisé des résultats de test *boundary scan*.

RECOMMANDATIONS

Les résultats de détection, localisation et classification des défectuosités obtenus par le système de diagnostic automatisé prototype et montrés au chapitre 5, ainsi que les résultats du système recommandeur obtenus par simulation et présentés au chapitre 6 nous indiquent qu'il serait pertinent d'envisager l'adaptation et la mise en œuvre du système en environnement de production à faible volume et haute complexité.

Dans un premier temps, il faudrait procéder à la génération d'un modèle au niveau *boundary scan* d'une carte commerciale afin d'en obtenir un dictionnaire de panne par émulation sur l'équipement de test utilisé en production. Le système de diagnostic automatisé prototype pourrait ensuite être testé sur la ligne produisant cette carte et comparé avec le procédé de diagnostic conventionnel.

Une fois le moteur diagnostic opérationnel avec une carte commerciale, il serait également pertinent de tester le système recommandeur avec des rétroactions réelles plutôt que simulées afin d'en évaluer la performance.

La pondération des critères du système recommandeur devrait également faire l'objet d'un ajustement en conditions réelles. À ce sujet, une opération de *data mining* des données de réparation historiques de la carte pourrait s'avérer judicieuse et permettre d'établir une pondération nominale basée sur les causes physiques historiques.

Le *data mining* des données historiques pourrait également être utile pour le raffinement du simulateur de cas présenté au chapitre 6. Un élément qui mériterait une évaluation plus approfondie à partir des données historiques est le taux d'incidence du phénomène d'apparition en grappe des défectuosités. Évaluer de façon plus précise l'impact de la localité spatiale et temporelle sur la distribution des causes physiques de défectuosités permettrait d'améliorer à la fois le simulateur et le système recommandeur.

Puisque le système possède déjà une représentation interne de la disposition physique des cartes, il pourrait être utile de combiner aux résultats de test *boundary scan*, les résultats de tests d'imagerie en lumière visible ou rayons X effectués en amont sur la chaîne de production. En comparant ces images avec son modèle interne et l'historique des observations précédentes contenues dans la base de cas, le système pourrait offrir une localisation plus précise des défectuosités ainsi que de meilleures recommandations de réparation.

En effet, à l'exception de la génération de dictionnaires de pannes par émulation décrite au chapitre 3, aucun des éléments du système proposé n'est inextricablement lié au *boundary scan*. L'architecture proposée peut être utilisée avec toutes les sources de données disponibles sur la chaîne de production, ce qui inclut les images et les mesures analogiques effectuées avec l'équipement de test électrique en circuit (ICT). La combinaison des données de plusieurs sources est probablement la stratégie offrant le meilleur potentiel. C'est aussi une façon de maximiser l'utilisation des données disponibles dans notre environnement cible.

ANNEXE I

IMPLÉMENTATION DU TAP CONTROLLER

```
--------------------------------------------------------------------------------
-- Title      : TAP_controller
-- Project    : jtagemu
--------------------------------------------------------------------------------
-- File       : TAP_controller.vhd
-- Author     : simon
--------------------------------------------------------------------------------
-- Description: RTL model of standard JTAG TAP controller
--------------------------------------------------------------------------------
-- Copyright (c) 2016 Simon Pichette
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
--------------------------------------------------------------------------------
entity TAP_Controller is
  port(TMS       : in  std_logic;
       TCK       : in  std_logic;
       TRST_n    : in  std_logic;
       Clock_IR  : out std_logic;
       Shift_IR  : out std_logic;
       Update_IR : out std_logic;
       Clock_DR  : out std_logic;
       Shift_DR  : out std_logic;
       Update_DR : out std_logic;
       Reset_n   : out std_logic;
       Enable    : out std_logic;
       Select_o  : out std_logic);
end TAP_Controller;
--------------------------------------------------------------------------------
architecture rtl of TAP_Controller is

  type state_type is (Exit2_DR, Exit1_DR, Shift_DRs, Pause_DR, Select_IR_scan,
                      Update_DRs, Capture_DR, Select_DR_scan, Exit2_IR, Exit1_IR,
                      Shift_IRs, Pause_IR, Run_test_Idle, Update_IRs, Capture_IR,
                      Test_logic_reset);
  signal state_reg, next_state : state_type;

begin

  -- combinational outputs from IEEE1149.1_2001 p.28 and IEEE1149.1_2013 p.34
  Clock_IR  <= '0' when TCK = '0' and (state_reg = Shift_IRs or
                                       state_reg = Capture_IR) else '1';
  Update_IR <= '1' when TCK = '0' and state_reg = Update_IRs else '0';

  Clock_DR  <= '0' when TCK = '0' and (state_reg = Shift_DRs or
                                       state_reg = Capture_DR) else '1';
  Update_DR <= '1' when TCK = '0' and state_reg = Update_DRs else '0';

  Select_o  <= '1' when state_reg = Exit2_IR or state_reg = Exit1_IR or
               state_reg = Shift_IRs or state_reg = Pause_IR or
               state_reg = Run_test_Idle or state_reg = Update_IRs or
```

```
              state_reg = Capture_IR or state_reg = Test_logic_reset else '0';

  -- sequential outputs from IEEE1149.1_2001 p.28 and IEEE1149.1_2013 p.34
  mealy_out : process(TCK, TRST_n)
  begin
    if TRST_n = '0' then
      Reset_n  <= '0';
      Enable   <= '0';
      Shift_IR <= '0';
      Shift_DR <= '0';
    else
      if falling_edge(TCK) then         -- DFF are clocked by inverted TCK

        -- defaults
        Reset_n  <= '1';
        Enable   <= '0';
        Shift_IR <= '0';
        Shift_DR <= '0';

        if state_reg = Test_logic_reset then
          Reset_n <= '0';
        end if;

        if state_reg = Shift_IRs then
          Enable   <= '1';
          Shift_IR <= '1';
        end if;

        if state_reg = Shift_DRs then
          Enable   <= '1';
          Shift_DR <= '1';
        end if;

      end if;
    end if;
  end process;  -- mealy_out

  state_register : process(TCK, TRST_n)
  begin
    if TRST_n = '0' then
      state_reg <= Run_test_Idle;
    else
      if rising_edge(TCK) then
        state_reg <= next_state;
      end if;
    end if;
  end process;  -- state_reg

  next_state_logic : process(state_reg, TMS)
  begin
    case state_reg is
      when Test_logic_reset =>
        if TMS = '1' then
          next_state <= Test_logic_reset;
        else
          next_state <= Run_test_Idle;
        end if;

      when Run_test_Idle =>
```

```
        if TMS = '1' then
          next_state <= Select_DR_scan;
        else
          next_state <= Run_test_Idle;
        end if;
      when Select_DR_scan =>
        if TMS = '1' then
          next_state <= Select_IR_scan;
        else
          next_state <= Capture_DR;
        end if;

      when Select_IR_scan =>
        if TMS = '1' then
          next_state <= Test_logic_reset;
        else
          next_state <= Capture_IR;
        end if;

      when Capture_DR =>
        if TMS = '1' then
          next_state <= Exit1_DR;
        else
          next_state <= Shift_DRs;
        end if;

      when Capture_IR =>
        if TMS = '1' then
          next_state <= Exit1_IR;
        else
          next_state <= Shift_IRs;
        end if;

      when Shift_DRs =>
        if TMS = '1' then
          next_state <= Exit1_DR;
        else
          next_state <= Shift_DRs;
        end if;

      when Shift_IRs =>
        if TMS = '1' then
          next_state <= Exit1_IR;
        else
          next_state <= Shift_IRs;
        end if;

      when Exit1_DR =>
        if TMS = '1' then
          next_state <= Update_DRs;
        else
          next_state <= Pause_DR;
        end if;

      when Exit1_IR =>
        if TMS = '1' then
          next_state <= Update_IRs;
        else
          next_state <= Pause_IR;
```

```
        end if;

      when Pause_DR =>
        if TMS = '1' then
          next_state <= Exit2_DR;
        else
          next_state <= Pause_DR;
        end if;
      when Pause_IR =>
        if TMS = '1' then
          next_state <= Exit2_IR;
        else
          next_state <= Pause_IR;
        end if;

      when Exit2_DR =>
        if TMS = '1' then
          next_state <= Update_DRs;
        else
          next_state <= Shift_DRs;
        end if;

      when Exit2_IR =>
        if TMS = '1' then
          next_state <= Update_IRs;
        else
          next_state <= Shift_IRs;
        end if;

      when Update_DRs =>
        if TMS = '1' then
          next_state <= Select_DR_scan;
        else
          next_state <= Run_test_Idle;
        end if;

      when Update_IRs =>
        if TMS = '1' then
          next_state <= Select_DR_scan;
        else
          next_state <= Run_test_Idle;
        end if;

      when others =>
        next_state <= Test_logic_reset;
    end case;
  end process;  -- next_state_logic

end;  -- architecture rtl of TAP_Controller
--------------------------------------------------------------------------------
```

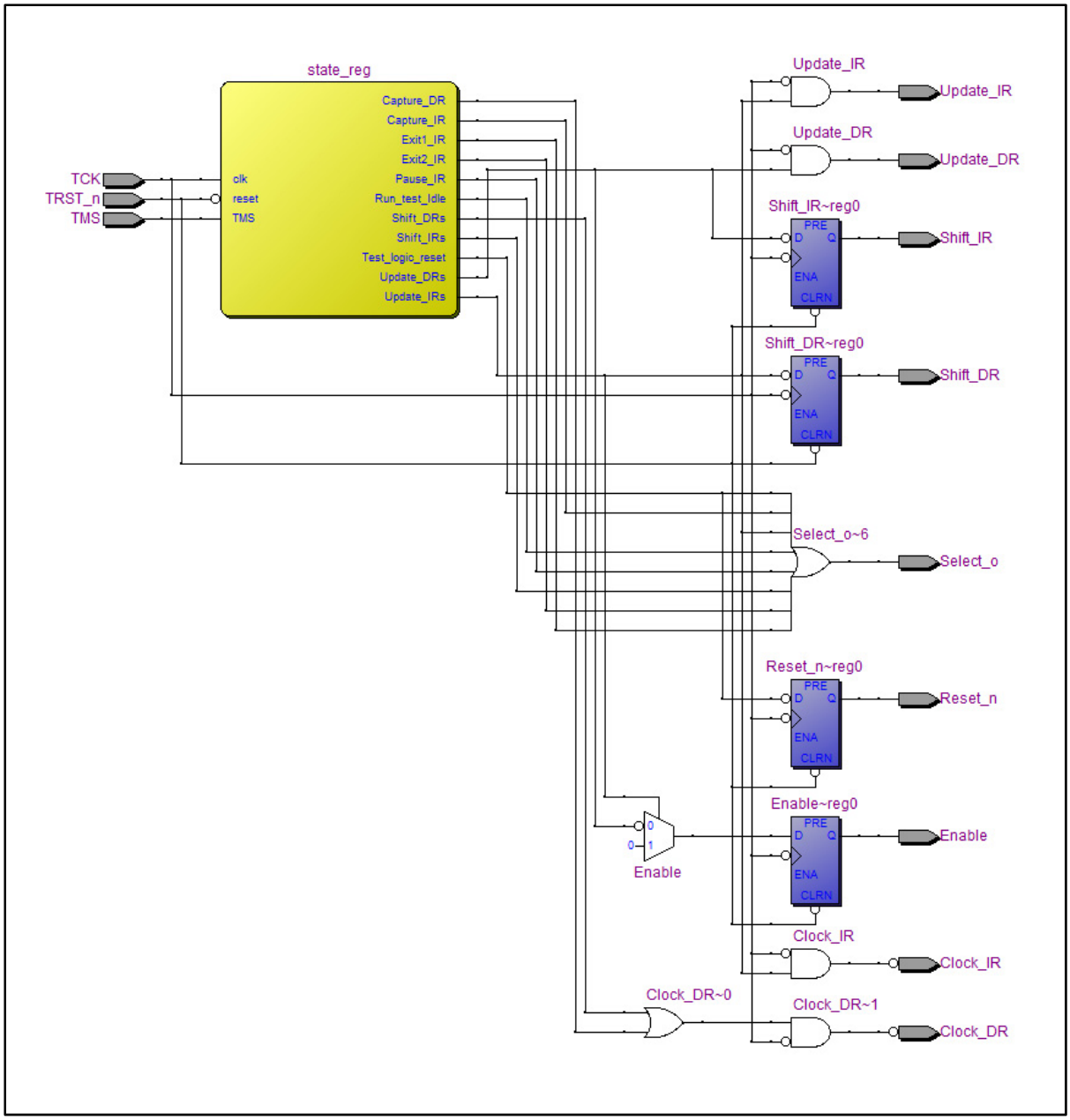


Figure-A I-1 Résultat de l'analyse RTL du TAP_Controller par Altera Quartus II

ANNEXE II

REGISTRE D'INSTRUCTION JTAG

Le registre d'instruction permet de charger des patrons de bits contrôlant le mode de fonctionnement de la chaîne JTAG. Il est formé d'une séquence de cellules IR_1. Nous avons reproduit cette cellule en VHDL telle qu'illustrée dans (Parker, 2016) à la page 21. Le résultat de l'analyse RTL de l'outil de synthèse est présenté à la figure-A II-1. La page suivante montre le code source VHDL.

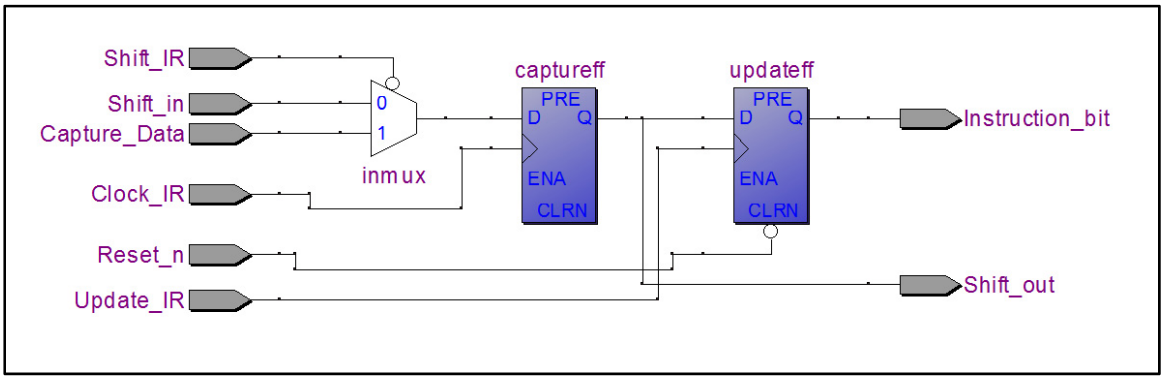


Figure-A II-1 Résultat de l'analyse RTL de la cellule IR_1 par l'outil de synthèse

```
--------------------------------------------------------------------------------
-- Title      : IR_1
-- Project    : jtagemu
--------------------------------------------------------------------------------
-- File       : IR_1.vhd
-- Author     : simon
--------------------------------------------------------------------------------
-- Description: RTL model of standard JTAG Instruction Register Cell
--------------------------------------------------------------------------------
-- Copyright (c) 2016 Simon Pichette
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
--------------------------------------------------------------------------------
entity IR_1 is
  port(
    Clock_IR        : in  std_logic;
    Shift_IR        : in  std_logic;
    Update_IR       : in  std_logic;
    Shift_in        : in  std_logic;
    Capture_Data    : in  std_logic;
    Shift_out       : out std_logic;
    Instruction_bit : out std_logic;
    Reset_n         : in  std_logic);
end IR_1;
--------------------------------------------------------------------------------
architecture rtl of IR_1 is
  signal inmux     : std_logic;
  signal captureff : std_logic;
  signal updateff  : std_logic;
begin
  process(Clock_IR)
  begin
    if rising_edge(Clock_IR) then
      captureff <= inmux;
    end if;
  end process;

  process(Update_IR, Reset_n)
  begin
    if Reset_n = '0' then
      updateff <= '0';
    else
      if rising_edge(Update_IR) then
        updateff <= captureff;
      end if;
    end if;
  end process;
  inmux           <= Capture_Data when Shift_IR = '0' else Shift_in;
  Instruction_bit <= updateff;
  Shift_out       <= captureff;
end; -- rtl of IR_1
--------------------------------------------------------------------------------
```

ANNEXE III

BOUNDARY REGISTER JTAG

Le *boundary register* nécessite l'utilisation de cellules spécialisées, selon le type et la direction des broches de la puce, et afin de permettre le chargement et la lecture des signaux présents sur les broches. Certaines cellules sont également dédiées au contrôle d'autres cellules, notamment pour la gestion des broches bidirectionnelles, qui nécessitent trois cellules chacune. Le standard 1149.1 2016 présente dix configurations standard pour les cellules de *boundary scan*. Pour modéliser les circuits utilisés dans notre projet, nous avons eu besoin des cellules BC_1, BC_2 et BC_4, que nous avons reproduites en VHDL tels qu'illustrées dans (Parker, 2016) aux pages 93, 94 et 96. Les résultats d'analyse RTL sont illustrés aux figures-A III-1, III-2 et III-3. Le code source VHDL est illustré aux figures III-4, III-5 et III-6.

Le choix des cellules implémentées revient au concepteur de chaque dispositif compatible JTAG. Les cellules diffèrent par leur taille et leur complexité et ne permettent pas toutes l'ensemble des opérations de test JTAG. La cellule BC_1 sert d'élément de base et est la plus flexible. La cellule BC_2 permet une plus grande testabilité de la chaîne JTAG elle-même, mais n'est pas appropriée pour l'implémentation de broches bidirectionnelles. La cellule BC_4 est beaucoup plus petite et introduit moins de délais sur la chaîne de balayage par l'élimination du multiplexeur du chemin de données. Cependant elle ne peut être chargée en parallèle et n'est pas appropriée pour le test des broches d'entrées ou bidirectionnelles. Le choix du type de cellule utilisée pour chaque nœud de la chaîne de balayage ainsi que la broche associée et d'autres détails à propos de l'implémentation du standard JTAG sont regroupé dans un fichier de description formulé dans un langage appelé BSDL.

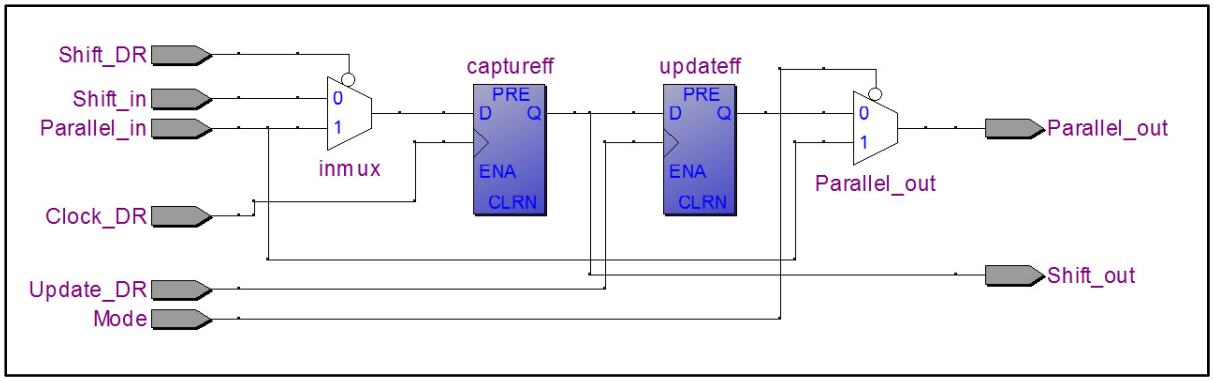


Figure-A III-1 Résultat de l'analyse RTL de la cellule BC_1 par Altera Quartus II

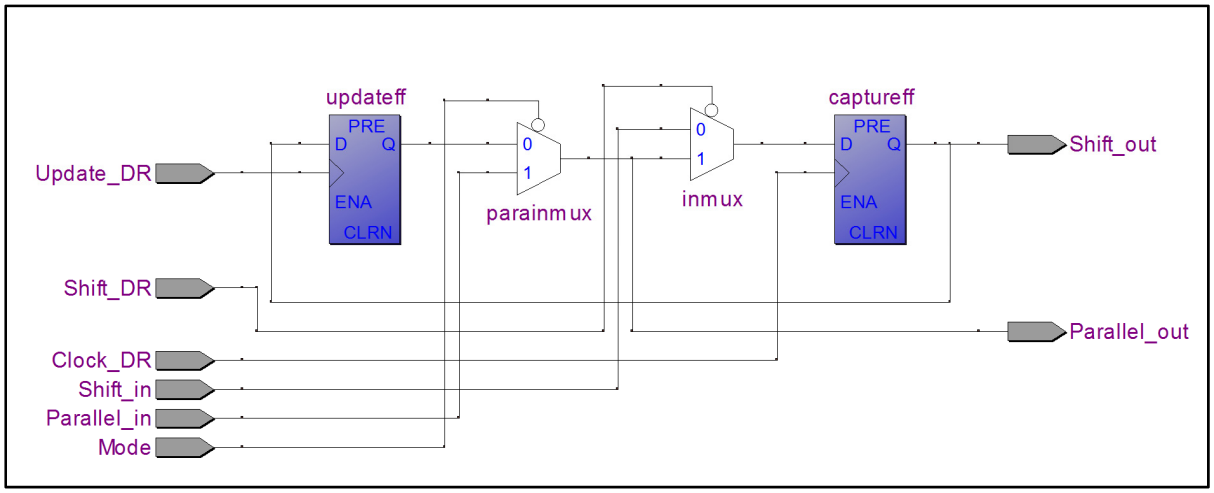


Figure-A III-2 Résultat de l'analyse RTL de la cellule BC_2 par Altera Quartus II

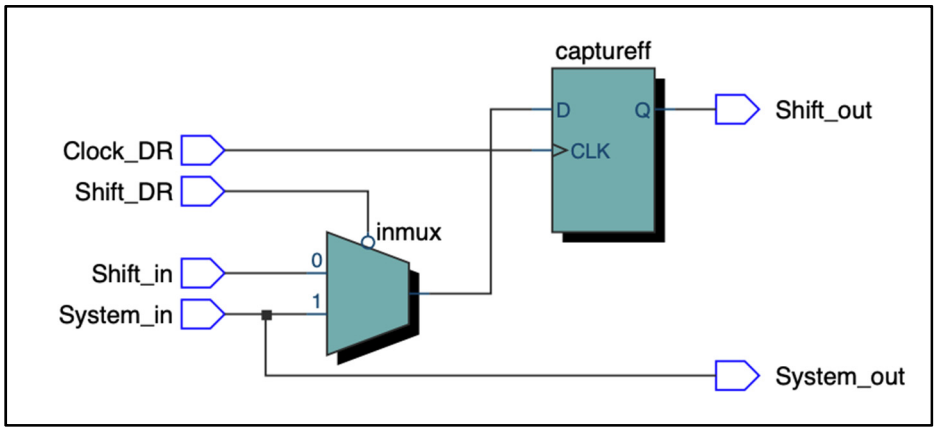


Figure-A III-3 Résultat de l'analyse RTL de la cellule BC_4 par Altera Quartus II

```
--------------------------------------------------------------------------------
-- Title      : BC_1
-- Project    : jtagemu
--------------------------------------------------------------------------------
-- File       : BC_1.vhd
-- Author     : simon
-- Description: RTL model of standard JTAG Boundary Scan Register Cell that
--              supports all instructions.
--------------------------------------------------------------------------------
-- Copyright (c) 2016 Simon Pichette
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
--------------------------------------------------------------------------------
entity BC_1 is
  port(Clock_DR     : in  std_logic;
       Shift_DR     : in  std_logic;
       Update_DR    : in  std_logic;
       Shift_in     : in  std_logic;
       Parallel_in  : in  std_logic;
       Mode         : in  std_logic;
       Shift_out    : out std_logic;
       Parallel_out : out std_logic);
end BC_1;
--------------------------------------------------------------------------------
architecture rtl of BC_1 is
  signal inmux     : std_logic;
  signal captureff : std_logic;
  signal updateff  : std_logic;
begin
  process(Clock_DR)
  begin
    if rising_edge(Clock_DR) then
      captureff <= inmux;
    end if;
  end process;

  process(Update_DR)
  begin
    if rising_edge(Update_DR) then
      updateff <= captureff;
    end if;
  end process;

  inmux        <= Parallel_in when Shift_DR = '0' else Shift_in;
  Parallel_out <= Parallel_in when Mode = '0'     else updateff;
  Shift_out    <= captureff;

end;  -- rtl of BC_1
--------------------------------------------------------------------------------
```

Figure-A III-4 Code source VHDL de la cellule BC_1

```
--------------------------------------------------------------------------------
-- Title      : BC_2
-- Project    : jtagemu
--------------------------------------------------------------------------------
-- File       : BC_2.vhd
-- Author     : simon
--------------------------------------------------------------------------------
-- Description: RTL model of standard JTAG Boundary Scan Register Cell with
--              parallel input mux
--------------------------------------------------------------------------------
-- Copyright (c) 2016 Simon Pichette
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
--------------------------------------------------------------------------------
entity BC_2 is
  port(Clock_DR     : in  std_logic;
       Shift_DR     : in  std_logic;
       Update_DR    : in  std_logic;
       Shift_in     : in  std_logic;
       Parallel_in  : in  std_logic;
       Mode         : in  std_logic;
       Shift_out    : out std_logic;
       Parallel_out : out std_logic);
end BC_2;
--------------------------------------------------------------------------------
architecture rtl of BC_2 is
  signal inmux     : std_logic;
  signal captureff : std_logic;
  signal updateff  : std_logic;
  signal parainmux : std_logic;

begin
  process(Clock_DR)
  begin
    if rising_edge(Clock_DR) then
      captureff <= inmux;
    end if;
  end process;

  process(Update_DR)
  begin
    if rising_edge(Update_DR) then
      updateff <= captureff;
    end if;
  end process;

  parainmux    <= Parallel_in when Mode = '0'     else updateff;
  inmux        <= parainmux   when Shift_DR = '0' else Shift_in;
  Parallel_out <= parainmux;
  Shift_out    <= captureff;
end;  -- rtl of BC_2
--------------------------------------------------------------------------------
```

Figure-A III-5 Code source VHDL de la cellule BC_2

```
--------------------------------------------------------------------------------
-- Title      : BC_4
-- Project    : jtagemu
--------------------------------------------------------------------------------
-- File       : BC_4.vhd
-- Author     : simon
--------------------------------------------------------------------------------
-- Description: RTL model of standard JTAG Boundary Scan Register Cell with
--              no update latch and no series multiplexer
--------------------------------------------------------------------------------
-- Copyright (c) 2020 Simon Pichette
--------------------------------------------------------------------------------
--  Sources
--    Boundary Scan Handbook, 4E, Kenneth Parker
--    IEEE1149.1-2001 Standard
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
--------------------------------------------------------------------------------
entity BC_4 is
  port( Clock_DR   : in std_logic;
        Shift_DR   : in  std_logic;
        Shift_in   : in  std_logic;
        System_in  : in  std_logic;
        Shift_out  : out std_logic;
        System_out : out std_logic);
end BC_4;
--------------------------------------------------------------------------------
architecture rtl of BC_4 is
  signal inmux     : std_logic;
  signal captureff : std_logic;
begin

  process(Clock_DR)
  begin
    if rising_edge(Clock_DR) then
      captureff <= inmux;
    end if;
  end process;

  inmux      <= System_in when Shift_DR = '0' else Shift_in;
  System_out <= System_in;
  Shift_out  <= captureff;

end; -- rtl of BC_4
--------------------------------------------------------------------------------
```

Figure-A III-6 Code source VHDL de la cellule BC_4

ANNEXE IV

OUTILS POUR LA TRANSFORMATION BSDL-VHDL

splitio.pl

```
#!/usr/bin/perl -w

## splitio.pl

## split inout vhdl/bsdl port entries into separate in and out std_logic ports
use strict;
use warnings;


while(my $line = <>) {
    chomp $line;
    $line =~ s/^\s+//;    # left trim
    my ($name, $attribs) = split(/:/,$line,2);

    $name =~ s/\s+$//;   # right trim
    my ($direction, $rest) = split(' ',$attribs,2);

    if (lc $direction ne 'linkage') {

        if (lc $direction eq 'inout') {
            print "$name\_i : in  std_logic;\n";
            print "$name\_o : out std_logic;\n";
        } else {
            print "$name: $direction std_logic;\n";
        }
    }
}
```

map_bs.pl

```
#!/usr/bin/perl -w

## mapbs.pl
## Reads bsd file fragment to correctly map input and output boundary scan cells
## to vhd model vectors
use strict;
use warnings;
use Data::Dumper;
no warnings "uninitialized";    # suppress useless warnings for optional fields

my @inputs;
my @outputs;

while(my $line = <>) {
    chomp $line;

    # split line at first '('
    my ($cellnum, $attribs) = split(/\(/,$line,2);

    # trim everything past ')'
    ($attribs) = split(/\)/, $attribs);

    # split fields on ','
    my ($type, $port, $function, $safe, $ccell,
        $disval, $disrslt) = split(/,/, $attribs, 7);

    # extract integers from numeric fields
    $cellnum =~ s/\D//g;
    $ccell =~ s/\D//g;
    $disval =~ s/\D//g;

    # left trim text fields
    $port =~ s/^\s+//;
    $function =~ s/^\s+//;
    $safe =~ s/^\s+//;
    $disrslt =~ s/^\s+//;

    ### boundary register entry is now completely parsed. ###
    #print "'$cellnum', '$type', '$port', '$function', '$safe',
    #       '$ccell', '$disval', '$disrslt'\n";

    # generate connections for vhdl file

    if (lc $function eq 'output3') {
        push @outputs, "  $port\_o <= Parallel_out_vec($cellnum);";
    }

    if (lc $function eq 'input') {
        push @inputs, "  Parallel_in_vec($cellnum) <= $port\_i;";
    }
}
print "-- inputs\n";
foreach (@inputs) { print "$_\n";}
print "\n-- outputs\n";
foreach (@outputs) { print "$_\n";}
```

ANNEXE V

TRANSFORMATION BSDL VERS VHDL

SN74BCT8244A.bsd

```
--------------------------------------------------------------------------------
--  TI SN74BCT8244A                                                           --
--     IEEE Std 1149.1 (JTAG) Boundary-Scan Test Device                       --
--     with Octal Buffers                                                     --
--------------------------------------------------------------------------------
--  Created by    : Texas Instruments Incorporated                            --
--  Documentation : SN74BCT8244A Data Sheet (SCBS042)                         --
--  Product Status: Released to Production (RTP)                              --
--  BSDL revision : 2.1                                                       --
--  BSDL status   : Production                                                --
--  Date created  : 05/01/94                                                  --
--  Last modified : 07/26/97                                                  --
--  Modification history -                                                    --
--      - misc clean-up, cosmetic only                                        --
--------------------------------------------------------------------------------
entity sn74bct8244a is
    generic (PHYSICAL_PIN_MAP : string := "DW");

    port (OE_NEG1:in bit;
          Y1:out bit_vector(1 to 4);
          Y2:out bit_vector(1 to 4);
          A1:in bit_vector(1 to 4);
          A2:in bit_vector(1 to 4);
          OE_NEG2:in bit;
          GND, VCC:linkage bit;
          TDO:out bit;
          TDI, TMS, TCK:in bit;
          NC:linkage bit_vector(1 to 4));

    use STD_1149_1_1990.all; -- Get standard attributes and definitions

    attribute PIN_MAP of sn74bct8244a : entity is PHYSICAL_PIN_MAP;

    constant JT : PIN_MAP_STRING := "OE_NEG1:1, Y1:(2,3,4,5)," &
                  "Y2:(7,8,9,10), A1:(23,22,21,20)," &
                  "A2:(19,17,16,15), OE_NEG2:24, GND:6," &
                  "VCC:18, TDO:11, TDI:14, TMS:12, TCK:13";

    constant DW : PIN_MAP_STRING := "OE_NEG1:1, Y1:(2,3,4,5)," &
                  "Y2:(7,8,9,10), A1:(23,22,21,20)," &
                  "A2:(19,17,16,15), OE_NEG2:24, GND:6," &
                  "VCC:18, TDO:11, TDI:14, TMS:12, TCK:13";

    constant NT : PIN_MAP_STRING := "OE_NEG1:1, Y1:(2,3,4,5)," &
                  "Y2:(7,8,9,10), A1:(23,22,21,20)," &
                  "A2:(19,17,16,15), OE_NEG2:24, GND:6," &
                  "VCC:18, TDO:11, TDI:14, TMS:12, TCK:13";

    constant FK : PIN_MAP_STRING := "OE_NEG1:9, Y1:(10,11,12,13)," &
                  "Y2:(16,17,18,19), A1:(6,5,4,3)," &
```

```
                "A2:(2,27,26,25), OE_NEG2:7, GND:14, VCC:28," &
                "TDO:20, TDI:24, TMS:21, TCK:23, NC:(1,8,15,22)";

    attribute TAP_SCAN_IN    of TDI : signal is true;
    attribute TAP_SCAN_MODE  of TMS : signal is true;
    attribute TAP_SCAN_OUT   of TDO : signal is true;
    attribute TAP_SCAN_CLOCK of TCK : signal is (20.0e6, BOTH);

    attribute INSTRUCTION_LENGTH of sn74bct8244a : entity is 8;
    attribute INSTRUCTION_OPCODE of sn74bct8244a : entity is
            "EXTEST (00000000, 10000000), " &
            "BYPASS (11111111, 10000100, 00000101, 10001000, 00000001), " &
            "SAMPLE (00000010, 10000010), " &
            "INTEST (00000011, 10000011), " &
            "HIGHZ  (00000110, 10000110), " &  -- Bypass with outputs high-z
            "CLAMP  (00000111, 10000111), " &  -- Bypass with bs value
            "RUNT   (00001001, 10001001), " &  -- Boundary run test
            "READBN (00001010, 10001010), " &  -- Boundary read normal mode
            "READBT (00001011, 10001011), " &  -- Boundary read test mode
            "CELLTST(00001100, 10001100), " &  -- Boundary selftest normal mode
            "TOPHIP (00001101, 10001101), " &  -- Boundary toggle out test mode
            "SCANCN (00001110, 10001110), " &  -- BCR scan normal mode
            "SCANCT (00001111, 10001111)  " ;  -- BCR scan test mode

    attribute INSTRUCTION_CAPTURE of sn74bct8244a : entity is "10000001";
    attribute INSTRUCTION_DISABLE of sn74bct8244a : entity is "HIGHZ";
    attribute INSTRUCTION_GUARD   of sn74bct8244a : entity is "CLAMP";

    attribute REGISTER_ACCESS of sn74bct8244a : entity is
            "BOUNDARY (EXTEST, INTEST, SAMPLE, READBN, READBT, CELLTST),"  &
            "BYPASS   (BYPASS, HIGHZ, CLAMP, RUNT, TOPHIP)," &
            "BCR[2]   (SCANCN, SCANCT)" ;

    attribute BOUNDARY_CELLS    of sn74bct8244a : entity is "BC_1";
    attribute BOUNDARY_LENGTH   of sn74bct8244a : entity is 18;
    attribute BOUNDARY_REGISTER of sn74bct8244a : entity is

    "0  (BC_1, Y2(4), output3, X, 16, 1, Z),"  & -- these outputs controlled
    "1  (BC_1, Y2(3), output3, X, 16, 1, Z),"  & -- by cell 16
    "2  (BC_1, Y2(2), output3, X, 16, 1, Z),"  &
    "3  (BC_1, Y2(1), output3, X, 16, 1, Z),"  &
    "4  (BC_1, Y1(4), output3, X, 17, 1, Z),"  & -- these outputs controlled
    "5  (BC_1, Y1(3), output3, X, 17, 1, Z),"  & -- by cell 17
    "6  (BC_1, Y1(2), output3, X, 17, 1, Z),"  &
    "7  (BC_1, Y1(1), output3, X, 17, 1, Z),"  &
    "8  (BC_1, A2(4), input, X),"   &
    "9  (BC_1, A2(3), input, X),"   &
    "10 (BC_1, A2(2), input, X),"   &
    "11 (BC_1, A2(1), input, X),"   &
    "12 (BC_1, A1(4), input, X),"   &
    "13 (BC_1, A1(3), input, X),"   &
    "14 (BC_1, A1(2), input, X),"   &
    "15 (BC_1, A1(1), input, X),"   &
    "16 (BC_1, OE_NEG2, input, X)," &
    "16 (BC_1, *, control, 1),"     &
    "17 (BC_1, OE_NEG1, input, X)," &
    "17 (BC_1, *, control, 1)" ;

end sn74bct8244a;
```

SN74BCT8244A.vhd

```
--------------------------------------------------------------------------------
-- Title      : SN74BCT8244A
-- Project    : jtagemu
--------------------------------------------------------------------------------
-- File       : SN74BCT8244A.vhd
-- Author     : simon
--------------------------------------------------------------------------------
-- Description: JTAG model of SN74BCT8244A octal buffer with standard JTAG
--              boundary register cell
--------------------------------------------------------------------------------
-- Copyright (c) 2017 Simon Pichette
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
--------------------------------------------------------------------------------
entity SN74BCT8244A is
  port (
    OE_NEG1 : in  std_logic;
    Y1      : out std_logic_vector(1 to 4);
    Y2      : out std_logic_vector(1 to 4);
    A1      : in  std_logic_vector(1 to 4);
    A2      : in  std_logic_vector(1 to 4);
    OE_NEG2 : in  std_logic;
    TMS     : in  std_logic;
    TCK     : in  std_logic;
    TDI     : in  std_logic;
    TDO     : out std_logic);
end SN74BCT8244A;
--------------------------------------------------------------------------------
architecture jtagmodel of SN74BCT8244A is
  component BC_1 is
    port(Clock_DR : in std_logic;
      Shift_DR     : in  std_logic;
      Update_DR    : in  std_logic;
      Shift_in     : in  std_logic;
      Parallel_in  : in  std_logic;
      Mode         : in  std_logic;
      Shift_out    : out std_logic;
      Parallel_out : out std_logic);
  end component BC_1;

  component IR_1 is
    port(Clock_IR : in std_logic;
      Shift_IR        : in  std_logic;
      Update_IR       : in  std_logic;
      Shift_in        : in  std_logic;
      Capture_Data    : in  std_logic;
      Shift_out       : out std_logic;
      Instruction_bit : out std_logic;
      Reset_n         : in  std_logic);
  end component IR_1;

  component TAP_Controller is
    port(TMS : in std_logic;
      TCK       : in  std_logic;
      TRST_n    : in  std_logic;
      Clock_IR  : out std_logic;
```

```
      Shift_IR  : out std_logic;
      Update_IR : out std_logic;
      Clock_DR  : out std_logic;
      Shift_DR  : out std_logic;
      Update_DR : out std_logic;
      Reset_n   : out std_logic;
      Enable    : out std_logic;
      Select_o  : out std_logic);
  end component TAP_Controller;

  -- internal signals
  signal TDO_mux          : std_logic;
  signal TDO_out          : std_logic;
  signal TDO_q            : std_logic;
  signal Capture_DR_sig   : std_logic;
  signal Capture_IR_sig   : std_logic;
  signal Shift_IR_sig     : std_logic;
  signal Clock_DR_sig     : std_logic;
  signal Clock_IR_sig     : std_logic;
  signal Shift_DR_sig     : std_logic;
  signal Update_DR_sig    : std_logic;
  signal Update_IR_sig    : std_logic;
  signal Mode_sig         : std_logic := '0';
  signal Reset_n_sig      : std_logic;
  signal Enable_sig       : std_logic;
  signal Select_o_sig     : std_logic;
  signal IR_Select        : std_logic_vector(1 downto 0) := "10";
  signal brchain          : std_logic_vector(18 downto 0);
  signal irchain          : std_logic_vector(8 downto 0);
  signal Capture_Data_vec : std_logic_vector(7 downto 0)  := "10000001";
  signal Instruction      : std_logic_vector(7 downto 0)  := "11111111";
  signal Parallel_in_vec  : std_logic_vector(17 downto 0) := (others => '0');
  signal Parallel_out_vec : std_logic_vector(17 downto 0) := (others => '0');

  -- simple data registers without parallel-hold
  signal Bypass_cell : std_logic;
  signal ID_Code     : std_logic_vector(31 downto 0) := x"05046093";
  signal User_Code   : std_logic_vector(31 downto 0) := x"A5A55A5A";

  signal data_out : std_logic;
begin

  -- TDI distribution (no mux)
  irchain(8)  <= TDI; -- instruction register
  brchain(18) <= TDI; -- boundary register
                      -- TDI is also sent to Device ID reg,
                      --   User_Code reg and Bypass Reg

  -- TDO selection logic
  TDO_mux1 : with IR_Select select
  TDO_mux <=
    brchain(0)   when "00",
    ID_Code(0)   when "01",
    User_Code(0) when "10",
    Bypass_cell  when others;

  TDO_mux2 : TDO_out <= TDO_mux when Select_o_sig = '0' else irchain(0);

  TDO_reg : process(TCK)
```

```
  begin
    if falling_edge(TCK) then
      TDO_q <= TDO_out;
    end if;
  end process;

  -- TDO tri-state buffer emulation
  TDO <= TDO_q when Shift_IR_sig = '1' or Shift_DR_sig = '1' else '0';
  -- end TDO selection logic

  Bypass_register : process(Clock_DR_sig)
  begin
    if rising_edge(Clock_DR_sig) then
      if Shift_DR_sig = '0' then -- Capture_DR state
        Bypass_cell <= '0';
      else
        Bypass_cell <= TDI; -- Shift_DR state
      end if;
    end if;
  end process; -- Bypass_reg

  Device_ID_register : process(Clock_DR_sig)
  begin
    if rising_edge(Clock_DR_sig) then
      if Shift_DR_sig = '0' then -- Capture_DR state
        ID_Code <= x"05046093";
      else
        ID_Code <= TDI & ID_Code(31 downto 1);
      end if;
    end if;
  end process; -- Device_ID_register

  User_Code_register : process(Clock_DR_sig)
  begin
    if rising_edge(Clock_DR_sig) then
      if Shift_DR_sig = '0' then -- Capture_DR state
        User_Code <= x"A5A55A5A";
      else
        User_Code <= TDI & User_Code(31 downto 1);
      end if;
    end if;
  end process; -- User_ID_register

  Instruction_register : for i in 8 downto 1 generate
      ircellx : IR_1 port map (
        Clock_IR        => Clock_IR_sig,
        Shift_IR        => Shift_IR_sig,
        Update_IR       => Update_IR_sig,
        Shift_in        => irchain(i),
        Capture_Data    => Capture_Data_vec(i-1),
        Shift_out       => irchain(i-1),
        Instruction_bit => Instruction(i-1),
        Reset_n         => Reset_n_sig
      );
  end generate Instruction_register;

  Instruction_decoder : process(Update_IR_sig, Reset_n_sig)
  begin
    if Reset_n_sig = '0' then
```

```
      IR_Select <= "11"; -- Bypass on reset
    else
      if falling_edge(Update_IR_sig) then
        case Instruction is
          when "00000010" => -- Sample/Preload
            IR_Select <= "00";
            Mode_sig  <= '0';
          when "10000010" => -- Sample/Preload
            IR_Select <= "00";
            Mode_sig  <= '0';
          when "00000000" => -- EXTEST
            IR_Select <= "00";
            Mode_sig  <= '1';
          when "10000000" => -- EXTEST
            IR_Select <= "00";
            Mode_sig  <= '1';
          when others =>
            IR_Select <= "11";
            Mode_sig  <= '0';
        end case;
      end if;
    end if;
  end process; -- Instruction_decoder

  Boundary_register : for i in 18 downto 1 generate
      brcellx : BC_1 port map (
        Clock_DR     => Clock_DR_sig,
        Shift_DR     => Shift_DR_sig,
        Update_DR    => Update_DR_sig,
        Shift_in     => brchain(i),
        Parallel_in  => Parallel_in_vec(i-1),
        Mode         => Mode_sig,
        Shift_out    => brchain(i-1),
        Parallel_out => Parallel_out_vec(i-1)
      );
  end generate Boundary_register;

  -- connect external pins to boundary register

  -- inputs
  Parallel_in_vec(8)  <= A2(4);
  Parallel_in_vec(9)  <= A2(3);
  Parallel_in_vec(10) <= A2(2);
  Parallel_in_vec(11) <= A2(1);
  Parallel_in_vec(12) <= A1(4);
  Parallel_in_vec(13) <= A1(3);
  Parallel_in_vec(14) <= A1(2);
  Parallel_in_vec(15) <= A1(1);
  Parallel_in_vec(16) <= OE_NEG2;
  Parallel_in_vec(17) <= OE_NEG1;

  IO_WRITE : process(Parallel_out_vec(18))
  begin
    if (Parallel_out_vec(18) = '0') then
      Parallel_in_vec(20) <= OE_RESET;
    end if;
  end process;

  -- outputs
```

```
  Y2(4) <= Parallel_out_vec(0);
  Y2(3) <= Parallel_out_vec(1);
  Y2(2) <= Parallel_out_vec(2);
  Y2(1) <= Parallel_out_vec(3);
  Y1(4) <= Parallel_out_vec(4);
  Y1(3) <= Parallel_out_vec(5);
  Y1(2) <= Parallel_out_vec(6);
  Y1(1) <= Parallel_out_vec(7);

  -- Tri-state buffer control
  OE_RESET <= data_out when Parallel_out_vec(18) = '1' else 'Z';

  IO_READ : process (Parallel_out_vec(18))
  begin
    if (Parallel_out_vec(18) = '0') then
      data_out <= Parallel_out_vec(19);
    else
      data_out <= '0';
    end if;
  end process;

  TAP_FSM : TAP_Controller
    port map (
      TMS       => TMS,
      TCK       => TCK,
      TRST_n    => '1',
      Clock_IR  => Clock_IR_sig,
      Shift_IR  => Shift_IR_sig,
      Update_IR => Update_IR_sig,
      Clock_DR  => Clock_DR_sig,
      Shift_DR  => Shift_DR_sig,
      Update_DR => Update_DR_sig,
      Reset_n   => Reset_n_sig,
      Enable    => Enable_sig,
      Select_o  => Select_o_sig
    );

end; -- jtagmodel of SN74BCT8244A
--------------------------------------------------------------------------------
```

ANNEXE VI

VÉRIFICATION DU SIMULATEUR

Afin de vérifier le fonctionnement du simulateur, nous avons procédé à un grand nombre d'essais de génération de cas où nous avons comptabilisé le nombre de cas générés pour chaque classe de défectuosité (tableau-A VI-1, reprise du tableau 4.2) à l'exception de la classe 0, et chaque classe de défectuosité physique (tableau-A VI-2, reprise du tableau 6.2).

Tableau-A VI-1 Classes de défectuosités associées à un NodeResult

No.	Classe de défaut	Définition
0	Aucune défectuosité	Tous les résultats mesurés du nœud sont conformes aux résultats attendus.
1	Stuck-at 0	Le nœud comporte des erreurs et tous les résultats mesurés sont au niveau logique bas.
2	Stuck-at 1	Le nœud comporte des erreurs et tous les résultats mesurés sont au niveau logique haut.
3	Court-circuit	Le nœud comporte des erreurs et tous les résultats mesurés sont identiques à ceux d'un autre nœud.
4	Pont dominé par les 1	Le nœud comporte des erreurs et tous les résultats mesurés sont identiques à ceux d'un autre nœud lorsque ce dernier est au niveau logique haut. Les deux nœuds forment une porte logique OU-cablé (*Wired-OR*).
5	Pont dominé par les 0	Le nœud comporte des erreurs et tous les résultats mesurés sont identiques à ceux d'un autre nœud lorsque ce dernier est au niveau logique bas. Les deux nœuds forment une porte logique ET-cablé (Wired-AND).

Tableau-A VI-2 Classes de causes physiques utilisées par le prototype

Classe	Définition
Pièce manquante	La pièce en cause n'est pas présente sur la carte assemblée
Pièce mal alignée	La pièce est mal positionnée sur la carte après assemblage
Excès de soudure	Un excès de pâte pour ce composant cause une défaillance
Manque de soudure	Une quantité de pâte insuffisante pour ce composant cause une défaillance
Trace manquante	Une trace sur la carte est manquante ou brisée
Trace court-circuitée	Une trace sur la carte est connectée à une autre trace ou à une pastille

Nous avons ensuite comparé les totaux obtenus avec les probabilités d'occurrence demandées, puis répété l'opération pour les configurations suivantes :

1. Occurrences de base seulement
2. Occurrences de base et particularités de la carte B
3. Occurrences de base et particularités de la ligne 2
4. Occurrences de base et particularités de la carte B sur la ligne 2
5. Occurrences de base et particularités de la carte A sur la ligne 2

Le tableau-A VI-3, reprise du tableau 6.4, montre les occurrences de base des différentes causes physiques pour chacune des classes de défectuosités.

Tableau-A VI-3 Distribution de base des causes physiques par classe de défectuosité

Classe de défectuosité	**Pièce manquante**	**Alignement**	**Soudure**		**Routage**	
			Excès	**Manque**	**Circuit ouvert**	**Court-circuit**
Stuck-at-0	26%	13%	0%	34%	27%	0%
Stuck-at-1	26%	13%	0%	34%	27%	0%
Court-circuit	0%	19%	45%	0%	0%	36%
Pont dominé par 0	0%	19%	45%	0%	0%	36%
Pont dominé par 1	0%	19%	45%	0%	0%	36%

Les matrices présentées pour la vérification sont toutes organisées comme ce tableau. Les vecteurs représentant la proportion de cas générés pour chaque classe de défectuosité ont le format suivant :

[stuck_at_0, stuck_at_1, short, One_dominant_bridge, Zero_dominant_bridge].

Occurrences de base

Les résultats attendus pour des classes de défectuosités équiprobables et la distribution des causes physique de base sont montrés à la figure-A VI-1. Les sorties du simulateur indiquant les résultats sont présentés à la figure-A VI-2.

```
Distribution des classes de défectuosités (%)
[20, 20, 20, 20, 20]

Occurrences des causes physiques, par classe (%)
[26, 13, 0, 34, 27, 0]
[26, 13, 0, 34, 27, 0]
[ 0, 19, 45, 0, 0, 36]
[ 0, 19, 45, 0, 0, 36]
[ 0, 19, 45, 0, 0, 36]
```

Figure-A VI-1 Résultats attendus avec occurrences de base seulement

```
Simulator case generation stats for 20000 cases
----------------------------------------
     Fault Class     | total | percentage
----------------------------------------
Stuck_at_0           | 4026  | 20.1 %
Stuck_at_1           | 4058  | 20.3 %
Short                | 3891  | 19.5 %
One_dominant_bridge  | 3957  | 19.8 %
Zero_dominant_bridge | 4068  | 20.3 %
----------------------------------------
Physical cause distribution (%)
----------------------------------------
26.5 13.6 00.0 33.6 26.3 00.0
26.0 13.1 00.0 33.4 27.5 00.0
00.0 18.7 45.4 00.0 00.0 35.9
00.0 18.9 43.5 00.0 00.0 37.6
00.0 18.9 45.1 00.0 00.0 36.0
----------------------------------------
```

Figure-A VI-2 Résultats obtenus avec occurrences de base seulement

Occurrences de base et particularité de la carte B

Les résultats attendus pour une distribution des classes de défectuosités et des causes physique influencées par les particularités de la carte B sont présentées à la figure-A VI-3. Les sorties du simulateur indiquant les résultats sont présentés à la figure-A VI-4.

```
Distribution des classes de défectuosités (%)
[12, 12, 25, 25, 25]

Occurrences des causes physiques, par classe (%)
[26, 13, 0, 34, 27, 0]
[26, 13, 0, 34, 27, 0]
[ 0, 11, 49, 0, 0, 40]
[ 0, 11, 49, 0, 0, 40]
[ 0, 11, 49, 0, 0, 40]
```

Figure-A VI-3 Résultats attendus avec particularités de la carte B

```
Simulator case generation stats for 20000 cases
----------------------------------------
     Fault Class     | total | percentage
----------------------------------------
Stuck_at_0           | 2420  | 12.1 %
Stuck_at_1           | 2444  | 12.2 %
Short                | 5089  | 25.4 %
One_dominant_bridge  | 4946  | 24.7 %
Zero_dominant_bridge | 5101  | 25.5 %
----------------------------------------
Physical cause distribution (%)
----------------------------------------
25.7 13.0 00.0 34.8 26.6 00.0
27.1 13.6 00.0 32.4 26.8 00.0
00.0 11.1 48.4 00.0 00.0 40.5
00.0 11.2 49.9 00.0 00.0 39.0
00.0 11.5 50.3 00.0 00.0 38.2
----------------------------------------
```

Figure-A VI-4 Résultats obtenus avec particularités de la carte B

Occurrences de base et particularité de la ligne 2

Les résultats attendus pour une distribution des classes de défectuosités et des causes physique influencées par les particularités de la ligne 2 sont présentées à la figure-A VI-5. Les sorties du simulateur indiquant les résultats sont présentés à la figure-A VI-6.

```
Distribution des classes de défectuosités (%)
[12, 12, 25, 25, 25]

Occurrences des causes physiques, par classe (%)
[26, 13, 0, 34, 27, 0]
[26, 13, 0, 34, 27, 0]
[ 0, 12, 66, 0, 0, 22]
[ 0, 12, 66, 0, 0, 22]
[ 0, 12, 66, 0, 0, 22]
```

Figure-A VI-5 Résultats attendus avec particularités de la ligne 2

```
Simulator case generation stats for 20000 cases
----------------------------------------
     Fault Class     | total | percentage
----------------------------------------
Stuck_at_0           | 2449  | 12.2 %
Stuck_at_1           | 2489  | 12.4 %
Short                | 4984  | 24.9 %
One_dominant_bridge  | 5099  | 25.5 %
Zero_dominant_bridge | 4979  | 24.9 %
----------------------------------------
Physical cause distribution (%)
----------------------------------------
25.8 14.0 00.0 34.0 26.2 00.0
26.7 13.1 00.0 33.7 26.5 00.0
00.0 12.7 66.8 00.0 00.0 20.5
00.0 12.2 67.1 00.0 00.0 20.7
00.0 12.6 66.6 00.0 00.0 20.8
----------------------------------------
```

Figure-A VI-6 Résultats obtenus avec particularités de la ligne 2

Occurrences de base et particularité de la carte B sur la ligne 2

Les résultats attendus pour une distribution des classes de défectuosités et des causes physique influencées par les particularités de la carte B combinés à celles de la ligne 2 sont présentées à la figure-A VI-7. Les sorties du simulateur indiquant les résultats sont présentés à la figure-A VI-8. Les particularités de la carte B et de la ligne 2 sont synergiques, ce qui augmente dramatiquement les probabilités d'occurrence de certaines défectuosités.

```
Distribution des classes de défectuosités (%)
[4, 4, 31, 31, 31]

Occurrences des causes physiques, par classe (%)
[26, 13, 0, 34, 27, 0]
[26, 13, 0, 34, 27, 0]
[ 0,  4, 70, 0, 0, 26]
[ 0,  4, 70, 0, 0, 26]
[ 0,  4, 70, 0, 0, 26]
```

Figure-A VI-7 Résultats attendus avec particularités de la carte B sur la ligne 2

```
Simulator case generation stats for 20000 cases
----------------------------------------
     Fault Class     | total | percentage
----------------------------------------
Stuck_at_0           |  803  |  4.0 %
Stuck_at_1           |  812  |  4.1 %
Short                | 6208  | 31.0 %
One_dominant_bridge  | 6201  | 31.0 %
Zero_dominant_bridge | 5976  | 29.9 %
----------------------------------------
Physical cause distribution (%)
----------------------------------------
25.7 14.4 00.0 33.4 26.5 00.0
25.0 13.2 00.0 36.7 25.1 00.0
00.0 04.2 71.0 00.0 00.0 24.8
00.0 03.8 71.6 00.0 00.0 24.5
00.0 04.5 69.0 00.0 00.0 26.5
----------------------------------------
```

Figure-A VI-8 Résultats obtenus avec particularités de la carte B sur la ligne 2

Occurrences de base et particularité de la carte A sur la ligne 2

Les résultats attendus pour une distribution des classes de défectuosités et des causes physique influencées par les particularités de la carte A combinés à celles de la ligne 2 sont présentées à la figure-A VI-9. Les sorties du simulateur indiquant les résultats sont présentés à la figure-A VI-10. Les particularités de la carte A et de la ligne 2 n'influencent pas les mêmes classes de défectuosités ni les mêmes causes physique, la distribution obtenue est donc moins concentrée que dans le cas de l'exemple précédent.

```
Distribution des classes de défectuosités (%)
[23, 23, 18, 18, 18]

Occurrences des causes physiques, par classe (%)
[17, 8,  0, 57, 17, 0]
[17, 8,  0, 57, 17, 0]
[ 0, 12, 66, 0, 0, 22]
[ 0, 12, 66, 0, 0, 22]
[ 0, 12, 66, 0, 0, 22]
```

Figure-A VI-9 Résultats attendus avec particularités de la carte A sur la ligne 2

```
Simulator case generation stats for 20000 cases
----------------------------------------
     Fault Class     | total | percentage
----------------------------------------
Stuck_at_0           | 4662  | 23.3 %
Stuck_at_1           | 4648  | 23.2 %
Short                | 3644  | 18.2 %
One_dominant_bridge  | 3572  | 17.9 %
Zero_dominant_bridge | 3474  | 17.4 %
----------------------------------------
Physical cause distribution (%)
----------------------------------------
16.8 08.3 00.0 58.0 16.9 00.0
16.7 08.7 00.0 58.1 16.5 00.0
00.0 10.7 68.8 00.0 00.0 20.5
00.0 11.7 66.7 00.0 00.0 21.6
00.0 13.1 66.5 00.0 00.0 20.4
----------------------------------------
```

Figure-A VI-10 Résultats obtenus avec particularités de la carte A sur la ligne 2

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

Amati, L., Bolchini, C., Frigerio, L., Salice, F., Eklow, B., Suvatne, A., & Martin, M. (2009, octobre). *An Incremental Approach to Functional Diagnosis.* Paper presented at the Defect and Fault Tolerance in VLSI Systems, 2009. DFT '09. 24th IEEE International Symposium on.

Amrouch, H., Chowdhury, A. B., Jin, W., Karri, R., Khorrami, F., Krishnamurty, P., Polian, I., van Santen, V. M., Tan, B. & Tan, X. D. (2021, octobre) *Special Session: Machine Learning for Semiconductor Test and Reliability.* Paper presented at the 2021 IEEE International Test Conference (ITC).

Binu, D., & Kariyappa, B. S. (2017). A survey on fault diagnosis of analog circuits: Taxonomy and state of the art. *International Journal of Electronics and Communications, 73*(1), 68-83.

Bolchini, C., Quintarelli, E., Salice, F., & Garza, P. (2013, octobre). *A data mining approach to incremental adaptive functional diagnosis.* Paper presented at the Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2013 IEEE International Symposium on.

Bolchini, C., & Cassano, L. (2014, octobre). *Machine learning-based techniques for incremental functional diagnosis: A comparative analysis.* Paper presented at the Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2014 IEEE International Symposium on.

Boydon, C. J. I. S., Wu, Y.-H., & Wu, C.-H. (2021). *Data-driven Scheduling for High-mix and Low-volume Production in Semiconductor Assembly and Testing*. Dans 2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)

Buchanan, B. G. & Wilkins, D. C. (1993) *Readings in Knowledge Acquisition and Learning : Automating the Construction and Improvement of Expert Systems*. Morgan Kaufman Publishing.

Buettner, R., Bilo, M., Bay, N., & Zubac, T. (2020, juillet). *A Systematic Literature Review of Medical Image Analysis Using Deep Learning*. Paper presented at the 2020 IEEE Symposium on Industrial Electronics & Applications.

Chess, B., & Larrabee, T. (1999). Creating small fault dictionaries [logic circuit fault diagnosis]. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 18*(3), 346-356.

Chowdhury, A. B., Tan, B., Garg, S., & Karri, R. (2022). Robust Deep Learning for IC Test Problems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 41*(1), 183-195.

Dendani, N., Khadir, M., & Guessoum, S. (2012, novembre). *Hybrid Approach for Fault Diagnosis Based on CBR and Ontology: using jCOLIBRI framework.* Paper presented at the 2012 IEEE International Conference on Complex Systems (ICCS).

Evans, E. (2004). *Domain-Driven Design : Tackling Complexity in the Heart of Software.* Addison-Wesley Professionnal.

Fenton, W. G., McGinnity, T. M., & Maguire, L. P. (2001). Fault diagnosis of electronic systems using intelligent techniques: a review. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 31*(3), 269-281.

Fenton, W. G., McGinnity, T. M., & Maguire, L. P. (2002). Fault Diagnosis of Electronic Systems : Using Artificial Intelligence. *IEEE Instrumentation & Measurement Magazine*, *5*(3), 16-20.

Gomez-Uribe, C. A. & Hunt, N. (2015). The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Transactions on Management Information Systems 6*(4), Article 13, 1-19.

Huang, W., Wei, P., Zhang, M., & Liu, H. (2019, juillet). *HRIPCB: a challenging dataset for PCB defects detection and classification*. Paper presented at 3rd Asian Conference on Artificial Intelligence Technology (ACAIT 2019).

Hyafil, L. & Rivest, R. L. (1976). Constructing optimal binary decision trees is NP-complete. *Information Processing Letters, 5*(1), 15-17.

Ibarra, O. H., & Sahni, S. (1975). Polynomially complete fault detection problems. *IEEE Tramsactions on Computers, C24*(3), 242-250.

Jin, S., Ye, F., Zhang, Z., Chakrabarty, K. (2016) Efficient Board-Level Functional Fault Diagnosis With Missing Syndromes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 35*(6), 985-998.

Lirov, Y. (1989). Electronic Circuit Diagnostic Expert Systems – A Survey. *Computers and Mathematics with Applications, 18*(4), 381-398.

Liu, M., Li, X., Chakrabarty, K., & Gu, X. (2022). Knowledge Transfer in Board-Level Functional Fault Diagnosis Enabled by Domain Adaptation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 41*(3), 762-775.

Meixner, A. (2020). Demand Grows for Reducing PCB Defects. *Semiconductor Engineering*. Repéré à https://semiengineering.com/demand-grows-for-reducing-pcb-defects/

Nigh, C., Bhargava, G., & Blanton, R. D. (2021, octobre). *AAA: Automated On-ATE AI Debug of Scan Chain Failures*. Paper presented at the 2021 IEEE International Test Conference (ITC).

Okusa, K., Okazaki, T., & Yasuda, S. (2020). *A Statistical Study on Highly Accurate Quality Prediction for High-mix Low-Volume Semiconductor Products*. Dans 2020 International Symposium on Semiconductor Manufacturing (ISSM*)*.

Parker, K. P. (2016). *The Boundary Scan Handbook, Fourth Edition*. London: Springer International Publishings.

Pichette, S. & Thibeault, C. (2022). *Knowledge-Intensive Diagnostics using Case-Based Reasoning and Synthetic Case Generation.* Manuscrit soumis pour publication à IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

Richter, M. M., & Aamodt, A. (2005). Case-based reasoning foundations. *The Knowledge Engineering Review*, *20*(3), 203-207.

Richter, M. M. & Weber, R. O. (2013). *Case-Based Reasoning : A Textbook*. Berlin Heidelberg : Springer Verlag.

Rowland, J. G. & Jain, L. C. (1993). Knowledge Based Systems for Instrumentation Diagnosis, System Configuration and Circuit and System Design. *Engineering Applications of Artificial Intelligence*, *6*(5), p. 437-446.

Siemens EDA. (2021). ODB++ Design Format Specification, Release 8.1 Update 3. Repéré à https://odbplusplus.com/wp-content/uploads/sites/2/2021/02/odb_spec_user.pdf

Simpson, W. R., & Sheppard, J. W. (1996, septembre). *Encapsulation and diagnosis with fault dictionaries.* Paper presented at the AUTOTESTCON '96, Test Technology and Commercialization.

Sprovieri, J. (2004). Managing High-Mix, Low-Volume Assembly. *assemblymag.com*. Repéré à https://www.assemblymag.com/articles/83764-managing-high-mix-low-volume-assembly

Stone, J., Greenwald, M., Partridge, C., & Hughes, J. (1998). Performance of checksums and CRCs over real data. *IEEE/ACM Transactions on Networking*, *6*(5), 529-543.

Sugimatsu, H., Hori, S., Sawada, A., & Azuma, K. (1994, novembre). *A CBR application: service productivity improvement by sharing experience.* Paper presented at the Emerging Technologies and Factory Automation, 1994. ETFA '94., IEEE Symposium on.

Totton, K. A. E., & Limb, P. R. (1991, novembre). *Experience in using neural networks for electronic diagnosis.* Paper presented at the 1991 Second International Conference on Artificial Neural Networks.

Tourangeau, S., & Eklow, B. (2006, octobre). *Test Economics - What can a Board/System Test Engineer do to Influence Supply Operation Metrics.* Paper presented at the Test Conference, 2006. ITC '06. IEEE International.

Ucamco NV. (2021). The Gerber Layer Format Specification Revision 2021.11. Repéré à https://www.ucamco.com/files/downloads/file_en/451/gerber-layer-format-specification-revision-2021-11_en.pdf

Watt, J, Borhani, R., & Katsaggelos, A. K. (2020). *Machine Learning Refined : Foundations, Algorithms and Applications, Second Edition*. Cambridge University Press.

Winston, P. (1984). *Artificial Intelligence.* New York: Addison-Wesley.

Xilinx. (2018). UltraScale Architecture Libraries Guide, UG974 (v2018.3). Repéré à https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug974-vivado-ultrascale-libraries.pdf

Ye, F., Zhang, Z., Chakrabarty, K., & Gu, X. (2012, novembre). *Adaptive Board-Level Functional Fault Diagnosis Using Decision Trees.* Paper presented at the Test Symposium (ATS), 2012 IEEE 21st Asian.

Ye, F., Zhang, Z., Chakrabarty, K., & Gu, X.. (2013, mai). *Information-theoretic syndrome and root-cause analysis for guiding board-level fault diagnosis.* Paper presented at the Test Symposium (ETS), 2013 18th IEEE European.

Ye, F., Zhang, Z., Chakrabarty, K., & Gu, X. (2014). Board-Level Functional Fault Diagnosis Using Multikernel Support Vector Machines and Incremental Learning. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 33*(2), 279-290.

Ye, F., Zhang, Z., Chakrabarty, K., & Gu, X. (2015). Information-Theoretic Syndrome Evaluation, Statistical Root-Cause Analysis, and Correlation-Based Feature Selection for Guiding Board-Level Fault Diagnosis, *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 34*(6), 1014-1026.

Zhang, X. (2012, octobre). *Model-based design of diagnostic system.* Paper presented at the System Science, Engineering Design and Manufacturing Informatization (ICSEM), 2012 3rd International Conference on.

Zhang, Z., & Chakrabarty, K. (2011, septembre). *Smart Diagnosis: Efficient Board-level Diagnosis and Repair using Artificial Neural Networks.* Paper presented at the 2011 IEEE International Test Conference (ITC).