Méthodologie de test pour cartes mixtes analogiques-numériques

Bertrand Gilles

THÈSE / UNIVERSITÉ DE BRETAGNE OCCIDENTALE
sous le sceau de l'Université européenne de Bretagne

pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE
Mention : Informatique
École Doctorale SICMA

présentée par

Bertrand GILLES

Préparée à l'équipe d'accueil 3883 - LISyC
Université de Brest

Méthodologie de test pour cartes mixtes analogiques-numériques

Thèse soutenue le 7 juillet 2009
devant le jury composé de :

Jacques TISSEAU
Professeur, ENIB / *président du jury*
Michel RENOVELL
Directeur de recherche, CNRS - LIRMM / *rapporteur*
Chantal ROBACH
Professeur, ESISAR - INP Grenoble / *rapporteur*
Vincent BEROULLE
Maître de conférences, ESISAR - INP Grenoble / *examinateur*
Dominique DUHAUT
Professeur, Université de Bretagne Sud / *examinateur*
Yvon KERMARREC
Professeur, Télécom Bretagne / *examinateur*
Laurent NANA TCHAMNDA
Professeur, Université de Bretagne Occidentale / *directeur de thèse*
Valérie-Anne NICOLAS
Maître de conférences, Université de Bretagne Occidentale / *examinateur*
Michel LE GOFF
Responsable de l'agence ISIS-MPP de Brest / *invité*

« ... in palmis semper parens juventus. In palmis resurgo. »
Von Martius

Remerciements

Je tiens tout d'abord à remercier les membres de mon jury de thèse :

- Jacques Tisseau, professeur, directeur de l'ENIB et du LISyC, qui m'a fait l'honneur de présider ce jury.

- Michel Renovell, directeur de recherche CNRS – LIRMM, ainsi que Chantal Robach, professeur, directeur de l'ESISAR – INP Grenoble, d'avoir bien voulu accepter la lourde tâche de rapporteur. Je les remercie vivement pour leur lecture attentive du document ainsi que pour leurs commentaires avisés.

- Dominique Duhaut, professeur à l'Université de Bretagne Sud, Yvon Kermarrec, professeur à Télécom Bretagne, et Vincent Beroulle, maître de conférences à l'ESISAR – INP Grenoble, d'avoir lu en détail le document.

- Laurent Nana Tchamnda, professeur à l'Université de Bretagne Occidentale, qui a dirigé ma thèse, et Valérie-Anne Nicolas, maître de conférences à l'Université de Bretagne Occidentale, qui l'a co-encadrée. Leur soutien a été important tout au long de mon travail. Leurs relectures du document ont été très précieuses, ainsi que leurs conseils avisés lors de la préparation de la soutenance.

- Michel Legoff, directeur de l'agence brestoise de la société ISIS-MPP, qui a collaboré activement aux travaux présentés dans ce document.

Je remercie également Bruno Castel de la société ISIS-MPP qui m'a fait largement bénéficier de sa grande expérience du test matériel.

J'ai une pensée émue pour Lionel Marcé, professeur à l'Université de Bretagne Occidentale, qui nous a quittés en mars 2006. Il fut mon premier directeur de thèse. Je lui suis reconnaissant de la confiance qu'il m'a toujours témoignée.

La thèse est un travail long et difficile. Aussi, les amis sont importants. Qu'ils soient ici remerciés.

Le soutien de la famille est également essentiel. Je remercie mes très chers parents, ma « petite » sœur Karine, John, Paul, Claire et Alex qui ont toujours cru en moi. Merci à Karine pour l'organisation de la « cérémonie » du 21 juillet 2009 à Belfast !

Enfin, merci à Lydie pour son amour et sa patience.

Brest, le 21 septembre 2009.

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Introduction

Le test est une activité qui intervient dans les différentes phases du cycle de vie d'un circuit, d'une carte ou d'un système électronique, avec différents objectifs. Ainsi, au niveau composant, le concepteur d'un circuit intégré oriente sa conception dans le but de rendre ce dernier plus testable (*Design For Testability*). Par ailleurs, en production, il est nécessaire de tester de manière exhaustive tous les circuits produits afin d'éliminer ceux présentant un défaut de fabrication. Au niveau carte, le test de production consiste souvent à tester unitairement chaque composant monté sur la carte, ainsi que les interconnexions entre les composants. Au niveau système, l'activité de test consiste à vérifier au final que le système satisfait ses spécifications fonctionnelles.

Un circuit électronique est caractérisé par son type : analogique, numérique ou mixte. Un circuit mixte est un circuit qui possède à la fois des fonctions analogiques et des fonctions numériques. Une carte mixte est composée de circuits de types différents. De nombreux travaux ont été effectués sur le test de production et sur la conception en vue d'un meilleure testabilité des circuits. Les résultats de ces travaux ont abouti à l'élaboration de nombreuses méthodes et outils de test.

Les travaux présentés dans ce document s'intéressent au test en maintenance de cartes mixtes. Il n'existe pas, à notre connaissance, de méthodologie de test et d'outil de test dédiés spécifiquement au test de cartes électroniques mixtes en phase de maintenance. Ce manque est principalement dû au contexte de maintenance lui-même, pour au moins deux raisons.

Premièrement, la phase de maintenance cible des cartes électroniques à longue durée de vie. On peut s'étonner de considérer une grande durée de vie alors que l'explosion des systèmes embarqués grand public (téléphone portable, assistant personnel, appareil photo numérique, navigateur GPS, $\cdots$), les prix de plus en plus bas et le marketing incitent les clients à renouveler de plus en plus rapidement leur matériel. Cependant, certains systèmes, et tout particulièrement les systèmes militaires et avioniques, se doivent de rester opérationnels pendant plusieurs dizaines d'années. Il est alors important, pour la sûreté de fonctionnement, de vérifier que les fonctionnalités des cartes électroniques composant ces systèmes ne se dégradent pas au cours du temps (maintenance préventive). D'autre part, il est également nécessaire de pouvoir diagnostiquer les pannes dans les cartes électroniques dans le but de les réparer (maintenance corrective). C'est particulièrement important dans le cas où les lots de rechanges sont épuisés.

Deuxièmement, la phase de maintenance se trouve très en aval du cycle de vie de la carte. Ce positionnement tardif, couplé à la longévité des cartes ciblées, entraîne très

souvent une connaissance réduite de la carte pour les personnes impliquées dans le test en maintenance. En effet, des informations concernant les spécifications et la conception peuvent (involontairement ou volontairement pour des raisons de confidentialité) être totalement ou partiellement indisponibles.

Actuellement, les ingénieurs de test en maintenance doivent faire face à diverses situations. En effet, ils doivent tester des cartes qui peuvent être numériques, analogiques ou mixtes, et pour lesquelles la qualité et la quantité d'information est variable. Ils peuvent donc rencontrer des situations idéales où la totalité de la documentation d'origine est disponible, des situations intermédiaires où la documentation est partiellement disponible, et des situations difficiles où aucune documentation n'est disponible. Aussi, les ingénieurs de test se doivent de posséder une expérience importante. Leurs méthodes, souvent empiriques induisent des coûts élevés pour le test. Clairement, il manque des méthodologies de test qui permettent d'automatiser au moins partiellement le test de cartes mixtes en phase de maintenance.

Le travail effectué dans le cadre de cette thèse vise deux buts. Le premier but est de proposer une méthodologie de test adaptée au test en maintenance des cartes mixtes. Le deuxième but consiste à développer un outil *semi-automatique* qui met en oeuvre cette méthodologie. L'outil doit être capable de générer automatiquement des données de test à partir de la modélisation d'une carte et de stratégies et tactiques de test prenant en compte le savoir-faire des ingénieurs en matière de test.

Le document de thèse se décompose en deux parties :

- La première partie du document présente un état de l'art sur le test de circuits intégrés et de cartes électroniques. Une carte électronique étant principalement composée de circuits intégrés, il nous a semblé important de commencer cette partie par un panorama des principales méthodes de test des circuits intégrés numériques, analogiques et mixtes. Dans ce cadre, nous avons mis l'accent sur le test de production car c'est celui qui a très certainement été le plus automatisé. Nous abordons ensuite le test de cartes à proprement parler, puis nous terminons cet état de l'art par un bilan.
- La deuxième partie du document présente notre travail de thèse. Nous commençons par présenter la méthodologie que nous proposons. Nous décrivons ensuite la mise en oeuvre de la méthodologie et introduisons l'outil résultant de cette dernière. Nous présentons ensuite les résultats obtenus à partir de différents cas d'étude et terminons par un bilan de notre travail et quelques perspectives.

Première partie

État de l'art du test des circuits et des cartes électroniques

Chapitre 1

Contexte

Nous avons précisé en introduction que nous nous intéressons au test de cartes mixtes en phase de maintenance. Nous nous positionnons ainsi dans un contexte caractérisé par un niveau particulier (niveau carte), un type de carte particulier (carte mixte) et une phase du cycle de vie particulière (maintenance). Avant d'aborder le test de cartes, il nous semble nécessaire d'examiner le test de circuits intégrés étant donné qu'une carte électronique est composée en grande partie de circuits intégrés. De même, la nature (analogique, numérique ou mixte) d'un circuit influe sur la manière dont celui-ci peut être testé. Aussi est-il important de s'intéresser aux méthodes de test développées spécifiquement pour chaque nature de circuits. Les objectifs du test d'un circuit varient en fonction des différentes phases qui constituent le cycle de vie du circuit. Le test de circuits intégrés en phase de production bénéficie de travaux importants de la part de la communauté du test. La complexité toujours croissante des circuits et les délais de mise sur le marché nécessitent des méthodes de test de production automatisées avec un temps d'application le plus court possible. L'aspect automatisation fortement ciblé par le test de production nous intéresse car, dans le contexte complexe de la maintenance, nous cherchons aussi à automatiser le plus possible le processus de test.

Nous commençons par présenter le test dans le cycle de vie des circuits intégrés au chapitre 2, puis nous décrivons au chapitre 3 les différentes méthodes adaptées au test de production des circuits intégrés numériques, analogiques et mixtes. Ces deux chapitres permettent de mieux comprendre le test de cartes électroniques abordé au chapitre 4. Nous terminons cette partie par un bilan au chapitre 5.

Chapitre 2

Le test dans le cycle de vie des circuits intégrés

L'objectif du test d'un circuit intégré n'est pas unique. Il varie en fonction des différentes phases qui constituent le cycle de vie du circuit. On distingue ainsi quatre types de tests [BA00] qui sont le *test de caractérisation*, le *test de production*, le *test de déverminage* et le *test de contrôle d'entrée*.

Le test de caractérisation est réalisé sur les premiers prototypes du circuit fabriqués à l'issue de la phase de conception. Son objectif est de vérifier que la conception du circuit est correcte et que celui-ci satisfait ses spécifications. Lorsque la conception du circuit est validée, le circuit entre alors en phase de production, c'est-à-dire qu'il est fabriqué à plus ou moins grande échelle. Le test de production est relatif à la fabrication du circuit et son but est de s'assurer que celle-ci est correcte.

Le test de production se doit de détecter les défauts de fabrication des circuits et ainsi de différencier les bons circuits des mauvais. La durée de vie des circuits ayant passé le test de production est très inégale : certains circuits deviennent défaillants rapidement (mortalité infantile et défaillances aléatoires) alors que d'autres fonctionneront correctement pendant longtemps. En effet, la fiabilité des composants électroniques est décrite par une courbe en forme de « baignoire » [Wil02].

Le but du test de déverminage est de s'assurer de la fiabilité des circuits en les faisant fonctionner pendant un certain temps dans des conditions environnementales stressantes. Ainsi, les circuits potentiellement défaillants sont éliminés.

Le test de contrôle d'entrée concerne les circuits qui ont déjà été livrés chez un client qui est un fabricant de cartes ou de systèmes électroniques. Il est nécessaire, pour un systémier, de tester à nouveau les circuits avant leur assemblage sur un circuit imprimé (*Printed Circuit Board ou PCB*). En effet, plus un défaut dans un circuit est détecté tard dans le processus d'intégration du système, plus le coût de remplacement du circuit est important. La nature du test de contrôle d'entrée dépend du domaine d'application du système final et de ses contraintes. Ainsi, ce test peut, pour un circuit et une application donnée, reprendre un sous-ensemble plus ou moins complet du test de production

ou être complètement personnalisé.

Il existe deux grandes approches pour le test de circuits : l'approche structurelle et l'approche fonctionnelle. Celles-ci sont décrites ci-après.

Le test structurel fait appel au concept du test en « boîte blanche ». Une boîte blanche représente un circuit dont la structure est connue. Le but du test consiste à vérifier l'intégrité du circuit en se basant sur sa structure. Un *défaut* représente une différence imprévue entre la conception et l'implémentation du circuit. L'élaboration du test consiste à créer le ou les vecteurs de test qui permettent de détecter une *défaillance* du circuit, i.e. l'effet d'un défaut visible aux sorties primaires du circuit. Cette approche du test est dite *orientée défaut (Defect-oriented Test ou DOT)*. Celle-ci repose sur l'utilisation d'algorithmes basés sur des *modèles de fautes*. Un modèle de faute représente à un niveau d'abstraction plus ou moins élevé (électrique, logique, ...) un défaut. La représentation abstraite d'un défaut doit combiner une bonne précision dans la modélisation du défaut physique et une simplicité de mise en oeuvre concernant la génération des données de test. Nous présenterons plus en détail certains modèles de fautes et leur utilisation dans la génération des données de test lorsque nous aborderons le test structurel des circuits numériques et des circuits analogiques/mixtes.

Le test fonctionnel utilise le concept du test en « boîte noire ». Contrairement au principe de la boîte blanche, la boîte noire représente un circuit dont la structure est inconnue. L'approche du test se focalise alors sur la fonctionnalité du circuit, i.e. son comportement et ses performances. Il consiste à vérifier que le circuit satisfait des spécifications fonctionnelles. Ainsi, le test est *basé sur les spécifications (Specification-based Test)*. Il consiste à mesurer les paramètres fonctionnels du circuit en analysant ses réponses pour des stimuli d'entrée donnés obtenus par la couverture totale ou partielle des spécifications.

Chapitre 3

Test de circuits intégrés

Un critère possible de classification des circuits électroniques est la nature des signaux qu'ils gèrent. Ainsi, il est possible de distinguer les circuits *numériques*, les circuits *analogiques* et les circuits *hybrides* ou *mixtes*. Les circuits numériques gèrent uniquement des signaux dont le niveau prend deux valeurs logiques : zéro et un. Par opposition, les circuits analogiques manipulent des signaux continus en amplitude et dans le temps, et qui peuvent ainsi prendre une infinité de valeurs. Alors que la frontière séparant les circuits numériques et analogiques est franche, il en va autrement pour les circuits mixtes qui contiennent à la fois des composants numériques et analogiques, gérant ainsi des signaux hétérogènes. Par la suite, nous considérons que la *nature* d'un circuit correspond à celle des signaux qu'il gère.

Dans ce chapitre, nous nous intéressons aux méthodes développées pour le test de production des circuits intégrés comme nous l'avons évoqué au chapitre 2. Nous avons choisi de nous concentrer sur le test de production car c'est certainement celui qui a été le plus automatisé. L'automatisation du test de production permet de tester rapidement les circuits fabriqués à grande échelle. Le but du test de production pour un circuit est de détecter les défauts de fabrication de celui-ci, la conception du circuit étant supposée bonne. Pour comprendre le fondement de ces méthodes de test, il est nécessaire de connaître l'origine et l'effet des défauts de fabrication affectant les circuits intégrés (section 3.1). Par ailleurs, le type de circuit influe sur la manière de tester. Les méthodes de test pour les circuits numériques sont décrites en section 3.2. Celles relatives aux circuits analogiques et mixtes sont décrites en section 3.3.

3.1 Défauts de fabrication

Les défauts de fabrication d'un circuit intégré sont généralement de deux types : les défauts dont l'origine provient de l'environnement de fabrication et les défauts causés par une variation du processus de fabrication [Rob96]. Une particule de poussière qui se dépose sur la tranche de silicium du circuit au cours de sa fabrication constitue un exemple de défaut dû à l'environnement. Un exemple de défaut causé par une variation du processus de fabrication est un mauvais alignement des masques.

Les variations du processus de fabrication engendrent deux types de défauts : les défauts globaux et les défauts locaux. Un défaut global correspond à une variation systématique d'un même paramètre. Par exemple, une variation hors tolérance de la tension de seuil de tous les transistors du circuit est représentative d'un défaut global. Par opposition, un défaut local correspond à de petites variations aléatoires qui apparaissent entre des composants adjacents. Ces petites variations sont appelées *erreur d'appariement (mismatch error)*.

Un défaut sur un composant du circuit provoque une défaillance *catastrophique* ou *paramétrique*. Une défaillance catastrophique correspond à une destruction ou un comportement incontrôlable du composant alors qu'une défaillance paramétrique va dégrader son fonctionnement.

Il est important de comprendre que l'effet des défauts est très différent suivant la nature du circuit [Rob96]. Nous présentons ci-après l'effet des défauts sur les circuits numériques et analogiques.

3.1.1 Effet des défauts sur les circuits numériques

Les circuits numériques ne sont pas sensibles aux erreurs d'appariement. Cela est dû au fait que le circuit est composé de transistors fonctionnant en mode de commutation (bloqué ou saturé), i.e. en niveaux logiques (un ou zéro). En d'autres termes, de légères variations affectant les caractéristiques des transistors physiquement proches d'un composant ne modifient pas la fonction logique du circuit.

Il n'en est pas de même concernant les effets des défauts globaux ou ceux dus à l'environnement de fabrication qui sont aléatoires. Prenons l'exemple d'un défaut environnemental où une particule de poussière se dépose sur un transistor MOS lors du dépôt de polysilicium. Si la particule couvre entièrement la grille d'un transistor MOS, cette dernière sera détruite et s'en suivra alors une défaillance catastrophique. Dans le cas d'un recouvrement partiel, une partie de la grille existe. Il s'en suit alors une défaillance paramétrique correspondant à un fonctionnement dégradé.

3.1.2 Effet des défauts sur les circuits analogiques

Comme les circuits numériques, les circuits analogiques sont affectés par les défauts globaux ainsi que ceux dus à l'environnement de fabrication. Cependant, contrairement aux circuits numériques, les circuits analogiques sont également très sensibles aux erreurs d'appariement. En effet, un circuit analogique accomplit une fonction qui est caractérisée par un ensemble de paramètres et il existe des relations de dépendance entre ses paramètres et les valeurs de ses composants. Ainsi, une légère variation de la valeur d'un composant par rapport à sa valeur nominale modifie un (ou plusieurs) paramètre du circuit. Illustrons ceci en considérant le filtre analogique RL du premier ordre montré en figure 3.1.

La fréquence de coupure du filtre est un de ses paramètres et est donnée par la relation :

$$F_0 = \frac{R}{2\pi L}$$

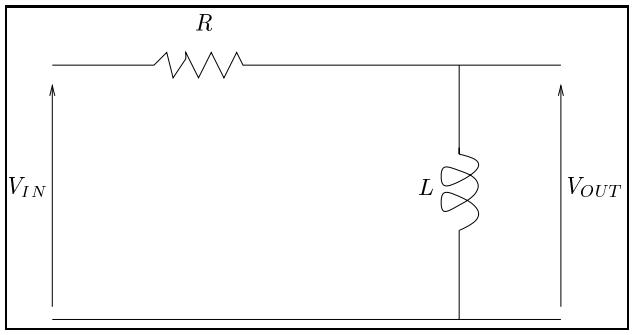

FIG. 3.1 – Filtre RL

On voit aisément qu'une erreur d'appariement modifiant les valeurs nominales respectives de la résistance R et de l'inductance L va avoir pour conséquence de modifier la fréquence de coupure du filtre F_0 (défaillance paramétrique).

De manière générale, les erreurs d'appariement limitent les performances des circuits analogiques et contribuent fortement à rendre le test de circuits analogiques et mixtes difficile.

3.2 Test de circuits numériques

Nous présentons les principales méthodes de test structurel (section 3.2.1) et fonctionnel (section 3.2.2) pour les circuits numériques.

3.2.1 Test structurel de circuits numériques

Nous avons déjà présenté, dans le chapitre 2, le test structurel comme un test orienté défaut et introduit la notion de modèle de faute. Nous présentons dans cette section un aperçu des principaux modèles de fautes utilisés pour les circuits numériques intégrés. Nous discutons ensuite de la simulation de fautes, puis de la génération automatique des vecteurs de test. Enfin, nous présentons les techniques de conception en vue d'une meilleure testabilité.

3.2.1.1 Modèles de fautes

Le premier modèle de faute à avoir été largement étudié est le *modèle de collage simple (single stuck-at line model)* [Hay85]. Dans ce modèle, on considère les collages permanents à 0 ou à 1 sur les différentes lignes du circuit décrit sous forme de portes logiques interconnectées. Un collage consiste à couper virtuellement une ligne et appliquer un signal constant qui peut prendre les valeurs logiques 0 ou 1. Ce modèle a été et reste largement utilisé, principalement parce qu'il permet de représenter de nombreux défauts et qu'il est indépendant de la technologie. Une extension de ce modèle est le *modèle de collage multiple* [ABF90] dans lequel plusieurs lignes peuvent être collées simultanément à 0 ou à 1.

Le *modèle de court-circuit* (bridging fault) [MJR86] a été introduit dans le but de modéliser des défauts non couverts par les modèles précédents. Ce modèle considère le cas où plusieurs lignes du circuit sont en contact (court-circuit).

Les modèles de fautes décrits précédemment ne permettent pas de représenter des défauts affectant une technologie particulière. Par exemple, on constate des défauts spécifiques affectant les transistors utilisés dans la technologie CMOS [Cas76]. De nouveaux modèles de fautes ont été introduits pour représenter ces défauts [Wad78] : *transistor collé ouvert (stuck-off)*, *transistor collé passant (stuck-on)*, *court-circuit (short)* et *circuit ouvert (open)*.

Plus récemment, toujours pour la technologie CMOS, de nouveaux modèles plus réalistes ont été proposés, comme par exemple les courts-circuits résistifs [RHB95] : des études ont montré qu'un court-circuit entre lignes possède une résistance intrinsèque [RMBF92] qui est négligée par le modèle de court-circuit classique décrit plus haut.

Enfin, nous terminons cette présentation succincte des modèles de fautes en introduisant les *fautes de délais* qui modélisent les *défauts temporels*. Un défaut temporel sur un circuit affecte les délais de propagation (retards) dans celui-ci. Il en résulte que certaines valeurs logiques sont positionnées en retard en différents points du circuit. Cela provoque dans le meilleur des cas un résultat bon hors délai et dans le cas le moins favorable un résultat faux. Une très bonne synthèse des différents modèles de fautes de délais est effectuée dans [MA97].

3.2.1.2 Simulation de fautes

Le but de la simulation de fautes est de déterminer, pour un circuit, l'ensemble des fautes détectées par un vecteur d'entrée donné. La simulation de fautes consiste, pour un vecteur d'entrée donné et une liste de fautes (préétablie ou déduite de la structure du circuit), à classer chacune des fautes comme *détectée* ou *non détectée*. Une faute est détectée lorsque la réponse (obtenue par simulation) d'un circuit affecté par cette faute, pour un vecteur d'entrée donné est différente de celle du circuit sain.

La simulation de fautes peut être utilisée après ou pendant la génération des vecteurs de test. Après le processus de génération des vecteurs de test (section 3.2.1.3), elle permet de calculer le *taux de couverture (fault coverage)* de l'ensemble des vecteurs de test obtenus. Le taux de couverture est le rapport entre le nombre de fautes détectées et le nombre de fautes à détecter. Utilisée pendant le processus de génération des vecteurs de test, elle permet de produire un ensemble de vecteurs de test assurant un taux de couverture donné, pour un modèle de faute donné.

Nous introduisons maintenant les principaux algorithmes de simulation de fautes. La simulation de fautes série est l'algorithme de simulation le plus simple. Pour une liste initiale de n fautes, il consiste à simuler tout d'abord le circuit sain, puis de manière consécutive, les n circuits défectueux affectés chacun d'une des fautes de la liste. Cette approche est simple mais lente. Dans le but d'accélérer le temps d'exécution, la simulation parallèle [Ses65] utilise les opérations bit à bit des machines. Le nombre de circuits

qui peuvent être simulés en même temps est fonction de la longueur du mot machine. La simulation de fautes déductive [Arm72] n'utilise pas de liste de fautes préétablie. Les fautes qui pourront être détectées dans un circuit sont déduites à partir de la structure du circuit sain et du vecteur de test appliqué. La simulation de fautes concurrente est basée sur le fait que le comportement d'un circuit défaillant est généralement peu différent du circuit sain. Seules les parties du circuit fautif qui possèdent un comportement différent du circuit sain sont simulées [Fuj85].

3.2.1.3 Génération de vecteurs de test

Nous décrivons maintenant les différentes méthodes de génération des vecteurs de test. Un vecteur de test représente le stimulus (ensemble des valeurs d'entrée) à appliquer au *circuit sous test (Circuit Under Test ou CUT)* dans le but de détecter un défaut du circuit. Nous présentons tout d'abord de manière succincte les méthodes de test exhaustives et aléatoires, puis les méthodes déterministes de manière plus détaillée.

Le test exhaustif consiste à choisir comme jeu de test l'ensemble de toutes les combinaisons possibles de valeurs d'entrée. Cette approche, qui paraît naturelle, est irréalisable en pratique pour un circuit possédant un nombre de broches d'entrée élevé. En effet, pour un circuit possédant n entrées, le nombre de vecteurs de test est 2^n. Cela conduit à un grand nombre de vecteurs et un temps d'application global prohibitif. De plus, cela ne permet pas de détecter des fautes qui nécessitent une séquence précise de plusieurs vecteurs de test. C'est le cas par exemple lorsque l'on veut tester les fautes de délais présentées dans la section 3.2.1.1 ou tester les fautes de collage dans les circuits séquentiels (ce type de circuit est présenté dans le paragraphe traitant du test déterministe). Ainsi, il est nécessaire d'adopter d'autres méthodes de test qui sont présentées ci-après.

Le test aléatoire consiste, comme son nom l'indique, à choisir de manière aléatoire un ensemble de vecteurs de test à appliquer au circuit et à vérifier, par simulation, si ceux-ci permettent de détecter une liste de défauts potentiels. Les défauts sont exprimés en utilisant des modèles de fautes donnés, comme par exemple le modèle de collage présenté dans la section 3.2.1.1. Le test aléatoire possède l'avantage d'être simple à mettre en oeuvre. Par contre, un nombre important de vecteurs de test est nécessaire pour obtenir une bonne couverture de fautes, ce qui peut induire un temps de test long.

Le test déterministe consiste à générer un ensemble de vecteurs de test permettant de détecter un défaut ciblé du circuit, en utilisant un modèle de faute donné. Un nombre important de travaux se sont concentrés sur le test de circuits combinatoires (les valeurs de sortie ne dépendent que des valeurs d'entrée), avec l'utilisation du modèle de collage simple sur une représentation au niveau porte logique. Les *algorithmes de génération automatique de vecteurs de test (Automatic Test Pattern Generation ou ATPG)* les plus connus sont le D-algorithme [Rot66], le PODEM (Path-Oriented DEcision Making) [Goe81] et le FAN (FANout-oriented Test Generation) [Fuj85]. Le D-algorithme a été

véritablement le premier algorithme ATPG. Le PODEM et le FAN en proposent des améliorations successives. Une comparaison intéressante de ces trois algorithmes est présentée dans [KM98]. Ces algorithmes reposent sur la technique dite de *sensibilisation des chemins*. Cette technique se décompose en trois phases : la *phase de sensibilisation* de la faute, la *phase de propagation aval* et la *phase de propagation amont*. Nous expliquons maintenant plus précisément ces trois phases en prenant comme exemple le circuit simple représenté figure 3.2. Ce circuit présente un défaut modélisé par une faute qui colle à zéro (sa0) la sortie de la porte AND numéro 1 (point noir). La sensibilisation consiste à produire sur le site de la faute (l'endroit du collage) la valeur opposée de celle du collage. La notation x/y décorant une ligne signifie que cette ligne vaut x pour le circuit sain et y pour le circuit défaillant. La phase de propagation aval consiste à propager la faute (valeur du collage) en trouvant un chemin allant du site de la faute jusqu'à une sortie primaire du circuit. La valeur de cette sortie primaire doit être différente entre le bon et le mauvais circuit. La phase de propagation amont (appelée également *justification*) consiste à trouver le vecteur de test qui permet la sensibilisation de la faute et la propagation aval. Pour le circuit pris en exemple, nous pouvons nous apercevoir que le vecteur de test $(1, 1, 1, 1, 0)$ permet de différencier le bon circuit du mauvais. Il convient de remarquer que dans notre exemple, nous avons trouvé un chemin reliant le site de la faute à une sortie primaire du circuit qui permet l'aboutissement de la phase de justification. Or, ce n'est pas toujours le cas car des conflits peuvent apparaître sur certaines équipotentielles du circuit (une équipotentielle du circuit ne peut pas être à la fois dans l'état logique zéro et dans l'état logique un) lors de la phase de justification. Il faut alors essayer de résoudre ces conflits. Si la résolution des conflits est impossible dans la phase de justification, il faut trouver un autre chemin liant le site de la faute à une sortie primaire du circuit et justifier à nouveau. Ce principe est connu sous le nom de retour arrière (*backtracking*).

La *contrôlabilité* d'un circuit est sa capacité et sa facilité à valuer ses noeuds internes à partir de ses entrées primaires. L'*observabilité* d'un circuit est sa capacité et sa facilité à observer la valeur de ses noeuds internes à partir de ses sorties primaires. La *testabilité* d'un circuit est sa capacité à être testé. Celle-ci dépend de la contrôlabilité et de l'observabilité du circuit. Dans la méthode de sensibilisation des chemins, la sensibilisation et la justification garantissent la contrôlabilité alors que la propagation aval garantit l'observabilité.

Nous avons cité des algorithmes ATPG qui sont applicables uniquement aux circuits combinatoires. Nous discutons maintenant des algorithmes ATPG qui concernent les circuits séquentiels. Contrairement à un circuit combinatoire dont les valeurs de sortie dépendent uniquement des valeurs d'entrée, les valeurs de sortie d'un circuit séquentiel dépendent à la fois de ses valeurs d'entrée et de ses états internes. Un état interne correspond à une valeur stockée en mémoire.

Dans le but de mieux comprendre les problèmes relatifs au test des circuits séquentiels, nous décrivons de manière détaillée le type de circuit séquentiel le plus simple : le circuit séquentiel synchrone simple horloge. Celui-ci contient des éléments de logique combinatoire et des éléments mémoire un bit (*Bascule D sur front ou flip-flop*) synchronisés par une même horloge. Il est fréquemment représenté sous la forme d'une machine

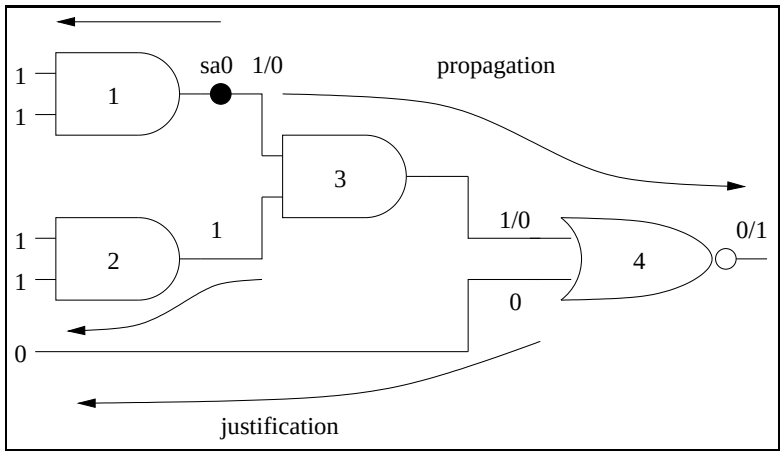

FIG. 3.2 – Sensibilisation des chemins

à états finis (figure 3.3) [BA00].

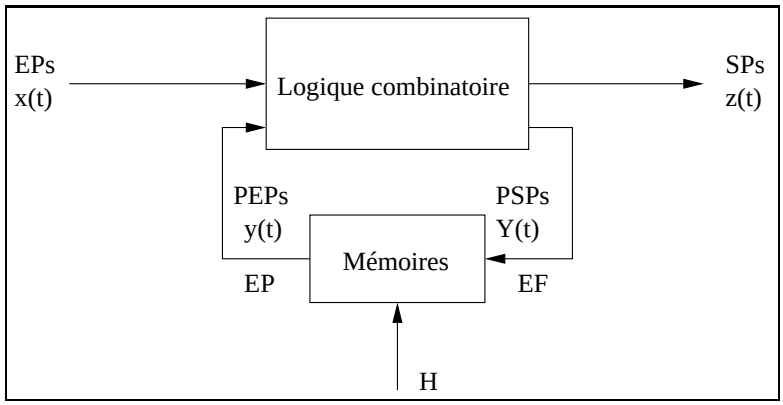

FIG. 3.3 – Représentation d'un circuit séquentiel

Le bloc combinatoire possède un ensemble d'entrées primaires (EPs) et de sorties primaires (SPs). Il possède également un ensemble de pseudo-entrées primaires (PEPs) et de pseudo-sorties primaires (PSPs) internes inaccessibles. Les pseudo-entrées primaires proviennent des flip-flops et forment l'état présent (EP) du circuit. Les pseudo-sorties primaires qui alimentent les flip-flops forment l'état futur (EF) du circuit. Le circuit reste dans le même état (et possède donc le même comportement) entre deux coups d'horloge (H) et change d'état (et donc de comportement) au coup d'horloge suivant. Ainsi, le circuit peut être vu comme une succession temporelle de circuits combinatoires différents. C'est la représentation du circuit en *tranches de temps (time frame)*, comme le montre la figure 3.4.

Un seul vecteur de test permet de sensibiliser une faute de collage et de propager celle-ci pour un circuit combinatoire. Un circuit séquentiel pouvant être vu comme une succession de circuits combinatoires différents correspondant à ses différents états,

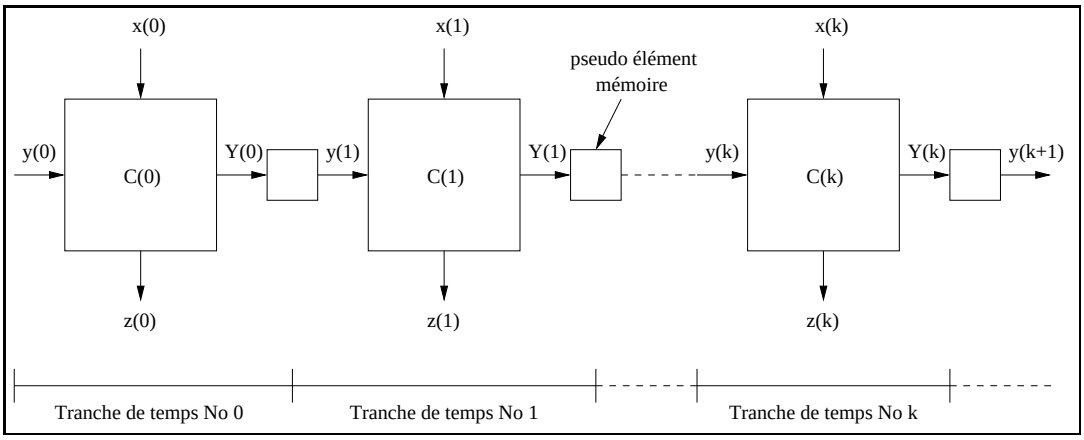

FIG. 3.4 – Représentation en tranches de temps d'un circuit séquentiel

nous voyons qu'il est nécessaire d'utiliser une séquence de vecteurs de test. Un premier ensemble de vecteurs permet de mettre le circuit dans un état où la faute peut être sensibilisée. Le deuxième ensemble de vecteurs permet de mettre le circuit dans l'état où la faute est observable (propagation vers une sortie primaire). Cette approche de la génération des vecteurs de test pour les circuits séquentiels est appelée *approche basée sur l'analyse topologique du circuit*. Une autre approche, appelée *approche basée sur la simulation*, consiste à construire une séquence de test, par phases successives, en combinant la génération et la simulation de séquences d'essai. Une bonne synthèse des différentes classes d'algorithmes existants est effectuée dans [Che96].

La conception en vue d'une meilleure testabilité [ABB+04] (*Design for testability ou DFT*) regroupe un ensemble de techniques de conception qui améliorent la contrôlabilité et l'observabilité du circuit, le rendant ainsi plus testable. Les circuits numériques complexes sont peu testables car ils possèdent un très grand nombre de noeuds internes par rapport au nombre de leurs entrées et sorties primaires. On distingue deux grandes familles de techniques DFT : les techniques *ad hoc* et les *techniques structurées*.

Les *techniques ad hoc* constituent en fait un ensemble de « bonnes pratiques » issues de l'expérience [ABF90]. Par exemple, pour un circuit difficile à tester, une solution simple consiste à ajouter des points de test qui pourront augmenter l'observabilité et/ou la contrôlabilité du circuit. Nous pouvons également citer la technique de partitionnement qui consiste comme son nom l'indique, à partitionner un circuit complexe en circuits plus petits, ce qui permet de réduire le coût du test. En effet, le temps de génération des vecteurs de test est proportionnel au moins au carré du nombre d'éléments composant le circuit [ABB+04].

Les *techniques structurées* impliquent l'utilisation de signaux et de composants logiques supplémentaires qui vont permettre d'augmenter la contrôlabilité et l'observabilité du circuit à tester. Et ceci, sans augmenter de manière significative le nombre d'entrées et de sorties dédiées au test, évitant ainsi un des gros inconvénients des méthodes ad hoc [ABB+04]. Les techniques structurées les plus communément utilisées

sont le *scan path* et l'*auto-test intégré (Built-In Self Test ou BIST)*. Nous les décrivons maintenant succinctement.

Le scan path permet, dans un mode de test du circuit, d'interconnecter des points de mémorisation du circuit implémentés à l'aide de bascules (mémoires un bit élémentaire) [BA00]. Les points de mémorisation forment alors un registre à décalage, comme le montre la figure 3.5. Le mode test (le signal T vaut un) autorise ainsi le chargement série (via l'entrée primaire X_n et cadencé par le signal d'horloge H) des différentes valeurs de contrôle y_k et leur application ainsi que la récupération série des valeurs observées Y_k (via la sortie primaire Z_m). [ABB+04] explique de manière détaillée cette technique.

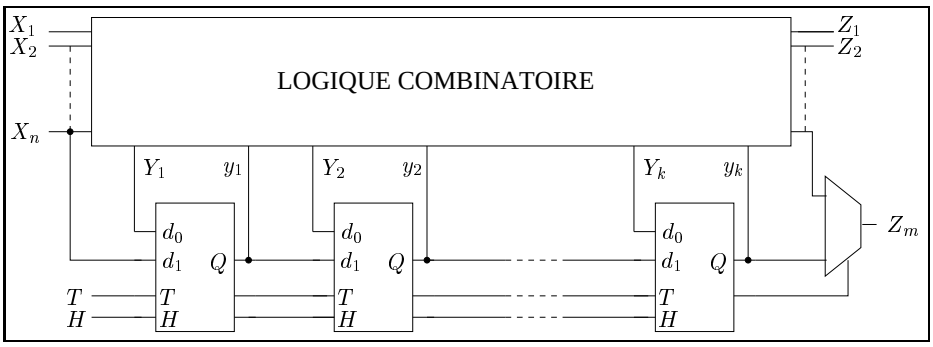

FIG. 3.5 – Architecture de la technique scan path

L'auto-test intégré est une technique de DFT dans laquelle la génération du test, son application et l'analyse de la réponse du circuit sous test sont effectués à travers l'ajout de composants matériels supplémentaires. L'architecture matérielle la plus simple d'un auto-test intégré est représentée figure 3.6. En mode test, le circuit sous test (bloc CST) répond aux stimuli envoyés par le générateur de vecteurs de test (bloc GVT). Les réponses du circuit sous test (bloc CST) sont analysées par l'analyseur de réponse (bloc AR) et comparées aux réponses du circuit réputé bon qui sont stockées en mémoire (bloc MEM). Le résultat du test est binaire (circuit sain ou défectueux). Il existe plusieurs types de générateurs de vecteurs de test ainsi que plusieurs types d'analyseurs de réponse [AKS93]. Citons, à titre d'exemple, les générateurs de vecteurs de test pseudo-aléatoires et les analyseurs de réponses basés sur l'utilisation des *registres à décalage avec rebouclage linéaire (Linear Feedback Shift Register ou LFSR)*, largement utilisés. Le lecteur intéressé par la théorie des LFSR pourra consulter [Gol82].

3.2.2 Test fonctionnel de circuits numériques

Au niveau fonctionnel, un circuit numérique est le plus souvent décrit par un langage de haut niveau comme par exemple VHDL [IEE93] (VHSIC Decription Language où l'acronyme VHSIC signifie Very High Speed Integrated Circuit), Verilog [IEE01a] ou System C [IEE01b]. Ces langages permettent non seulement de spécifier la conception du circuit, mais aussi de le synthétiser automatiquement à partir d'outils dédiés. L'automate à états finis (*Finite State Machine* ou FSM) est un formalisme très utilisé pour décrire le comportement d'un circuit séquentiel [Har03]. Les différents modèles de

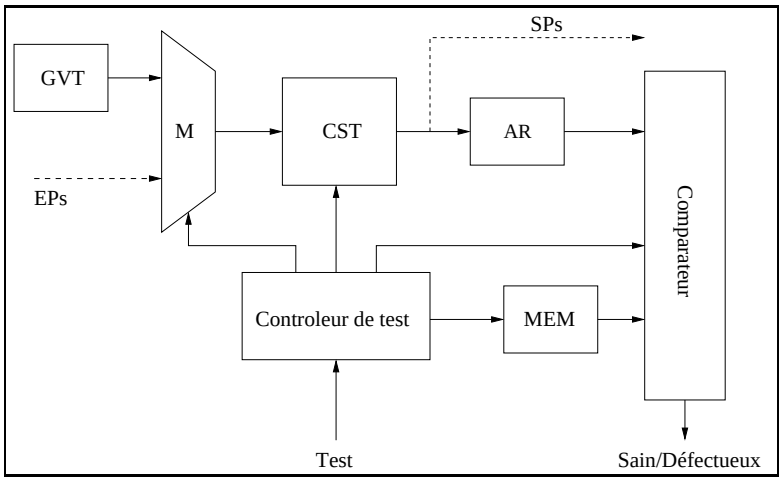

FIG. 3.6 – Structure générale de l'auto-test intégré au niveau composant

fautes utilisés pour les FSM sont la *couverture des états* et la *couverture des transitions*. La couverture des états consiste à atteindre tous les états et la couverture des transitions consiste à franchir toutes les transitions. Les circuits numériques devenant de plus en plus complexes, le formalisme FSM montre ses limites à cause de l'explosion du nombre des états. Ainsi, dans le but de contourner ce problème, d'autres formalismes comme par exemple les automates à états finis étendus (*Extended Finite State Machine* ou EFSM) [CK93] ont été proposés. Un EFSM ne nécessite pas de représenter explicitement les registres de données internes du circuit. Les opérations sur les registres sont modélisées dans les transitions, diminuant ainsi le nombre des états. Il est possible d'extraire un EFSM à partir de la description comportementale du circuit écrite en langage de haut niveau tel que chaque transition corresponde à une instruction de la description comportementale. La génération des vecteurs de test, qui consiste à couvrir les transitions du EFSM, garantit que chaque instruction dans la description comportementale de haut niveau est exécutée au moins une fois [CK93]. Les résultats de travaux récents sur les ATPG fonctionnels basés sur l'utilisation des EFSM sont disponibles dans [FMP05, GFMP06].

3.3 Test de circuits analogiques et mixtes

Il existe des différences importantes de complexité entre le test des circuits intégrés numériques et le test de circuits analogiques [Vin98]. Nous présentons maintenant ces principales différences.

Les circuits analogiques possèdent peu de composants, de l'ordre de quelques centaines, alors que les circuits numériques complexes possèdent plusieurs centaines de millions de transistors. Le niveau d'intégration des circuits numériques, toujours croissant, influe directement sur la complexité du test, alors que ce n'est pas le cas pour les circuits analogiques. Par ailleurs, des différences comportementales existent entre

les deux types de circuits. Les signaux analogiques possèdent une infinité de valeurs (même lorsque leur variation est limitée), ce qui n'est évidemment pas le cas des signaux numériques. Il est impossible, contrairement à un signal numérique (défini par une valeur unique à un instant donné), de connaître avec précision la valeur d'un signal analogique. Il est ainsi nécessaire d'associer une *tolérance* qui détermine un intervalle de bonnes valeurs pour un signal analogique. Les tolérances dépendent des variations du processus de fabrication et des erreurs de mesure.

Les variations du processus de fabrication ont un impact significatif sur les paramètres d'un circuit analogique (cf. section 3.1.2).

La fonctionnalité et la performance d'un circuit numérique sont distinctes : la fonctionnalité d'un circuit décrit les relations entre les sorties et les entrées (table de vérité), alors que la performance spécifie les délais dans les chemins critiques. Un circuit peut fonctionner correctement mais ne pas satisfaire ses spécifications vis-à-vis de ses contraintes temporelles. Des modèles de fautes orientés vers la fonctionnalité ou la performance d'un circuit ont été développés séparément. Une telle séparation n'est pas envisageable pour les circuits analogiques ou mixtes où fonctionnalité et performance sont liées. Il n'y a pas de lien clairement établi entre le taux de couverture par rapport à un modèle de faute structurel donné et les performances du circuit.

De manière générale, chaque classe de circuit analogique/mixte possède un ensemble de spécifications distinct. Ainsi, par exemple, les spécifications d'un filtre sont très différentes de celles d'un convertisseur analogique-numérique. Les spécifications étant liées à une classe de circuit, il n'existe pas de technique générale de test.

Les différences de complexités décrites ci-dessus font que les techniques de test pour les circuits analogiques/mixtes et les circuits numériques sont différentes. Nous présentons maintenant les principales méthodes de test structurel (section 3.3.1) et fonctionnel (section 3.3.2) pour les circuits analogiques et mixtes.

3.3.1 Test structurel de circuits analogiques et mixtes

Nous présentons dans cette section un aperçu des principaux modèles de fautes utilisés pour les circuits intégrés analogiques et mixtes, puis un aperçu de la simulation de fautes. Nous discutons ensuite de la génération automatique des vecteurs de test.

3.3.1.1 Modèles de fautes

En section 3.1.2, nous avons expliqué que les circuits analogiques sont exposés, comme les circuits numériques, aux défauts globaux. Ils sont, de plus, contrairement aux circuits numériques, particulièrement sensibles aux erreurs d'appariement, qui peuvent affecter de manière significative les performances. Il est donc indispensable de modéliser les deux types de défaillances. Les modèles de fautes classiques que l'on trouve en analogique sont les modèles de fautes *catastrophique* et *paramétrique*. Dans le modèle de faute catastrophique, une défaillance catastrophique est modélisée par un composant qui est mis en court-circuit ou en circuit ouvert. Dans le modèle de faute paramétrique, une défaillance paramétrique (due à une erreur d'appariement) est modélisée, pour un com-

posant, comme une déviation de l'un de ses paramètres en dehors de sa tolérance. Dans un circuit analogique, des variations paramétriques mineures de plusieurs composants peuvent autant impacter les performances du circuit qu'une variation paramétrique plus importante d'un seul composant. Ces variations multiples ne peuvent pas être représentées par un modèle de faute unique. Il est donc nécessaire d'utiliser des modèles de fautes paramétriques multiples. Cela contraste fortement avec les modèles de fautes numériques les plus répandus qui sont des modèles simples où une seule faute est présente à un instant donné. Une difficulté supplémentaire vient du fait que des défaillances paramétriques multiples affectant un circuit peuvent se compenser [Rob96]. Par exemple, un circuit composé de deux amplificateurs montés en cascade et ayant chacun un gain non conforme à sa spécification peut donner un gain correct si le produit des deux gains est dans la fenêtre de tolérance. Pour terminer, il est important de rappeler qu'il n'y a pas de lien clairement établi entre le taux de couverture par rapport à un modèle de faute structurel donné et les fonctionnalités/performances du circuit.

3.3.1.2 Simulation de fautes

Nous avons vu en section 3.3.1.1 que les modèles de fautes pour les circuits analogiques affectent soit la topologie du circuit (modèle de faute catastrophique), soit les valeurs des composants du circuit. L'utilisation de ces modèles de fautes fait que la simulation de fautes repose principalement sur des simulateurs analogiques de type SPICE [Tui88, Riv94]. Ce type de simulateur calcule les réponses d'un circuit analogique à partir d'un fichier d'entrée. Ce fichier d'entrée décrit les stimuli d'entrée, les composants, leur valeurs respectives et la manière dont ceux-ci sont interconnectés.

3.3.1.3 Génération de vecteurs de test

Comme pour les circuits numériques, un vecteur de test représente le stimulus à appliquer au circuit sous test dans le but de détecter un défaut physique (section 3.2.1.3). Les stimuli analogiques sont plus complexes que les stimuli numériques. Ils ne sont pas constitués de séquences discrètes binaires, mais correspondent à des signaux analogiques qui sont par définition continus en temps et en amplitude. De plus, un stimulus analogique peut être caractérisé par une forme particulière adaptée pour un type de test donné. Par exemple, on teste un filtre analogique du premier ordre en lui injectant un signal sinusoïdal (fréquence pure).

Les recherches concernant la génération automatique des vecteurs de test pour les circuits analogiques datent des années 60-70 [DR79]. Nous présentons ici différentes méthodes de test paramétrique. Le test paramétrique considère les fautes (paramétriques) qui affectent la valeur des composants et qui ont pour effet de changer les performances du circuit. Les performances du circuit sont caractérisées par un ensemble de paramètres appelés paramètres de sortie. Par exemple, le gain est un paramètre de sortie d'un circuit réalisant une fonction d'amplification. Le principe de ce type de test est de trouver les stimuli qui permettent d'observer les changements de valeur des paramètres de sortie dûs à des variations des valeurs des composants. Le test paramétrique permet donc de

localiser les défauts dans un circuit. Il est à noter que dans la littérature, ce type de test peut être vu comme étant du test fonctionnel [Vin98] : on s'intéresse alors plus aux performances du circuit (au travers de ses paramètres de sortie) qu'à sa structure (valeur des composants et topologie). Dans le but d'éviter toute confusion, nous considérons ici que le test paramétrique est un test de type structurel et non fonctionnel. Le test fonctionnel fera l'objet de la section 3.3.2.

Dans [HK93a], les auteurs présentent une méthode de génération automatique de tests pour les circuits analogiques basée sur la *sensibilité*. La sensibilité représente l'effet de la déviation de la valeur d'un composant d'un circuit sur le changement de la valeur d'un paramètre de sortie. La méthode génère une liste de paramètres de sortie qui doivent être mesurés pour couvrir l'ensemble des fautes considérées. Cependant, les formes des stimuli analogiques qui permettent de mesurer les paramètres de sortie ne sont pas générées. Cette méthode a été généralisée pour prendre en compte les modèles de fautes multiples [HK93b]. Elle a également été utilisée pour tester les parties analogiques de circuits mixtes [AHK95].

Dans [RB96], les auteurs proposent un algorithme de génération de tests pour les circuits analogiques et mixtes qui utilise les graphes à flot de signaux (Signal Flow Control ou SFG). L'algorithme construit un graphe à flot de signaux modélisant le circuit, puis inverse la direction de ses arcs. Il utilise les poids symboliques des arcs pour représenter les valeurs des composants défaillants et calcule la forme du stimulus d'entrée qui permet de détecter une faute. Une faute correspond à la variation de la valeur d'un composant qui provoque une variation d'un paramètre de sortie.

3.3.2 Test fonctionnel de circuits analogiques et mixtes

Le test fonctionnel est basé sur les spécifications du circuit. Il consiste à vérifier que le circuit sous test satisfait ses spécifications. Dans ce but, l'ensemble des paramètres fonctionnels (paramètres de sortie) caractéristiques du circuit sont mesurés et comparés aux valeurs attendues dans le cahier des charges. Le test fonctionnel ne considère pas la structure du circuit.

Les circuits analogiques réalisent des fonctions très différentes. On distingue plusieurs classes de circuits comme par exemple les filtres, les amplificateurs et les convertisseurs. En considérant la classe des filtres, on s'aperçoit qu'il existe un grand nombre de types de filtres différents. A cause de cette diversité, chaque circuit est caractérisé par un ensemble de paramètres spécifiques, et donc, par un ensemble de mesures spécifiques à effectuer.

Pour effectuer la mesure d'un paramètre particulier, il est nécessaire d'utiliser un type de test adapté. Celui-ci impose les caractéristiques des stimuli (dont la forme des signaux) à appliquer au circuit sous test. Par exemple, une méthode très utilisée pour mesurer la fréquence de coupure d'un filtre analogique linéaire passe-bas consiste à étudier la réponse en fréquence du filtre en un point particulier qui correspond à sa fréquence de coupure. Cette méthode impose donc d'injecter comme stimulus un signal sinusoïdal dont la fréquence est égale à la fréquence de coupure du filtre, afin de pouvoir mesurer l'amplitude et la phase de la réponse du circuit. D'autres types de

stimuli comme les signaux d'amplitude continue (*Direct Current ou DC*), les signaux carrés, ou les signaux d'autres formes peuvent être utilisés, en fonction de la nature des paramètres à mesurer. Par exemple, la mesure du seuil d'un comparateur s'effectue en utilisant un signal DC. Cependant, les signaux sinusoïdaux sont souvent utilisés pour tester les circuits analogiques ou mixtes. C'est le cas pour les méthodes de test spectrales que nous décrivons maintenant succinctement.

L'analyse fréquentielle (analyse de Fourier) de la réponse d'un circuit linéaire à une fréquence pure (*tone* en anglais) permet de quantifier le degré de non-linéarité du circuit. Ce degré de non-linéarité du circuit se traduit, dans le spectre de puissance du signal de sortie, par l'apparition de fréquences harmoniques qui viennent s'ajouter à la fréquence fondamentale. Le test *single tone* est basé sur ce principe. Il permet de mesurer le taux de distorsion harmonique qui compare la puissance de la fondamentale à celle des harmoniques. Le test *multi tone* utilise comme stimulus d'entrée une somme de fréquences pures différentes et permet de mesurer un ensemble de paramètres dits de transmission, comme par exemple le taux d'inter-modulation.

Nous pouvons nous apercevoir, au travers de ces quelques exemples, de la diversité des méthodes de test et des stimuli associés. Il en résulte que les ressources matérielles nécessaires pour tester les circuits analogiques et mixtes sont plus nombreuses et plus complexes que pour les circuits numériques. En effet, pour ces derniers, les stimuli correspondent toujours à des niveaux logiques valant zéro ou un.

Le banc de test analogique le plus simple est représenté figure 3.7. Il permet d'effectuer un test par mesure directe. Il est constitué de différents générateurs de signaux analogiques et différents instruments de mesure.

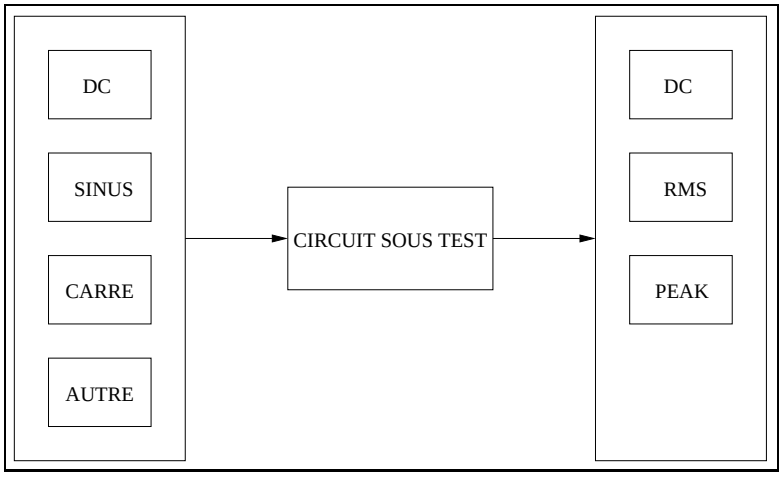

FIG. 3.7 – Banc de test analogique

A l'heure actuelle, la plupart des circuits analogiques et mixtes sont testés à l'aide de bancs de test utilisant un *processeur numérique de signal (Digital Signal Processing ou DSP)*. La figure 3.8 montre l'architecture matérielle de ce type de banc. Le DSP permet la synthèse des stimuli analogiques. La synthèse d'un signal consiste à construire celui-ci à partir de données numériques chargées en mémoire auxquelles est appliquée

une conversion numérique-analogique. Il est ainsi possible d'émuler les générateurs de signaux analogiques traditionnels utilisés en mesure directe. Les stimuli numériques sont envoyés directement au circuit sous test lorsque celui-ci est de type mixte. Les réponses analogiques du circuit sous test sont acquises, puis numérisées et stockées en mémoire alors que les réponses numériques (cas d'un circuit mixte) sont acquises et stockées en mémoire sans transformation. Afin d'obtenir un test précis, le synthétiseur doit délivrer les données numériques à des intervalles de temps précis pour la construction d'un signal analogique délivré au circuit sous test. L'unité de capture doit également échantillonner et numériser précisément un signal analogique délivré par le circuit sous test. On parle alors de synchronisation du synthétiseur et de l'unité de capture.

Le DSP effectue également les post-traitements sur les réponses du circuit sous test. Ces traitements peuvent être simples, comme des mesures de type voltmètre en mesure directe, ou nettement plus complexes comme par exemple l'analyse spectrale nécessitée par les méthodes de test multi-tone. L'intégralité du test est ainsi gérée par le programme de test qui s'exécute sur le DSP.

Nous n'avons présenté que très succinctement le test de circuits analogiques et mixtes. Le lecteur intéressé trouvera des informations très détaillées dans [Mah87, BR01].

Pour terminer, nous expliquons brièvement l'échantillonnage cohérent (*coherent sampling*), car c'est une notion fondamentale qui conditionne la précision des tests effectués sur un banc de test basé sur un DSP. Dans la méthode single tone, l'analyse spectrale de la réponse du circuit sous test est effectuée en utilisant la transformée de Fourier rapide (Fast Fourier Transform ou FFT) qui est un algorithme permettant de calculer très rapidement la transformée de Fourier discrète. L'échantillonnage cohérent établit une relation qui maximise l'information contenue dans le stimulus sinusoïdal, et qui implique la plus grande précision dans le calcul de la FFT [Mah87]. Cette relation est donnée par

$$\frac{F_{test}}{F_e} = \frac{M}{N}$$

où F_{test} représente la fréquence du signal single tone, F_e la fréquence d'échantillonnage, N le nombre de points utilisés pour le calcul de la FFT et M le nombre entier de périodes du signal, M et N étant premiers entre eux. L'échantillonnage cohérent est également applicable à un signal multi tone car la somme de signaux single tone cohérents reste un signal cohérent.

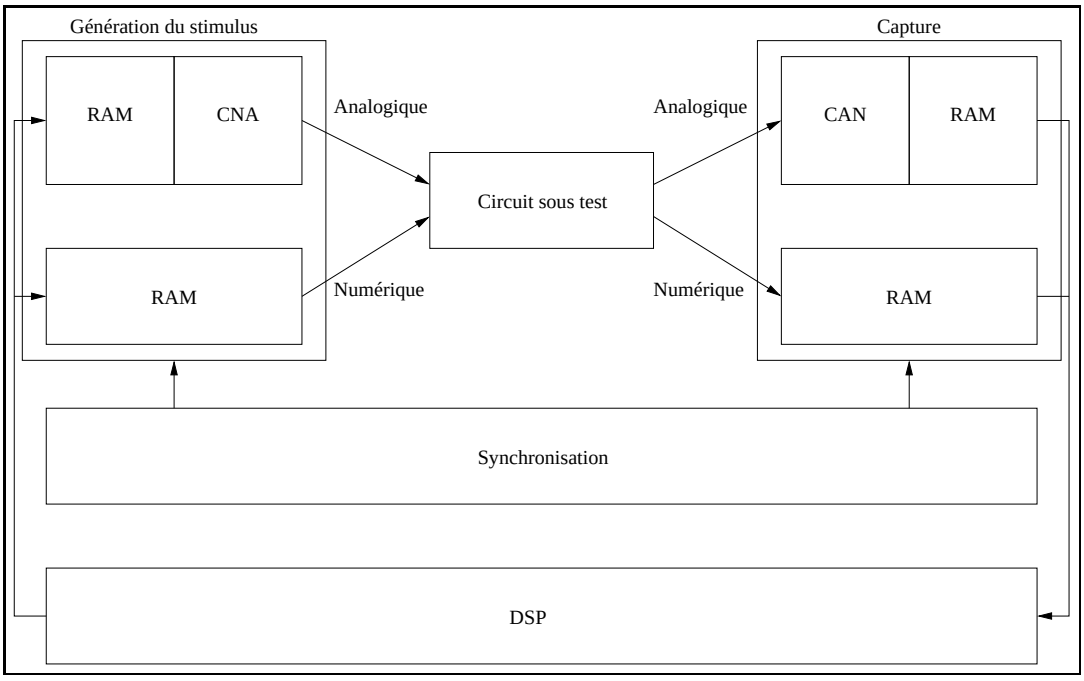

FIG. 3.8 – Banc de test basé sur un DSP

Chapitre 4

Test de cartes électroniques

Dans le chapitre précédent, nous avons présenté les techniques de test structurelles et fonctionnelles pour les circuits intégrés numériques, analogiques et mixtes. Dans ce chapitre, nous nous intéressons au test de cartes, appelé également test au niveau carte (*Board Level Test*). Le test de cartes repose, comme le test de circuits, sur deux approches distinctes : l'approche structurelle et l'approche fonctionnelle. Par rapport aux objectifs du test de circuit, les objectifs du test de carte sont plus globaux. Ainsi, dans l'approche structurelle, le test consiste à tester les circuits montés sur la carte, mais aussi à vérifier la structure de la carte. Par conséquent, il est important de vérifier par exemple que les bons circuits sont montés correctement sur la carte et qu'il n'y a pas de circuits manquants. Par ailleurs, il est également important de vérifier que les interconnexions entre les circuits sont correctes. Il est nécessaire de pouvoir détecter par exemple des courts-circuits ou des coupures sur les interconnexions. Dans l'approche fonctionnelle, le test consiste à vérifier que le comportement de la carte satisfait ses spécifications. Le test unitaire de composants est insuffisant : il est nécessaire de vérifier les interactions entre les composants montés sur la carte.

Afin de faciliter le test, des normes de test au niveau carte ont été développées. Ces normes prévoient la testabilité de la carte dès sa conception, reprenant et étendant ainsi le concept de conception en vue d'une meilleure testabilité abordé dans le chapitre précédent. L'idée est d'acheminer facilement les vecteurs de test et de récupérer facilement les réponses. Le but est de tester non seulement les circuits mais également les interconnexions entre ces circuits. Les normes de test peuvent être vues comme un moyen de faciliter le test structurel au niveau carte.

Nous commençons par présenter les techniques de test structurelles et fonctionnelles (section 4.1), puis les principales normes de test proposant la conception en vue d'une meilleure testabilité (section 4.2). Nous terminons en précisant la position de la communauté du test matériel vis-à-vis du test de cartes (section 4.3).

4.1 Techniques de test

Nous présentons dans cette section les deux approches du test au niveau carte : le test structurel et le test fonctionnel.

4.1.1 Test structurel

Les techniques automatiques d'inspection visuelle (*Automatic Optical Inspection ou AOI*) et d'inspection par rayons X (*Automatic X-Ray Inspection ou AXI*) sont des méthodes d'inspection visuelle en amont du cycle de fabrication des cartes. Elles permettent de détecter les défauts de la carte nue.

Le test *in situ (In Circuit Testing ou ICT)* consiste à tester individuellement chaque composant de la carte comme s'il n'était pas monté sur la carte. L'idée du test in situ est la suivante : si tous les composants d'une carte sont bons et si toutes les interconnexions entre les composants sont bonnes, alors la carte est considérée comme bonne. Dans la pratique, la carte à tester est posée sur un *lit à clous (bed of nails)*. Le lit à clous est composé d'un ensemble de pointes. Chaque pointe entre en contact avec un point de soudure de la carte, le contact s'effectuant par le dessous de la carte. Ainsi, une pointe est en contact avec une ligne d'interconnexion de la carte. La figure 4.1 montre le principe du test in situ. Les stimuli de test d'un composant de la carte et ses réponses aux stimuli sont respectivement appliqués et récupérées par les pointes adéquates du lit à clous. Les avantages de cette technique sont les suivants :

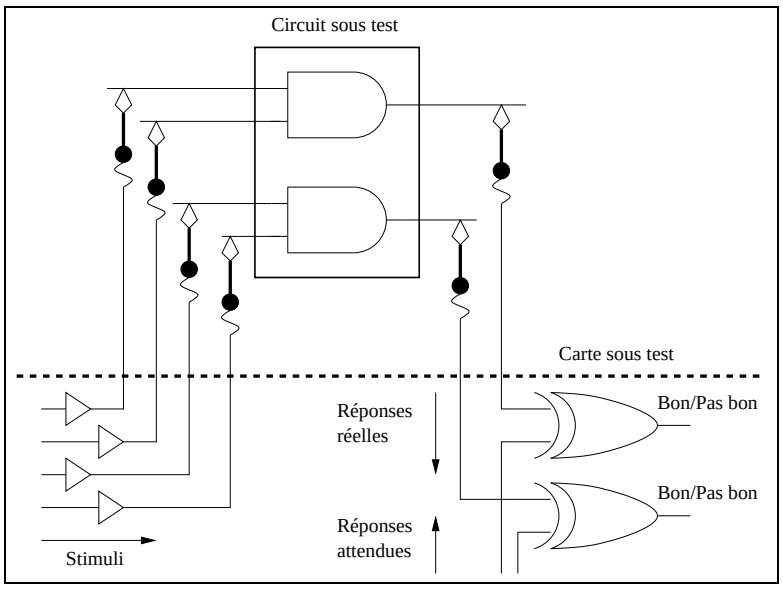

FIG. 4.1 – Principe du test in situ

– alors qu'il peut exister un très grand nombre de cartes, il n'existe qu'un nombre plus limité de composants. Et comme les composants sont testés individuellement lors de leur fabrication, des bibliothèques de vecteurs de test sont disponibles,

- l'exécution des vecteurs de test est généralement rapide car leur taille est petite,
- lorsque le test d'un composant échoue, la carte peut être réparée en remplaçant le composant défectueux (il faut évidemment que ce remplacement soit technologiquement possible).

Les inconvénients sont les suivants :

- le lit à clous est spécifique à la carte à tester. Pour une carte moyenne, le nombre de pointes est de l'ordre d'un millier. Le nombre de pointes, plus le fait que celles-ci doivent être positionnées précisément sur les points de soudure de la carte font que le coût de fabrication d'un lit à clous est élevé,
- le test in situ devient inapplicable avec la densité croissante des composants sur une carte,
- considérer que l'on teste individuellement chaque composant comme s'il n'était pas monté sur une carte alors qu'en fait, il l'est, pose un problème. Un vecteur de test conçu pour tester un composant isolé électriquement n'est pas toujours adapté au test du même composant entouré des autres composants de la carte.
- de par son principe même, le test in situ ne permet pas de tester les interactions entre les composants. En effet, le principe fondamental du test in situ est de tester unitairement un composant en y accédant via le lit à clous et en le testant en utilisant la bibliothèque de vecteurs de test adéquate.

4.1.2 Test fonctionnel

Le test fonctionnel consiste à tester les fonctionnalités (comportement) de la carte. La carte peut être testée dans son environnement normal (par exemple, la carte est insérée dans un fond de panier (*backplane*)) ou montée dans un testeur fonctionnel. Un testeur fonctionnel simule l'environnement normal de la carte en lui fournissant en temps réel les signaux nécessaires.

Les avantages sont les suivants :

- le test fonctionnel permet de vérifier les interactions entre les composants. C'est le seul test qui est capable de vérifier le bon comportement d'une carte, i.e. que la carte satisfait à ses spécifications,
- il présente un vrai intérêt pour la validation de la conception d'une carte et le test en maintenance (réutilisation d'un test existant ou développement spécifique).

Les inconvénients sont les suivants :

- le test fonctionnel donne un résultat de type go/nogo, ce qui signifie que la carte fonctionne correctement ou pas. Lorsque le test échoue, il faut utiliser du test de composants orienté diagnostic,
- pour chaque nouvelle carte, il est nécessaire de concevoir un nouveau test fonctionnel. Pour une carte de taille moyenne, le temps de développement moyen du logiciel de test est typiquement de six mois. Le développement est donc cher et le délai n'est pas compatible avec du test de production.
- les testeurs fonctionnels coûtent très chers.

4.2 Normes de test au niveau carte

La complexité croissante des cartes électroniques et l'évolution des technologies utilisées pour leur fabrication rendent le test in situ de moins en moins applicable. En particulier, avec la technologie de *montage en surface (Surface Mount Technology ou SMT)*, les composants sont soudés sur un côté de la carte, sans perçage de celle-ci. Il est ainsi impossible (à moins de prévoir des broches de test spécifiques) de contrôler et d'observer les signaux de ces composants en utilisant un lit à clous car ces signaux ne sont pas accessibles de l'autre côté de la carte. Par ailleurs, les cartes électroniques actuelles sont multi-couches et les lignes de connexions dans les couches internes sont également inaccessibles. L'inapplicabilité du test in situ a conduit à repenser la manière d'acheminer les vecteurs de test (et les réponses) vers les composants d'une carte électronique. Ceci a conduit à l'élaboration de différentes normes.

Dans cette section, nous présentons les principales normes de test au niveau carte. Nous abordons tout d'abord la norme IEEE 1149.1, appelée également « boundary scan », qui est la plus ancienne et qui s'intéresse aux cartes électroniques composées uniquement de circuits numériques. Nous présentons ensuite deux normes qui constituent chacune une extension du boundary scan : la norme IEEE 1149.6 propose une solution aux limitations du boundary scan dues aux technologies actuelles, et la norme IEEE 1149.4 traite les cartes possédant des circuits analogiques et mixtes.

4.2.1 Norme IEEE 1149.1 « Boundary Scan »

La norme IEEE 1149.1 [IEE90] constitue une réponse pour le test de cartes composées de circuits intégrés numériques. La norme possède deux niveaux de spécifications. Elle spécifie d'une part l'architecture générale des composants, et d'autre part, la façon dont les composants doivent être reliés entre eux sur la carte. L'utilisation de la norme permet principalement d'une part, d'isoler logiquement les circuits (compatibles) en vue de leur test et d'autre part, de tester les interconnexions entre les composants (détection de courts-circuits entre lignes et de lignes ouvertes).

La figure 4.2 montre l'architecture d'un composant numérique respectant la norme IEEE 1149.1. La partie hachurée représente la logique interne du circuit avant sa mise à la norme. Chaque broche du circuit (rectangle noir) est connectée à une entrée ou une sortie de la logique du circuit par une *cellule de boundary scan (boundary scan cell)*. Ces cellules sont reliées en série entre elles pour former un registre à décalage appelé *registre de boundary scan (boundary scan register)*. La norme définit des registres supplémentaires : le registre d'instruction et les registres de données. Le registre d'instruction stocke en série l'instruction courante. L'instruction spécifie les opérations à effectuer et le registre de données à sélectionner.

Le port d'accès de test (*Test Access Port ou TAP)* est une interface standardisée permettant l'accès et le contrôle des différents registres. Ces registres seront décrits plus tard. Le port est composé de quatre signaux, avec une broche externe dédiée à chacun d'entre eux. Ces signaux sont décrits ci-après :

– entrée des données de test (*Test Data Input ou TDI*) : ce signal d'entrée charge

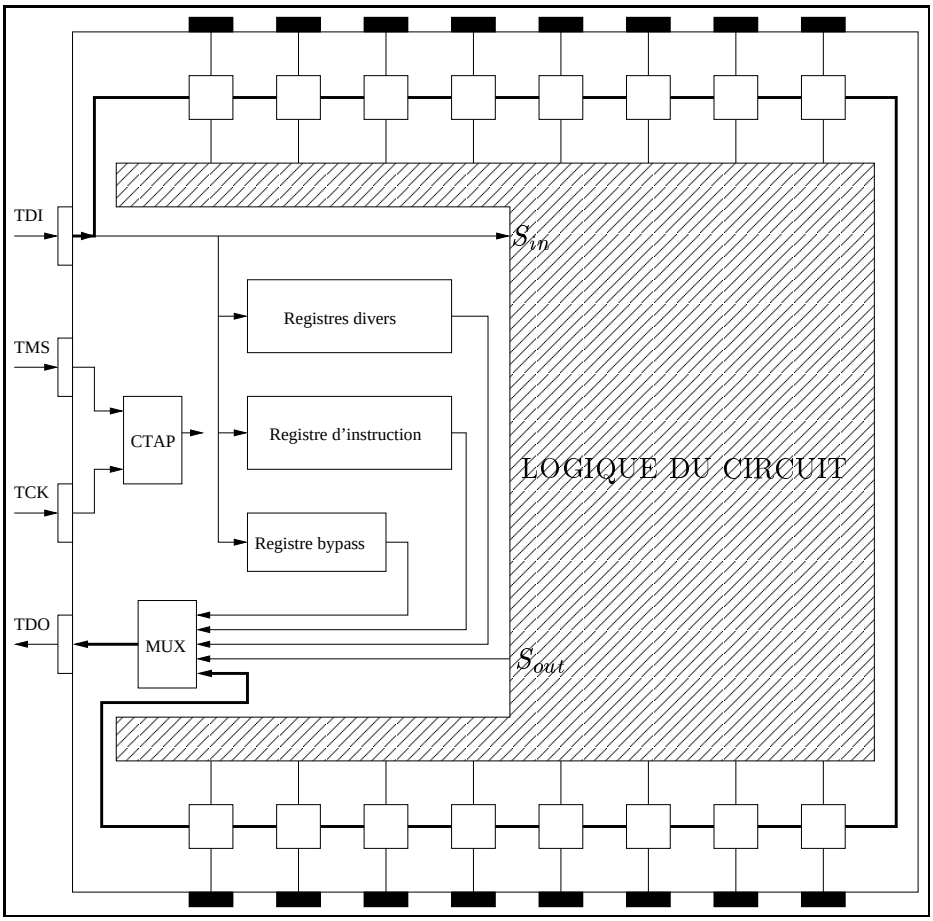

FIG. 4.2 – Architecture générale d'un composant respectant la norme IEEE 1149.1

les données de test et les instructions dans les registres appropriés,
- sortie des données de test (*Test Data Output ou TDO*) : ce signal de sortie fournit les données provenant du registre de boundary scan ou des autres registres,
- horloge de test (*Test Clock ou TCK*) : ce signal d'entrée contrôle le chargement des différents registres,
- sélection du mode de test (*Test Mode Selection ou TMS*) : ce signal d'entrée détermine l'état du contrôleur TAP (*TAP Controller*).

Le contrôleur TAP (boîte CTAP sur la figure 4.2) est une machine à états finis de seize états. Le contrôleur change uniquement d'état sur un front montant du signal TCK, l'état suivant dépendant de la valeur du signal TMS. L'automate du contrôleur est donné dans [IEE90, BA00]. À chaque état du contrôleur est associé un comportement du boundary scan. Nous décrivons brièvement quelques uns de ces comportements. Dans l'état *Test Logic Reset*, toute la circuiterie de test est déconnectée et le circuit fonctionne normalement. Dans l'état *Shift-DR*, le registre de données sélectionné entre les broches

TDI et TDO est décalé d'un bit sur un front montant du signal TCK (un bit rentre par TDI et un bit sort par TDO). Par exemple, le registre boundary scan permet de charger un vecteur de test ou de récupérer la réponse du circuit sous test de manière sérielle. Dans l'état *Update-DR*, les données du registre sélectionné sont accessibles de manière parallèle. Par exemple, le registre de boundary scan permet d'appliquer un vecteur de test à la logique interne du circuit. Pour terminer, dans l'état *Capture-DR*, les données sont chargées parallèlement dans le registre de données sélectionné. Par exemple, le registre de boundary scan peut récupérer de manière parallèle les réponses du circuit sous test.

L'ensemble des instructions définies par la norme est donné dans [IEE90, BA00]. Nous en décrivons maintenant quelques unes. L'instruction *intest* permet le test de la logique interne du circuit. Le vecteur de test est chargé via la broche TDI dans le registre de boundary scan (état Shift-DR du contrôleur), puis est appliqué à la logique du circuit (état Update-DR). La réponse du circuit sous test est ensuite stockée dans le registre de boundary scan (état Capture-DR). Le contrôleur retourne ensuite dans l'état Shift-DR de façon à charger un nouveau vecteur de test via TDI et récupérer le résultat du test précédent via TDO. L'instruction *Extest* permet le test des interconnexions entre les différents circuits de la carte. Le mode Extest est illustré figure 4.3. Dans ce mode, la logique interne des circuits est déconnectée. Le vecteur de test V1 est chargé en série via la broche TDI dans le registre de boundary scan et appliqué aux broches externes du circuit numéro 1. Dans notre exemple, le contenu du vecteur V1 est calculé de manière à ce que des valeurs logiques « un » soient appliquées sur les broches externes du circuit numéro 1 qui sont reliées à celles du circuit numéro 2, les autres valeurs étant quelconques. Les signaux sur les broches externes (V2) du circuit numéro 2 sont ensuite stockés dans son registre de boundary scan, puis récupérés en série via la broche TDO. Dans notre exemple, le vecteur V2 contient des « un » qui sont les valeurs des signaux sur les broches du circuit numéro 2 reliées au circuit numéro 1 (les vecteurs V1 et V2 sont différents car les broches interconnectées pour chacun des deux circuits sont différentes). Les deux lignes d'interconnexions sont alors considérées comme bonnes (non ouvertes). Dans la réalité, les vecteurs de test sont plus complexes, dans le but par exemple de tester le court-circuit de deux lignes d'interconnexions. Pour finir, l'instruction *runbist* permet de déclencher un auto-test et de récupérer son résultat, en utilisant respectivement l'entrée S_{in} et la sortie S_{out} représentées figure 4.2.

Les registres de données sont au minimum le registre de boundary scan et le registre de *bypass*. Le registre de bypass est un registre de un bit qui permet de raccourcir la chaîne de scan, ce qui permet d'acheminer plus rapidement les vecteurs de test à un autre composant situé en aval. La norme prévoit qu'il peut exister également des registres de données supplémentaires.

La norme est flexible en ce qui concerne la configuration du chemin d'acheminement des vecteurs de test. La figure 4.4 montre une configuration possible où les registres de boundary scan des différents circuits de la carte sont reliés en série entre eux, formant une seule et unique *chaîne de scan (boundary scan chain)*. Afin de ne pas alourdir le schéma, le TAP et les différents registres de chaque circuit n'ont pas été représentés, et

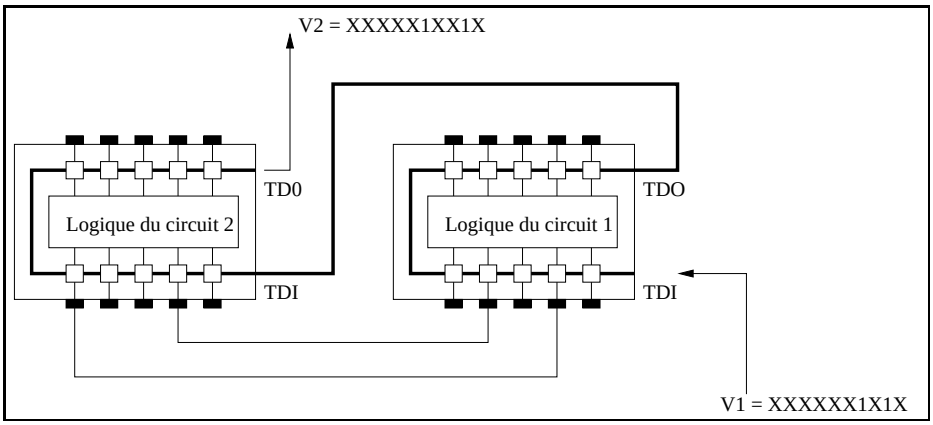

FIG. 4.3 – Mode Extest

les entrées/sorties primaires de la carte n'ont pas été annotées.

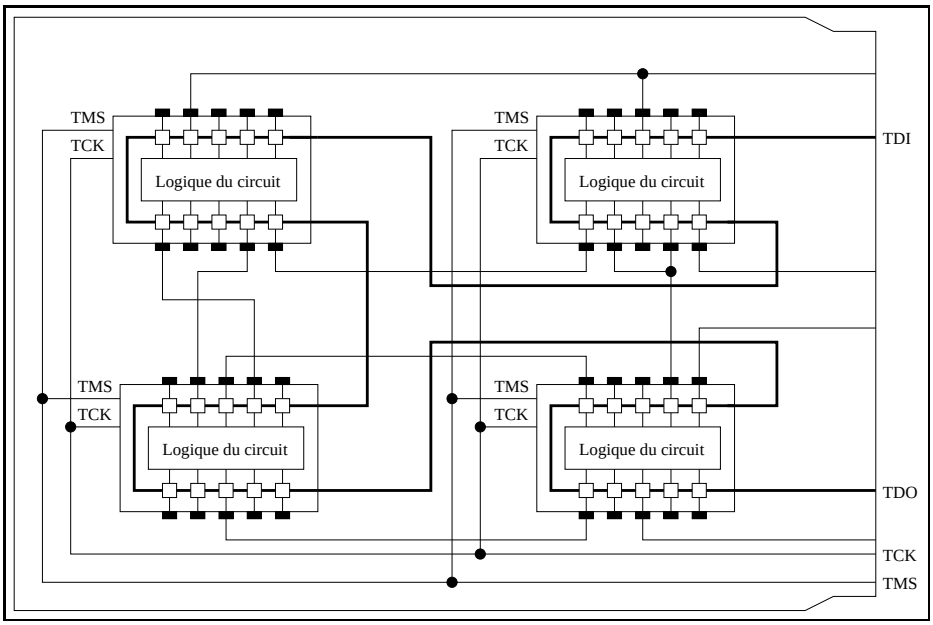

FIG. 4.4 – Norme IEEE 1149.1 au niveau carte

Nous terminons cette rapide description de la norme boundary scan en introduisant son langage de description (*Boundary Scan Description Language ou BDSL)* [IEE94]. Ce langage permet de décrire la circuiterie de test du boundary scan. Il peut être utilisé par exemple par des outils de conception assistée par ordinateur (CAO) pour synthétiser la circuiterie de test d'un composant. Le langage BSDL est implémenté comme un sous-ensemble du langage VHDL.

La norme IEEE 1149.1 est largement acceptée et utilisée dans l'industrie par les

fabriquants de circuits intégrés, les outils de conception et de test, ainsi que par les fournisseurs d'équipements de test.

4.2.2 Norme IEEE 1149.6

L'utilisation d'interconnexions à couplage capacitif, appelées également connexions AC (*AC coupled interconnects*) comme lignes de transmission entre circuits devient de plus en plus répandue. Les interconnexions à très haute vitesse (multi-gigahertz) entre les entrées/sorties de composants sérialisateur-désérialisateur (*Serializer-Deserializer ou SerDes*), que l'on trouve par exemple dans les routeurs ou les switchs, sont à couplage capacitif. Ce type d'interconnexion, illustré figure 4.6, est composé d'un élément de terminaison résistif (R) et d'un élément capacitif (C). Une interconnexion AC permet de relier deux circuits numériques qui ne sont pas compatibles par rapport aux tensions représentant les niveaux logiques (*DC-incompatible*). Des raisons technologiques et économiques rendent souhaitable l'utilisation et la connexion de circuits DC-incompatibles [Par01]. Une interconnexion AC rend deux circuits DC-incompatibles compatibles en bloquant la tension continue (*DC voltage ou offset*) du signal, ce qui a pour effet de décaler les tensions représentant les niveaux logiques un et zéro (*AC voltages*).

La norme IEEE 1149.1 ne permet pas de tester efficacement les interconnexions AC à l'aide de l'instruction Extest décrite dans la section 4.2.1. Afin de mieux comprendre cette problématique, nous décrivons maintenant la mise en oeuvre de l'instruction Extest pour deux types d'interconnexions [Whe03] : les interconnexions classiques au sens de la norme IEEE 1149.1, appelées interconnexions directes ou DC (*DC interconnect*) et les interconnexions AC.

La figure 4.5 représente le test d'une interconnexion DC. En mode test, lorsque l'instruction Extest est exécutée, la cellule de boundary scan du circuit émetteur est déconnectée du coeur numérique. Elle est ensuite positionnée à la valeur de test (un sur l'exemple) via le registre de boundary scan lorsque le contrôleur TAP est dans l'état Shift-DR. La donnée de test est mise à jour (OUT' vaut un) sur le front descendant de l'horloge TCK lorsque le contrôleur TAP est dans l'état Update-DR, puis copiée dans la cellule du circuit récepteur (capture) sur un front montant de l'horloge, dans l'état Capture-DR. La durée entre la mise à jour de la donnée de test et sa capture est égale à 2,5 périodes de l'horloge TCK.

La figure 4.6 représente le test d'une interconnexion AC. Le séquencement des états du contrôleur TAP est identique à celui utilisé pour le test d'une interconnexion DC. Ceci est normal car dans les deux cas, l'instruction Extest est utilisée. Nous pouvons cependant remarquer qu'après la mise à jour de la donnée de test par le circuit émetteur (OUT' vaut un), le signal à l'entrée de la cellule d'entrée du circuit récepteur s'affaiblit en fonction du temps (IN). Cet affaiblissement est dû à la capacité (C) qui se décharge vers la masse à travers la résistance (R). Le réseau formé par (R) et (C) est un filtre dont la constante de temps vaut RC. Lorsque la durée entre la mise à jour de la donnée de test et sa capture (qui vaut 2,5 périodes de l'horloge TCK) est supérieure à 5 fois la constante de temps du filtre, la tension (IN) franchit le seuil de différentiation des niveaux logiques. Le niveau logique alors capturé dans la cellule (IN') vaut zéro. Le test

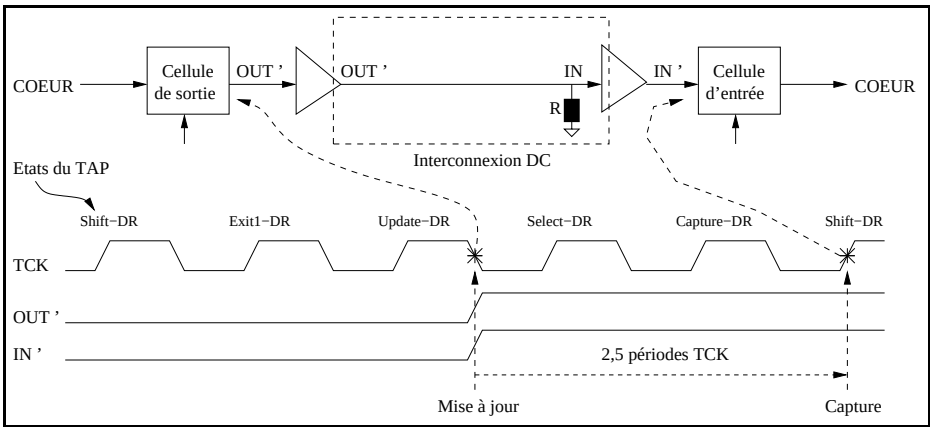

FIG. 4.5 – Test d'une interconnexion DC avec l'instruction Extest

de l'interconnexion est alors faussé.

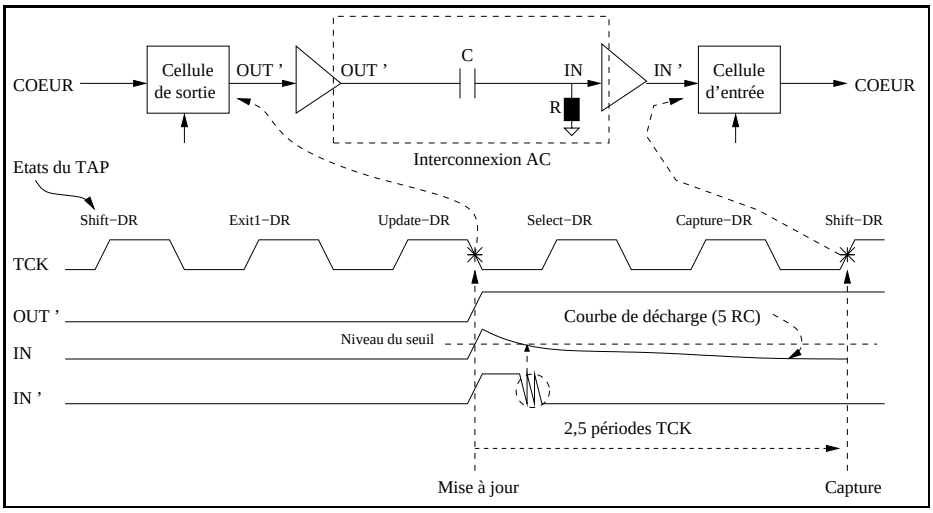

FIG. 4.6 – Test d'une interconnexion AC avec l'instruction Extest

La norme IEEE 1149.6 [IEE03] permet de résoudre ce problème. Cette norme est une extension de la norme IEEE 1149.1 avec laquelle elle reste compatible. Elle permet ainsi de tester des interconnexions DC et AC. Concernant les interconnexions AC, elle considère également le cas des interconnexions différentielles car ce type d'interconnexion est fréquemment utilisé, principalement à cause d'une meilleure tolérance aux fautes et d'une meilleure immunité au bruit. Une interconnexion différentielle est représentée figure 4.7.

Afin d'apporter une solution au problème des signaux s'affaiblissant dans le temps, la norme IEEE 1149.6 impose que la capture doit s'effectuer sur les transitions du signal et non plus sur le niveau du signal (IEEE 1149.1). Pour ce faire, les cellules d'entrée du côté récepteur reliées aux lignes IN et IN* sont modifiées (cellules d'entrée 1149.6 sur la

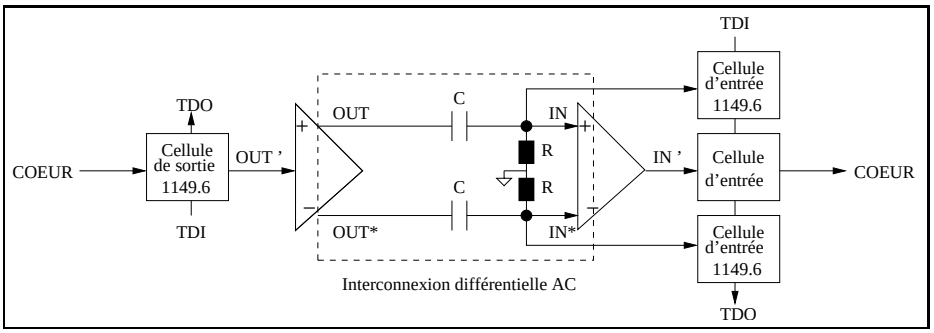

FIG. 4.7 – Test d'une interconnexion différentielle AC

figure 4.7). Une cellule d'entrée 1149.6 détecte les transitions montantes et les transitions descendantes du signal affaibli et reconstitue le signal numérique original. La détection est basée sur le principe du comparateur à hystéresis auto-référencé [EBP02]. La norme IEEE 1149.6 définit deux nouvelles instructions *Extest_ Pulse* et *Extest_ Train* qui sont présentées de manière détaillée dans [EBP02]. Par ailleurs, un ensemble de considérations pratiques de la norme, et plus particulièrement au niveau carte, est abordé dans [EBRB03].

4.2.3 Norme IEEE 1149.4

La norme IEEE 1149.4 [IEE99], appelée également *bus de test analogique (Analog Test Bus ou ATB)* est une extension de la norme boundary scan. Son objectif principal est de permettre le test des interconnexions d'une carte mixte. Un second objectif est de tester la partie analogique des circuits mixtes de la carte. Nous présentons d'abord l'architecture matérielle d'un circuit respectant la norme. Puis nous expliquons le principe des tests d'interconnexion et le test de la partie analogique.

La figure 4.8 montre l'architecture générale d'un composant respectant la norme IEEE 1149.4. La circuiterie de test, issue de celle du boundary scan, est enrichie. Nous retrouvons les quatre signaux TDI, TDO, TMS et TCK qui forment le TAP, le registre d'instruction et les registres de données (entre les broches TDI et TDO), et le contrôleur TAP. La norme ajoute deux signaux analogiques AT1 et AT2. Le signal AT1 est habituellement utilisé comme stimulus de test, tandis que le signal AT2 est la réponse du circuit sous test. Les signaux AT1 et AT2 forment le port d'accès du test analogique (*Analog Test Access Port ou ATAP*). La norme IEEE 1149.4 fait évoluer le registre de boundary scan. Il est composé des cellules DBM (*Digital Boundary Module*), des cellules ABM (*Analog Boundary Module*) et du circuit d'interface du bus de test (*Test Bus Interface Circuit ou TBIC*). Les cellules DBM sont connectées aux entrées/sorties numériques du circuit et les cellules ABM sont connectées aux entrées/sorties analogiques du circuit. Le TBIC interface les signaux AT1 et AT2 avec le circuit mixte. Il permet de connecter ou d'isoler le bus de test analogique interne du composant mixte au bus de test analogique externe. Le bus de test analogique interne est composé des deux lignes internes AB1 et AB2 qui distribuent chacune des cellules ABM. Le TBIC

permet également la réalisation des tests d'interconnexion entre les broches AT1 et AT2 des composants mixtes, et la caractérisation des mesures analogiques [BA00]. Le bus de

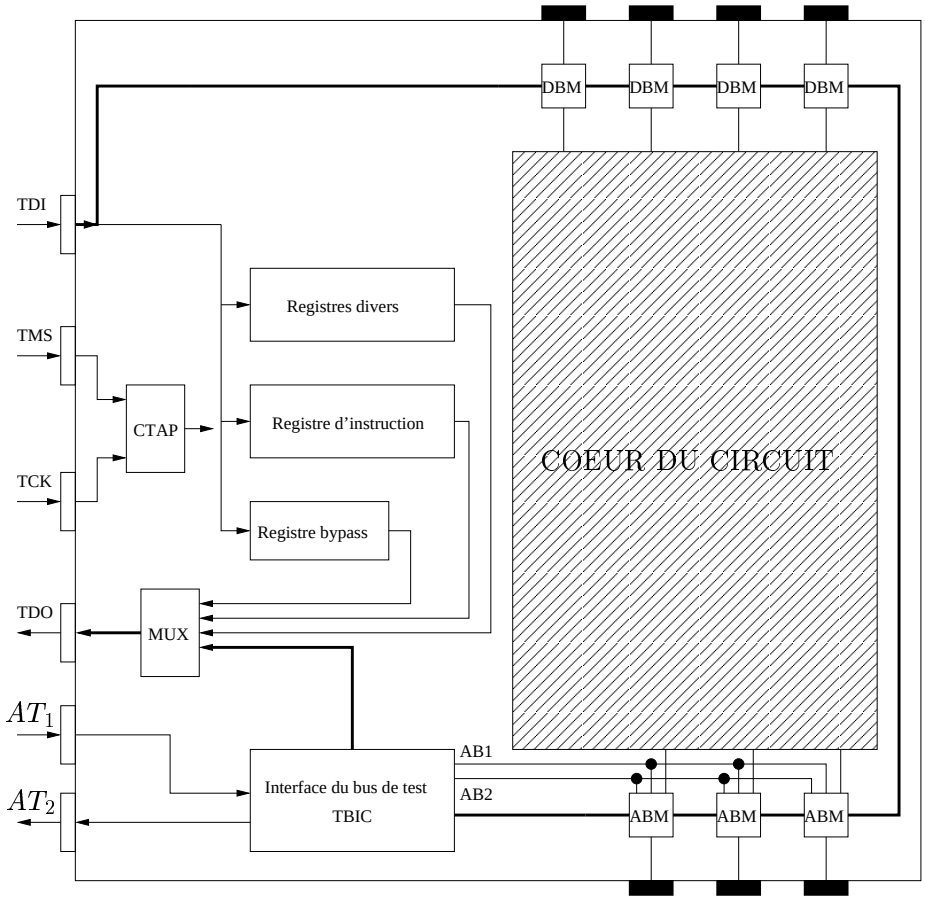

FIG. 4.8 – Architecture générale d'un composant respectant la norme IEEE 1149.4

test analogique externe, au niveau carte, composé de deux lignes, permet d'acheminer les signaux analogiques entre le testeur et les broches de l'ATAP de chacun des circuits composant la carte, comme le montre la figure 4.9 [ABB+04].

Dans un but de concision, nous ne décrivons pas les différentes configurations de fonctionnement du TBIC, ni l'architecture matérielle des cellules ABM et DBM. Le lecteur intéressé pourra consulter [IEE99, BA00, ABB+04]. Nous décrivons maintenant deux instructions particulières de la norme : *extest* et *intest*. Une présentation détaillée de toutes les instructions se trouve dans [IEE99, BA00, ABB+04]. L'instruction *extest* permet de tester les interconnexions simples de la même façon que le boundary scan, comme décrit en section 4.2.1. Le mode opératoire est exactement le même pour les lignes numériques. Pour les lignes analogiques, on applique sur celles-ci des tensions analogiques continues représentatives des états logiques zéro et un. L'instruction extest permet également d'effectuer un test paramétrique sur une interconnexion étendue. Dans ce type d'interconnexion, un composant analogique discret (comme par exemple

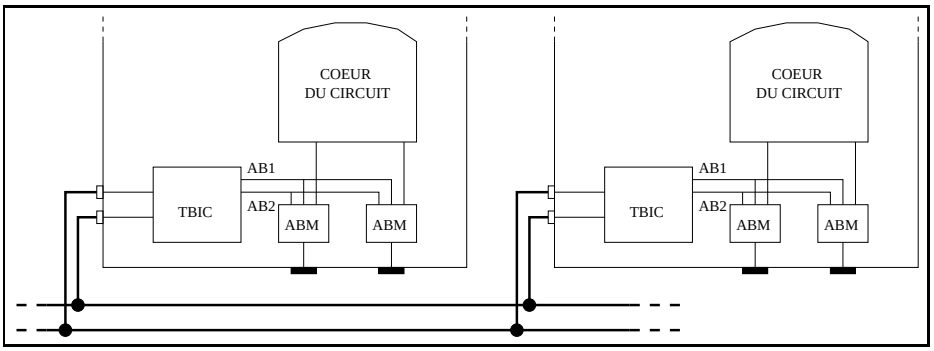

FIG. 4.9 – Bus de test analogique associé à la norme IEEE 1149.4

une résistance ou une capacité) relie une broche de chacun des deux circuits (ou d'un même circuit). Les composants analogiques discrets restent très utilisés dans les cartes mixtes. Ils permettent notamment, dans le cas des résistances, de dissiper plus de puissance que s'ils étaient intégrés dans les circuits mixtes. Par ailleurs, les valeurs des composants discrets sont plus précises que leurs équivalents intégrés. Le test paramétrique consiste à mesurer l'impédance du composant analogique discret présent sur la ligne. Le mode opératoire détaillé est donné dans [IEE99, BA00, ABB+04].

L'instruction *intest* permet le test interne d'un circuit mixte. Lorsque le circuit supporte cette instruction optionnelle, le registre de boundary scan doit obligatoirement posséder des cellules DBM qui interfacent les blocs numériques et analogiques du circuit (figure 4.10). Ces cellules additionnelles permettent de contrôler et d'observer les signaux d'interface entre les deux types de blocs. Le bloc numérique est déconnecté et testé suivant le principe de l'instruction *intest* de la norme IEEE 1149.1 (voir section 4.2.1). Le bloc analogique reste connecté. A un instant donné, une seule broche analogique peut être contrôlée via AT1 et une seule broche analogique peut être observée via AT2. Pour terminer, l'instruction *probe* (la seule nouvelle instruction définie par la norme par rapport au boundary scan) permet l'interaction réelle entre les blocs analogiques et numériques en laissant le bloc numérique connecté.

La norme IEEE 1149.4 est récente et elle ne bénéficie pas encore, à l'heure actuelle, d'une large acceptation des fabriquants de circuits et des concepteurs de cartes. Une des principales raisons de ceci vient du fait que l'objectif de la norme est de faciliter le test des circuits et des cartes mixtes, et non de fournir les moyens de test. Un système de test externe (source, système de mesure, logiciel) est nécessaire. On ne trouve pas à l'heure actuelle ce genre de système sur étagère. Cela contraste fortement avec le boundary scan dont l'une des raisons du succès est que la mise en oeuvre des tests est effectuée en utilisant une solution à base de PC ne nécessitant pas ou requérant peu de matériel supplémentaire.

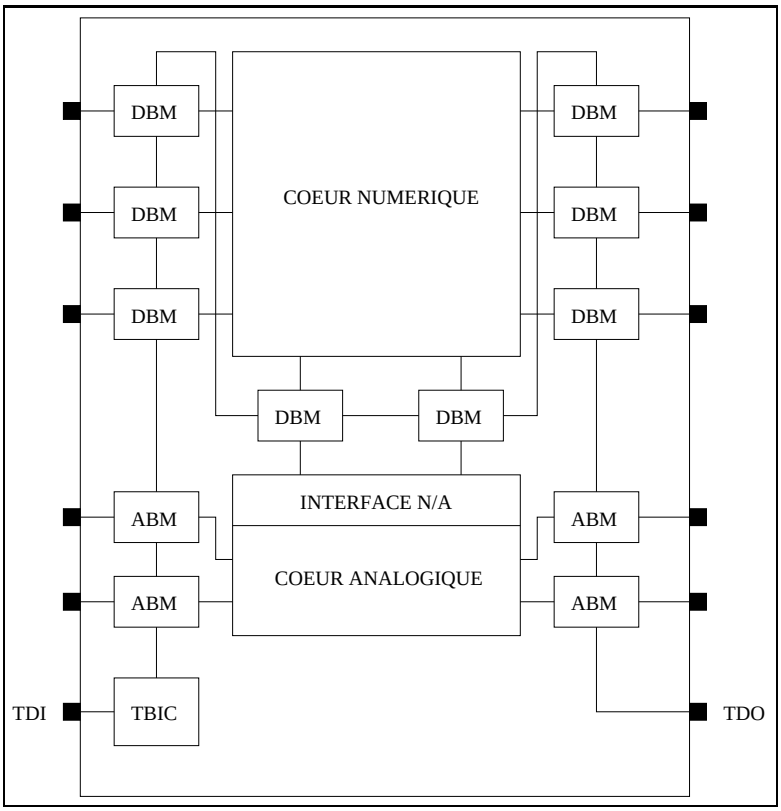

FIG. 4.10 – Circuiterie de test de l'instruction INTEST

4.3 La communauté du test matériel et le test de cartes

A ce jour, la communauté du test matériel s'intéresse beaucoup plus au test de circuits et de systèmes sur puce (*System On Chip* ou *Soc*) mixtes et RF (*Radio frequency*) [JBR+08] qu'au test de cartes. Nous en voulons pour preuve que la plus grande conférence internationale dédiée au test matériel (*International Test Conference ou ITC*) a consacré, ces dix dernières années, moins de 10% de ses publications et panels au test de cartes. Et parmi ceux-ci, la plupart se sont intéressés au boundary scan, ou plus récemment au test des interconnexions entre composants de type mémoire [P1501, ERT06, Ehr07]. Pourtant, le test de cartes reste un challenge important pour les fabricants de cartes qui doivent traiter des volumes de production importants et des cartes de plus en plus complexes [Ekl02].

Chapitre 5

Bilan

Les chapitres précédents présentant l'état de l'art du test des circuits et des cartes électroniques font apparaître clairement plusieurs points importants :

- beaucoup de travaux de recherche s'intéressent au test de production des circuits intégrés (niveau composant),
- comparativement, la communauté du test matériel est peu active en ce qui concerne le test de cartes électroniques,
- l'essentiel du test au niveau carte consiste en l'élaboration ou l'évolution de normes de test.

Le manque d'intérêt pour le test de cartes est principalement dû à une idée reçue. Celle-ci consiste à penser que le test de la carte nue à l'aide des méthodes d'inspection de type optique ou rayon X, puis le test de la carte assemblée avec des composants réputés bons (*Known Good ou KG*) [Ekl04] en utilisant le test in situ (voir section 4.1.1) ou le boundary scan (voir section 4.2.1), est suffisant. À première vue, cette approche semble correcte car les techniques utilisées sont bien maîtrisées et semblent adéquates pour détecter la plupart des défauts de fabrication de la carte (courts-circuits, circuits ouverts, composants manquants, mal montés, ou inadéquats). Cette manière de voir les choses explique pourquoi la communauté s'intéresse au test de composants plutôt qu'à celui des cartes. Pourtant, ce point de vue présente deux grandes faiblesses. D'une part, l'utilisation des technologies récentes comme les liaisons par radio RF ou les liaisons optiques et série à haut débit commence à montrer les limites des techniques de test au niveau carte. Par exemple, le test in situ devient infaisable à cause de problèmes liés à l'intégrité des signaux. Un autre exemple concerne le boundary scan qui devient inapplicable pour les circuits numériques à haute vitesse (voir section 4.2.2). D'autre part, il devient difficile de supposer que les composants montés sur la carte sont exempts de défauts. En effet, les circuits intégrés sont toujours plus complexes. Il devient de plus en plus difficile de vérifier leur bonne conception. De plus, pour les tests de production, les circuits actuels commencent à montrer les limites des équipements automatiques de test.

D'autres raisons sont également données pour expliquer le désintérêt concernant le test de cartes [But02]. Premièrement, le niveau d'intégration croissant et la prolifération

des systèmes sur puce ont déporté le problème du test de carte vers le problème du test de puce. Deuxièmement, la révolution des appareils portables (téléphone, PDA, ...). Si l'on prend l'exemple des téléphones portables, ceux-ci ne sont guère plus qu'un système sur puce, un écran, un clavier, une alimentation et une circuiterie de support. Cette architecture matérielle fait que le support, i.e. la carte « disparaît ». Encore une fois, ces raisons expliquent pourquoi la recherche actuelle met fortement l'accent sur le test de composants.

Mais il existe encore de gros systèmes sur cartes, à longue durée de vie, et à niveau de fiabilité requis élevé. Pour ces cas, le test fonctionnel au niveau carte semble dorénavant prometteur [Nel03]. Les méthodes de test structurelles au niveau composant et au niveau carte ne permettent pas d'affirmer que la carte respectera au final ses spécifications fonctionnelles. Contrairement au test structurel, le test fonctionnel au niveau carte permet de vérifier les interactions entre les différents composants de la carte. Le test fonctionnel peut être utilisé en phase de production mais également plus en amont du cycle de vie. Ainsi, le test fonctionnel peut servir à valider la conception de la carte. De même, le test fonctionnel peut être utilisé plus en aval du cycle de vie. Il peut servir en phase de maintenance à vérifier d'une part que le comportement de la carte ne varie pas au cours du temps (maintenance préventive). D'autre part, le test fonctionnel peut, en phase de maintenance, constituer une aide à la localisation de panne dans la carte et à la réparation (maintenance corrective).

Dans cette thèse, nous nous intéressons au test de cartes électroniques mixtes en phase de maintenance. A notre connaissance, il n'existe pas de méthodologie de test spécifique au test en maintenance de cartes électroniques mixtes. Dans la deuxième partie du document, nous proposons et mettons en oeuvre une telle méthodologie.

Deuxième partie

Proposition et mise en oeuvre d'une méthodologie pour le test de cartes mixtes

Chapitre 6

Démarche

L'objet de notre étude est le test de cartes électroniques mixtes en phase de maintenance. Dans le bilan présenté à la fin de la première partie, nous avons montré que les travaux de recherche actuels relatifs au test de cartes électroniques ne sont pas adaptés au test en maintenance.

Dans cette partie du document, nous présentons une méthodologie de test qui offre une réponse à la problématique du test de cartes mixtes en phase de maintenance. Cette méthodologie constitue le premier but de notre travail de thèse. Elle repose sur :

- un formalisme de représentation uniforme qui permet de modéliser les différentes fonctionnalités (analogique, numérique, mixte) de la carte, ainsi que la connexion aux équipements de test, et qui est bien adapté à une connaissance plus ou moins fine des spécifications de la carte,
- la prise en compte de l'expertise et du savoir-faire des ingénieurs de test (pratiques industrielles) sous forme de tactiques de test exprimées dans le même formalisme,
- la génération automatique des données de test fonctionnelles à partir de stratégies de test globales (ciblant la maintenance préventive) et de stratégies de test locales (ciblant la maintenance corrective).

Notre méthodologie de test et son application à un cas d'étude sont présentés au chapitre 7.

Le deuxième but de notre travail est de mettre en oeuvre cette méthodologie en réalisant un outil prototype. En effet, nous pensons que la méthodologie ne sera utile et utilisable par des professionnels du test uniquement que si ceux-ci disposent d'un outil qui répond à leurs besoins. Cet outil doit :

- être convivial et facile à utiliser,
- proposer une bibliothèque de composants couramment rencontrés sur les cartes mixtes,
- être ouvert en autorisant la prise en compte de nouveaux composants et de nouvelles tactiques de test,
- permettre de générer automatiquement sans intervention de l'utilisateur les données de test liées au stratégies de test globales,
- permettre de générer de manière interactive les données de test liées au stratégies

de test locales.
Les détails de la mise en oeuvre de la méthodologie et la présentation de l'outil prototype se trouvent au chapitre 8.

Durant nos travaux, nous n'avons pas eu l'opportunité de disposer physiquement de cartes mixtes ni de banc de test adéquat. En conséquence, afin de vérifier l'adéquation de notre méthodologie au test en maintenance de cartes mixtes, nous avons défini un ensemble de modèles de cartes mixtes à tester dont certains sont issus de cartes réelles. En utilisant notre méthodologie, nous avons modélisé chacune des cartes, défini les tactiques de test et généré les données de test pour différentes stratégies de test. Puis, nous avons simulé le comportement des cartes sur les données de test générées en utilisant un outil de simulation adapté et comparé les sorties observées avec celles prédites. Ce protocole de validation est détaillé au chapitre 9.

Chapitre 7

Méthodologie pour le test de cartes mixtes

Dans ce chapitre, nous décrivons une méthodologie de test pour les cartes électroniques mixtes en phase de maintenance permettant l'automatisation, en grande partie, du processus de génération des données de test. Nous commençons par exposer les fondements de cette méthodologie (section 7.1). Nous présentons ensuite la modélisation des signaux (section 7.2) et la méthodologie de modélisation d'une carte mixte (section 7.3), puis le processus de génération des données de test (section 7.4). Nous terminons ce chapitre par un bilan (section 7.5).

7.1 Principe de la méthodologie

Notre but est d'aider véritablement les ingénieurs de test en maintenance dans leur tâche. Ainsi, il nous semble indispensable d'intégrer leur expertise (pratiques industrielles) et de proposer une automatisation (au moins partielle) du processus de génération des données de test. De plus, toujours dans le but de procurer une véritable aide, le formalisme de représentation induit par le besoin d'automatisation doit être adapté au caractère mixte des cartes et aisément appréhendable par les futurs utilisateurs. La figure 7.1 résume ces idées.

Nous avons vu dans le chapitre 5 que la maintenance de cartes électroniques mixtes nécessite une approche fonctionnelle du test. La nécessité d'avoir une approche fonctionnelle du test implique une modélisation fonctionnelle de la carte. Ainsi, dans la méthodologie proposée, une carte est modélisée par un ensemble de fonctions numériques, analogiques et mixtes. À cette fin, la méthodologie permet de choisir des fonctions prédéfinies ou d'en créer de nouvelles. En particulier, une bibliothèque de fonctions analogiques et mixtes, modélisant le comportement de composants classiques (filtres, comparateurs, $\cdots$) et de sources (générateurs) est proposée. Il est également possible de spécifier des fonctions de type « boîte noire » lorsqu'un composant de la carte est inconnu. Une fonction boîte noire ne contient alors qu'un ensemble de vecteurs de test. Dans tous les cas, la modélisation fonctionnelle d'un composant est décrite par un *mo-*

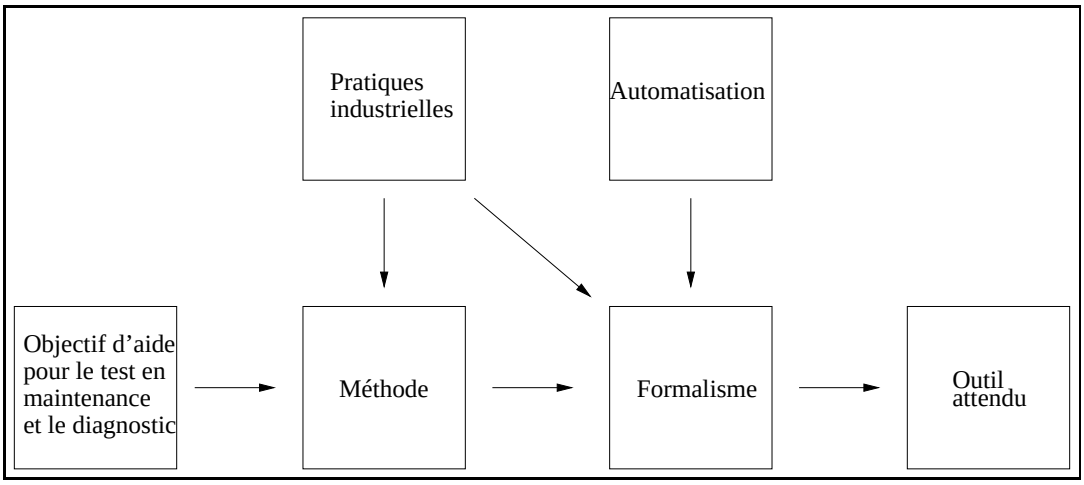

FIG. 7.1 – Méthodologie : les idées principales

dèle fonctionnel.

Le plus souvent, la modélisation du comportement d'une carte n'est pas suffisante pour produire des données de test pertinentes. Ainsi, par exemple, la fonction de transfert d'un filtre analogique passe-haut ne permet pas d'inférer les points en fréquence intéressants à tester. C'est alors qu'intervient le savoir-faire de l'ingénieur de test qui va effectuer par exemple une mesure dans la bande passante et une mesure à la fréquence de coupure en considérant que ces deux mesures sont suffisantes pour tester efficacement le filtre. Cette expertise apparaît sous la forme de *modèles de test* et de *tactiques de test* dans notre approche, illustrée figure 7.2.

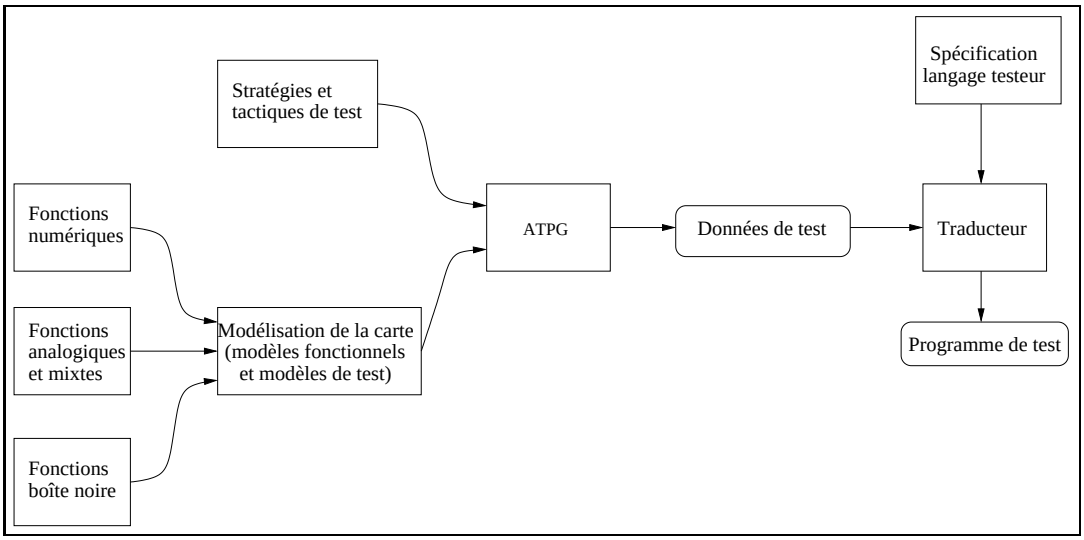

FIG. 7.2 – Processus de génération des données de test

Nous avons abordé dans l'introduction de ce document la nécessité d'effectuer de la

maintenance préventive et de la maintenance corrective. Afin de répondre à ces besoins, la méthodologie propose des *stratégies de test*. Une stratégie de test peut être *globale* ou *locale*. Une stratégie de test globale permet de tester l'ensemble des fonctionnalités de la carte et constitue une réponse au besoin de la maintenance préventive. Une stratégie de test locale permet de tester plus finement un comportement particulier de la carte. Elle constitue ainsi une aide à la localisation de panne et répond au besoin de la maintenance corrective.

Le générateur des données de test (boîte ATPG sur la figure 7.2) génère les données de test de la carte à partir de sa modélisation (modèles fonctionnels et modèles de test) et des stratégies et éventuelles tactiques de test choisies. Ces données de test peuvent ensuite être encodées dans un programme de test s'exécutant sur un testeur.

Après cette présentation des fondements de notre méthodologie, nous présentons dans la section qui suit l'approche de modélisation de signaux adoptée dans cette dernière.

7.2 Modélisation des signaux

Nous commençons par donner les définitions des types de signaux qui nous intéressent, dans le cadre des cartes mixtes.

Définition 7.1 *Un signal analogique est un signal à temps continu [Kun88].*

Définition 7.2 *Un signal discret est une suite de valeurs numériques [Kun88].*

7.2.1 Exemples de signaux analogiques

Un exemple de signal analogique est le signal sinusoïdal $s(t-t_0)$ de période T_0 représenté figure 7.3 et défini par :

$$\begin{aligned} s(t-t_0) &= A\sin(2\pi F_0(t-t_0)) && (7.1)\\ &= A\sin(2\pi F_0 t-\phi_0) && (7.2)\end{aligned}$$

avec $\phi_0 = 2\pi F_0 t_0$, où F_0 est la fréquence du signal ($F_0 = 1/T_0$) et t_0 son retard. ϕ_0 représente le déphasage du signal.

Un autre exemple de signal analogique est le signal rectangulaire $rect(t-t_0)$ de période T représenté figure 7.4 et défini par :

$$rect(t-t_0) = \begin{cases} 1 & si \quad t \in [t_0, t_0+\Delta T_1] \\ 0 & si \quad t \in]t_0+\Delta T_1, t_0+\Delta T_0+\Delta T_1[\end{cases} \qquad (7.3)$$

où t_0 est le retard, avec la relation :

$$T = \Delta T_0 + \Delta T_1 \qquad (7.4)$$

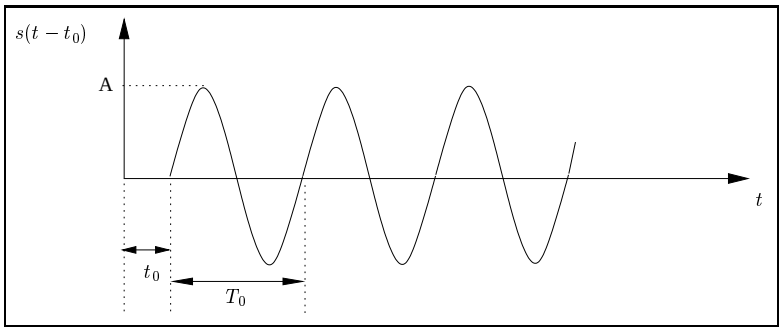

FIG. 7.3 – Signal analogique sinusoïdal

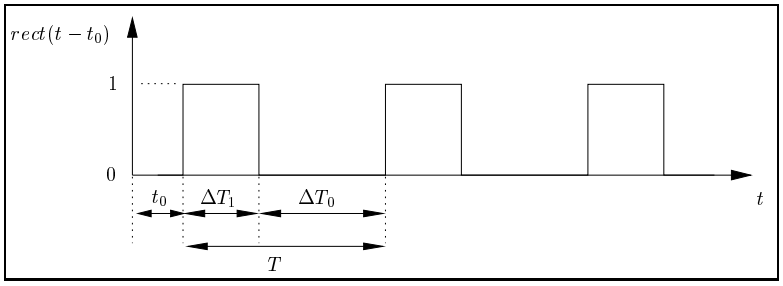

FIG. 7.4 – Signal analogique rectangulaire

7.2.2 Exemple de signal discret

Un exemple de signal discret est le signal sinusoïdal discret $s((n-k_0)\tau)$ de période T_0 défini par :

$$s((n-k_0)\tau) = A\sin(2\pi F_0(n\,\tau - k_0\,\tau)) \quad (7.5)$$
$$= A\sin(2\pi F_0 n\,\tau - \phi_0) \quad (7.6)$$

avec $\phi_0 = 2\pi F_0 k_0\,\tau$, où F_0 est la fréquence du signal et k_0 son retard. n et k_0 sont des entiers relatifs et τ représente la période d'échantillonnage. Le signal (7.5) est représenté sur la figure 7.5 avec $k_0 = 1$ et $\tau = \frac{T_0}{4}$. Ce signal correspond à la séquence de points noirs sur cette même figure. Le signal représenté en pointillés représente le signal analogique de même forme que le signal discret.

7.2.3 Représentation du temps

Lorsqu'on teste une carte électronique, il est nécessaire de connaître la valeur de signaux à des instants précis. C'est par exemple le cas lorsque l'on veut tester la synchronisation de deux signaux.

Définition 7.3 *Un échantillon est la valeur d'un signal analogique à un instant donné dans le domaine de définition du signal.*

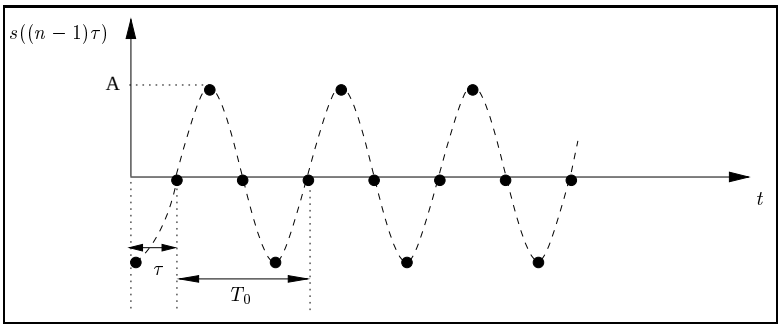

FIG. 7.5 – Signal discret sinusoïdal

Pour calculer un échantillon d'un signal à l'instant t, il est nécessaire de se synchroniser à cet instant. Cette synchronisation s'effectue par l'utilisation d'un événement estampillé temporellement par t, délivré par une *horloge* et appelé *top*. Dans la pratique, nous aurons besoin d'obtenir non pas un seul échantillon d'un signal, mais un ensemble d'échantillons. Par conséquent, il est nécessaire de considérer non pas un seul top, mais un ensemble de tops possédant une référence de temps unique. Cela nous amène à la définition suivante d'une horloge :

Définition 7.4 *Une horloge délivre des tops qui sont des points de synchronisation par rapport à une référence de temps unique. La différence entre l'instant d'un top d'horloge et la référence de temps est une date. Toutes les horloges sont calées sur la même référence de temps.*

Il est important de préciser qu'un top n'est par définition ni un signal analogique, ni un signal discret. Il est modélisé par la structure *top* possédant un unique champ qui correspond à l'instant du top :

$$top(instant) \tag{7.7}$$

Nous présentons maintenant la modélisation des signaux analogiques, puis celle des signaux discrets.

7.2.4 Modélisation des signaux analogiques

Nous proposons de modéliser les signaux analogiques par une structure

$$siga(forme, param1, param2, param3)$$

Un signal est caractérisé par plusieurs paramètres qui sont représentés comme des champs de la structure *siga*. Le premier champ de *siga*, appelé toujours $forme$, désigne la forme du signal. Le nom et la signification des autres champs dépendent de la forme du signal. Nous avons limité, dans un but de simplicité, à trois le nombre de paramètres caractérisant un signal d'une forme donnée. Cependant, le nombre de paramètres de *siga* peut facilement être étendu afin de modéliser d'autres types de signaux que ceux déjà pris en compte.

Un signal analogique sinusoïdal non instancié est représenté par la structure $siga$ suivante :

$$siga(forme, ampl, frq, phi) \tag{7.8}$$

où $forme$ est toujours égal à $sinus$ et où $ampl$, frq, et phi représentent respectivement l'amplitude, la fréquence et le déphasage. Un signal sinusoïdal instancié est représenté par une valuation de (7.8). Ainsi, le signal (7.2) est représenté par :

$$siga(sinus, A, F_0, \phi_0) \tag{7.9}$$

De même, un signal analogique rectangulaire périodique non instancié est représenté par :

$$siga(forme, dt1, prd, dly) \tag{7.10}$$

où $forme$ est toujours égal à $rect$ et où $dt1$, prd et dly représentent respectivement la durée pour laquelle le signal vaut un sur une période, prd la période et dly le retard. Un signal rectangulaire instancié est représenté par une valuation de (7.10). Ainsi, le signal (7.3) est représenté par :

$$siga(rect, \Delta T_1, T, t_0) \tag{7.11}$$

La notation pointée permet d'accéder aux champs du signal. A titre d'exemple, pour le signal défini par l'expression

$$MonSignal = siga(sinus, A, F_0, \phi_0) \tag{7.12}$$

nous avons :

$$\begin{cases} MonSignal.forme = sinus \\ MonSignal.ampl = A \\ MonSignal.frq = F_0 \\ MonSignal.phi = \phi_0 \end{cases} \tag{7.13}$$

Cette notation est utilisée pour exprimer des conditions sur les signaux (cf. section 7.3.2.1).

Le tableau 7.1 indique les différentes formes de signaux modélisés. Il précise pour chaque signal, le nom de sa forme ainsi que le nom de ses champs. Le symbole « – » signifie que le champ est inutilisé.

TAB. 7.1 – Les différents signaux analogiques modélisés

Nom	Forme	Param1	Param2	Param3
signal sinusoïdal	sinus	ampl	frq	phi
signal rectangulaire	rect	dt1	prd	dly
signal DC	DC	valeur	-	-

7.2.5 Échantillonnage d'un signal analogique

Un échantillon d'un signal s modélisé par une structure $siga$ obtenu à l'instant t sera noté

$$E(s,t) \tag{7.14}$$

Ainsi, par exemple, l'échantillon du signal (7.2)

$$s = siga(sinus, A, F_0, \phi_0)$$

à l'instant $t = t_0 + T_0/4$ est noté

$$E(s, t_0 + T_0/4)$$

et vaut A.

De la même manière, l'échantillon du signal (7.3)

$$s = siga(rect, \Delta T_1, T, t_0)$$

à l'instant $t = t_0 + \frac{\Delta T_1}{2}$ est

$$E(s, t_0 + \frac{\Delta T_1}{2})$$

et vaut 1.

7.2.6 Modélisation des signaux discrets

Nous proposons de modéliser les signaux discrets par une structure

$$sigd(sa, date) \tag{7.15}$$

Le premier champ sa désigne un signal analogique de même forme que la séquence de valeurs formées par le signal discret. Le deuxième champ *date* permet d'identifier une valeur particulière de la séquence. Ainsi, la structure $sigd$ permet de représenter un signal discret (séquence de valeurs numériques) et de désigner la valeur du signal (une valeur particulière de la séquence) à l'instant *date*.

Ainsi, par exemple, l'expression

$$sigd(siga(sinus, A, F_0, \phi_0), 2\tau)$$

modélise le signal discret représenté figure 7.5 (la séquence de points noirs) et désigne la troisième valeur de la séquence (valeur de la courbe dessinée en pointillés à la date 2τ).

Comme pour les signaux analogiques, l'accès aux champs se fait en utilisant la notation pointée. Par exemple, si x est un signal discret, $x.sa$ permet d'accéder au signal analogique que le modèle véhicule. De même, $x.date$ permet d'accéder à son champ date.

7.3 Modélisation de la carte

Physiquement, une carte électronique est constituée d'un support sur lequel sont montés un ensemble de composants électroniques interconnectés. Ces composants effectuent des traitements et échangent entre eux des signaux. Certains composants (d'interface) reçoivent des signaux externes via les entrées primaires de la carte. D'autres envoient des signaux externes via les sorties primaires. Dans un environnement de test, comme par exemple un banc de test, les entrées et les sorties primaires de la carte sont reliées à des équipements de test qui permettent d'appliquer les stimuli de test (aux entrées primaires) et de mesurer les réponses (aux sorties primaires). D'un point de vue fonctionnel, on peut voir une carte comme une composition de blocs fonctionnels qui interagissent entre eux et/ou avec l'extérieur par des signaux.

Dans notre approche, nous considérons ainsi une modélisation fonctionnelle hiérarchique à deux niveaux : le *niveau carte* et le *niveau bloc*. Le niveau carte représente une carte comme étant composée d'un ensemble de blocs fonctionnels qui échangent entre eux des signaux. Ce niveau permet également de représenter des blocs externes à la carte qui modélisent les connexions entre les entrées/sorties primaires de la carte et les équipements de test. Le niveau bloc permet de décrire la fonctionnalité d'un bloc.

Nous exposons tout d'abord la modélisation au niveau carte (section 7.3.1), puis celle au niveau bloc (section 7.3.2). Enfin, nous donnons un exemple de modélisation d'une carte mixte (section 7.3.3).

7.3.1 Niveau carte

Comme nous l'avons mentionné plus haut, une carte est modélisée au « niveau carte » par un ensemble de blocs fonctionnels qui échangent des signaux. Un bloc fonctionnel est un bloc qui possède des entrées et des sorties et qui décrit une fonction liant ses entrées et ses sorties. La figure 7.6 montre un exemple de cette décomposition où les blocs fonctionnels sont représentés par les rectangles b_1 à b_5. Les échanges de signaux entre les blocs fonctionnels sont représentés par des liens orientés. Par ailleurs, il est nécessaire de modéliser les connexions entre les entrées/sorties primaires de la carte et les équipements (de test) externes. Ces connexions sont modélisées en utilisant des blocs d'entrée/sortie. Un *bloc d'entrée* est un bloc externe à la carte qui modélise la connexion d'un équipement (de test) externe aux entrées de la carte. De même, un *bloc de sortie* est un bloc externe à la carte qui modélise la connexion d'un équipement (de test) externe aux sorties de la carte. De plus, comme nous l'avons mentionné en section 7.2, nous avons besoin d'horloges afin de connaître la valeur de signaux à des instants donnés. Un *bloc horloge* est un bloc d'entrée particulier qui modélise une horloge. Dans la figure 7.6, les blocs d'entrée (respectivement de sortie) sont représentés par les carrés s_1, s_2 (respectivement m_1, m_2). Les blocs d'entrée s_1 et s_2 représentent des sources externes qui fournissent des signaux aux entrées primaires de la carte. Les blocs de sortie m_1 et m_2 représentent des points de mesure externes qui observent les signaux délivrés par les sorties primaires de la carte. Les entrées et sorties primaires de la carte sont représentées par des carrés noirs. Le bloc h_1 représente une horloge.

Un bloc fonctionnel modélise une fonction analogique, numérique ou mixte. La modélisation au niveau bloc est présentée dans la section suivante.

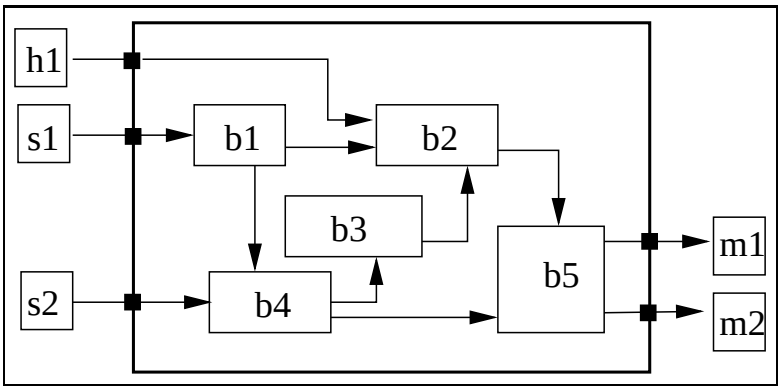

FIG. 7.6 – Modélisation au niveau carte

7.3.2 Niveau bloc

A chaque bloc (fonctionnel, entrée/sortie et horloge) est associé un *modèle fonctionnel*. De plus, un *modèle de test* est associé à chaque bloc fonctionnel (mais pas aux blocs d'entrée/sortie et horloge qui correspondent à des éléments extérieurs à la carte). Le modèle fonctionnel d'un bloc spécifie son comportement en décrivant les relations entre ses entrées et ses sorties. Le modèle de test d'un bloc décrit la manière dont celui-ci peut être testé de manière efficace. Ces deux modèles utilisent le formalisme des automates (ou machines) à états finis communicants (*Communicating Finite State Machine ou CFSM*) [LY96, BZ83]. Nous avons choisi ce formalisme car il est bien adapté pour modéliser des systèmes formés de processus communicants. Une carte électronique est composée d'un ensemble de blocs. Chaque bloc peut être vu comme un processus qui communique avec d'autres blocs par échange de signaux. De plus, le formalisme des CFSM est bien connu dans la communauté des ingénieurs de test, ce qui peut aider à l'appropriation de la méthodologie. Nous présentons maintenant le formalisme des automates à états finis communicants. Nous détaillons ensuite les notions de modèle fonctionnel et de modèle de test.

7.3.2.1 Les automates à états finis communicants

Définition 7.5 *Un automate à états finis est composé d'un ensemble fini d'états et de transitions. Il est associé à un graphe orienté (appelé diagramme des transitions) dont les sommets et les arcs correspondent respectivement aux états et transitions de l'automate [HU79].*

Définition 7.6 *Un automate à états finis est dit communicant lorsqu'il communique avec d'autres automates en utilisant des files d'attente d'entrée. Deux automates communiquent lorsque l'un poste un message dans la file d'entrée de l'autre [Hie01].*

Les transitions d'un automate à états finis communicant sont décorées avec des étiquettes. Une étiquette est une expression qui décrit les conditions et les actions au franchissement de la transition (expression logique, envoi et/ou réception de messages entre CFSM). L'envoi d'un message *msg* vers un CFSM B est noté [LY96] :

$$B\,!\,msg \tag{7.16}$$

De même, la réception d'un message *msg* émis par un CFSM B est notée [LY96] :

$$B\,?\,msg \tag{7.17}$$

Les transitions d'un CFSM peuvent être de différents types. Une transition est dite *émettrice* si son étiquette décrit uniquement l'envoi de messages. Une transition est dite *réceptrice* si son étiquette décrit uniquement la réception de messages. Une transition est *émettrice/réceptrice* si son étiquette décrit l'émission et la réception de messages. La figure 7.7 montre un exemple de deux CFSM possédant chacun une transition émettrice et une transition réceptrice.

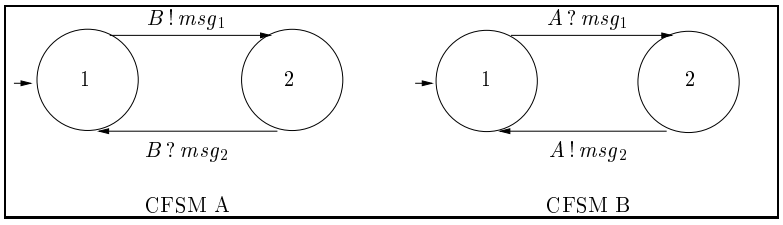

FIG. 7.7 – Exemple simple d'automates à états finis communicants.

Dans notre approche, la réception d'un message est bloquante et l'émission non bloquante. La communication entre deux CFSM est synchronisée par la réception d'un message. Pour illustrer le principe de la réception bloquante et de l'envoi non bloquant, prenons l'exemple de la figure 7.7. Le CFSM A atteint immédiatement l'état 2 à partir de l'état initial [1] 1 en envoyant le message msg_1 vers le CFSM B sans attendre que le CFSM B ait reçu le message (envoi non bloquant). Une fois dans l'état 2, A ne repasse dans l'état 1 que lorsque msg_2 est reçu de B (réception bloquante).

Nous pouvons voir aisément que dans cet exemple, les transitions réceptrices sont franchies inconditionnellement dès qu'un message est reçu. Cela traduit un comportement très simple. Pourtant, il est nécessaire de modéliser des comportements plus complexes où l'état suivant d'un CFSM dépend à la fois de son état courant et de la valeur du message reçu (modélisation par exemple des circuits séquentiels). Cela revient à dire que le franchissement d'une transition réceptrice devient conditionnel. Pour ce faire, une garde apparaît dans l'étiquette d'une transition réceptrice, étendant la notation (7.17) à

$$B\,?msg\ \rightarrow G \tag{7.18}$$

où G représente la garde qui est une expression booléenne.

[1] L'état initial du CFSM est représenté graphiquement avec une flèche droite entrante.

Définition 7.7 *La sémantique d'une réception avec garde (aussi appelée réception gardée) est la suivante : lorsque le message est reçu, la garde est évaluée et la transition correspondante n'est franchie que si la garde est vraie.*

La figure 7.8 montre une version enrichie des automates présentés figure 7.7 où les réceptions des CFSM A et B possèdent des gardes. Maintenant, le CFSM A ne peut revenir dans son état initial (état 1) que si msg_1 et msg_2 valent respectivement 2 et 1 (l'opérateur == représente le test d'égalité).

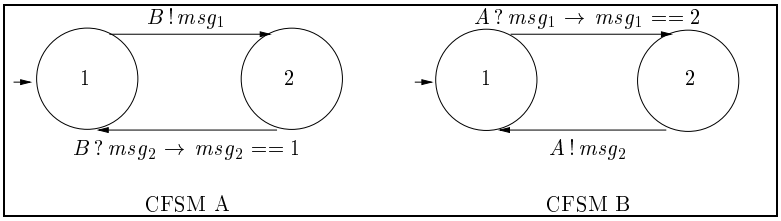

FIG. 7.8 – Exemple simple d'automates à états finis communicants avec des réceptions gardées.

En fait, une garde exprime, de manière générale, un ensemble de conditions qui doivent être satisfaites pour le franchissement d'une transition. La modélisation de certains comportements nécessite d'associer également des conditions aux émissions de messages. C'est par exemple le cas pour la modélisation du signal de sortie d'un circuit analogique qui est d'une forme particulière.

Définition 7.8 *La sémantique d'une émission avec garde (aussi appelée émission gardée) est la suivante : la garde est évaluée et la transition correspondante n'est franchie, i.e. le message n'est envoyé, que si la garde est vraie.*

Nous associons ainsi une garde à une émission de message en étendant la notation (7.16) à

$$\gamma \,:\, B\,!\,msg \tag{7.19}$$

où la garde γ représente une condition préalable à l'envoi du message msg vers le CFSM B. La figure 7.9 montre des conditions qui sont ajoutées pour l'émission du message msg_2 par le CFSM B. Sur cet exemple, la valeur de msg_2 est bornée dans l'intervalle $[0, 3]$. Notons que sur cet exemple, la communication entre les CFSM A et B est possible car la garde de la transition sortante de l'état 2 de A est compatible avec la garde de la transition sortante de l'état 2 de B. Si la transition réceptrice de A devenait $B\,?\,msg_2 \rightarrow msg_2 == 4$, il n'y aurait plus de communication possible entre A et B.

Jusqu'ici, nous avons illustré les mécanismes de communication des CFSM par des transitions n'échangeant qu'un seul message. Il est cependant intéressant de modéliser la réception ou l'envoi en séquence de plusieurs messages par un CFSM, i.e. de pouvoir représenter la réception bloquante de plusieurs messages et l'envoi non bloquant de plusieurs messages.

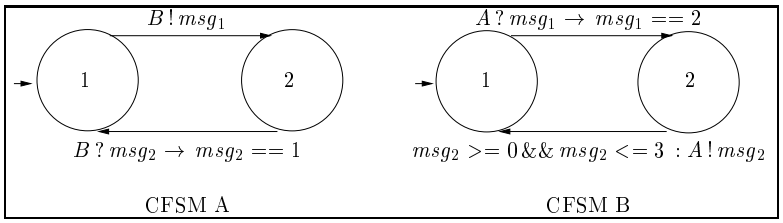

FIG. 7.9 – Exemple simple d'automates à états finis communicants avec des réceptions et émissions gardées

La notation

$$B\,?\,msg_B, C\,?\,msg_C, \cdots \rightarrow G \tag{7.20}$$

représente la réception gardée bloquante de plusieurs messages provenant de plusieurs CFSM et la notation

$$\gamma\ :\ B\,!\,msg_B, C\,!\,msg_C, \cdots \tag{7.21}$$

représente l'envoi gardé non bloquant de plusieurs messages vers plusieurs CFSM. Les conditions exprimées par les gardes G dans (7.20) et γ dans (7.21) s'appliquent à l'ensemble des envois/réceptions. Notons que chaque message peut être un signal atomique ou une structure regroupant plusieurs signaux. La figure 7.10 montre un exemple d'utilisation de réception multiple. Il s'agit de la transition de l'état 1 vers l'état 2 dans le CFSM B. En effet, il y a attente de la réception du message msg_1 envoyé par A et du message msg_3 envoyé par C. Il est à noter sur cet exemple que la garde de cette même transition exprime une condition uniquement sur la valeur de msg_3.

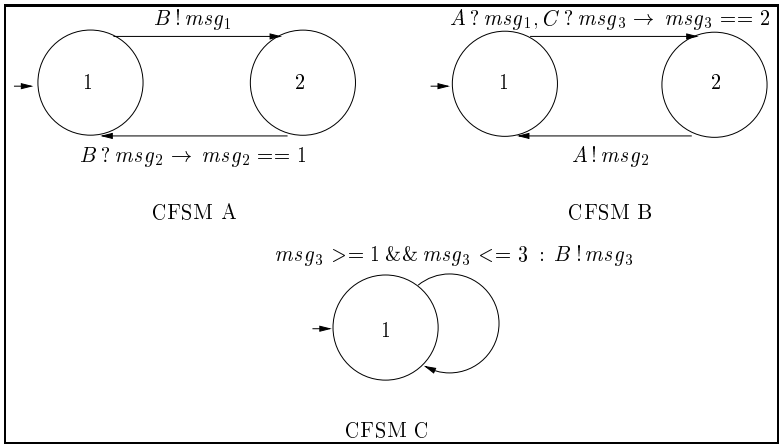

FIG. 7.10 – Exemple simple d'automates à états finis communicants utilisant la réception bloquante de plusieurs messages.

La forme la plus générale des transitions d'un CFSM est celle des transitions émettrices/réceptrices. La notation

$$B\,?\,msg_B, C\,?\,msg_C, \cdots \rightarrow \\ G\,[\gamma\ :\ D\,!\,msg_D, E\,!\,msg_E, \cdots] \tag{7.22}$$

représente la réception gardée de plusieurs messages provenant de plusieurs CFSM, puis l'émission gardée de plusieurs messages vers plusieurs CFSM (les gardes G et γ représentent respectivement les conditions sur les réceptions et les émissions). La notation crochet est utilisée pour distinguer la garde de réception de celle d'émission.

Définition 7.9 *La sémantique d'une émission/réception avec garde (aussi appelée émission/réception gardée) est la suivante : lorsque les messages sont reçus, la garde G associée à la réception des messages est évaluée. Si G est vraie, alors la garde γ associée à l'émission des messages est évaluée. Si γ est vraie, alors les messages sont émis et la transition est franchie. Sinon, les messages ne sont pas émis et la transition n'est pas franchie.*

La figure 7.11 montre l'exemple du CFSM B constitué d'un unique état et d'une unique transition émettrice/réceptrice.

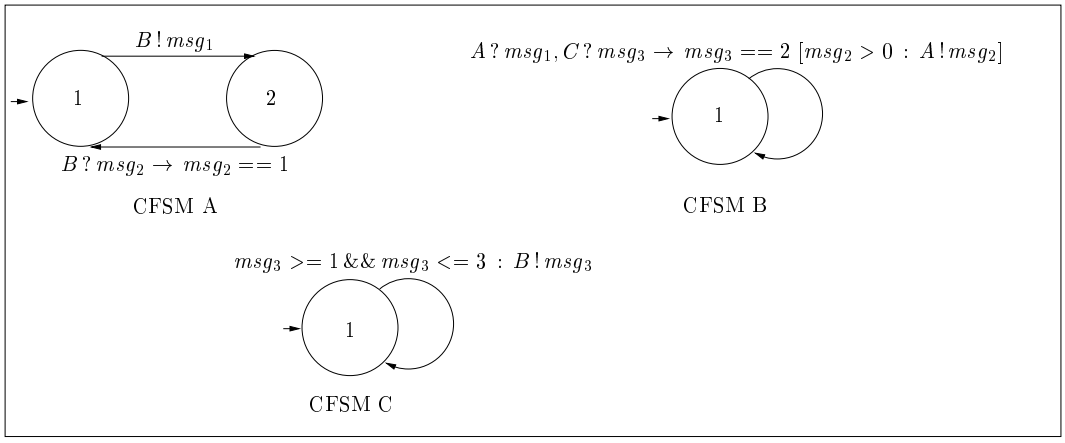

FIG. 7.11 – Exemple simple d'automate à états finis communicant possédant une transition émettrice/réceptrice.

Expression de conditions relatives aux signaux Dans le cadre de la modélisation des blocs d'une carte en CFSM, les gardes des transitions des CFSM permettent de décrire précisément les signaux échangés. Les signaux sont modélisés de la manière décrite en section 7.2. Une condition sur un signal est alors exprimée par un ensemble de conditions sur les champs de la structure *siga* ou *sigd* modélisant ce signal. Dans l'exemple présenté sur la figure 7.12, les conditions sur les signaux analogiques échangés par les CFSM sont exprimées par les gardes G_A, G_B, et γ_B qui valent respectivement

$$G_A : instanceOf(z) == siga\ \&\&\ z.forme == rect\ \&\&\ z.prd == 0.001$$

$$G_B : instanceOf(x) == siga \,\&\&\, x.forme == sinus \,\&\&$$
$$instanceOf(y) == siga \,\&\&\, y.forme == sinus \,\&\&$$
$$y.frq <= 1000$$

$$\gamma_B : instanceOf(z) == siga \,\&\&\, z.forme == rect$$

La fonction $instanceOf(s)$ renvoie la nature du signal s qui est analogique ou discret. La garde G_A signifie que le signal z reçu par A doit être un signal analogique ($instanceOf(z) == siga$) de forme rectangulaire ($z.forme == rect$) dont la période est égale à 0.001 seconde ($z.prd == 0.001$). La garde G_B signifie que les deux signaux x et y reçus par B doivent être des signaux analogiques ($instanceOf(z) == siga$, $instanceOf(y) == siga$) de forme sinusoïdale ($x.forme == sinus$, $y.forme == sinus$) et que la fréquence du signal y doit être inférieure ou égale à 1000 Hz ($y.frq <= 1000$). Finalement, la garde γ_B exprime le fait que le signal z envoyé par B doit être un signal analogique de forme rectangulaire.

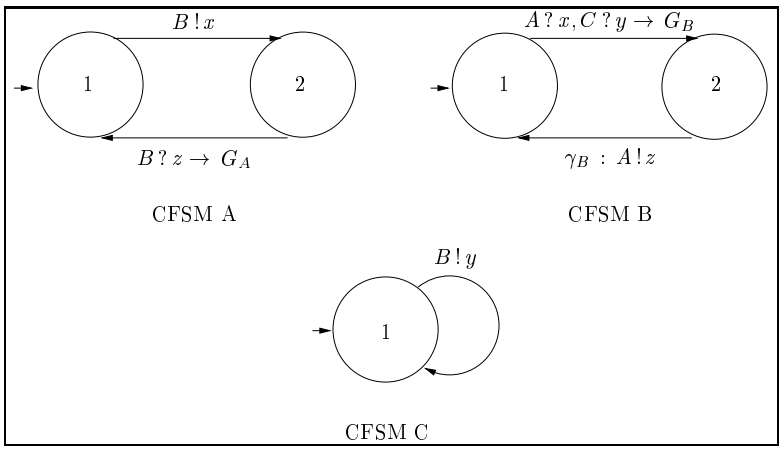

FIG. 7.12 – Exemple simple d'automates à états finis communicants échangeant des signaux

Une description plus formelle des étiquettes de transitions est donnée par une grammaire définissant la syntaxe du langage pour l'étiquette d'une transition [GNN07a] en annexe A.

La méthodologie de test que nous proposons utilise le formalisme des CFSM afin de modéliser une carte. Les CFSM étant introduits, nous présentons maintenant la modélisation d'un bloc fonctionnel à travers ses deux modèles associés : le modèle fonctionnel et le modèle de test.

7.3.2.2 Modèle fonctionnel

Nous avons introduit en section 7.3.1 les blocs fonctionnels, les blocs d'entrée/sortie et les blocs horloge. Nous présentons maintenant les modèles fonctionnels associés.

Définition 7.10 *Le modèle fonctionnel d'un bloc est un modèle qui spécifie le comportement de ce bloc en décrivant les relations entre ses entrées et ses sorties.*

Modèle fonctionnel d'un bloc fonctionnel Le modèle fonctionnel d'un bloc fonctionnel est constitué d'un ensemble de CFSM traitant les signaux d'entrée et de sortie du bloc. Ces signaux sont modélisés, suivant leur nature analogique ou discrète, par des structures *siga* ou *sigd* (cf. section 7.2). La relation entre les signaux de sortie et ceux d'entrée est exprimée par des conditions sur les transitions des CFSM (cf. section 7.3.2.1).

La figure 7.13 montre l'exemple du modèle fonctionnel d'un filtre analogique passe-haut du premier ordre dont la fonction de transfert complexe est

$$H(f) = \frac{1}{1 + j\frac{f_c}{f}}$$

où f désigne la variable fréquence et fc la fréquence de coupure du filtre. Ce modèle fonctionnel exprime le comportement du filtre lorsque le signal d'entrée est un signal sinusoïdal. Les CFSM *Amont* et *Aval* apparaissant dans les étiquettes des transitions sont les modèles fonctionnels des blocs liés respectivement à l'entrée et à la sortie du bloc F. Le CFSM F n'envoie vers le CFSM *Aval* un signal analogique sinusoïdal (de sortie) que s'il a reçu un signal analogique sinusoïdal (en entrée) provenant du CFSM *Amont* (garde de la transition réceptrice). Les caractéristiques du signal sinusoïdal de sortie dépendent de celles du signal sinusoïdal d'entrée et sont exprimées par la condition $\Gamma = \gamma_1$ && γ_2 && γ_3 dans la transition émettrice :

$$\Gamma = \begin{cases} \gamma_1 = (V_0 == \frac{x.ampl}{\sqrt{1+\frac{f_c^2}{F_0^2}}}) \\ \gamma_2 = (F_0 == x.frq) \\ \gamma_3 = (\phi_0 == x.phi - \arctan\left(\frac{f_c}{F_0}\right)) \end{cases} \tag{7.23}$$

Γ exprime l'atténuation, la fréquence et le déphasage du signal sinusoïdal en sortie du filtre.

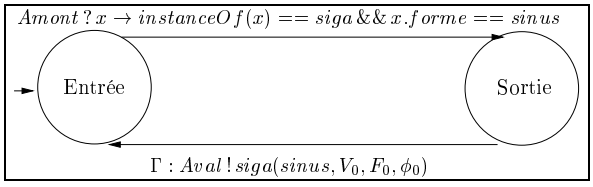

FIG. 7.13 – Modèle fonctionnel d'un bloc filtre passe-haut du premier ordre pour un signal d'entrée sinusoïdal

Dans l'exemple précédent, nous considérons un signal d'entrée sinusoïdal. Ce choix est motivé par le fait que ce stimulus d'entrée est très souvent utilisé pour caractériser le fonctionnement de ce type de filtre. Cependant, il est possible de considérer un autre modèle fonctionnel caractérisant la réponse du filtre à un autre stimulus d'entrée. Ainsi, le modèle fonctionnel est défini pour des formes particulières des signaux d'entrée et dépend de la forme des signaux d'entrée à considérer. Le choix des stimuli dépend des blocs environnants.

Modèle fonctionnel d'un bloc d'entrée et d'un bloc de sortie Le modèle fonctionnel d'un bloc d'entrée est constitué d'un CFSM envoyant un signal d'entrée modélisé par une structure *siga* ou *sigd*. Dans le même esprit, le modèle fonctionnel d'un bloc de sortie est modélisé par un CFSM recevant un signal de sortie modélisé par une structure *siga* ou *sigd*. Comme pour les modèles fonctionnels des blocs fonctionnels, des conditions peuvent êtres exprimées sur le signal d'entrée ou de sortie. Ces conditions peuvent s'exprimer sur l'ensemble ou une partie des caractéristiques du signal. À titre d'exemple, nous considérons une carte électronique très simple constituée uniquement du filtre analogique passe-haut dont le modèle fonctionnel F a été présenté dans le paragraphe précédent (figure 7.13). La figure 7.14 montre le modèle fonctionnel d'un bloc d'entrée pour cette carte. Ce bloc représente une source de tension délivrant le signal *siga*. Physiquement, cette source correspond à un générateur de signaux. Les conditions sur les champs de la structure *siga* correspondent à des conditions sur le signal délivré par la source à l'entrée primaire de la carte. Ainsi, la condition $\Gamma = \gamma_1$ && γ_2 && γ_3 && γ_4 && γ_5 dans la transition émettrice

$$\Gamma = \begin{cases} \gamma_1 = (V_0 > 3) \\ \gamma_2 = (V_0 < 12) \\ \gamma_3 = (F_0 > 1000) \\ \gamma_4 = (F_0 < 10000) \\ \gamma_5 = (\phi_0 == 0) \end{cases} \tag{7.24}$$

signifie que l'amplitude du signal d'entrée sinusoïdal est comprise entre 3 et 12 volts, que la bande passante est comprise entre 1000 et 10000 Hz, et que la phase est nulle. La figure 7.15 montre le modèle fonctionnel d'un bloc de sortie pour la carte. Ce bloc identifie un point de mesure et pourrait correspondre physiquement à un appareil de mesure. Ici, le signal reçu du modèle fonctionnel du filtre n'est pas conditionné. Dans la réalité, cela correspond simplement à l'observation du signal à la sortie primaire de la carte.

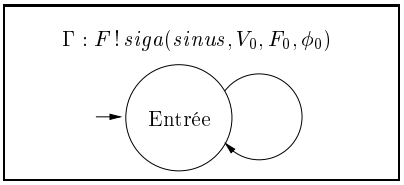

FIG. 7.14 – Modèle fonctionnel d'un bloc d'entrée (source)

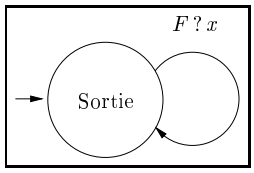

FIG. 7.15 – Modèle fonctionnel d'un bloc de sortie (point de mesure)

Modèle fonctionnel d'une horloge Le modèle fonctionnel d'une horloge est constitué d'un CFSM qui envoie des tops d'horloge calculés par rapport à une référence de temps. La figure 7.16 montre le modèle fonctionnel de l'horloge la plus simple que l'on puisse considérer (CFSM H) et illustre l'utilisation des horloges dans la modélisation des cartes. Le CFSM H envoie un top estampillé à l'instant z ($top(z)$) vers le CFSM B. La référence de temps n'apparaît pas explicitement mais il ne faut pas oublier qu'elle existe. Lorsque $top(z)$ est reçu par le CFSM B et que le signal sinusoïdal x émis par le CFSM A est également reçu par le CFSM B, ce dernier envoie le couple (x, z) vers le CFSM C qui ne fait que recevoir x et z. Il est important de remarquer sur cet exemple que le couple (x, z) est un échantillon au sens de la définition 7.3, i.e. l'association d'un signal et d'une date. Sur cet exemple, la valeur de l'instant z du top n'est pas conditionnée. La valeur de z peut être conditionnée de manière à obtenir un échantillon à un instant précis. Ainsi, si la condition sur l'envoi du top par le CFSM H devient par exemple

$$z == 10 \ : \ B \,!\, top(z)$$

alors la condition $z == 10$ exprime que z doit être égal à 10 unités de temps. Ainsi, l'échantillon obtenu par B sera toujours obtenu pour cet instant. Cette unique datation est restrictive. Nous pouvons obtenir un modèle fonctionnel de l'horloge délivrant par exemple des tops à intervalles réguliers. Il suffit de modifier la condition d'envoi du top :

$$z\%T_H == 0 \ : \ B \,!\, top(z)$$

La condition $z\%T_H == 0$ exprime que l'horloge envoie des tops pour des instants z qui sont des multiples d'une constante T_H, où % représente l'opérateur modulo et T_H la période de l'horloge. Cette condition garantit la périodicité des tops envoyés car z est calé sur une référence de temps. Cette référence de temps est commune à toutes les horloges utilisées dans la modélisation de la carte.

La valeur initiale d'une horloge est égale à zéro par défaut et elle avance sur demande lors du processus de génération de données de test.

Dans notre approche, les modèles fonctionnels des blocs fonctionnels de la carte, les modèles fonctionnels des blocs d'entrée et de sortie, et les modèles fonctionnels des horloges forment un ensemble d'automates à états finis communicants qui modélisent de manière homogène une carte électronique. L'ensemble des comportements de la carte est représenté par l'ensemble des interactions possibles entre ces automates.

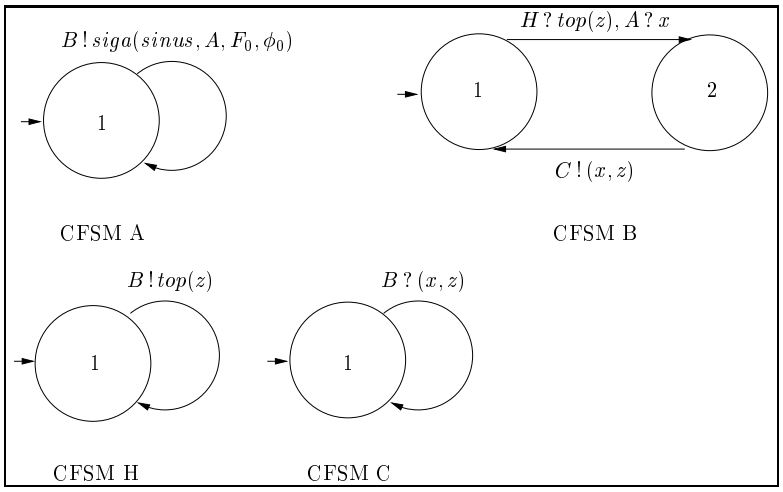

FIG. 7.16 – Utilisation d'une horloge

7.3.2.3 Modèle de test

Le modèle de test d'un bloc fonctionnel est une représentation spécifique aux besoins du test car comme dit en section 7.1, très souvent le modèle fonctionnel n'est pas le plus adéquat pour les besoins du test.

Définition 7.11 *Le modèle de test d'un bloc fonctionnel est un modèle qui décrit la manière dont celui-ci peut être efficacement testé.*

Le modèle de test d'un bloc fonctionnel est constitué d'un ensemble de CFSM qui traitent, comme pour le modèle fonctionnel, des signaux d'entrée et de sortie modélisés par des structures *siga* et/ou *sigd*. Le modèle de test par défaut d'un bloc est identique à son modèle fonctionnel. Cependant, les besoins du test conduisent souvent à extraire certains comportements spécifiques d'un bloc. Ainsi, les conditions exprimées dans les transitions du modèle de test sont généralement plus fortes que celles exprimées par le modèle fonctionnel. Le modèle de test définit un ensemble de comportements particuliers jugé suffisant pour tester correctement le bloc. Il peut être ainsi vu comme une restriction comportementale du modèle fonctionnel auquel des conditions supplémentaires ont été ajoutées.

L'exemple de la figure 7.17 montre le modèle de test que nous proposons pour le filtre analogique passe-haut du premier ordre dont le modèle fonctionnel est montré figure 7.13. Ce modèle de test définit deux données de test correspondant à des stimuli analogiques sinusoïdaux.

La première donnée de test est relative au comportement du filtre lorsque la fréquence du signal sinusoïdal d'entrée est égale à dix fois la fréquence de coupure du

filtre. La condition $\Gamma_1 = \gamma_{11}$ && γ_{12} && γ_{13} && γ_{14} && γ_{15}

$$\Gamma_1 = \begin{cases} \gamma_{11} = (F_0 == x.frq) \\ \gamma_{12} = (V_0 \leq x.ampl + \delta_{11}) \\ \gamma_{13} = (V_0 \geq x.ampl - \delta_{11}) \\ \gamma_{14} = (\phi_0 \leq 0 + \delta_{12}) \\ \gamma_{15} = (\phi_0 \geq 0 - \delta_{12}) \end{cases} \tag{7.25}$$

donne les caractéristiques du signal sinusoïdal sortant dans ce cas. A cette fréquence, le signal d'entrée n'est quasiment pas atténué (γ_{12} et γ_{13}), ni déphasé (γ_{14} et γ_{15}). Cette première donnée de test permet donc de tester le comportement du filtre dans sa bande passante. Les constantes positives δ_{11} et δ_{12} définissent respectivement un intervalle de tolérance pour l'amplitude et la phase du signal sortant. Ces intervalles de tolérance prennent en compte la légère atténuation du signal de sortie du filtre, ainsi que son faible déphasage. Ils prennent également en compte les déviations paramétriques du filtre.

La deuxième donnée de test est quant à elle relative au comportement du filtre lorsque la fréquence du signal d'entrée est égale à la fréquence de coupure du filtre. La condition $\Gamma_2 = \gamma_{21}$ && γ_{22} && γ_{23} && γ_{24} && γ_{25}

$$\Gamma_2 = \begin{cases} \gamma_{21} = (F_0 == x.frq) \\ \gamma_{22} = (V_0 \leq x.ampl\frac{\sqrt{2}}{2} + \delta_{21}) \\ \gamma_{23} = (V_0 \geq x.ampl\frac{\sqrt{2}}{2} - \delta_{21}) \\ \gamma_{24} = (\phi_0 \leq -\frac{\pi}{4} + \delta_{22}) \\ \gamma_{25} = (\phi_0 \geq -\frac{\pi}{4} - \delta_{22}) \end{cases} \tag{7.26}$$

donne les caractéristiques du signal sinusoïdal sortant dans ce cas. A cette fréquence, le signal d'entrée est affaibli de 3 dB (γ_{22} et γ_{23}) et est déphasé de $-\frac{\pi}{4}$ radians (γ_{24} et γ_{25}). Les constantes positives δ_{21} et δ_{22} définissent des intervalles de tolérance qui prennent en compte les déviations paramétriques du filtre. Cette deuxième donnée de test permet donc de tester le comportement du filtre à sa fréquence de coupure.

Nous venons de présenter le modèle de test d'un bloc fonctionnel. Concernant les blocs d'entrée ou de sortie et les horloges, aucun modèle de test ne leur est associé. La raison vient du fait que nous ne cherchons pas à tester le comportement de ces blocs extérieurs à la carte : ils sont réputés bons. Néanmoins, des conditions peuvent êtres ajoutées aux modèles fonctionnels des blocs d'entrées (générateurs de signaux) et de sortie (points de mesure). Cela pourrait être vu comme un modèle de test, cependant l'idée n'est pas de tester les générateurs ou les points de mesure mais d'orienter le test de la carte vers des données de test particulières. Cette orientation correspond à une *tactique de test*, notion qui est détaillée en section 7.4.2.

7.3.3 Exemple d'application

Dans cette section, nous appliquons notre méthodologie de modélisation à un cas d'étude simple (carte TCB - Test Case Board) [GNN07b]. Nous donnons tout d'abord la

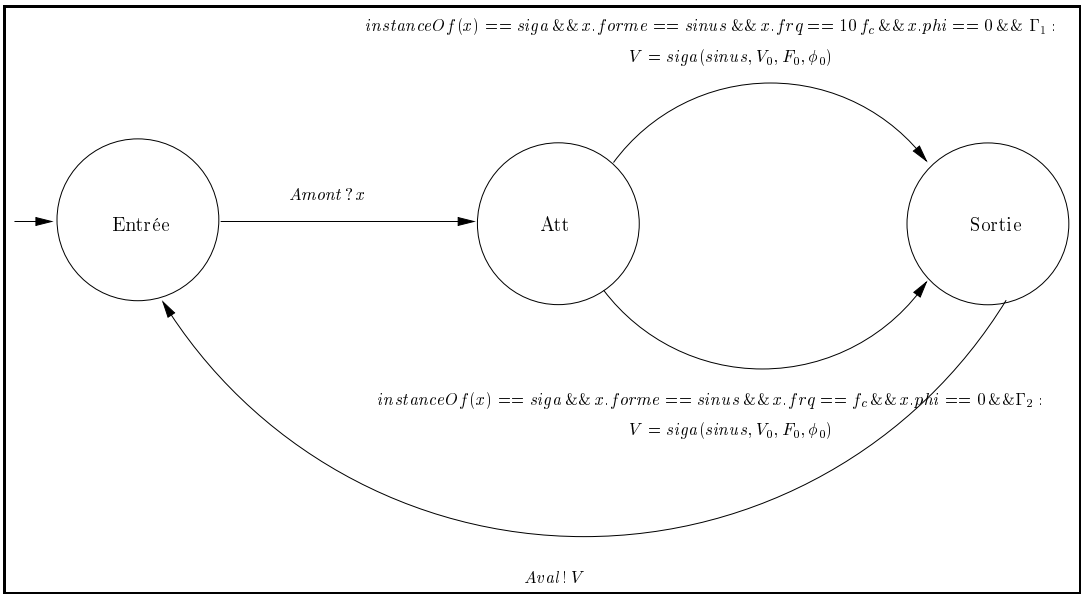

FIG. 7.17 – Modèle de test d'un bloc filtre passe-haut du premier ordre pour un signal d'entrée sinusoïdal

description fonctionnelle de la carte TCB. Nous présentons ensuite sa modélisation au niveau carte, puis nous décrivons la modélisation au niveau bloc en donnant les différents modèles fonctionnels. Les modèles de test des blocs sont présentés en section 7.4.4. Nous utilisons la carte TCB pour illustrer les différents aspects de notre méthodologie dans le reste de ce document.

7.3.3.1 Description de la carte

La carte TCB est une carte mixte qui possède une entrée analogique. La fonction principale de la carte est de vérifier périodiquement la tension instantanée du signal d'entrée en la comparant à un seuil de tension déterminé. Le résultat de la comparaison est une valeur logique datée écrite dans la mémoire RAM de la carte. Un filtre analogique passe-haut, situé en amont de la carte, élimine les tensions continues appliquées à l'entrée de la carte.

7.3.3.2 Modélisation au niveau carte

La figure 7.18 montre la modélisation de la carte TCB au niveau carte. Sur cette figure, le périmètre de la carte est délimité par le rectangle en pointillés et les blocs se trouvant dans ce périmètre sont des blocs fonctionnels. La carte est composée de deux blocs analogiques F et C, d'un bloc mixte Adc et d'un bloc numérique Mem. Le bloc F représente un filtre analogique passe-haut du premier ordre, C un comparateur à un seuil, Adc est un convertisseur analogique-numérique et Mem, une mémoire RAM. Les blocs S et MP sont respectivement des blocs d'entrée et de sortie et représentent

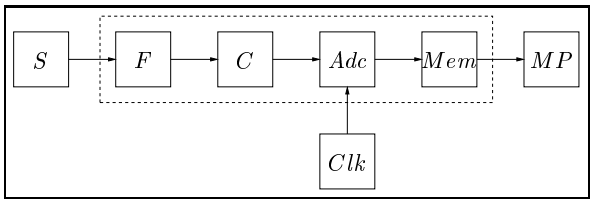

FIG. 7.18 – Modélisation de la carte TCB au niveau carte

respectivement une source de tension analogique et un point de mesure numérique. Le bloc *Clk* représente une horloge.

7.3.3.3 Modélisation au niveau bloc

Nous avons défini en section 7.3.2.2 la notion de modèle fonctionnel d'un bloc. Nous décrivons ci-après les modèles fonctionnels de chacun des blocs de la carte TCB. Par convention, le nom du modèle fonctionnel est identique au nom du bloc auquel il est associé.

Modèle fonctionnel de la source La figure 7.19 montre le modèle fonctionnel de la source S. Le CFSM S envoie le signal x vers le CFSM F.

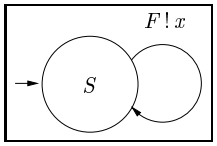

FIG. 7.19 – Modèle fonctionnel de la source S

Modèle fonctionnel du point de mesure La figure 7.20 montre le modèle fonctionnel du point de mesure MP. Le CFSM MP attend le signal discret x émis par le CFSM Mem. Le point de mesure MP permet, pour les besoins du test, de « voir » le contenu de la mémoire Mem.

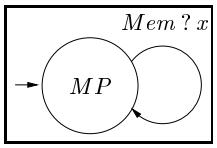

FIG. 7.20 – Modèle fonctionnel du point de mesure MP

Modèle fonctionnel de l'horloge La figure 7.21 montre le modèle fonctionnel de l'horloge. Le CFSM *Clk* envoie des tops périodiques aux instants y vers le CFSM *Adc*. La période entre les tops vaut T_e.

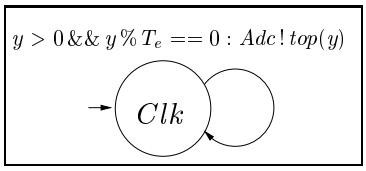

FIG. 7.21 – Modèle fonctionnel de l'horloge Clk

Modèle fonctionnel du filtre analogique Le modèle fonctionnel du filtre analogique passe-haut du premier ordre est représenté figure 7.22. Le CFSM F décrit le comportement du filtre lorsqu'un signal analogique sinusoïdal lui est appliqué en entrée. F reçoit ainsi un signal analogique sinusoïdal émis par le CFSM S et envoie un signal analogique sinusoïdal vers le CFSM C. Ce modèle fonctionnel a déjà été décrit en section 7.3.2.2. Nous rappelons juste ici que la condition $\Gamma = \gamma_1 \;\&\&\; \gamma_2 \;\&\&\; \gamma_3$

$$\Gamma = \begin{cases} \gamma_1 = (V_0 == \frac{x.ampl}{\sqrt{1+\frac{f_c^2}{F_0^2}}}) \\ \gamma_2 = (F_0 == x.frq) \\ \gamma_3 = (\phi_0 == x.phi - \arctan\left(\frac{f_c}{F_0}\right)) \end{cases} \tag{7.27}$$

exprime les caractéristiques du signal de sortie en fonction de celles du signal d'entrée.

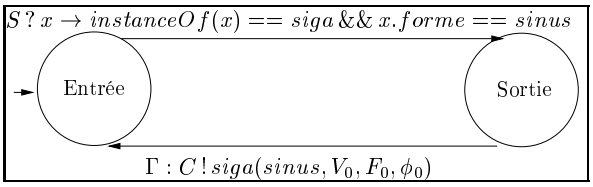

FIG. 7.22 – Modèle fonctionnel du filtre analogique F

Modèle fonctionnel du comparateur La figure 7.23 montre le modèle fonctionnel du comparateur C. Le CFSM C décrit le comportement du comparateur lorsqu'un signal analogique sinusoïdal est appliqué à son entrée. Le CFSM C attend le signal x émis par F. Deux comportements distincts sont décrits lorsque le signal x reçu est un signal analogique sinusoïdal. Si l'amplitude maximale du signal est strictement inférieure à la valeur du seuil S du comparateur, alors C envoie le signal analogique DC nul vers le CFSM *Adc*. Si l'amplitude maximale du signal dépasse la valeur du seuil S du comparateur, alors C envoie le signal analogique rectangulaire $siga(rect, \Delta T_1, T_0, t_0)$ vers le CFSM *Adc* en vérifiant la condition $\Gamma = \gamma_1 \;\&\&\; \gamma_2 \;\&\&\; \gamma_3$:

$$\Gamma = \begin{cases} \gamma_1 = (T_0 == \frac{1}{x.frq}) \\ \gamma_2 = (\Delta T_1 == \frac{1}{2\,x.frq} - \frac{1}{\pi\,x.frq} \arcsin\left(\frac{S}{x.ampl}\right)) \\ \gamma_3 = (t_0 == \frac{1}{2\pi\,x.frq} \arcsin\left(\frac{S}{x.ampl}\right) + \frac{x.phi}{2\pi\,x.frq}) \end{cases} \tag{7.28}$$

Ces conditions expriment les caractéristiques du signal de sortie en fonction des caractéristiques du signal d'entrée dans le cas du dépassement du seuil.

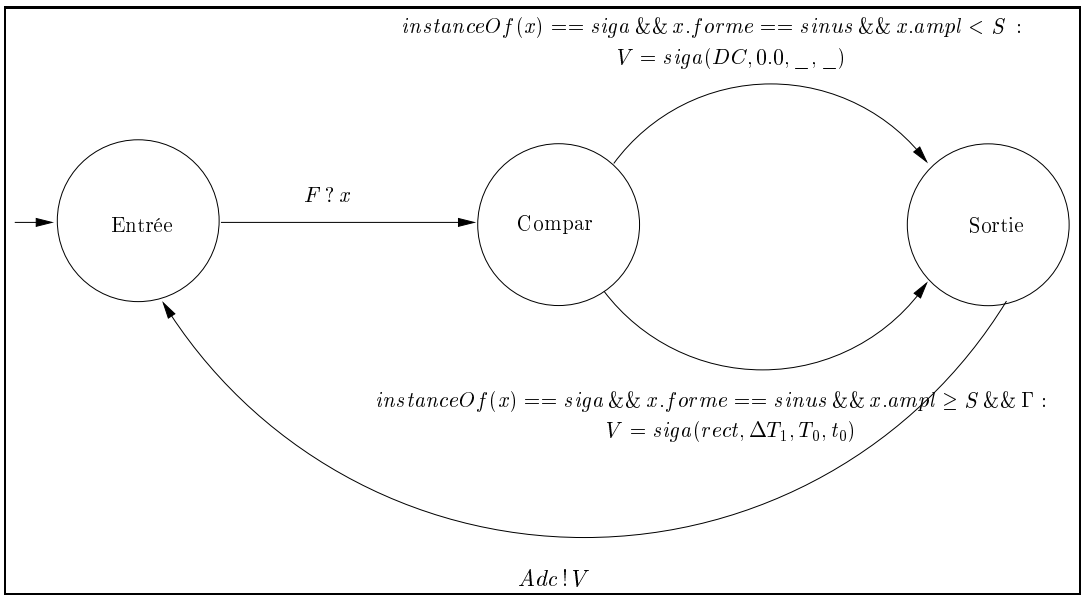

FIG. 7.23 – Modèle fonctionnel du comparateur C

Modèle fonctionnel du convertisseur analogique-numérique La figure 7.24 montre le modèle fonctionnel du convertisseur analogique-numérique *Adc*. Le CFSM *Adc* attend la réception d'un top d'horloge daté à l'instant z envoyé par le CFSM *Clk*. Lorsque ce top est reçu, *Adc* attend la réception du signal envoyé par le CFSM C. Puis, *Adc* envoie le signal discret $sigd(x, z)$ vers le CFSM Mem si le signal x est un signal analogique.

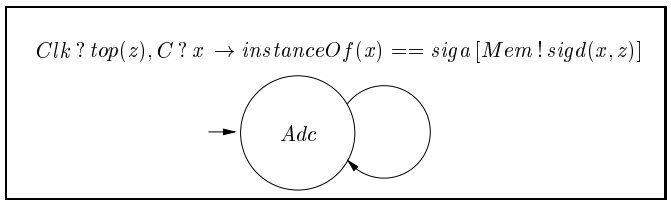

FIG. 7.24 – Modèle fonctionnel du convertisseur analogique-numérique ADC

Modèle fonctionnel de la mémoire Le modèle fonctionnel de la mémoire Mem est représenté figure 7.25. Le CFSM *Mem* reçoit le signal discret x envoyé par le CFSM *Adc* et le renvoie au CFSM MP (pour observabilité).

Nous venons de présenter les modèles fonctionnels des blocs de la carte TCB. Avec cet exemple, il est intéressant de noter que les CFSM permettent de modéliser de manière

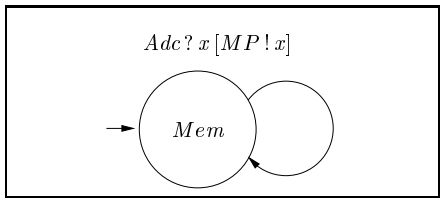

FIG. 7.25 – Modèle fonctionnel de la mémoire Mem

uniforme le comportement de blocs de différentes natures (analogique, numérique et mixte). L'ensemble des CFSM précédemment décrits intègre les interactions entre les blocs fonctionnels de la carte et constitue le modèle fonctionnel de la carte TCB. Nous reviendrons sur cet exemple en section 7.4.4 pour illustrer les modèles de test.

7.4 Test de la carte

Nous avons exposé en section 7.1 les idées directrices de notre méthodologie de test ainsi que notre approche du processus de génération des données de test pour les cartes mixtes. Nous avons présenté en section 7.3 la manière dont nous modélisons une carte électronique. Nous avons introduit deux niveaux de modélisation : le niveau carte et le niveau bloc. Le modèle fonctionnel et le modèle de test d'un bloc ont été décrits.

Dans cette section, nous détaillons l'approche choisie pour le test de la carte en nous basant sur cette modélisation et l'utilisation de stratégies et tactiques de test. Les niveaux de modélisation carte et bloc présentés en section 7.3 permettent également, du point de vue du test, d'intégrer l'expertise des ingénieurs de test en définissant des *stratégies de test* et des *tactiques de test*. Nous commençons ainsi par présenter la notion de stratégie de test (section 7.4.1), puis celle de tactique de test (section 7.4.2). Nous décrivons ensuite le principe et le fonctionnement de la génération des données de test d'une carte (section 7.4.3). Nous présentons enfin les données de test obtenues pour la carte TCB dont la modélisation fonctionnelle a été décrite en section 7.3.3.

7.4.1 Stratégies de test

Une stratégie de test met en oeuvre des critères de test permettant la couverture fonctionnelle de la carte. Nous avons vu en section 7.3 qu'une carte électronique mixte est modélisée de manière homogène sous la forme d'un ensemble d'automates à états finis communicants. Les stratégies de test que nous proposons sont exprimées dans ce même formalisme. Ainsi, les stratégies de test que nous proposons implantent des critères comme le test des états, des transitions, et des chemins. Nous distinguons les *stratégies de test globales* et les *stratégies de test locales*. Une stratégie de test globale vise à évaluer le comportement global de la carte. Une stratégie de test locale cible un comportement particulier de la carte.

Nous introduisons ci-après certaines propriétés relatives au test basé sur les automates à états finis qui nous seront utiles par la suite.

Propriété 7.1 *La couverture des transitions d'un CFSM est un critère qui s'applique aux jeux de test. Un jeu de test satisfait ce critère lorsqu'il permet d'activer au moins une fois chaque transition du CFSM.*

Propriété 7.2 *Un jeu de test teste un état s'il permet d'activer l'ensemble des combinaisons possibles de transitions entrantes et sortantes de cet état.*

Propriété 7.3 *Un jeu de test vérifie le critère de couverture des états s'il permet de tester au moins une fois chacun des états du CFSM.*

Propriété 7.4 *Un jeu de test teste une transition s'il permet d'activer cette transition.*

Propriété 7.5 *Un jeu de test teste un chemin reliant deux états dans l'ensemble des CFSM s'il permet d'activer en séquence toutes les transitions de ce chemin.*

Nous présentons maintenant les différentes stratégies de test de notre approche en commençant par les stratégies de test globales, puis les stratégies de test locales.

7.4.1.1 Stratégies de test globales

Dans notre méthodologie, **le test de la carte consiste par défaut à tester unitairement**, mais **immergé dans son contexte d'exécution simulé, chaque bloc fonctionnel** de la carte en utilisant son modèle de test associé. Ce choix est intéressant car les spécifications unitaires du bloc sont testées, et en plus à travers son environnement. Ainsi, le test d'intégration est partiellement réalisé. Les données de test pour un bloc sont générées **par défaut en effectuant la couverture des transitions de son modèle de test**. Nous avons vu en section 7.3.2.3 que le modèle de test décrit un ensemble particulier de comportements dont le test est jugé suffisant pour valider un bloc. Ces comportements particuliers sont décrits par des conditions sur les transitions des CFSM décrivant le modèle de test. La couverture des transitions du modèle de test d'un bloc permet de générer les données de test de ce bloc en vérifiant l'ensemble des conditions exprimées dans les transitions.

En pratique, les données de test obtenues pour un bloc ne sont pas directement utilisables. En effet, fréquemment, les blocs testés correspondent à des composants enfouis dans la carte qui ne possèdent pas de point d'accès direct. Les données de test pour un bloc doivent donc être calculées afin de pouvoir être appliquées aux entrées primaires de la carte (contrôlabilité du test), et les réponses du bloc à ces stimuli de test doivent pouvoir être observées aux sorties primaires de la carte (observabilité du test). Il s'en suit qu'il est nécessaire, afin d'assurer la contrôlabilité et l'observabilité du test du bloc, de propager les données de test du bloc en amont vers les entrées primaires et en aval vers les sorties primaires. Cette propagation est réalisée en utilisant les modèles fonctionnels des blocs traversés. Durant la propagation, les données de test du bloc sous test doivent satisfaire les conditions exprimées dans les modèles fonctionnels traversés. La vérification de ces conditions permet de calculer les données de test finales du bloc applicables à la carte. Dans l'exemple de la figure 7.26, le bloc b_3 (enfoui) est testé. Les données de test de b_3 sont obtenues à partir de son modèle de test représenté par un rectangle blanc. Les flèches en pointillé montrent le sens des propagations vers l'amont

des données de test du bloc. Les propagations vers l'aval suivent les flèches noires en trait continu qui symbolisent les communications entre les blocs. Les rectangles hachurés représentent les modèles fonctionnels des blocs fonctionnels de la carte traversés durant la propagation. Les carrés hachurés représentent les modèles fonctionnels des blocs d'entrée, des blocs de sortie et du bloc horloge atteints à l'issue de la propagation. Il est à noter que des propagations amont peuvent être induites par des propagations aval. De même, des propagations amont peuvent engendrer des propagations aval.

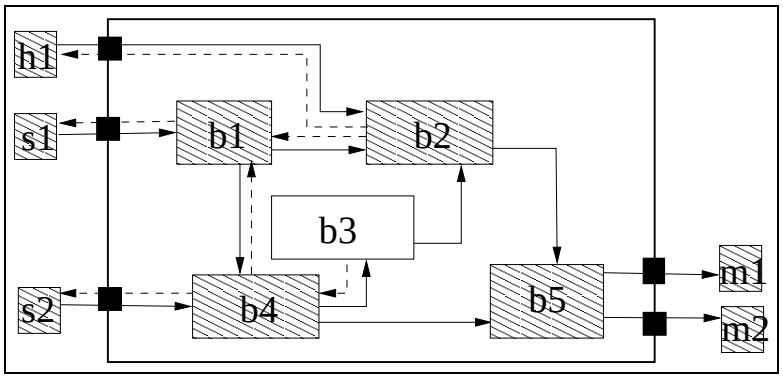

FIG. 7.26 – Propagation des données de test d'un bloc à travers les modèles fonctionnels.

Le choix des modèles fonctionnels comme support de la propagation est motivé par le souci de maximiser les chances de réussir la propagation. L'utilisation des modèles de test des blocs fonctionnels traversés induirait des conditions plus fortes que celles des modèles fonctionnels et une réduction des domaines de définition. En effet, il faut garder à l'esprit que le modèle de test exprime une restriction comportementale (cf. section 7.3.2.3). Dans le cas où la propagation relative à un bloc échoue, les données de test générées pour ce bloc sont incomplètes ou inexistantes. Dans le rapport de test final, on indique alors pour ce bloc :

- son nom,
- les transitions couvertes de son modèle de test,
- les transitions non couvertes de son modèle de test,
- les contraintes non satisfaites dans les modèles fonctionnels supports de la propagation.

L'analyse pour un bloc des transitions non couvertes de son modèle de test et des contraintes non satisfaites dans les modèles fonctionnels supports de la propagation permet de déterminer si :

- le modèle de test du bloc n'est pas adapté pour être utilisé sur la carte, ce qui ne constitue pas une erreur,
- la modélisation de la carte est incorrecte (problème de spécification), ce qui constitue une erreur.

Lorsque tous les blocs de la carte sont testés, nous avons couvert toutes les transitions de tous les modèles de test de la carte. **Le caractère global de la stratégie de test est lié à l'activation exhaustive de l'ensemble de ces transitions**. Nous

proposons également une deuxième stratégie de test globale qui consiste à générer les données de test d'un bloc en effectuant non pas la couverture des transitions de son modèle de test, mais en effectuant la couverture de ses états. Le mécanisme de propagation décrit ci-dessus reste valable. **Le caractère global de cette stratégie de test est lié au test de tous les états de tous les modèles de test de la carte**. La couverture de tous les états est plus forte que la couverture de toutes les transitions (au sens des propriétés 7.2 et 7.4).

Nous venons de présenter les stratégies de test globales comme des stratégies qui réalisent des tests de couverture. Une manière efficace de réaliser des tests de couverture est de chercher des chemins qui permettent chacun de visiter le plus grand nombre d'états possible (couverture des états) ou de franchir le plus grand nombre possible de transitions (couverture des transitions) dans le but d'obtenir un nombre minimal de chemins. La vérification des conditions sur chaque chemin permet alors de calculer les données de test finales. On obtient ainsi un nombre minimal de données de test qui sont elles-mêmes de longueur maximales.

7.4.1.2 Stratégies de test locales

Contrairement aux stratégies de test globales, les stratégies de test locales orientent la génération des données de test dans le but de tester un ou des comportements particuliers de la carte. Exprimées dans le formalisme des automates à états finis communicants, ces stratégies vont consister à générer les données de test qui permettent de tester (au sens des propriétés 7.2, 7.4, 7.5) un état donné, ou une transition donnée ou un chemin donné. Dans un souci d'efficacité, et contrairement au choix fait pour les stratégies globales, nous cherchons cette fois à parcourir des chemins de longueur minimale et ainsi donc de générer le plus souvent des données de test de longueurs minimales. Nous précisons maintenant chacune des stratégies de test locales que nous venons d'introduire.

Test d'une transition Les données de test générées pour tester une transition sont obtenues à partir des conditions qu'elle exprime. Comme pour les stratégies de test globales, les données de test obtenues ne sont pas directement utilisables (à moins que la carte possède un point d'accès direct pour le test). Ainsi, il est également nécessaire, afin d'assurer la contrôlabilité et l'observabilité du test, de propager les données de test de la transition vers les entrées et sorties primaires de la carte en utilisant les modèles fonctionnels des blocs traversés.

Test d'un état Les données de test générées pour tester un état sont obtenues à partir des conditions exprimées par chaque combinaison transition entrante/transition sortante de l'état. Puis, comme pour le test d'une transition présenté ci-dessus, les propagations nécessaires sont effectuées.

Test d'un chemin Les données de test générées pour tester un chemin sont obtenues à partir des conditions sur les transitions formant le chemin et en faisant les propagations

nécessaires.

7.4.2 Tactiques de test

Les tactiques de test permettent d'orienter finement la génération des données de test. Une tactique de test diffère d'une stratégie de test. Comme nous l'avons expliqué en section 7.4.1, une stratégie de test (globale ou locale) est exprimée en terme de parcours de chemin dans un ensemble de CFSM. **Une tactique de test s'exprime sous la forme de conditions supplémentaires associées à une ou un ensemble de transitions**. Le modèle de test d'un bloc de la carte introduit en section 7.3.2.3 peut être vu comme implantant une tactique de test locale à ce bloc qui ne prend pas en compte son environnement. Une tactique de test peut permettre de tester plus finement un bloc en tenant compte de son environnement. Elle peut permettre également de tester un aspect précis du comportement d'une carte. Les affinements obtenus par les tactiques de test peuvent être variés. Il est difficile d'énumérer les différentes tactiques de test. Trois catégories de tactiques de test ont toutefois retenu plus particulièrement notre attention : les tactiques de test visant à améliorer l'observabilité des données de test générées, celles ayant pour but de rendre la contrôlabilité des données de test générées meilleure, et les tactiques de test ayant pour but de permettre l'obtention de données de test pour un comportement particulier qui ne peut être testé à partir des modèles fonctionnels et des modèles de test « standards ». Nous donnons ci-après un exemple pour chacune des ces trois catégories de tactiques de test.

• Exemple d'amélioration de l'observabilité de données de test générées : les conditions Γ_1 et Γ_2 données respectivement par les expressions (7.25) et (7.26) pour le modèle de test du filtre analogique passe-haut représenté sur la figure 7.17 peuvent être modifiées de la manière suivante :

$$\Gamma'_1 = \begin{cases} \gamma_{11} = (F_0 == x.frq) \\ \gamma_{12} = (V_0 \leq x.ampl + \delta_{11} + \mathbf{\Delta_{obs11}}) \\ \gamma_{13} = (V_0 \geq x.ampl - \delta_{11} - \mathbf{\Delta_{obs11}}) \\ \gamma_{14} = (\phi_0 \leq 0 + \delta_{12} + \mathbf{\Delta_{obs12}}) \\ \gamma_{15} = (\phi_0 \geq 0 - \delta_{12} - \mathbf{\Delta_{obs12}}) \end{cases} \tag{7.29}$$

$$\Gamma'_2 = \begin{cases} \gamma_{21} = (F_0 == x.frq) \\ \gamma_{22} = (V_0 \leq x.ampl\frac{\sqrt{2}}{2} + \delta_{21} + \mathbf{\Delta_{obs21}}) \\ \gamma_{23} = (V_0 \geq x.ampl\frac{\sqrt{2}}{2} - \delta_{21} - \mathbf{\Delta_{obs21}}) \\ \gamma_{24} = (\phi_0 \leq -\frac{\pi}{4} + \delta_{22} + \mathbf{\Delta_{obs22}}) \\ \gamma_{25} = (\phi_0 \geq -\frac{\pi}{4} - \delta_{22} - \mathbf{\Delta_{obs22}}) \end{cases} \tag{7.30}$$

L'ajout de constantes supplémentaires Δ_{obs11}, Δ_{obs12}, Δ_{obs21} et Δ_{obs22} dans les conditions autorise l'expression de contraintes sur l'environnement du bloc ainsi que sur le chemin à parcourir jusqu'à la sortie. Par exemple, la constante Δ_{obs11} ajoutée dans les conditions γ_{12} et γ_{13} peut permettre de rendre observable la perturbation de l'amplitude V_0 du signal de sortie du filtre induite par l'interaction avec un autre composant de la carte.

• Exemple d'amélioration de la contrôlabilité de données de test générées : la condition Γ donnée par l'expression (7.24) pour le modèle fonctionnel de la source représenté sur la figure 7.14 peut être modifiée de la manière suivante :

$$\Gamma' = \begin{cases} \gamma_1 = (V_0 > 3 - \mathbf{\Delta_{ctrl11}}) \\ \gamma_2 = (V_0 < 12 + \mathbf{\Delta_{ctrl11}}) \\ \gamma_3 = (F_0 > 1000 - \mathbf{\Delta_{ctrl12}}) \\ \gamma_4 = (F_0 < 10000 + \mathbf{\Delta_{ctrl12}}) \\ \gamma_5 = (\phi_0 \leq 0 + \mathbf{\Delta_{ctrl13}}) \\ \gamma_6 = (\phi_0 \geq 0 - \mathbf{\Delta_{ctrl13}}) \end{cases} \tag{7.31}$$

L'ajout de constantes supplémentaires Δ_{ctrl11}, Δ_{ctrl12} et Δ_{ctrl13} dans les conditions autorise l'expression de contraintes sur le chemin de l'entrée au bloc sous test. Par exemple, les constantes Δ_{ctrl11}, Δ_{ctrl12} et Δ_{ctrl13} ajoutées dans la condition Γ' peuvent permettre de prendre en compte une perturbation, due à la carte, des caractéristiques du signal délivré par la source.

• Exemple de test pour un comportement particulier : l'obtention d'une donnée de test permettant de vérifier la synchronisation entre les parties analogiques et numériques de la carte TCB est rendue possible en ajoutant des conditions sur le CFSM modélisant le point de mesure (cf. section 7.4.4).

7.4.3 Génération des données de test : principe de fonctionnement

Dans cette section, nous donnons le principe de fonctionnement des différents algorithmes de génération de données de test (ATPG) associés aux différentes stratégies de test décrites en section 7.4.1. Les algorithmes sont génériques et s'expriment en terme de parcours d'automates. Ils sont présentés de manière plus formelle et plus détaillée en section 8.3.

7.4.3.1 Algorithme ATPG associé aux stratégies de test globales

L'algorithme 1 génère un jeu de données de test pour une carte électronique en utilisant une stratégie de test globale (couverture des transitions ou couverture des états des modèles de test). Par défaut, les blocs les plus internes à la carte sont testés les premiers, ceux qui se trouvent aux interfaces en dernier. Ce choix permet d'optimiser la génération des données de test. On peut cependant introduire plus de souplesse à ce niveau en proposant à l'utilisateur d'associer un rang de test aux blocs de la carte.

7.4.3.2 Algorithmes ATPG associés aux stratégies de test locales

Test d'une transition ou d'un état L'algorithme 2 génère les données de test permettant de tester une transition ou un état d'un modèle de test d'un bloc.

Test d'un chemin L'algorithme 3 génère les données de test permettant de tester un chemin. Ce chemin peut être défini sur les modèles fonctionnels et/ou les modèles de test de la carte.

Algorithme 1 Test global de la carte

Entrées: Soit C une carte électronique décomposée en un ensemble de blocs et SG une stratégie de test globale

Sorties: un jeu de test pour C

Début

tantque il reste des blocs à traiter pour C **faire**

choisir un bloc B à traiter

générer les séquences de test aux entrées/sorties de la carte en utilisant le modèle de test du bloc B et les modèles fonctionnels des autres blocs et la stratégie SG

si échec partiel ou total **alors**

consigner les échecs de la génération dans le rapport de test

fin si

incorporer les séquences de test du bloc B au jeu de test de C

fin tantque

Fin

Algorithme 2 Test d'une transition (respectivement un état)

Entrées: Soit B un bloc d'une carte électronique C, soit T une transition (respectivement E un état) du modèle de test du bloc B

Sorties: Les données de test permettant de tester T (respectivement E)

Début

générer une séquence de test pour T (respectivement des séquences de test pour E) en :

- utilisant le modèle de test de B
- vérifiant les conditions exprimées par T dans le modèle de test de B (respectivement les conditions exprimées par l'ensemble des couples transition entrante/transition sortante autour de E dans le modèle de test de B)
- utilisant les modèles fonctionnels des autres blocs

si échec partiel ou total **alors**

mentionner dans le rapport de test que T ne peut être testé (respectivement que E ne peut être testé ou testé complètement)

fin si

Fin

Algorithme 3 Test d'un chemin

Entrées: Soit C une carte électronique décomposée en un ensemble de blocs EB et soit M un chemin (séquence de transitions) défini sur un sous-ensemble de EB

Sorties: Les données de test couvrant M

Début

générer une séquence de test des entrées aux sorties de la carte, couvrant M, en vérifiant la conjonction de conditions exprimées par l'ensemble de transitions impliquées par M

si échec de la génération **alors**

mentionner dans le rapport de test que M ne peut être testé en précisant les échecs rencontrés

fin si

Fin

7.4.4 Exemple d'application

Nous avons présenté en section 7.3.3 la modélisation fonctionnelle de la carte TCB. Ayant introduit la notion de modèle de test en section 7.3.2.3, les stratégies de test en section 7.4.1 et les tactiques de test en section 7.4.2, nous illustrons maintenant notre processus de test sur la carte TCB. À cette fin, nous présentons d'abord les modèles de test des blocs de la carte TCB (section 7.4.4.1). Puis nous décrivons des tactiques de test utilisées dans le modèle de test du filtre (section 7.4.4.1) et sur le point de mesure de la carte TCB (section 7.4.4.2). Nous décrivons ensuite l'application de la stratégie de test globale de couverture des transitions des modèles de test des blocs de la carte TCB (section 7.4.4.3). Nous avons choisi cette stratégie ici, car c'est celle qui est proposée par défaut dans notre méthodologie de test.

7.4.4.1 Modèles de test

Pour la carte TCB, nous dérivons des modèles de test uniquement pour les blocs fonctionnels internes, i.e. le filtre, le comparateur, le convertisseur analogique-numérique et la mémoire. Nous obtenons ainsi quatre modèles de test. Nous rappelons que par défaut, le modèle de test est identique au modèle fonctionnel.

Modèles de test du convertisseur analogique-numérique et de la mémoire Dans un premier temps, par souci de simplification, les modèles de test utilisés pour le convertisseur analogique-numérique Adc et la mémoire Mem sont respectivement les modèles de test par défaut (modèles fonctionnels).

Modèle de test du comparateur La figure 7.27 montre le modèle de test que nous proposons pour le comparateur C. Notre modèle de test décrit deux comportements

dont le test est jugé suffisant pour valider le bloc associé. Ainsi, nous vérifions la réponse du comparateur pour des signaux dont les amplitudes maximales sont respectivement légèrement inférieure $(S-\delta)$ et supérieure $(S+\delta)$ au seuil S du comparateur. La valeur de la constante δ dépend des caractéristiques physiques du comparateur. La condition Γ reste identique à celle (7.28) du modèle fonctionnel.

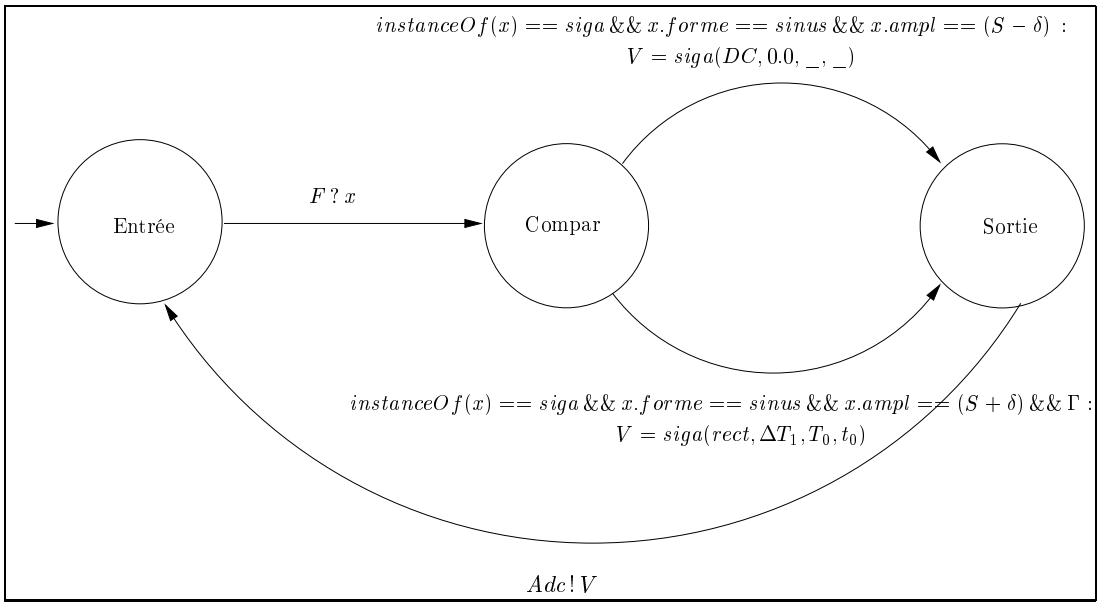

FIG. 7.27 – Modèle de test du comparateur C

Modèle de test du filtre La figure 7.28 montre le modèle de test que nous proposons pour le filtre. Ce modèle de test est une extension du modèle de test décrit en section 7.3.2.3. Il contient une tactique de test qui prend en compte l'environnement du filtre en rendant observables ses réponses à la sortie primaire de la carte (en garantissant que l'amplitude instantanée des réponses sinusoïdales du filtre est supérieure au seuil du comparateur). Sans cette tactique de test, la valeur de la sortie primaire pourrait être toujours égale à zéro, rendant ainsi impossible la distinction entre les deux réponses du filtre (signal affaibli et signal non affaibli). Les ensembles de conditions Γ'_1 et Γ'_2 sont définis respectivement par $\Gamma'_1 = \gamma_{11}$ && γ_{12} && γ_{13} && γ_{14} && γ_{15} && γ_{16} et $\Gamma'_2 = \gamma_{21}$ && γ_{22} && γ_{23} && γ_{24} && γ_{25} && γ_{26} :

$$\Gamma'_1 = \begin{cases} \gamma_{11} = (F_0 == x.frq) \\ \gamma_{12} = (V_0 \leq x.ampl + \delta_{11}) \\ \gamma_{13} = (V_0 \geq x.ampl - \delta_{11}) \\ \gamma_{14} = (\phi_0 \leq 0 + \delta_{12}) \\ \gamma_{15} = (\phi_0 \geq 0 - \delta_{12}) \\ \gamma_{16} = (V_0 - \delta_{11} > S) \end{cases} \tag{7.32}$$

$$\Gamma'_2 = \begin{cases} \gamma_{21} = (F_0 == x.frq) \\ \gamma_{22} = (V_0 \leq x.ampl\frac{\sqrt{2}}{2} + \delta_{21}) \\ \gamma_{23} = (V_0 \geq x.ampl\frac{\sqrt{2}}{2} - \delta_{21}) \\ \gamma_{24} = (\phi_0 \leq -\frac{\pi}{4} + \delta_{22}) \\ \gamma_{25} = (\phi_0 \geq -\frac{\pi}{4} - \delta_{22}) \\ \gamma_{26} = (V_0 - \delta_{21} > S) \end{cases} \tag{7.33}$$

Γ'_1 et Γ'_2 expriment respectivement les caractéristiques des deux réponses du filtre en fonction des signaux d'entrée. Les conditions γ_{16} et γ_{26} correspondent à la tactique de test susmentionnée.

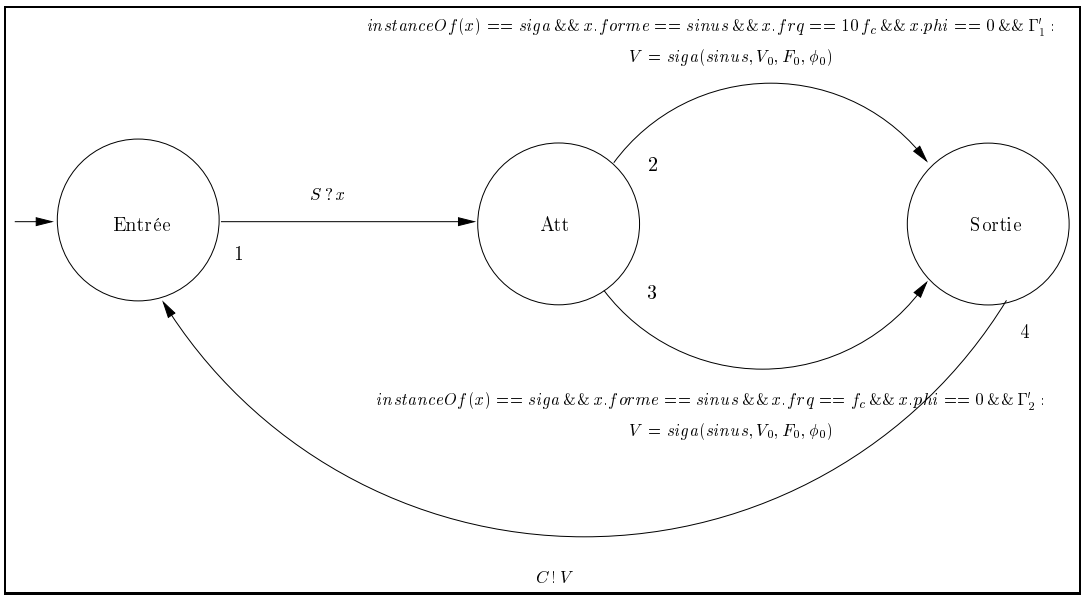

FIG. 7.28 – Modèle de test du filtre F

7.4.4.2 Tactique de test pour l'intégration à l'échelle de la carte

La figure 7.29 montre une tactique de test appliquée au point de mesure (condition supplémentaire dans le modèle fonctionnel du point de mesure). Elle permet de tester la synchronisation entre la partie analogique et la partie numérique de la carte TCB. Lorsque la partie analogique et la partie numérique de la carte sont synchronisées, les valeurs datées écrites dans la mémoire sont toutes égales à un et chaque date correspond à un top de l'horloge *Clk* (figure 7.30). Dans notre tactique de test, nous avons choisi d'écrire des « un » en mémoire à chaque top d'horloge. Nous aurions pu également utiliser une tactique de test différente et n'écrire que des zéro en mémoire.

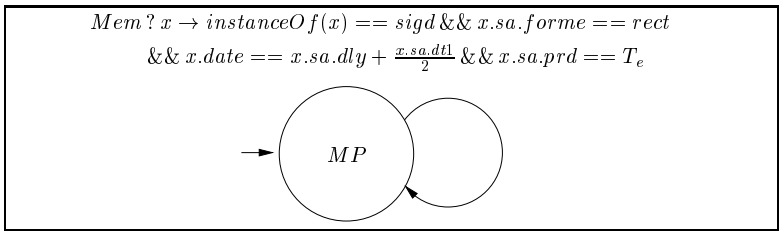

FIG. 7.29 – Tactique de test basée sur la synchronisation analogique-numérique

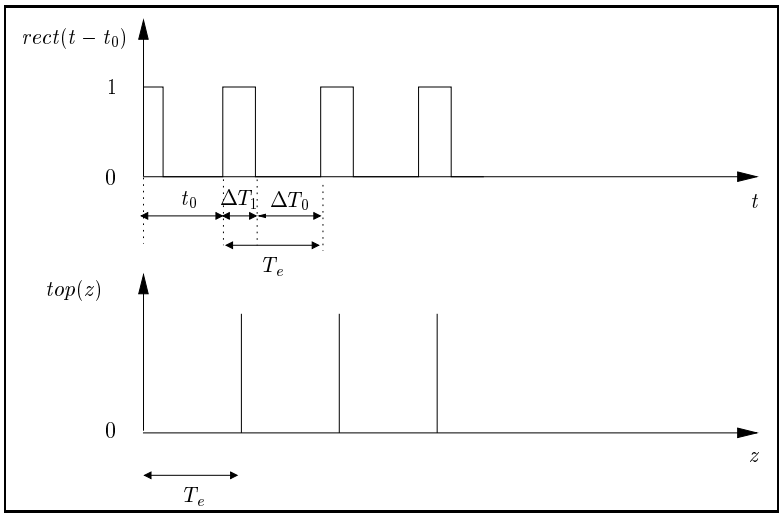

FIG. 7.30 – Synchronisation analogique-numérique de la carte TCB

7.4.4.3 Processus de test

Pour illustrer notre approche du test sur la carte TCB, nous choisissons ici d'appliquer principalement une stratégie de test globale (algorithme 1) de couverture des transitions des modèles de test. En complément, nous appliquons aussi une stratégie de test locale (algorithme 2) pour le test de la transition du point de mesure (synchronisation analogique-numérique). Le jeu de données de test obtenu comporte 5 données de test et est [GNN07b, NGN08] :

$$TDS = \{TDS_{filtre}, TDS_{comparateur}, TDS_{num}\} \tag{7.34}$$

où

$$TDS_{filtre} = \{TD1_{filtre}, TD2_{filtre}\} \tag{7.35}$$

$$TDS_{comparateur} = \{TD1_{comparateur}, TD2_{comparateur}\} \tag{7.36}$$

$$TDS_{num} = \{TD1_{num}\} \tag{7.37}$$

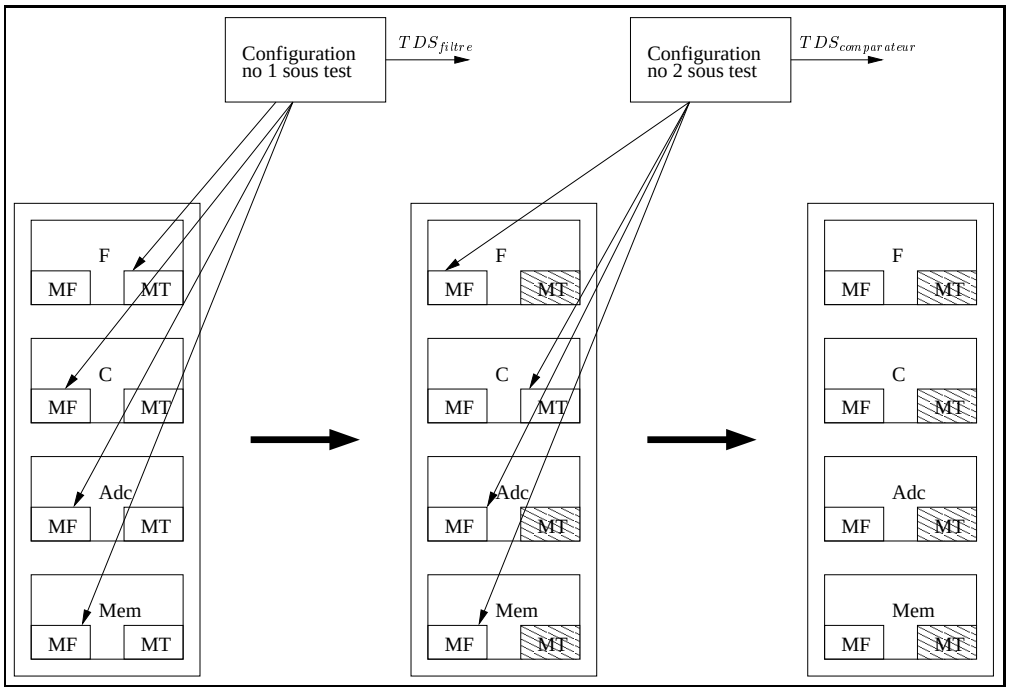

FIG. 7.31 – Processus de génération d'un jeu de test pour la carte TCB

TDS_{filtre}, $TDS_{comparateur}$ et TDS_{num} sont les données de test qui permettent de tester respectivement le filtre, le comparateur et la partie numérique de la carte TCB.

Le processus de génération d'un jeu de test pour la carte TCB est illustré figure 7.31. Un bloc de la carte est identifié par un rectangle portant son nom et les deux cases MF et MT représentent respectivement son modèle fonctionnel et son modèle de test. Le modèle de test est hachuré lorsque toutes ses transitions sont couvertes.

La première configuration sous test produit le jeu de test TDS_{filtre} du filtre F à partir de son modèle de test et des modèles fonctionnels du comparateur C, du convertisseur analogique-numérique Adc et de la mémoire Mem. A la fin de cette première itération du processus de génération, le modèle de test de F est couvert. Les modèles de test de Adc et de Mem sont également couverts, car les modèles fonctionnels de Adc et Mem sont couverts et les modèles de test de ces deux blocs sont identiques à leurs modèles fonctionnels.

La deuxième configuration sous test produit le jeu de test $TDS_{comparateur}$ de C à partir de son modèle de test et des modèles fonctionnels de F, Adc et Mem. A la fin de cette deuxième itération, tous les modèles de test des blocs de la carte sont couverts et on s'arrête.

Le jeu de données de test TDS ne contient donc pas explicitement des données de test relatives aux blocs Adc et Mem car ces blocs sont testés implicitement par les jeux de test TDS_{filtre} et $TDS_{comparateur}$.

La donnée de test TDS_{num} est une donnée de test supplémentaire, basée sur la synchronisation analogique-numérique, qui est obtenue à partir des modèles fonctionnels

des quatre blocs de la carte, et en utilisant le test de la transition de la tactique de test représentée figure 7.29.

Nous présentons maintenant l'expression littérale de chaque donnée de test. Une donnée de test (TD) est composée d'un couple de signaux d'entrée et d'un signal de sortie. Le couple d'entrée est de la forme (S, Clk) et le singleton de sortie est de la forme (MP) où S représente le signal de la source analogique, Clk le signal d'horloge et MP le signal au point de mesure numérique. Nous rappelons que T_e est la période de l'horloge Clk et f_c est la fréquence de coupure du filtre passe-haut.

$$\begin{array}{rll} TD1_{filtre} = & (In = & (siga(sinus, V_{IN}, 10\,f_c, 0), top(T_e)), \\ & Out = & sigd(siga(rect, \Delta T_1', T_0', t_0'), T_e)) \end{array} \tag{7.38}$$

$$\begin{array}{rll} TD2_{filtre} = & (In = & (siga(sinus, V_{IN}, f_c, 0), top(T_e)), \\ & Out = & sigd(siga(rect, \Delta T_1'', T_0'', t_0''), T_e)) \end{array} \tag{7.39}$$

Ces deux données de test permettent de tester respectivement le filtre dans sa bande passante et à sa fréquence de coupure. $V_{IN}, \Delta T_1', T_0', t_0', \Delta T_1'', T_0''$ et t_0'' sont calculés en vérifiant les conditions exprimées par le modèle de test du filtre et celles exprimées par les modèles fonctionnels des autres blocs.

Nous proposons d'examiner un peu plus en détail l'obtention de ces données de test. Nous avons choisi ici la stratégie globale de couverture des transitions des modèles de test. Ces deux données de test proviennent donc de la couverture des transitions du modèle de test du filtre. Cette couverture est réalisée itérativement en visant d'abord la couverture de la transition du modèle de test du filtre la plus éloignée de l'état initial : la transition numéro 4 (voir la numérotation des transitions du modèle de test du filtre sur la figure 7.28). Pour couvrir la transition numéro 4, on choisit le plus long chemin de l'état initial à la transition numéro 4. Un chemin possible est par exemple :

$$Entrée \xrightarrow{1} Att \xrightarrow{2} Sortie \xrightarrow{4} Entrée$$

La sensibilisation de ce chemin (puis les propagations amont/aval) conduit à la donnée de test $TD1_{filtre}$ qui couvre les transitions numéro 1, 2 et 4 dans le modèle de test du filtre. A ce stade, le modèle de test du filtre n'est pas totalement couvert (il manque la transition numéro 3). On itère donc notre processus en cherchant la transition du modèle de test du filtre non couverte la plus éloignée de l'état initial : la transition numéro 3. Pour couvrir la transition numéro 3, on choisit le chemin :

$$Entrée \xrightarrow{1} Att \xrightarrow{3} Sortie \xrightarrow{4} Entrée$$

La sensibilisation de ce chemin (puis les propagations amont/aval) conduit à la donnée de test $TD2_{filtre}$ qui couvre les transitions numéro 1, 3 et 4. Toutes les transitions du modèle de test du filtre sont alors couvertes. Le bloc filtre est ainsi testé unitairement par les deux données de test $TD1_{filtre}$ et $TD2_{filtre}$.

$$\begin{array}{rll} TD1_{comparateur} = & (In = & (siga(sinus, V_{IN}', F_{IN}', \phi_{IN}'), top(T_e)), \\ & Out = & sigd(siga(DC, 0.0, \_, \_), T_e)) \end{array} \tag{7.40}$$

$$TD2_{comparateur} = \begin{array}{ll} (In = & (siga(sinus, V''_{IN}, F''_{IN}, \phi''_{IN}), top(T_e)), \\ Out = & sigd(siga(rect, \Delta T_1, T_0, t_0), T_e)) \end{array} \quad (7.41)$$

Ces deux données de test permettent de tester respectivement le comportement du comparateur en dessous et au-dessus de son seuil. $V'_{IN}, F'_{IN}, \phi'_{IN}, V''_{IN}, F''_{IN}, \phi''_{IN}, \Delta T_1, T_0$ et t_0 sont calculés en vérifiant les conditions exprimées par le modèle de test du comparateur et celles exprimées par les modèles fonctionnels des autres blocs. Elles sont obtenues par couverture des transitions du modèle de test du comparateur de la même façon que celle décrite pour le filtre.

$$TD1_{num} = \begin{array}{ll} (In = & (siga(sinus, V_{IN}, \frac{1}{T_e}, \phi_{IN}), top(T_e)), \\ Out = & sigd(siga(rect, \Delta T_1, T_e, t_0), T_e)) \end{array} \quad (7.42)$$

Cette donnée de test, basée sur la synchronisation analogique-numérique, permet de tester la partie numérique de la carte TCB. $V_{IN}, \phi_{IN}, \Delta T_1$ et t_0 sont calculés en vérifiant la condition exprimée au point de mesure (test de la transition de la tactique de test représentée sur la figure 7.29) et les conditions exprimées par les modèles fonctionnels des blocs de la carte.

7.5 Bilan

Dans ce chapitre, nous avons proposé une méthodologie de test pour les cartes mixtes articulée autour de modèles fonctionnels et de modèles de test. Cette méthodologie, basée sur une approche fonctionnelle, est adaptée pour traiter un niveau de connaissance variable des spécifications de la carte à tester. La modélisation des signaux proposée et les modèles fonctionnels associés à chaque bloc fonctionnel de la carte permettent de décrire les interactions entre les différents blocs et rendent notre approche très flexible par rapport aux différents types de composants (analogique, numérique, mixte) présents sur la carte.

Les modèles de test intègrent le savoir-faire des ingénieurs de test et permettent ainsi à la méthodologie de prendre en compte cette expertise fondamentale qui ne peut le plus souvent pas être déduite de la modélisation de la carte.

Au niveau du processus de génération des données de test, les stratégies de test proposées répondent bien au besoin du test en maintenance : les stratégies de test globales permettent de réaliser la maintenance préventive (vérification de la non dégradation des fonctionnalités au cours du temps) et les stratégies de test locales permettent de réaliser la maintenance corrective (test d'un comportement particulier, diagnostic).

De plus, les tactiques de test permettent d'orienter très finement la génération des données dans le but de réaliser un test particulier (notamment à des fins de diagnostic).

Chapitre 8

Mise en oeuvre de la méthodologie proposée

Dans le chapitre précédent, nous avons montré que dans notre approche, une carte électronique est modélisée par un ensemble d'automates à états finis communicants. Les stratégies de test que nous proposons se ramènent à parcourir des chemins (un ensemble de transitions) dans l'ensemble des CFSM modélisant la carte à tester. Les données de test sont alors générées en vérifiant les conditions associées aux transitions.

La programmation par contraintes est très pertinente pour la génération de données de test [Kor90, DO91, GBR98], car elle permet de travailler sur des plages de valeurs dans un premier temps, et les solveurs utilisent des algorithmes optimisés pour la résolution de contraintes (de différents types). Le testeur pourra ensuite instancier les données de test pour prendre en compte des contraintes applicatives ou environnementales particulières. Par ailleurs, la programmation logique est tout à fait appropriée pour décrire des parcours de chemins dans des automates, et donc nos stratégies de test.

Pour ces raisons, nous avons choisi d'implanter notre méthodologie en utilisant la programmation logique par contraintes. Dans l'implantation de notre méthodologie de test, les conditions associées aux transitions sont traduites en contraintes (au sens de la programmation par contraintes). Les données de test sont obtenues en résolvant les contraintes correspondantes grâce aux solveurs proposés par la programmation par contraintes. Nous détaillons cette implantation en commençant par introduire le paradigme de la programmation logique par contraintes (section 8.1). Puis nous montrons comment la modélisation d'une carte (section 8.2) et la génération des données de test (section 8.3) sont implantées avec ce paradigme. Nous présentons ensuite le prototype d'outil *Copernicia* (section 8.4). Nous terminons ce chapitre en présentant un bilan de la mise en oeuvre réalisée (section 8.5).

8.1 La programmation logique par contraintes

La présentation que nous faisons de la programmation logique et de la programmation logique par contraintes est issue de [AW07]. Cet excellent ouvrage couvre plus

particulièrement la programmation logique par contraintes à travers l'utilisation de la plate-forme ECL^iPS^e. Cette plate-forme est conçue pour résoudre les problèmes de satisfaction de contraintes. Elle propose un langage qui permet de modéliser le problème, et différents algorithmes de résolution (solvers) [NSSS04]. Un point fort d'ECL^iPS^e est que la modélisation d'un problème est indépendante du solveur choisi. Ainsi, il est possible d'utiliser des solveurs différents sans retoucher la modélisation [WNS97]. Nous avons choisi également d'utiliser cette plate-forme car elle est gratuite et elle bénéficie de nombreux cas d'études industriels.

Avant d'aller plus loin dans la description de la programmation logique par contraintes (section 8.1.2), nous rappelons brièvement les caractéristiques principales de la programmation logique (section 8.1.1).

8.1.1 La programmation logique

La programmation logique est un paradigme de programmation simple et puissant bien adapté pour le calcul et la représentation de la connaissance. Le langage *Prolog (Programmation en Logique)* [GKPvC85] et la programmation logique par contraintes (PLC) sont des instances de ce paradigme. Le paradigme de la programmation logique diffère considérablement des autres paradigmes de programmation. La raison en est qu'il trouve ses origines dans la preuve automatique de théorèmes, dont il a repris la notion de déduction. Ainsi, pendant le processus de déduction, certaines valeurs sont calculées. Ce paradigme peut être résumé par les trois caractéristiques suivantes :

- une variable logique peut représenter n'importe quel type de donnée,
- lorsqu'on pose une question (but) à un programme, les variables sont minimalement substituées plutôt que d'être affectées et mises à jour,
- l'exécution d'un programme prend la forme d'un arbre de recherche qui inclut des points de choix où différents chemins peuvent être pris : si les substitutions deviennent inconsistantes, le programme revient en arrière au plus récent point de choix et essaie un autre chemin. Ce mécanisme est appelé retour-arrière (*backtracking*).

Un programme logique est constitué d'un ensemble de *prédicats*. Un prédicat est défini par un ensemble de *clauses*. Une clause est soit une *règle*, soit un *fait*. Un prédicat est caractérisé par son nom et son arité. En ECL^iPS^e, la syntaxe d'une règle est la suivante :

```
T :- Q.
```

où T représente la tête de la règle et Q le corps de la règle. La tête est formée d'un unique *littéral* constitué d'un symbole de prédicat et de ses arguments. Le corps de la règle est formé d'une conjonction de littéraux séparés par des virgules et peut être éventuellement vide. Le symbole `:-` est appelé le *séparateur de règle*. Une règle avec un corps vide s'appelle un fait et s'écrit

```
T.
```

Un exemple de règle est :

```
je_suis_moi(P,N) :- prenom(P), nom(N).
```

où l'arité du prédicat `je_suis_moi` vaut 2 (noté par convention `je_suis_moi/2`) et `prenom/1` et `nom/1` sont des prédicats définis respectivement par les faits :

```
prenom(bertrand).
```

et

```
nom(gilles).
```

`P` et `N` sont des variables logiques (débutent par une majuscule), `bertrand` et `gilles` sont des constantes symboliques (en minuscules).

Du point de vue sémantique, un programme Prolog peut être interprété comme un ensemble de formes propositionnelles de la logique des prédicats (ou logique du premier ordre). Une règle est interprétée comme une implication logique. Ainsi, la règle de notre exemple est interprétée comme :

$$\forall P. \forall N. (prenom(P) \wedge nom(N) \Rightarrow je\_suis\_moi(P, N)) \tag{8.1}$$

où les séparateurs `:-` et `,` dans la règle ont été respectivement substitués par les connecteurs logiques d'implication et de conjonction.

8.1.2 La programmation logique par contraintes (PLC)

La programmation par contraintes est une approche dans laquelle on spécifie des exigences (contraintes) et on trouve des solutions à ces exigences en utilisant des méthodes générales ou spécifiques du domaine [AW07]. Nous expliquons maintenant la notion essentielle, objet de la PLC, à savoir celle de problème de satisfaction de contraintes (*Constraint Satisfaction Problem ou CSP*).

De manière informelle, un problème de satisfaction de contraintes se présente sous la forme d'une séquence finie de variables, chacune étant définie sur un domaine, et un ensemble de contraintes, chacune portant sur un sous-ensemble des variables considérées. Résoudre le CSP consiste à trouver les valeurs de ses variables qui satisfont toutes les contraintes.

Plus formellement, un CSP implique une séquence finie de variables $X = x_1, \cdots, x_n$ de domaines respectifs $D_1, \cdots, D_n$, et un ensemble fini de contraintes C, chacune portant sur un sous-ensemble de X. Un CSP est noté $\langle C \ ; \ x_1 \in D_1, \cdots, x_n \in D_n \rangle$. Une solution de $\langle C \, ; x_1 \in D_1, \cdots, x_n \in D_n \rangle$ correspond aux valuations $\{(x_1, d_1), \cdots, (x_n, d_n)\}$, où chaque d_i est un élément du domaine D_i tel que, pour chaque contrainte $c \in C$ sur les variables $x_{k1}, \cdots, x_{km}$, on ait $(d_{k1} \cdots d_{km}) \in [c]$. La notation $[c]$ représente *l'interprétation* de la contrainte c, i.e. l'ensemble des valuations des variables telles que c soit vraie.

L'exemple suivant est un CSP simple utilisant des contraintes d'égalité et d'inégalité :

$$\langle x = y, y \neq z, z \neq u \ ; \ x \in \{1,2,3\}, y \in \{3,4,5\}, z \in \{1,2\}, u \in \{2\} \rangle. \tag{8.2}$$

Il est facile de vérifier que $\{(x,3), (y,3), (z,1), (u,2)\}$ est son unique solution.

La force de la programmation par contraintes est de proposer des solveurs pour résoudre les CSP. Ces solveurs sont des algorithmes optimisés pour résoudre les différents types de contraintes, en fonction des différents domaines des variables. Il est également possible de spécifier et de coder ses propres algorithmes de résolution.

La programmation logique par contraintes est la combinaison de la programmation logique et de la programmation par contraintes. Elle utilise ainsi le moteur de recherche Prolog basé sur l'unification pour l'aspect contrôle de l'exécution et la puissance des solveurs sur les domaines finis ou continus pour l'aspect calcul des données. Ayant introduit la programmation logique, puis les CSP, nous illustrons maintenant la manière de résoudre ces derniers en utilisant tout d'abord Prolog (sans solveur de contraintes spécifique), puis la plate-forme ECL^iPS^e(Prolog et solveurs) afin d'en démontrer l'intérêt.

Le programme Prolog suivant implante l'exemple de CSP donné par l'expression (8.2) :

```
resoudre(Liste) :- rechercher(Liste), testerContraintes(Liste).
rechercher(Liste) :- Liste = [X,Y,Z,U],
                     membre(X,[1,2,3]),
                     membre(Y,[3,4,5]),
                     membre(Z,[1,2]),
                     membre(U,[2]).
testerContraintes(Liste) :-  Liste = [X,Y,Z,U],
                             X = Y, Y \= Z, Z \=U.
membre(X, [X|_]).
membre(X, [_|L]) :- membre(X,L).
```

Nous résolvons notre problème en posant le but suivant :

```
resoudre([X,Y,Z,U]).
```

L'exécution Prolog de ce but provoque une recherche en profondeur d'abord avec retour-arrière. La figure 8.1 montre l'arbre de recherche construit. Chaque chemin dans l'arbre (entre le noeud racine et un noeud terminal) correspond à une instanciation possible pour les variables X, Y, Z et U. Les contraintes du CSP sont évaluées à la fin de chaque branche. Les flèches indiquent le sens du parcours effectué. Le chemin dessiné en gras représente le succès du but `resoudre`. Il donne la solution du CSP.

Cette approche est inefficace car toutes les instanciations possibles des variables du CSP sont générées. La taille de l'arbre de recherche (en nombre de feuilles ou de chemins) devient très grande lorsque le nombre de variables et leurs domaines respectifs augmentent. En effet, le calcul du nombre de feuilles (ou chemins) N de l'arbre est donné par :

$$N = \prod_i |D_i|$$

où $|D_i|$ représente la taille du domaine de la variable i. Dans notre exemple, N vaut 18.

Une approche plus efficace consiste, lors du parcours dans l'arbre de recherche, à évaluer les contraintes dès que possible. Cependant, cela nécessite d'entrelacer convenable-

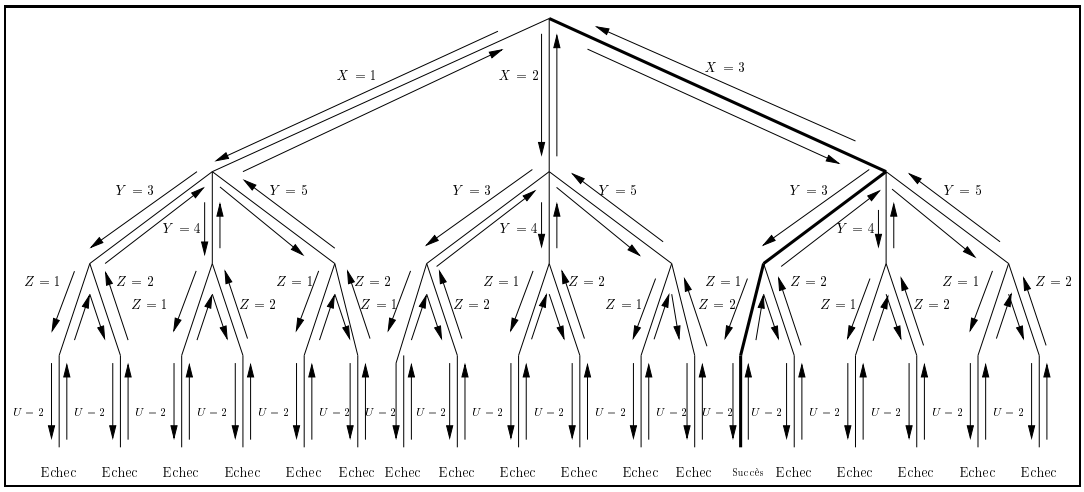

FIG. 8.1 – Recherche en profondeur d'abord avec retour-arrière.

ment le test des contraintes et l'instanciation des variables car les primitives spécifiques liées à l'évaluation d'expressions arithmétiques et la non-unifiabilité ne n'exécutent que sur des arguments clos ou suffisamment instanciés. Par exemple, le but `2 \= Y` échoue si `Y` n'est pas instancié. Le programme suivant montre clairement un entrelacement correct entre le test des contraintes et les instanciations de variables :

```
resoudre2(Liste) :- Liste = [X,Y,Z,U],
                    membre(X,[1,2,3]),
                    X = Y,
                    membre(Y,[3,4,5]),
                    membre(Z,[1,2]),
                    Y \= Z,
                    membre(U,[2]),
                    Z \= U.
```

La figure 8.2 montre l'arbre de recherche construit par l'exécution Prolog du but

`resoudre2([X,Y,Z,U]).`

Le fait d'évaluer les contraintes dès que possible au cours de la recherche permet de couper les branches de l'arbre qui ne correspondent à aucune solution dès la première inconsistance (ou violation) du système de contraintes. Le chemin dessiné en gras sur la figure 8.2 représente le succès du but et donne la solution du CSP.

L'utilisation de la programmation logique seule pour résoudre un CSP souffre de deux importantes limitations :

- l'entrelacement du test des contraintes et l'instanciation des variables est spécifique à chaque CSP et il n'est pas possible d'écrire un programme de résolution générique optimal de CSP,
- il n'est pas possible de déduire des contraintes d'un CSP les réductions possibles sur le domaine de ses variables. Par exemple, il est clair que dans le CSP (8.2),

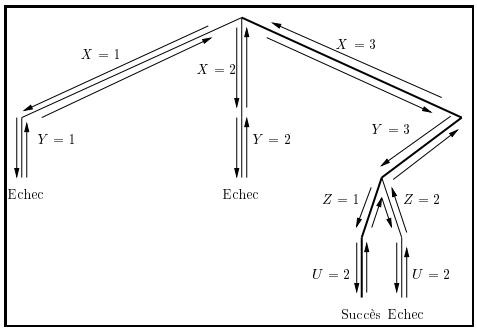

FIG. 8.2 – Recherche en profondeur d'abord avec retour-arrière en évaluant les contraintes dès que possible.

on aimerait déduire de la contrainte $x = y$ que les domaines respectifs de x et de y se réduisent au singleton $\{3\}$.

La programmation logique par contraintes reprend la souplesse de la programmation logique et s'affranchit des deux limitations décrites ci-dessus. La réduction des domaines des variables d'un CSP à partir de ses contraintes est appelée *propagation des contraintes*. La librairie `ic` (interval constraints) [ea03] proposée par la plate-forme ECL^iPS^e est un solveur de contraintes d'intervalles hybride qui permet de résoudre des CSPs dont les variables appartiennent aux domaines finis ou continus. L'utilisation du solveur `ic` permet de dépasser les limitations de la programmation logique en :

- retardant l'évaluation des contraintes (buts retardés) jusqu'à ce que leurs variables soient suffisamment instanciées,
- propageant les contraintes.

Ainsi, l'exécution du but :

```
resoudre3([X,Y,Z,U]).
```

pour le programme :

```
:-lib(ic).
resoudre3(Liste) :- declarerDomaines(Liste),
                    genererContraintes(Liste),
                    rechercherIc(Liste).

declarerDomaines(Liste) :-  Liste = [X,Y,Z,U],
                            X::[1..3], Y::[3..5],
                            Z::[1..2], U::[2..2].
genererContraintes(Liste) :-  Liste = [X,Y,Z,U],
                              X #= Y, Y #\= Z, Z #\= U.
rechercherIc(Liste) :- Liste = [X,Y,Z,U],
                       indomain(X),
                       indomain(Y),
                       indomain(Z),
                       indomain(U).
```

construit l'arbre de recherche dessiné figure 8.3 où la propagation des contraintes a réduit respectivement les domaines des variables x, y, z et u aux singletons $\{3\}$, $\{3\}$, $\{1\}$ et $\{2\}$. Dans cet exemple, l'unique chemin (dessiné en gras) de l'arbre de recherche correspond à l'unique solution du CSP.

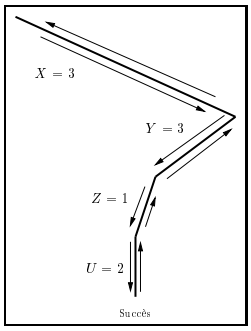

FIG. 8.3 – Recherche en profondeur d'abord avec retour-arrière avec propagation des contraintes

Nous avons introduit les bases de la programmation logique et de la programmation logique par contraintes. Dans la prochaine section, nous présentons la manière dont nous avons utilisé la PLC, à travers la plate-forme ECL^iPS^e, pour implanter notre modélisation d'une carte électronique.

8.2 Modélisation d'une carte en ECL^iPS^e

Dans cette section, nous montrons comment implanter la modélisation d'une carte électronique en ECL^iPS^e. Dans notre approche, une carte est modélisée par un ensemble de CFSM qui échangent entre eux des signaux. Les signaux sont représentés selon le formalisme présenté en section 7.2. Comme nous l'avons mentionné au chapitre précédent, le modèle fonctionnel et le modèle de test d'un bloc de la carte sont représentés chacun par un ensemble de CFSM. Chaque CFSM est implanté par des prédicats ECL^iPS^e appelés *prédicats de représentation*, utilisés par les algorithmes de génération de données de test. Afin d'optimiser le parcours de chemin effectué par les algorithmes, les états et les transitions des CFSM sont pondérés. Ces pondérations sont exprimées par les *prédicats de pondération*. Le calcul des données de test nécessite, à différentes phases, de mettre à jour, pour chaque CFSM, l'ensemble des états visités, des transitions franchies, et des signaux (contraints) émis et/ou reçus. Pour ce faire, un contexte global est défini pour ces données.

Les prédicats de représentation, les prédicats de pondération et la représentation du contexte sont présentés dans les sections 8.2.1, 8.2.2 et 8.2.3. La section 8.2.4 illustre leur utilisation sur l'exemple d'application introduit en section 7.3.3.

8.2.1 Prédicats de représentation

13 prédicats de représentation ont été définis afin de représenter les caractéristiques des blocs ainsi que les états, les transitions et les différents cheminements dans les automates. Dans un souci de clarté et de concision, nous donnons uniquement le nom et l'arité des prédicats. Les transitions et les états d'un CFSM sont identifiés par des numéros.

- `nombre_etats/2` associe à chaque automate son nombre d'états,
- `nombre_transitions/2` associe à chaque automate son nombre de transitions,
- `etat_initial/2` associe à chaque automate son état initial,
- `lsuiv/3` associe à chaque état de chaque automate la liste de ses états successeurs,
- `lprec/3` associe à chaque état de chaque automate la liste de ses états prédécesseurs,
- `ltrans_suiv/3` associe à chaque état de chaque automate la liste de ses transitions sortantes,
- `ltrans_prec/3` associe à chaque état de chaque automate la liste de ses transitions entrantes,
- `transition/7` décrit les transitions de chaque automate,
- `nombre_automates/1` définit le nombre d'automates qui modélisent la carte,
- `sources/1` définit la liste des automates modélisant les blocs d'entrée,
- `horloges/1` définit la liste des automates modélisant les blocs horloges,
- `sorties/1` définit la liste des couples automates/transition modélisant les blocs de sortie (un seul couple par automate correspondant à la transition finale),
- `caract_physiques/2` décrit la liste des caractéristiques physiques d'un bloc de la carte modélisé par un automate (par exemple, la fréquence de coupure d'un filtre ou le seuil d'un comparateur).

Le prédicat `transition/7` est défini par un ensemble de règles de la forme suivante :

```
transition(AutomateId, TransitionId, EtatDépart, EtatArrivée,
           CtxGlobalAvant, CtxGlobalAprès, Pas) :-
               conforme_spec(...),
               cumuler_contraintes(...),
               mettre_a_jour(...).
```

`TransitionId` est la transition de l'automate `AutomateId` qui va de l'état `EtatDépart` à l'état `EtatArivée`. Les variables `CtxGlobalAvant` et `CtxGlobalAprès` représentent respectivement le contexte global de l'ensemble des automates communicants avant et après le franchissement de la transition. Le paramètre `Pas` permet de borner la longueur de la séquence de test lors du processus de test. Le prédicat `conforme_spec` modélise les contraintes sur les signaux émis ou reçus par la transition. Ces contraintes sont cumulées à celles qui existent déjà sur le chemin courant par le prédicat `cumuler_contraintes`. Le prédicat `mettre_a_jour` met à jour le contexte global. Dans un souci de concision et de clarté, les arguments de ces trois prédicats ont été omis (en utilisant la notation « ... »).

Les prédicats de représentation `sources`, `horloges` et `sorties` traduisent les CFSM modélisant les blocs d'entrée/sortie de la carte (cf. section 7.3.1). Ces prédicats parti-

culiers sont appelés *prédicats d'entrée/sortie.*

8.2.2 Prédicats de pondération

Comme nous l'avons mentionné en section 7.4.1.1, nous cherchons à générer, dans le cadre des stratégies de test globales, un nombre minimal de données de test qui sont elles-mêmes de longueurs maximales. Nous avons également expliqué en section 7.4.1.2 que nous cherchons à générer, dans le cadre des stratégies de test locales, des données de test de longueurs minimales. Afin d'optimiser les parcours de chemins et d'atteindre ces objectifs, les algorithmes ATPG utilisent des *prédicats de pondération.* Nous les décrivons maintenant.

Le prédicat `poids/5` associe à chaque automate la longueur du plus court et du plus long chemin sans cycle pour franchir dans celui-ci un état donné ou une transition donnée, en partant de son état initial. La longueur du plus court chemin pour franchir un état (respectivement transition) en partant de l'état initial est appelée `poids inférieur` de l'état (respectivement transition). La longueur du plus long chemin pour franchir un état (respectivement une transition) en partant de l'état initial est appelé `poids supérieur` de l'état (respectivement transition). Le poids d'un état (respectivement transition) est composé de son poids inférieur et son poids supérieur. Le poids d'une transition est celui de son état de départ. Le prédicat `poids/5` est défini par un ensemble de clauses de la forme :

```
poids(1,AutomateId,EtatId,PoidsInférieur,PoidsSupérieur).
```

ou

```
poids(2,AutomateId,TransId,PoidsInférieur,PoidsSupérieur).
```

où `PoidsInférieur` et `PoidsSupérieur` sont respectivement la longueur du chemin le plus court et le plus long dans `AutomateId` pour franchir `EtatId` (lorsque le premier argument vaut 1) ou `TransId` (lorsque le premier argument vaut 2) en partant de l'état initial.

Le prédicat `poids_max/4` associe à chaque automate les poids inférieurs et supérieurs maximaux de ses états et les poids inférieurs et supérieurs maximaux de ses transitions. Ce prédicat est défini par un ensemble de clauses de la forme :

```
poids_max(X, AutomateId, PoidsInférieurMax, PoidsSupérieurMax).
```

où `X` égale 1 (état) ou 2 (transition), `PoidsInférieurMax` et `PoidsSupérieurMax` forment la pondération maximale des états (lorsque `X` égale 1) ou des transitions (lorsque `X` égale 2). Le poids inférieur maximal donne la longueur du plus court chemin garantissant l'accès à un état/transition quelconque à partir de l'état initial. Le poids supérieur maximal donne la longueur du plus long chemin sans cycle dans l'automate au départ de l'état initial.

8.2.3 Représentation du contexte

Nous avons expliqué en sections 7.4.1.1 et 7.4.1.2 que les stratégies de test globales et locales sont basées sur des parcours de chemins dans l'ensemble des CFSM qui mo-

délisent une carte électronique. Lorsqu'on parcourt un chemin, il faut mémoriser les informations associées à l'état de parcours de ce chemin qui sont nécessaires aux algorithmes de génération des données de test. L'ensemble de ces informations est appelé *contexte global*.

Le contexte global des automates communicants est représenté par le terme structuré

`env(liste_env_automates, liste_messages)`

qui contient la liste des contextes de chaque automate communicant (`liste_env_automates`) ainsi que les messages échangés par l'ensemble de ces automates (`liste_messages`).

Le contexte d'un automate est représenté par le terme structuré

`enva(etats_vus, trans_vues, mags_in, mags_out, mags_local)`

qui contient la liste des états visités dans l'automate (`etats_vus`), la liste des transitions franchies dans l'automate (`trans_vues`), la liste des magasins d'entrée de l'automate (`mags_in`), la liste des magasins de sortie de l'automate (`mags_out`) et la liste des magasins locaux de l'automate (`mags_local`).

Un magasin d'entrée (respectivement un magasin de sortie) est associé à chaque transition réceptrice (respectivement émettrice) de l'automate et permet de mémoriser les valeurs reçues (respectivement émises) par la transition. Ainsi, les magasins d'entrée/sortie permettent d'associer des valeurs aux transitions. Il est également intéressant de pouvoir associer des valeurs aux états, par exemple pour exprimer des propriétés d'un état indépendamment du chemin d'entrée et de sortie de cet état. La notion de magasin local correspond à cela. Un magasin local est associé à un état de l'automate.

Les prédicats :
- `extraire_env_mag_in/6`,
- `extraire_env_mag_out/6`,
- `extraire_env_mag_local/6`.

permettent d'accéder au contenu d'un magasin donné.

Le prédicat `ajout_env_elt/5` permet de modifier le contexte global.

Un exemple d'utilisation des prédicats présentés ci-dessus est donné en section 8.2.4.

8.2.4 Exemple

Nous avons traduit en prédicats ECL^iPS^e l'ensemble des modèles fonctionnels de la carte TCB présentée comme exemple d'application en section 7.3.3. Dans un esprit de concision, nous présentons dans cette section uniquement les extraits du code ECL^iPS^e correspondant aux CFSM représentant les modèles fonctionnels de la source, du filtre et du comparateur. Nous avons choisi de présenter plus précisément ces modèles (représentés respectivement par les figures 7.19, 7.22 et 7.23 du chapitre 7) car ils illustrent bien la communication entre automates et la propagation des contraintes sur les signaux analogiques élémentaires échangés par ces modèles. Le code ECL^iPS^e traduisant l'ensemble des modèles fonctionnels des blocs de la carte TCB est donné en annexe B.

La carte TCB est composée de 7 CFSM (un par bloc fonctionnel), ce qui se traduit au niveau du code par le fait :

```
nombre_automates(7).
```

Chaque automate est identifié par un numéro. Ainsi, la source, le comparateur et le filtre sont identifiés respectivement par les numéros 1, 5 et 7. Les autres numéros sont attribués à l'horloge (2), le point de mesure (3), la mémoire (4), et le convertisseur (6). L'automate numéro 1 modélise l'unique bloc d'entrée, l'automate numéro 2 modélise l'unique bloc horloge et l'automate numéro 3 modélise l'unique bloc de sortie (avec une seule transition numérotée 1) de la carte. Cela est traduit par les faits suivants :

```
sources([1]).
horloges([2]).
sorties([(3,1)]).
```

Les trois modèles auxquels nous nous intéressons plus particulièrement sont représentés par l'ensemble de faits suivants (se référer à la section 8.2.1 pour la description des prédicats) :

```
nombre_etats(1,1).
nombre_etats(5,3).
nombre_etats(7,2).
nombre_transitions(1,1).
nombre_transitions(5,4).
nombre_transitions(7,2).
etat_initial(1,1).
etat_initial(5,1).
etat_initial(7,1).
lsuiv(1,1,[1]).
lsuiv(5,1,[2]).
lsuiv(5,2,[3]).
lsuiv(5,3,[1]).
lsuiv(7,1,[2]).
lsuiv(7,2,[1]).
lprec(1,1,[1]).
lprec(5,1,[3]).
lprec(5,2,[1]).
lprec(5,3,[2]).
lprec(7,1,[2]).
lprec(7,2,[1]).
ltrans_suiv(1,1,[1]).
ltrans_suiv(5,1,[1]).
ltrans_suiv(5,2,[2,3]).
ltrans_suiv(5,3,[4]).
ltrans_suiv(7,1,[1]).
ltrans_suiv(7,2,[2]).
ltrans_prec(1,1,[1]).
```

```
ltrans_prec(5,1,[4]).
ltrans_prec(5,2,[1]).
ltrans_prec(5,3,[2,3]).
ltrans_prec(7,1,[2]).
ltrans_prec(7,2,[1]).
```

Chaque transition des automates est décrite par une règle `transition` présentée en section 8.2.1 :

```
transition(1,1,1,1,Env1,Env2,Pas) :-
        verifier_pas(Pas,_),
        conforme_spec(1,1,V,_),
        mettre_a_jour(1,1,V,Env1,Env2).
transition(5,1,1,2,Env1,Env2,Pas) :-
        verifier_pas(Pas,NouvPas),
        conforme_spec(5,1,V,_),
        cumuler_contraintes(5,1,V,Env1,Env1_inter,NouvPas),
        mettre_a_jour(5,1,V,Env1_inter,Env2).
transition(5,2,2,3,Env1,Env2,Pas) :-
        verifier_pas(Pas,NouvPas),
        conforme_spec(5,2,V,[V1]),
        cumuler_contraintes(5,2,V1,Env1,Env1_inter,NouvPas),
        mettre_a_jour(5,2,V,Env1_inter,Env2).
transition(5,3,2,3,Env1,Env2,Pas) :-
        verifier_pas(Pas,NouvPas),
        conforme_spec(5,3,V,[V1]),
        cumuler_contraintes(5,3,V1,Env1,Env1_inter,NouvPas),
        mettre_a_jour(5,3,V,Env1_inter,Env2).
transition(5,4,3,1,Env1,Env2,Pas) :-
        verifier_pas(Pas,NouvPas),
        conforme_spec(5,4,V,[V1]),
        cumuler_contraintes(5,4,V1,Env1,Env1_inter,NouvPas),
        mettre_a_jour(5,4,V,Env1_inter,Env2).
transition(7,1,1,2,Env1,Env2,Pas) :-
        verifier_pas(Pas,NouvPas),
        conforme_spec(7,1,V,_),
        cumuler_contraintes(7,1,V,Env1,Env1_inter,NouvPas),
        mettre_a_jour(7,1,V,Env1_inter,Env2).
transition(7,2,2,1,Env1,Env2,Pas) :-
        verifier_pas(Pas,NouvPas),
        conforme_spec(7,2,V,[V1]),
        cumuler_contraintes(7,2,V1,Env1,Env1_inter,NouvPas),
        mettre_a_jour(7,2,V,Env1_inter,Env2).
```

Avant de décrire le prédicat `conforme_spec`, nous présentons les faits du prédicat `caract_physiques` utilisés pour spécifier la fréquence de coupure du filtre (1000 Hz) et

la valeur du seuil du comparateur (5 V) :

```
caract_physiques(5, [5.0]).
caract_physiques(7, [1000.0]).
```

Le prédicat de représentation `conforme_spec` modélise les contraintes sur la valeur des signaux associés à une transition. Voici les clauses définissant ce prédicat dans notre exemple. Le cas le plus simple que l'on puisse rencontrer est le fait qui indique qu'il n'y a pas de contraintes sur le signal émis par le modèle fonctionnel de la source (cf. figure 7.19) :

```
conforme_spec(1,1,V,_).
```

Un autre cas simple est celui où il n'y a pas de contraintes sur le signal reçu par le modèle fonctionnel du comparateur (cf. figure 7.23) :

```
conforme_spec(5,1,V,_).
```

Un dernier cas simple est celui où le signal émis par le modèle fonctionnel du comparateur est celui stocké dans le magasin local de l'état « sortie » sans contraintes additionnelles (cf. figure 7.23) :

```
conforme_spec(5,4,V,[V]).
```

Les deux clauses suivantes modélisent les contraintes sur la valeur des signaux d'entrée et de sortie du modèle fonctionnel du comparateur dans le cas où

- l'amplitude instantanée du signal d'entrée est strictement inférieure au seuil du comparateur. Ces contraintes sont exprimées sur la figure 7.23. Le signal d'entrée est un signal sinusoïdal (la valeur « sinus » du champ forme est codée par l'entier 1) et le signal de sortie est un signal constant (la valeur « DC » du champ forme est codée par l'entier 3) dont la valeur est nulle (la valeur du deuxième champ est nulle). Les deux autres champs, qui non pas de signification pour ce type de signal, ont été mis à zéro,
- l'amplitude instantanée du signal d'entrée est supérieure ou égale au seuil du comparateur. Ces contraintes sont exprimées dans l'expression (7.28) et sur la figure 7.23. Le signal d'entrée est un signal sinusoïdal (la valeur « sinus » du champ forme est codée par l'entier 1) et le signal de sortie est un signal rectangulaire (la valeur « rect » du champ forme est codée par l'entier 2).

```
conforme_spec(5,2,V,[V1]) :-
        V1 = siga(Forme,Arg1,_,_),              % V1  = signal d'entrée
        caract_physiques(5,[S]),                % seuil = S = 5 V
        Forme $= 1,                             % forme de V1 = sinus
        Arg1  $< S,                             % amplitude < seuil
        V = siga(3,0,0,0).                      % V  = signal de
                                                % sortie DC = 0
conforme_spec(5,3,V,[Xr]) :-
        V  = siga(SeForme,SeArg1,SeArg2,SeArg3), % V  = signal de sortie
        Xr = siga(XrForme,XrArg1,XrArg2,XrArg3), % Xr = signal d'entrée
        caract_physiques(5,[S]),                % seuil = S = 5 V
```

```
        XrForme $= 1,                                % forme de Xr = sinus
        XrArg1  $>= S,                               % amplitude >= seuil
        SeForme $= 2,                                % forme de V  = rect
        SeArg2  $= 1 / XrArg2,
        R       $= S / XrArg1,
        SeArg1  $= 1 / (2 *  XrArg2) - 1 / (pi * XrArg2) * arcsin(R),
        SeArg3  $= 1 / (2 * pi * XrArg2) * arcsin(R) + XrArg3 / (2 * pi * XrArg2),
        SeArg1 $> 0,
        SeArg2 $> 0,
        SeArg3 $> 0.
```

Les deux clauses suivantes modélisent les contraintes sur la valeur des signaux d'entrée et de sortie du modèle fonctionnel du filtre exprimées dans l'expression (7.27) et sur la figure 7.22 :

```
conforme_spec(7,1,V,_) :-
        V = siga(Forme,_,_,_),      % le signal d'entrée est analogique
        Forme $= 1.                 % et de forme sinusoïdale
conforme_spec(7,2,V,[Xr]) :-
        V = siga(SeForme,SeArg1,SeArg2,SeArg3),  % V  = signal de sortie
        Xr = siga(_,XrArg1,XrArg2,XrArg3),       % Xr = signal d'entrée
        caract_physiques(7,[Fc]),                % fréquence de coupure
                                                 % Fc = 1000 Hz
        SeForme $= 1,                            % forme de V = sinus
        SeArg2  $= XrArg2,
        SeArg1  $= XrArg1 / sqrt(1 + (Fc/XrArg2)^2),
        SeArg3  $= XrArg3 - atan(Fc/XrArg2).
```

Le prédicat de représentation `cumuler_contraintes` permet de récupérer les contraintes rencontrées sur le chemin courant. Par souci de concision, nous donnons uniquement les clauses définissant ce prédicat pour le modèle fonctionnel du filtre :

```
cumuler_contraintes(7,1,V,Env1,Env1_inter,Pas) :-
        extraire_env_mag_out(Env1,1,1,V,Env1_inter,Pas).
```

qui récupère les contraintes du signal reçu de la source dans V (signal extrait du magasin de sortie associé à la transition numéro 1 du CFSM numéro 1 dans le contexte global Env1), V étant déjà contraint par la garde de la transition réceptrice (`conforme_spec(7,1,V,_)`).

```
cumuler_contraintes(7,2,V1,Env1,Env1_inter,Pas) :-
        extraire_env_mag_in(Env1,7,1,V1,Env1_inter,Pas).
```

qui récupère les contraintes du signal reçu par le filtre dans V1 (signal extrait du magasin d'entrée de la transition numéro 1 du CFSM numéro 7 dans le contexte global Env1), V1 étant utilisé pour calculer les contraintes modélisées par la garde de la transition émettrice (`conforme_spec(7,2,V,[V1])`).

Le prédicat `mettre_a_jour` met à jour le contexte global lorsque la transition d'un automate est franchie : la liste des états visités, la liste des transitions franchies, les signaux émis et/ou reçus et la liste des messages échangés entre les CFSM. Par souci de concision, nous donnons uniquement les clauses définissant ce prédicat pour le modèle fonctionnel du filtre :

```
mettre_a_jour(7,1,V,Env1_inter,Env2) :-
        ajout_env_elt(1,7,1,Env1_inter,Env10),                  % (a)
        ajout_env_elt(1,7,2,Env10,Env11),                       % (b)
        ajout_env_elt(2,7,1,Env11,Env12),                       % (c)
        ajout_env_elt(3,7,mag(1,V),Env12,Env13),                % (d)
        ajout_env_elt(5,_,reception(7,1,(1,1,V)),Env13,Env2).   % (e)
```

On ajoute dans la liste des états visités l'état numéro 1 (a) et l'état numéro 2 (b) de l'automate numéro 7. On ajoute dans la liste des transitions franchies la transition numéro 1 (c). La valeur V du signal reçu par la transition numéro 1 est stockée dans le magasin d'entrée associé à cette transition (d). Un message de réception est mémorisé (e) : « réception par la transition numéro 1 de l'automate numéro 7 de la valeur V émise par la transition numéro 1 de l'automate numéro 1 ». Ce faisant, le contexte global Env1_inter est mis à jour itérativement (via les contextes Env10, Env11, Env12 et Env13) pour aboutir au contexte Env2.

```
mettre_a_jour(7,2,V,Env1_inter,Env2) :-
        ajout_env_elt(1,7,1,Env1_inter,Env11),                  % (f)
        ajout_env_elt(2,7,2,Env11,Env12),                       % (g)
        ajout_env_elt(4,7,mag(2,V),Env12,Env13),                % (h)
        ajout_env_elt(5,_,emission(7,2,(5,1,V)),Env13,Env2).    % (i)
```

On ajoute dans la liste des états visités l'état numéro 1 (f) de l'automate numéro 7. On ajoute dans la liste des transitions franchies la transition numéro 2 (g). La valeur V du signal émis par la transition numéro 2 est stockée dans le magasin de sortie associé à cette transition (h). Un message d'émission est mémorisé (i) : « émission par la transition numéro 2 de l'automate numéro 7 de la valeur V vers la transition numéro 1 de l'automate numéro 5 ». Ce faisant, le contexte global Env1_inter est mis à jour itérativement (via les contextes Env11, Env12 et Env13) pour aboutir au contexte Env2.

Les poids des états des automates sont décrits par l'ensemble de faits suivants :

```
poids(1,1,1,1,1).
poids(1,5,1,1,1).
poids(1,5,2,2,2).
poids(1,5,3,3,3).
poids(1,7,1,1,1).
poids(1,7,2,2,2).
```

Les poids des transitions des automates sont dérivés par la règle suivante qui attribue à une transition le poids de son état de départ.

```
poids(2,A,N,P1,P2) :-
        existe_transition(A,N,X,_),
        poids(1,A,X,P1,P2).
```

Le prédicat `existe_transition/4` associe à chaque transition de chaque automate l'état de départ et l'état d'arrivée de celle-ci. Il est défini par les faits suivants :

```
existe_transition(1,1,1,1).
existe_transition(5,1,1,2).
existe_transition(5,2,2,3).
existe_transition(5,3,2,3).
existe_transition(5,4,3,1).
existe_transition(7,1,1,2).
existe_transition(7,2,2,1).
```

qui sont de la forme

```
existe_transition(AutomateId,TransId,EtatDépart,EtatArrivée).
```

où la transition `TransId` de l'automate `AutomateId` va de l'état `EtatDépart` vers l'état `EtatArrivée`.

Les pondérations maximales des états et des transitions des automates sont décrits par les faits suivants :

```
poids_max(1,1,1,1).
poids_max(1,5,3,3).
poids_max(1,7,2,2).
poids_max(2,1,1,1).
poids_max(2,5,3,3).
poids_max(2,7,2,2).
```

8.3 Mise en oeuvre des algorithmes de génération de données de test

Dans cette section, nous détaillons les algorithmes de génération de données de test. Nous commençons par présenter les algorithmes implantant les stratégies de test globales décrites en section 7.4.1.1 , puis ceux implantant les stratégies de test locales décrites en section 7.4.1.2. Par souci de clarté et de concision, nous nous limitons à la description des spécifications de ces algorithmes. Un extrait du code des algorithmes (algorithme de test d'une transition) est donné en annexe C. Le contexte global *env* contient, en fin d'exécution de chaque algorithme, les données de test.

8.3.1 Stratégies de test globales

Nous avons donné en section 7.4.3.1 le principe de l'algorithme 1 qui génère le jeu de test d'une carte électronique en utilisant une stratégie de test globale (couverture

des transitions ou couverture des états). Les détails d'implantation de cet algorithme sont présentés dans cette section, via les algorithmes 4, 5, 6, 7 et 8.

L'algorithme 4 permet d'exprimer le test global de la carte à partir du test de chacun de ses blocs. Il fait appel à l'algorithme 5 (respectivement 7) pour tester un bloc par couverture des transitions (respectivement couverture des états). L'algorithme 5 (respectivement 7) utilise l'algorithme 6 (respectivement 8) pour franchir une transition (respectivement un état). Dans l'algorithme 4, le paramètre N permet de borner la longueur maximale d'une donnée de test aux entrées/sorties de la carte. Par défaut, les blocs B_i les plus internes à la carte sont testés les premiers, ceux qui se trouvent aux interfaces en dernier. Ce choix permet d'optimiser la génération des données de test car souvent les blocs aux interfaces de la carte sont plus simples que les autres. On peut cependant introduire plus de souplesse à ce niveau en proposant à l'utilisateur d'associer un rang de test aux blocs de la carte.

Nous décrivons ci-après les fonctions utilisées dans l'algorithme 4 :

- $taille(L)$ renvoie le nombre d'éléments d'une liste L,
- $choix(L)$ renvoie un élément choisi dans la liste L,
- $modeleTest(B)$ renvoie le modèle de test du bloc B,
- $modeleFonc(B)$ renvoie le modèle fonctionnel du bloc B,
- $ajouter\_test(JT, JTB)$ ajoute le jeu de test JTB dans le jeu de test JT,
- $supprimer(L, L1)$ supprime dans une liste L les éléments d'une liste $L1$.

Les algorithmes 5, 6, 7 et 8 sont décrits dans les deux sous-sections suivantes.

8.3.1.1 Couverture des transitions d'un CFSM

L'algorithme 5 génère les séquences de test pour couvrir les transitions d'un CFSM. Nous décrivons ci-après les fonctions utilisées dans l'algorithme 5 :

- $creerEnv(E)$ crée, initialise et renvoie le contexte global d'un ensemble E d'automates communicants,
- $poidsInferieurMaxTransitions(A)$ renvoie la longueur du plus court chemin garantissant l'accès à une transition quelconque d'un automate A à partir de son état initial,
- $triTransOrd(A)$ renvoie la liste des transitions d'un automate A triées dans l'ordre des poids supérieurs décroissants dans le but de générer un nombre minimal de séquences de test, ces séquences de test étant elles-mêmes de longueur maximale,
- $taille(L)$ renvoie le nombre d'éléments d'une liste L,
- $premierElement(L)$ renvoie le premier élément d'une liste L,
- $franchirTransition(T, A, EA, ENV, TC\_Liste, Pas)$ est décrite par l'algorithme 6 et exhibe une donnée de test et un chemin valide des entrées aux sorties de la carte permettant de tester la transition désirée,
- $supprimer(L, L1)$ supprime dans une liste L les éléments d'une liste $L1$ (l'ordre des éléments restant de L est préservé).

Nous décrivons ci-après les fonctions utilisées dans l'algorithme 6 :

Algorithme 4 Test global de la carte

Entrées: Soit C une carte électronique composée d'une liste de blocs EB_Liste, soit SG une stratégie de test globale {Couverture des transitions ou couverture des états des modèles de test} et soit N un nombre de pas.

Sorties: un jeu de test JT pour C.

Variables:

B_i : bloc de la carte ;

A_{itest} : modèle de test (CFSM) de B_i ;

$A_{j_{fonc}}, j = 1 \cdots taille(EB\_Liste) - 1$: modèles fonctionnels (CFSM) ;

EA_{fonc} : ensemble de modèles fonctionnels (CFSM) ;

JTB : jeu de test ;

Début

$JT \leftarrow vide$;

tantque $taille(EB\_Liste) \neq 0$ **faire**

$B_i \leftarrow choix(EB\_Liste)$;

$A_{itest} \leftarrow modeleTest(B_i)$;

$EA_{fonc} \leftarrow vide$;

pour tout bloc $B_j \neq B_i$ de la carte **faire**

$A_{j_{fonc}} \leftarrow modeleFonc(B_j)$;

$EA_{fonc} \leftarrow EA_{fonc} \bigcup A_{j_{fonc}}$;

fin pour

si SG == couverture des transitions **alors**

$JTB \leftarrow couverture\_transitions(A_{itest}, EA_{fonc}, N)$ {Algorithme 5}

sinon

{SG == couverture des états}

$JTB \leftarrow couverture\_etats(A_{itest}, EA_{fonc}, N)$ {Algorithme 7}

fin si

$ajouter\_test(JT, JTB)$;

$supprimer(EB\_Liste, B_i)$;

fin tantque

Fin

- $trouverCheminsAmont(T, A, EA, ENV, C, Ch, Pas)$ renvoie vrai si des chemins C et Ch sont trouvés, faux sinon. C est un chemin valide dans l'automate A reliant son état initial à la transition T. $Ch = \bigcup_i Ch_i$ où i est un automate impliqué dans une communication (directe ou indirecte) avec A sur le chemin C ($i \in EA$). Ch_i est un chemin valide dans l'automate i reliant son état initial à la transition impliquant une communication (directe ou indirecte) avec A sur le chemin C,
- $franchir(T, A, ENV)$ renvoie vrai si le franchissement de la transition T de l'automate A est un succès, faux sinon,
- $trouverCheminsAval(T, A, EA, ENV, C', Ch', Pas)$ renvoie vrai si des chemins C' et Ch' sont trouvés, faux sinon. C' est un chemin valide dans l'automate A impliquant une communication (directe ou indirecte) entre la transition T et les automates de sortie. $Ch' = \bigcup_i Ch'_i$ où i est un automate impliqué dans une communication (directe ou indirecte) avec les automates de sortie sur le chemin C' ($i \in EA$). Ch'_i est un chemin valide dans l'automate i reliant l'état atteint dans ce dernier à la transition impliquant une communication (directe ou indirecte) avec les automates de sortie sur le chemin C',
- $trans(CH)$ renvoie l'ensemble de transitions constituant le chemin CH,
- $liste(E)$ renvoie la liste des éléments de l'ensemble E.

La sémantique des conjonctions utilisées correspond à la résolution par le solveur de contraintes du problème de l'extraction du chemin permettant de franchir la transition T.

À la fin de l'algorithme 5, le contexte global ENV contient les séquences de test générées, ainsi que le détail des transitions couvertes et non couvertes de l'automate.

8.3.1.2 Couverture des états d'un CFSM

L'algorithme 7 génère les séquences de test pour couvrir les états d'un CFSM. Nous décrivons ci-après les fonctions utilisées dans l'algorithme 7 :

- $poidsInferieurMaxEtats(A)$ renvoie la longueur du plus court chemin garantissant l'accès à un état quelconque d'un automate A à partir de son état initial,
- $triEtatsOrd(A)$ renvoie la liste des états d'un automate A triés dans l'ordre des poids supérieurs décroissants dans le but de générer un nombre minimal de séquences de test, ces séquences de test étant elles-mêmes de longueur maximale,
- $franchirEtat(E, A, EA, ENV, Pas)$ est décrite par l'algorithme 8 et exhibe les données de test et les chemins valides des entrées aux sorties de la carte permettant de tester l'état désiré.

Nous décrivons ci-après les fonctions utilisées dans l'algorithme 8 :

- $nbCouplesTransitionsES(E)$ renvoie le nombre de couples de transitions entrantes/sortantes autour de l'état E.

La sémantique des conjonctions utilisées correspond à la résolution par le solveur de contraintes du problème de l'extraction des chemins permettant de tester l'état.

Algorithme 5 Génération des données de test pour couvrir les transitions d'un CFSM

Entrées: Soit A un CFSM, soit EA l'ensemble des CFSM pouvant communiquer avec A et soit N un nombre de pas.

Sorties: Les séquences de test pour couvrir les transitions de A en au plus N pas en amont et en aval.

Variables:

L : entier ;

ENV : contexte global de l'ensemble des automates communicants $\{A\} \cup EA$;

$TAord\_Liste$: liste des transitions de A triées dans l'ordre des poids supérieurs décroissants ;

TC_Liste : liste des transitions couvertes de A ;

Début

$ENV \leftarrow creerEnv(\{A\} \cup EA)$;

$L \leftarrow poidsInferieurMaxTransitions(A)$;

si $N < L$ **alors**

AfficherErreur « la couverture de 100% des transitions de A en au plus N pas est impossible ⇒ augmenter la valeur de N »

sinon

$TAord\_Liste \leftarrow triTransOrd(A)$;

tantque $taille(TAord\_Liste) \neq 0$ **faire**

$T \leftarrow premierElement(TAord\_Liste)$;

si $franchirTransition(T, A, EA, ENV, TC\_Liste, N)$ **alors**

$supprimer(TAord\_Liste, TC\_Liste)$;

sinon

$supprimer(TAord\_Liste, T)$;

AfficherErreur « la transition T n'est pas couverte »

fin si

fin tantque

fin si

Fin

Algorithme 6 $FranchirTransition(T, A, EA, ENV, TC\_Liste, Pas)$

L'algorithme renvoie vrai et donne la liste des transitions couvertes TC_Liste dans le CFSM A si la transition T du CFSM A est testée, faux sinon. La séquence de test correspondante de longueur respectant Pas est dans ENV à la fin de l'algorithme.

Paramètres:

Pas : entier ; (entrée)

T : transition à tester ; (entrée)

A : CFSM possédant T ; (entrée)

EA : ensemble des CFSM pouvant communiquer avec A ; (entrée)

ENV : contexte global de l'ensemble des automates communicants $\{A\} \cup EA$; (entrée/sortie)

TC_Liste : liste des transitions couvertes dans A lorsque T est testée ; (entrée/sortie)

Variables:

$NouvPas$: entier ;

C, C' : chemin valide dans A ;

Ch, Ch' : ensemble de chemins valides dans EA ;

Début

$TC\_Liste \leftarrow$ vide ;

$NouvPas \leftarrow Pas - 1$;

si $trouverCheminsAmont(T, A, EA, ENV, C, Ch, NouvPas) \wedge franchir(T, A, ENV) \wedge trouverCheminsAval(T, A, EA, ENV, C', Ch', NouvPas)$ **alors**

$TC\_liste \leftarrow liste(trans(C) \cup \{T\} \cup trans(C'))$;

renvoyer vrai ;

sinon

renvoyer faux ;

fin si

Fin

À la fin de l'algorithme 7, le contexte global ENV contient les séquences de test générées, ainsi que le détail des états couverts et non couverts de l'automate.

8.3.2 Stratégies de test locales

Nous avons donné en section 7.4.3.2 le principe de l'algorithme 2 qui génère les données de test permettant de tester un état ou une transition et le principe de l'algorithme 3 qui permet de tester un chemin. Les détails d'implantation de ces algorithmes sont présentés dans cette section via les algorithmes 9, 10 et 11. L'algorithme 9 teste une transition particulière d'un CFSM, l'algorithme 10 teste un état particulier d'un CFSM et l'algorithme 11 teste un chemin particulier dans un ensemble de CFSM. Ces trois algorithmes sont présentés dans les sous-sections suivantes.

8.3.2.1 Test d'une transition

L'algorithme 9 génère la séquence de test pour tester une transition d'un CFSM A. Le paramètre d'entrée N permet de borner la longueur maximale d'une séquence de test dans EA. La fonction $poidsInferieurTrans(A, T)$ donne la longueur du plus court chemin de l'état initial à l'état de départ de la transition T. À la fin de l'algorithme, le contexte global contient la séquence de test générée.

8.3.2.2 Test d'un état

L'algorithme 10 génère les séquences de test pour tester un état d'un CFSM A. Le paramètre d'entrée N permet de borner la longueur maximale d'une séquence de test dans EA. La fonction $poidsInferieurEtat(A, E)$ donne la longueur du plus court chemin de l'état initial à l'état E. À la fin de l'algorithme, le contexte global contient les séquences de test générées.

8.3.2.3 Test d'un chemin

L'algorithme 11 génère les données de test pour tester un chemin. Le paramètre d'entrée N permet de borner la longueur maximale d'une séquence de test. Nous décrivons ci-après les fonctions utilisées dans l'algorithme 11 :

- $transitionDebut(C)$ renvoie le couple automate/transition correspondant à la transition de début du chemin C,
- $longueur(C)$ renvoie la longueur du chemin C (nombre de ses transitions),
- $transitionFin(C)$ renvoie le couple automate/transition correspondant à la transition de fin du chemin C,
- $franchir(C, EA, ENV)\{\wedge_{i=1}^{longueur(C)} franchir(T_i, A_i, ENV)\}$ renvoie vrai si le franchissement du chemin C est un succès, faux sinon. Le chemin C est constitué des transitions T_i dans les CFSM A_i ordonnées de son début vers sa fin.

À la fin de l'algorithme 11, le contexte global contient la séquence de test générée.

Algorithme 7 Génération des données de test pour couvrir les états d'un CFSM

Entrées: Soit A un CFSM, soit EA l'ensemble des CFSM pouvant communiquer avec A et soit N un nombre de pas.

Sorties: Les séquences de test pour couvrir les états de A en au plus N pas en amont et en aval.

Variables:

L, RES : entier ;

ENV : contexte global de l'ensemble des automates communicants $\{A\} \cup EA$;

$EAord\_Liste$: liste des états de A triés dans l'ordre des poids supérieurs décroissants ;

Début

$ENV \leftarrow creerEnv(\{A\} \cup EA)$;

$L \leftarrow poidsInferieurMaxEtats(A)$;

si $N < L$ **alors**

AfficherErreur « la couverture de 100% des états de A en au plus N pas est impossible $\Rightarrow$ augmenter la valeur de N »

sinon

$EAord\_Liste \leftarrow triEtatsOrd(A)$;

tantque $taille(EAord\_Liste) \neq 0$ **faire**

$E \leftarrow premierElement(EAord\_Liste)$;

$RES \leftarrow franchirEtat(E, A, EA, ENV, N)$;

$supprimer(EAord\_Liste, E)$;

si $RES == COUVERTURE\_PARTIELLE$ **alors**

AfficherErreur « l'état E est partiellement couvert »

sinon

si $RES == ECHEC$ **alors**

AfficherErreur « l'état E n'est pas couvert »

fin si

fin si

fin tantque

fin si

Fin

Algorithme 8 $FranchirEtat(E, A, EA, ENV, Pas)$

L'algorithme renvoie COUVERTURE_TOTALE (respectivement COUVERTURE_PARTIELLE) si l'état E du CFSM A est testé complètement (respectivement partiellement), ECHEC sinon. Les séquences de test correspondantes de longueur respectant Pas sont dans ENV à la fin de l'algorithme.

Paramètres:

Pas : entier ; (entrée)

E : état à tester ; (entrée)

A : CFSM possédant E ; (entrée)

EA : ensemble des CFSM pouvant communiquer avec A ; (entrée)

ENV : contexte global de l'ensemble des automates communicants $\{A\} \cup EA$; (entrée/sortie)

Variables:

$NouvPas, NB$: entier ;

C, C' : chemin valide dans A ;

Ch, Ch' : ensemble de chemins valides dans EA ;

Début

$NB \leftarrow 0$;

$NouvPas \leftarrow Pas - 1$;

pour chaque couple de transition entrante/sortante T_e/T_s autour de E **faire**

si $trouverCheminsAmont(T_e, A, EA, ENV, C, Ch, NouvPas) \wedge franchir(T_e, A, ENV) \wedge franchir(T_s, A, ENV) \wedge trouverCheminsAval(T_s, A, EA, ENV, C', Ch', NouvPas)$ **alors**

$NB \leftarrow NB + 1$;

fin si

fin pour

si $NB == nbCouplesTransitionsES(E)$ **alors**

renvoyer $COUVERTURE\_TOTALE$;

sinon

si NB > 0 **alors**

renvoyer $COUVERTURE\_PARTIELLE$;

sinon

renvoyer ECHEC ;

fin si

fin si

Fin

Algorithme 9 Génération des données de test pour tester une transition d'un CFSM

Entrées: Soit T la transition à couvrir dans un CFSM A, soit EA l'ensemble des CFSM pouvant communiquer avec A et soit N un nombre de pas.

Sorties: La séquence de test pour tester T en au plus N pas en amont et en aval.

Variables:

L : entier ;

ENV : contexte global de l'ensemble des automates communicants $\{A\} \cup EA$;

TC_Liste : liste des transitions couvertes de A ;

Début

$ENV \leftarrow creerEnv(\{A\} \cup EA)$;

$L \leftarrow poidsInferieurTrans(A, T)$;

si $N < L$ **alors**

AfficherErreur « il est impossible de tester T en au plus N pas (T non atteignable en moins de N pas) $\Rightarrow$ augmenter la valeur de N »

sinon

si $franchirTransition(T, A, EA, ENV, TC\_Liste, N) ==$ **faux alors**

AfficherErreur « la transition T n'est pas couverte »

fin si

fin si

Fin

Algorithme 10 Génération des données de test pour tester un état d'un CFSM

Entrées: Soit E l'état à couvrir dans un CFSM A, soit EA l'ensemble des CFSM pouvant communiquer avec A et soit N un nombre de pas.

Sorties: Les séquences de test pour tester E en au plus N pas en amont et en aval.

Variables:

RES, L : entier ;

ENV : contexte global de l'ensemble des automates communicants $\{A\} \cup EA$;

Début

$ENV \leftarrow creerEnv(\{A\} \cup EA)$;

$L \leftarrow poidsInferieurEtat(A, E)$;

si $N < L$ **alors**

AfficherErreur « il est impossible de tester E en au plus N pas (E non atteignable en moins de N pas) $\Rightarrow$ augmenter la valeur de N »

sinon

$RES \leftarrow franchirEtat(E, A, EA, ENV, N)$;

si $RES == COUVERTURE\_PARTIELLE$ **alors**

AfficherErreur « l'état E est partiellement couvert »

sinon

si $RES == ECHEC$ **alors**

AfficherErreur « l'état E n'est pas couvert »

fin si

fin si

fin si

Fin

Algorithme 11 Génération des données de test pour tester un chemin traversant un ensemble de CFSM

Entrées: Soit C le chemin à couvrir dans un ensemble de CFSM $EA1$, soit $EA2$ l'ensemble des CFSM pouvant communiquer avec $EA1$ et soit N un nombre de pas.

Sorties: La séquence de test pour tester C en au plus N pas en amont et en aval.

Variables:

N_1, L_1, L_2 : entier ;

$EA = EA1 \cup EA2$: ensemble de CFSM ;

ENV : contexte global de EA ;

C', C'' : chemin valide dans un automate ;

Ch', Ch'' : ensemble de chemins valides dans EA ;

T_d, T_f : transition ;

A_d, A_f : CFSM ;

Début

$ENV \leftarrow creerEnv(EA)$;

$(A_d, T_d) \leftarrow transitionDebut(C)$;

$L_1 \leftarrow poidsInferieurTrans(A_d, T_d)$;

$L_2 \leftarrow longueur(C)$;

si $N < L_1 + L_2$ **alors**

AfficherErreur « il est impossible de tester C en au plus N pas (C non franchissable en moins de N pas) $\Rightarrow$ augmenter la valeur de N »

sinon

$N_1 \leftarrow N - 1$;

$(A_f, T_f) \leftarrow transitionFin(C)$;

si $trouverCheminsAmont(T_d, A_d, EA, ENV, C', Ch', N_1) \wedge franchir(C, EA, ENV) \wedge trouverCheminsAval(T_f, A_f, EA, ENV, C'', Ch'', N_1) ==$ **faux alors**

AfficherErreur « le chemin C n'est pas couvert »

fin si

fin si

Fin

8.4 Le prototype *Copernicia*

Nous avons présenté en section 8.2 comment modéliser une carte électronique sous la forme de prédicats ECL^iPS^e. Nous avons également détaillé en section 8.3 les algorithmes de génération de données de test. Il est évidemment fastidieux de modéliser une carte en écrivant « à la main » l'ensemble des prédicats nécessaires. En plus d'être fastidieuse, cette manière de procéder est génératrice d'erreurs. Afin d'être réellement utilisable, notre approche pour la modélisation et le test de cartes mixtes doit être implantée par un outil convivial. Nous avons développé le prototype d'un tel outil appelé *Copernicia*.

L'outil *Copernicia* fournit une interface homme-machine (IHM) écrite en C++ avec la bibliothèque graphique ILOG Views [ILO02]. L'IHM permet de modéliser graphiquement de manière conviviale une carte électronique au niveau carte (cf section 7.3.1) et au niveau bloc (cf section 7.3.2). Par ailleurs, l'outil autorise la spécification de tactiques de test (cf. section 7.4.2) et permet de générer des données de test selon les stratégies de test globales et locales mentionnées en sections 7.4.3.1 et 7.4.3.2. Il se dégage ainsi deux fonctionnalités principales correspondant aux deux modes d'utilisation distincts qui sont présentés dans les sections suivantes : le mode *modélisation* et le mode *génération des données de test*.

8.4.1 Mode « modélisation »

La figure 8.4 montre l'outil *Copernicia* en mode d'utilisation modélisation. Ce mode d'utilisation est accessible en cliquant sur un bouton dédié dans la barre d'outils de *Copernicia*. La fenêtre à fond blanc située dans le coin supérieur gauche de la fenêtre principale est utilisée pour décrire la carte électronique au niveau carte. Les autres fenêtres à fond gris sont utilisées pour la représentation des modèles fonctionnels et des modèles de test des différents blocs de la carte. La fenêtre à fond blanc située en bas (fenêtre de sortie) donne des informations sur la modélisation interne de la carte.

Une bibliothèque contenant les modèles fonctionnels et de test des composants électroniques courants (filtres analogiques, comparateurs, multiplexeurs, $\cdots$) est à disposition de l'utilisateur (accessible par une liste déroulante de composants dans la barre de menu de l'outil). Ce dernier peut cependant spécifier ses propres modèles (en utilisant un éditeur dédié), ce qui fait de *Copernicia* un outil ouvert. Dans le mode modélisation, l'utilisateur peut également modéliser des tactiques de test en intégrant des éléments supplémentaires dans les modèles de test ou fonctionnels de certains blocs de la carte (niveau bloc). Il peut aussi modéliser une tactique de test comme un bloc spécifique supplémentaire (niveau bloc) à intégrer dans la carte (niveau carte).

8.4.2 Mode « génération des données de test »

En mode génération de données de test (accessible en cliquant sur un bouton dédié dans la barre d'outils), l'utilisateur a la possibilité de générer des données de test en cliquant sur des éléments graphiques déterminés.

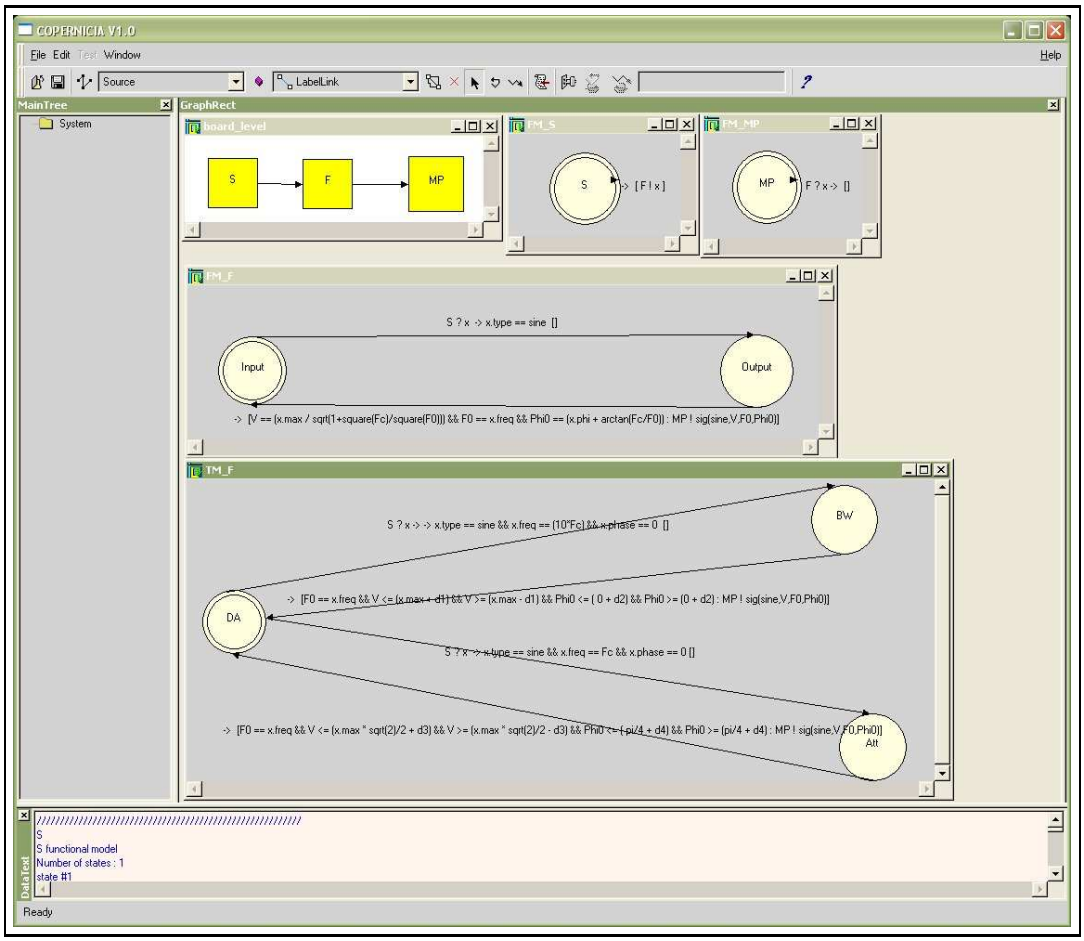

FIG. 8.4 – L'outil *Copernicia* en mode modélisation

• **Génération des données de test associées à une stratégie de test globale**

Deux boutons dédiés permettent de générer les données de test respectivement pour la couverture des transitions et la couverture des états de chacun des modèles de test des blocs de la carte (cf. section 7.4.3.1). Lorsque l'utilisateur clique sur ces boutons, l'outil *Copernicia* construit en mémoire un ensemble de représentations internes de la carte correspondant aux CFSM utilisés par l'algorithme 4 pour le test de chaque bloc. À partir de ces représentations internes, les prédicats de représentation et de pondération (cf. sections 8.2.1 et 8.2.2) sont générés automatiquement dans des fichiers $board_i.ecl$, $i = 1 \cdots n$, n étant le nombre de blocs internes de la carte (un fichier est généré pour chaque bloc de la carte à tester ; i est le numéro attribué au bloc), comme le montre la figure 8.5. Les données de test sont alors obtenues en exécutant l'algorithme 4 sur l'ensemble des fichiers $board_i.ecl$ avec la stratégie de test choisie. Pour ce faire, l'outil s'interface avec le moteur d'inférence ECL^iPS^e en utilisant une API (Application Programming Interface) décrite dans [NSSS04]. Les étapes de la génération des données

de test, comme le montre la figure 8.5, sont les suivantes :

Pour tout fichier $board_i.ecl$:

- compilation en ECL^iPS^e du fichier $board_i.ecl$,
- compilation du fichier `algos.ecl` contenant les algorithmes de génération de données de test,
- postage du but (appel de l'algorithme implantant la stratégie de test choisie),
- récupération des solutions (données de test générées),
- affichage des solutions dans la fenêtre de sortie et historisation dans un fichier (rapport de test).

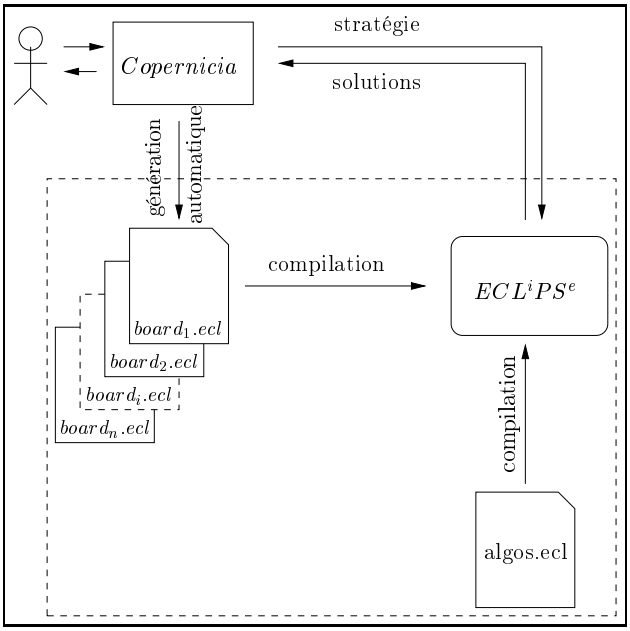

FIG. 8.5 – Interfaces externes de *Copernicia*

Rappelons que l'obtention d'une donnée de test finale repose sur une ultime étape d'instanciation. En effet, plusieurs données de test peuvent correspondre à un même chemin. Nos algorithmes vont ainsi rendre une donnée de test générique sous la forme d'un ensemble d'intervalles contraints qui représente un ensemble de données de test finales possibles pour un certain chemin. L'instanciation n'est pas faite par nos algorithmes ATPG mais peut être commandée à tout moment dans l'outil *Copernicia*. Garder cette généricité le plus tard possible est très intéressant. Cela permet en particulier à l'utilisateur, en fonction des contraintes supplémentaires liées à l'environnement de test de la carte, par exemple, de « choisir » l'instanciation qui lui convient le mieux.

• Génération des données de test associées à une stratégie de test locale

L'utilisateur a la possibilité de cliquer sur une transition (exécution de l'algorithme 9) ou un état d'un CFSM (exécution de l'algorithme 10). Il peut enfin sélectionner un chemin particulier qui peut traverser plusieurs CFSM (exécution de l'algorithme 11).

8.5 Bilan

Dans ce chapitre, nous avons présenté la mise en oeuvre de notre méthodologie en choisissant un langage d'implantation supportant la programmation logique par contraintes. Les modèles fonctionnels, les modèles de test des blocs et les tactiques de test étant constitués de CFSM, nos stratégies de test consistent à parcourir des chemins (des comportements) dans ces CFSM.

La génération des données de test est réalisée, pour chaque stratégie de test, en vérifiant l'ensemble des conditions rencontrées lors du franchissement des transitions lors des parcours de chemins. La vérification des conditions est implantée en utilisant la programmation par contraintes. Ainsi, vérifier les conditions revient à résoudre un ensemble de contraintes.

Dans cette réalisation, il est à noter que la programmation logique apporte beaucoup de souplesse pour parcourir des chemins dans un ensemble de CFSM (aspect contrôle) et la programmation par contraintes son efficacité et sa puissance pour le calcul des données de test (aspect données, calcul).

Chapitre 9

Validation de la méthodologie

Dans les deux chapitres précédents, nous avons présenté respectivement notre méthodologie de test pour les cartes électroniques mixtes (chapitre 7) et sa mise en oeuvre (chapitre 8). Dans ce chapitre, nous nous proposons d'étudier l'adéquation de notre méthodologie aux besoins du test de cartes électroniques en maintenance. Pour ce faire, nous nous appuyons sur une approche de validation. Cette dernière consiste à modéliser une carte donnée et à générer ses données de test à l'aide de notre méthodologie. Puis, en utilisant un outil de simulation adapté, à modéliser à nouveau la carte et simuler son comportement avec les données de test générées et enfin, à vérifier que les valeurs obtenues aux sorties primaires de la carte par simulation sont consistantes avec celles prédites par notre méthodologie et que l'ensemble des comportements ainsi simulés couvre les modèles, stratégies et tactiques de test mis en oeuvre. Nous validons ainsi l'aspect modélisation fonctionnelle de notre méthodologie (en confirmant les couples d'entrée/sortie de nos jeux de test par simulation) et son aspect test (en validant expérimentalement notre modélisation du processus de test et la sûreté de nos algorithmes de génération de données de test).

Nous décrivons tout d'abord le protocole de validation (section 9.1). Puis nous appliquons ce protocole de validation à un ensemble de trois cartes : les cartes TCB (cf. section 7.3.3), TCBE (une version étendue de la carte TCB) et la carte appelée CS1 constituée de composants analogiques et mixtes entrelacés (section 9.2). Nous terminons par un bilan (section 9.3).

9.1 Protocole de validation

Le protocole de validation consiste à :

1. définir un ensemble représentatif de cartes électroniques à tester,
2. appliquer notre méthodologie à chaque carte, i.e. modéliser la carte, définir les stratégies et les tactiques de test, générer les données de test (avec leur sorties associées),
3. simuler le comportement de chaque carte avec un logiciel de simulation adapté, plus particulièrement pour chaque donnée de test générée,

4. pour chaque donnée de test en entrée, vérifier la cohérence des signaux de sortie obtenus via la simulation par rapport aux signaux de sortie prédits par notre méthodologie,
5. vérifier que chaque jeu de test couvre les objectifs de test exprimés à travers les modèles, stratégies et tactiques de test utilisés.

Les objectifs principaux du protocole de validation sont :

1. évaluer l'adéquation de notre modélisation,
2. évaluer nos fonctionnalités de test (méthodologie de test globale, niveaux de test, modèles de test, stratégies de test, tactiques de test) et montrer qu'elles conviennent aux exigences, aux processus et aux objectifs du test en maintenance.

9.2 Mise en oeuvre du protocole

La mise en oeuvre du protocole nécessite de :
- définir un ensemble représentatif de cartes électroniques à tester,
- choisir un outil de simulation,
- itérer les étapes 2, 3, 4 et 5 du protocole sur chaque carte de l'ensemble représentatif.

Ces trois phases de mise en oeuvre sont détaillées dans les sous-sections suivantes.

9.2.1 Définition d'un ensemble représentatif de cartes

Notre ensemble représentatif de cartes est formé de trois cartes : la carte TCB introduite en section 7.3.3, la carte TCBE (cf. section 9.2.3.2) et la carte CS1 (cf. section 9.2.3.3). La carte TCB est une carte mixte très simple possédant des composants analogiques, numériques et mixtes, et des fonctionnalités temporelles simples. En outre, les composants de différentes natures ne sont pas entrelacés (l'entrelacement apparaît très souvent dans les cartes mixtes). La carte TCBE est une version étendue de la carte TCB qui possède un plus grand nombre de composants, ainsi que des fonctionnalités temporelles plus riches. Enfin, la carte CS1 possède des composants analogiques et mixtes différents des deux autres cartes, et qui sont en plus entrelacés. Pour ces raisons, il nous semble que ces trois cartes définissent un bon ensemble minimal d'exemples représentatifs de cartes électroniques à tester (premier point du protocole).

9.2.2 Choix de l'outil de simulation

Il existe plusieurs outils permettant de modéliser, simuler et analyser les systèmes dynamiques. Certains d'entre eux sont bien adaptés pour les systèmes tels que les cartes électroniques mixtes. Ils fournissent des éditeurs graphiques permettant de construire facilement des modèles complexes en interconnectant des blocs qui implantent des fonctions de base prédéfinies (générateurs de signaux, filtres analogiques, $\cdots$) ou des fonctions personnalisables. Sans être exhaustif, nous pouvons mettre en avant des outils commerciaux réputés tels que Simulink qui est une sur-couche de MATLAB [Mat03a],

SystemBuild qui fait partie de l'environnement MATRIXx [MAT], LabVIEW [Lab] et un outil du domaine public tel que Scicos qui fait partie de l'environnement Scilab [NS97]. Les fonctionnalités de Scicos sont similaires à celles de Simulink. En fait, Scilab, qui est développé par l'INRIA est connu comme la version de MATLAB dans le domaine public. La plupart des fonctionnalités offertes par Simulink sont également disponibles dans SystemBuild.

Afin de valider notre approche, nous avons choisi Simulink pour la modélisation et la simulation du comportement des cartes électroniques. Ce choix a été motivé par son adéquation par rapport aux objectifs visés et par l'expérience que nous avons déjà dans son utilisation au niveau des enseignements.

9.2.3 Itération du protocole de validation sur l'ensemble représentatif de cartes

Nous présentons dans les sous-sections suivantes la modélisation de chacune des cartes de l'ensemble représentatif (cf. section 9.2.1) à l'aide de notre méthodologie, puis la génération des données de test à partir des modèles produits. Chaque carte est ensuite simulée à l'aide de Simulink. Dans chacun des cas, la stratégie de test choisie est la stratégie globale de couverture des transitions. Nous avons opté pour une stratégie globale plutôt que locale car les jeux de test sont de taille plus importante pour une stratégie globale, ce qui permet d'avoir plus d'éléments d'étude sur l'ensemble de la carte et ainsi une validation plus significative. Dans les stratégies globales, nous avons choisi la couverture des transitions car elle est la stratégie globale minimale de notre méthodologie. En effet, l'autre stratégie globale qui consiste en la couverture des états implique la couverture des transitions.

9.2.3.1 La carte TCB

Dans cette section, nous présentons d'abord la description, puis la modélisation de la carte TCB et la génération des données de test par nos algorithmes ATPG. Nous présentons ensuite la modélisation de la carte TCB avec Simulink. Nous abordons enfin la simulation du comportement de la carte avec ces données de test et les résultats que nous avons obtenus.

9.2.3.1.1 Description de la carte

La description de la carte est disponible en section 7.3.3.1.

9.2.3.1.2 Modélisation de la carte TCB avec notre méthodologie

La modélisation complète de la carte TCB (modélisation au niveau carte, modèles fonctionnels et modèles de test des blocs) a été présentée en sections 7.3.3 et 7.4.4, ainsi qu'en annexe B.

9.2.3.1.3 Génération des données de test

Nous avons utilisé l'algorithme 4 décrit en section 8.3.1 et l'algorithme 5 décrit en section 8.3.1.1 (couverture des transitions) pour générer les données de test $TD1_{filtre}$, $TD2_{filtre}$, $TD1_{comp}$ et $TD2_{comp}$, et l'algorithme 9 décrit en section 8.3.2.1 (test d'une transition) pour générer la donnée de test $TD1_{num}$. L'expression littérale du jeu de données de test est donnée en section 7.4.4 avec les valeurs de paramètres suivantes :

- $S = 5\,V$ (seuil du comparateur),
- $f_c = 1000\,Hz$ (fréquence de coupure du filtre),
- $T_e = 0.002\,s$ (période de l'horloge),
- $\delta_{11} = \delta_{12} = \delta_{21} = \delta_{22} = 0.1$ (tolérances du modèle de test du filtre),
- $\delta = 0.2$ (tolérance du modèle de test du comparateur).

Les données de test générées à l'aide des algorithmes susmentionnés, puis instanciées sont les suivantes :

$$
\begin{array}{lcl}
TD1_{filtre} & = & (In = (siga(sinus, 5.5, 10000.0, 0.0), top(0.002)), \\
 & & Out = sigd(siga(rect, 0.0000148, 0.0001, 0.0000159), 0.002)) \\
TD2_{filtre} & = & (In = (siga(sinus, 7.778, 1000.0, 0.0), top(0.002)), \\
 & & Out = sigd(siga(rect, 0.000148, 0.001, 0.0000347), 0.002)) \\
TD1_{comp} & = & (In = (siga(sinus, 10.762, 498.295, 0.0), top(0.002)), \\
 & & Out = sigd(siga(DC, 0.0, \_, \_), 0.002)) \\
TD2_{comp} & = & (In = (siga(sinus, 14.158, 394.881, 0.0), top(0.002)), \\
 & & Out = sigd(siga(rect, 0.000224, 0.00253, 0.0000394), 0.002)) \\
TD1_{num} & = & (In = (siga(sinus, 12.298, 500.0, 5.819), top(0.002)), \\
 & & Out = sigd(siga(rect, 0.000273, 0.002, 0.00186), 0.002))
\end{array}
$$

Le rapport de test relatif à la donnée de test $TD1_{num}$ est donné en annexe D.

9.2.3.1.4 Modélisation de la carte TCB avec Simulink

La modélisation hiérarchique de la carte TCB effectuée avec Simulink est représentée sur les figures 9.1 et 9.2. La figure 9.1 représente le premier niveau de la modélisation où le rectangle central représente la carte TCB. L'entrée primaire de la carte (PI) est connectée à un générateur de tension analogique sinusoïdal. La sortie primaire de la carte (PO) est connectée à un bloc *po* qui modélise un point de mesure pour les données écrites dans la mémoire de la carte. Ces données (et leur date d'acquisition) sont écrites dans l'espace de travail de MATLAB (*workspace*). Des oscilloscopes affichent certains signaux durant la simulation. En particulier, les oscilloscopes connectés aux sorties respectives du filtre ($FilterOut$), de l'échantillonneur (Sampler Out) et du comparateur ($ComparatorOut$) rendent possible l'observation de signaux internes de la carte TCB. Finalement, la boîte *threshold* permet de régler la valeur du seuil du comparateur (ici à 5 V).

La figure 9.2 montre le deuxième niveau de modélisation. A ce niveau, la carte TCB est modélisée par un ensemble de diagramme blocs Simulink. Il est intéressant de noter

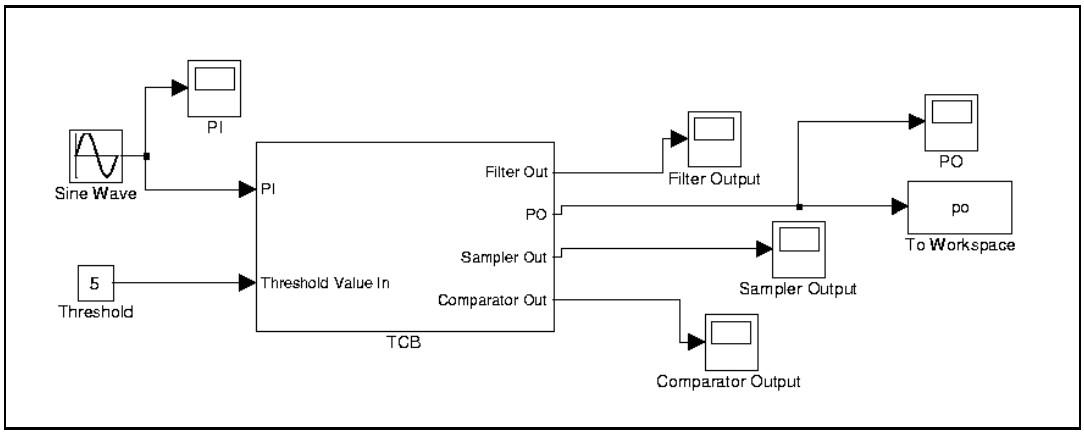

FIG. 9.1 – Modélisation de la carte TCB avec Simulink : premier niveau

qu'extérieurement, cette modélisation ressemble à celle que nous avons proposée avec notre méthodologie (voir figure 7.18). A chaque bloc défini au niveau carte de notre modélisation correspond un diagramme bloc Simulink :

- le filtre (F) est modélisé à l'aide du bloc prédéfini *fonction de transfert* (HP Transfer Fcn). Ce bloc implante la fonction de transfert du filtre analogique passe-haut du premier ordre défini par l'expression :

$$H(s) = \frac{s}{s + 1000 \,.\, 2\pi}$$

  où 1000 (Hz) est la valeur de la fréquence de coupure du filtre, avec $s = j\omega$, où j est le nombre imaginaire de module unité et ω la pulsation,
- le comparateur (C) est modélisé par un bloc personnalisé dont le comportement a été écrit en utilisant l'API des S-Fonctions [Mat03b],
- le convertisseur analogique-numérique (Adc) est modélisé à l'aide du bloc prédéfini *échantillonneur bloqueur d'ordre zéro* avec une fréquence d'échantillonnage de 500 Hz,
- la mémoire (Mem) est modélisée grâce au bloc prédéfini *expression générale*. L'expression utilisée est $u(1)$. Elle signifie que les valeurs de sortie du bloc sont identiques à celles de son entrée (fonction identité).

Plutôt que de mettre en dur la valeur du seuil du comparateur dans le corps de la S-Fonction qui modélise ce dernier, nous avons modélisé cette valeur de seuil par un bloc Simulink en entrée. Cela permet d'ajuster la valeur du seuil du comparateur sans avoir à recompiler le code du comparateur.

En comparant la modélisation obtenue via Simulink avec celle obtenue par notre méthodologie, nous pouvons noter que les deux approches reposent sur deux niveaux hiérarchiques de modélisation. Dans la modélisation Simulink, le premier niveau est utilisé dans le but de modéliser les entrées et les sorties de la carte. Le second niveau spécifie le comportement des composants de la carte et leurs liens. Dans notre méthodologie, le premier niveau spécifie les entrées et sorties de la carte mais également la manière

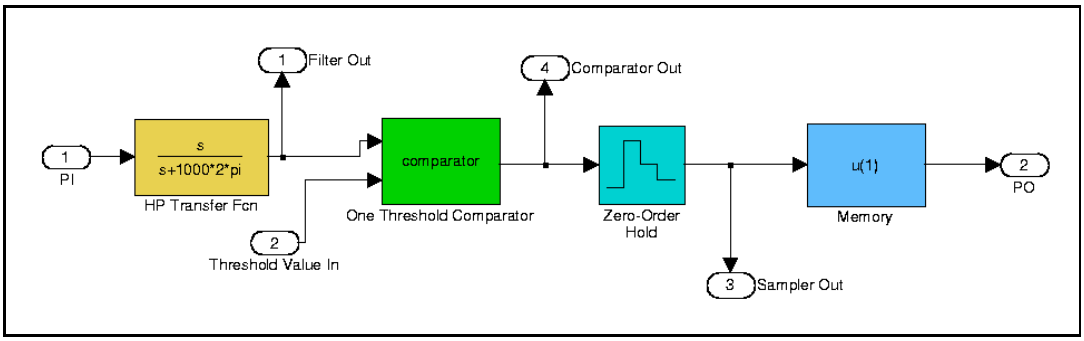

FIG. 9.2 – Modélisation de la carte TCB avec Simulink : deuxième niveau

dont les composants de la carte sont liés. Le second niveau est dédié à la spécification du comportement des composants de la carte. Les deux approches sont ainsi assez similaires. La véritable différence réside dans la représentation interne des blocs. En effet, un bloc Simulink est représenté de manière interne par un ensemble d'*équations différentielles ordinaires (ODE)* [Mat03a] et la simulation d'un modèle (un ensemble de blocs Simulink interconnectés) est basée sur l'intégration numérique d'un ensemble d'ODE effectuée par un solveur spécialisé[1]. Notre approche, quant à elle, repose sur les CFSM.

9.2.3.1.5 Simulation

Nous avons simulé avec Simulink le comportement de la carte TCB avec les données de test présentées en section 9.2.3.1.3. La simulation a été effectuée avec le solveur MATLAB ode45 (Dormand-Prince), de l'instant 0 à l'instant 0.01s. Dans cette section, nous nous concentrons sur la simulation avec la donnée de test $TD1_{num}$ générée pour la partie numérique. La figure 9.3 montre le stimulus de test analogique appliqué à l'entrée primaire.

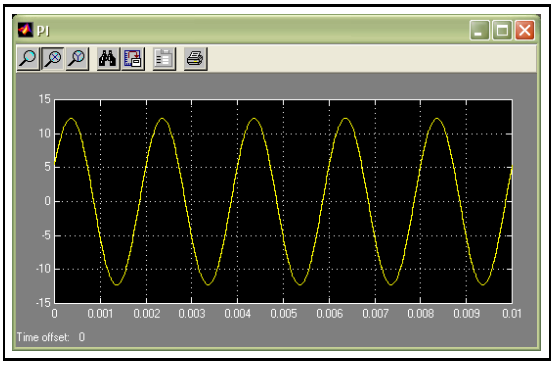

FIG. 9.3 – Stimulus de test analogique appliqué à l'entrée primaire

[1] MATLAB fournit un ensemble complet de solveurs

La figure 9.4 montre le signal de sortie du filtre. Ce signal est affaibli et décalé parce que la fréquence du signal sinusoïdal de test (500 Hz) est inférieure à la fréquence de coupure du filtre (1000 Hz). Grâce à l'utilisation de la tactique de test permettant la synchronisation analogique/numérique présentée en section 7.4.4.2, l'amplitude maximale du signal est légèrement supérieure à la valeur du seuil du comparateur (5 V) pour des instants multiples de la période d'échantillonnage.

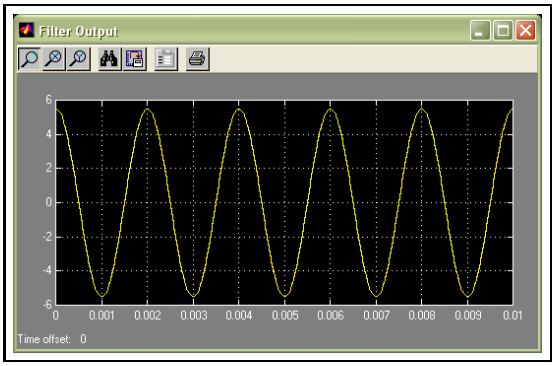

FIG. 9.4 – Signal de sortie du filtre

La figure 9.5 montre le signal de sortie du comparateur. Parce que l'amplitude instantanée du signal sinusoïdal d'entrée du comparateur est légèrement supérieure à la valeur de son seuil pour des instants multiples de la période d'échantillonnage, son signal de sortie est un signal rectangulaire. Les intervalles de temps pour lesquels la sortie du comparateur vaut un sont centrés sur les périodes correspondantes.

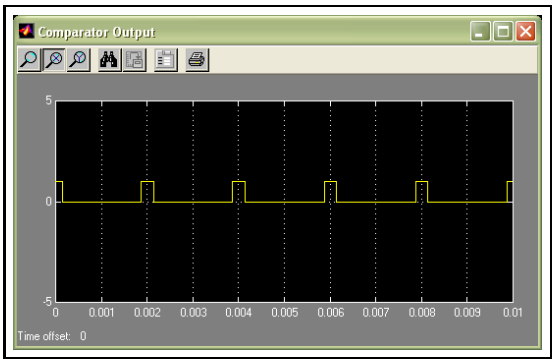

FIG. 9.5 – Signal de sortie du comparateur

La figure 9.6 montre le signal numérique présent à la sortie primaire de la carte. Nous pouvons remarquer que la mémoire prend la valeur un à l'instant $t_1 = 0.002s$ et qu'elle garde ensuite cette valeur jusqu'à la fin de la simulation. La mémoire vaut zéro à l'instant $t = 0$ car c'est la valeur par défaut qu'elle prend lors de l'initialisation de la carte.

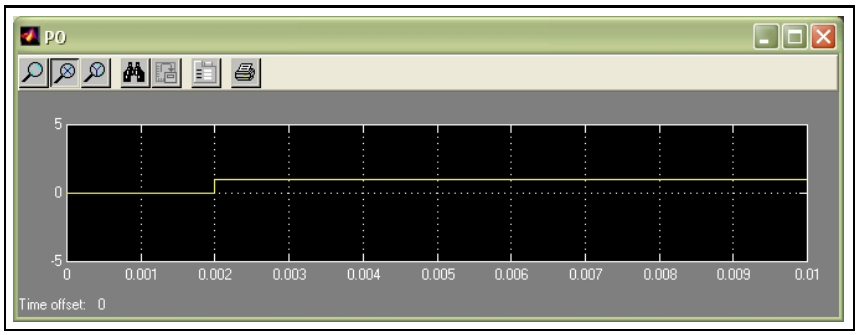

FIG. 9.6 – Signal présent à la sortie primaire de la carte TCB

9.2.3.1.6 Bilan

Les résultats de la simulation sont en accord avec les sorties prédites par les données de test. Ces résultats montrent que :

- la modélisation des signaux élémentaires proposée est suffisante pour représenter les signaux de la carte TCB,
- la modélisation fonctionnelle de la carte TCB au niveau carte et au niveau bloc (modèles fonctionnels) est correcte,
- l'application seule de la couverture des transitions des modèles de test permet de tester correctement le filtre, le comparateur, le convertisseur analogique-numérique et la mémoire,
- la tactique de test utilisée pour vérifier la synchronisation analogique/numérique permet de tester un comportement précis à l'échelle de la carte (test d'intégration).

Sur l'exemple de la carte TCB, nos fonctionnalités de test se sont révélées souples et nos algorithmes de génération de données de test sûrs.

9.2.3.2 La carte TCBE

Dans cette section, nous présentons d'abord la carte TCBE, puis sa modélisation et la génération des données de test avec notre méthodologie. Nous présentons ensuite la modélisation de la carte TCBE avec Simulink. Nous abordons enfin la simulation du comportement de la carte avec ces données de test et les résultats que nous avons obtenus.

9.2.3.2.1 Description de la carte

La carte TCBE peut être vue comme une extension de la carte TCB. Elle possède trois voies analogiques constituées chacune d'un filtre passe-haut du premier ordre et d'un comparateur. La fonction principale de la carte est de vérifier périodiquement les tensions instantanées des signaux d'entrée en les comparant à des seuils de tension donnés. Le résultat de la comparaison pour une voie est une valeur logique datée écrite dans une mémoire dédiée de la carte (il y a une mémoire par voie analogique).

9.2.3.2.2 Modélisation de la carte TCBE avec notre méthodologie

Nous présentons dans un premier temps la modélisation au niveau carte de la carte TCBE. Nous présentons ensuite les modèles fonctionnels, les modèles de test des blocs de la carte et les tactiques de test.

La figure 9.7 montre la modélisation de la carte TCBE au niveau carte. Sur cette figure, le périmètre de la carte est délimité par le rectangle en pointillés et les blocs se trouvant dans ce périmètre sont les blocs fonctionnels de la carte. La carte est composée de six blocs analogiques $F_1 \cdots F_3$ et $C_1 \cdots C_3$, d'un bloc mixte D et de trois blocs numériques $Mem_1 \cdots Mem_3$:

- les blocs $F_1 \cdots F_3$ représentent chacun un filtre analogique passe-haut du premier ordre,
- les blocs $C_1 \cdots C_3$ représentent chacun un comparateur à un seuil,
- le bloc D représente le contrôleur qui balaye cycliquement les voies de la carte,
- les blocs $Mem_1 \cdots Mem_3$ représentent chacun une mémoire.

Les blocs externes à la carte sont les suivants :

- les blocs d'entrée $S_1 \cdots S_3$ qui représentent chacun une source de tension analogique,
- les blocs de sortie $MP_1 \cdots MP_3$ qui représentent chacun un point de mesure (Measurement Point),
- le bloc Clk qui représente une horloge.

Nous présentons ci-après les modèles fonctionnels et les modèles de test des blocs susmentionnés.

Modèles fonctionnels des sources : le modèle fonctionnel de chaque source est identique [2] au modèle fonctionnel de la source S de la carte TCB représenté sur la figure 7.19.

Modèle fonctionnel de l'horloge : le modèle fonctionnel de l'horloge est identique [3] à celui de l'horloge Clk de la carte TCB représenté sur la figure 7.21.

Modèles fonctionnels des filtres analogiques : le modèle fonctionnel de chaque filtre est identique [4] à celui du filtre F de la carte TCB représenté sur la figure 7.22.

Modèles fonctionnels des comparateurs : le modèle fonctionnel de chaque comparateur est identique [5] à celui du comparateur de la carte TCB représenté sur la figure 7.23.

Modèle fonctionnel du contrôleur : le modèle fonctionnel du contrôleur numérique représenté figure 9.8 généralise le modèle fonctionnel du convertisseur analogique-numérique de la carte TCB (voir figure 7.24). Le signal de chaque comparateur est échantillonné, puis envoyé vers la mémoire adéquate, de manière cyclique.

[2] Il suffit de remplacer respectivement S par S_i et F par F_i avec $i = 1 \cdots 3$
[3] Il suffit de remplacer Adc par D
[4] Il suffit de remplacer respectivement S par S_i et C par C_i avec $i = 1 \cdots 3$
[5] Il suffit de remplacer F par F_i avec $i = 1 \cdots 3$ et Adc par D

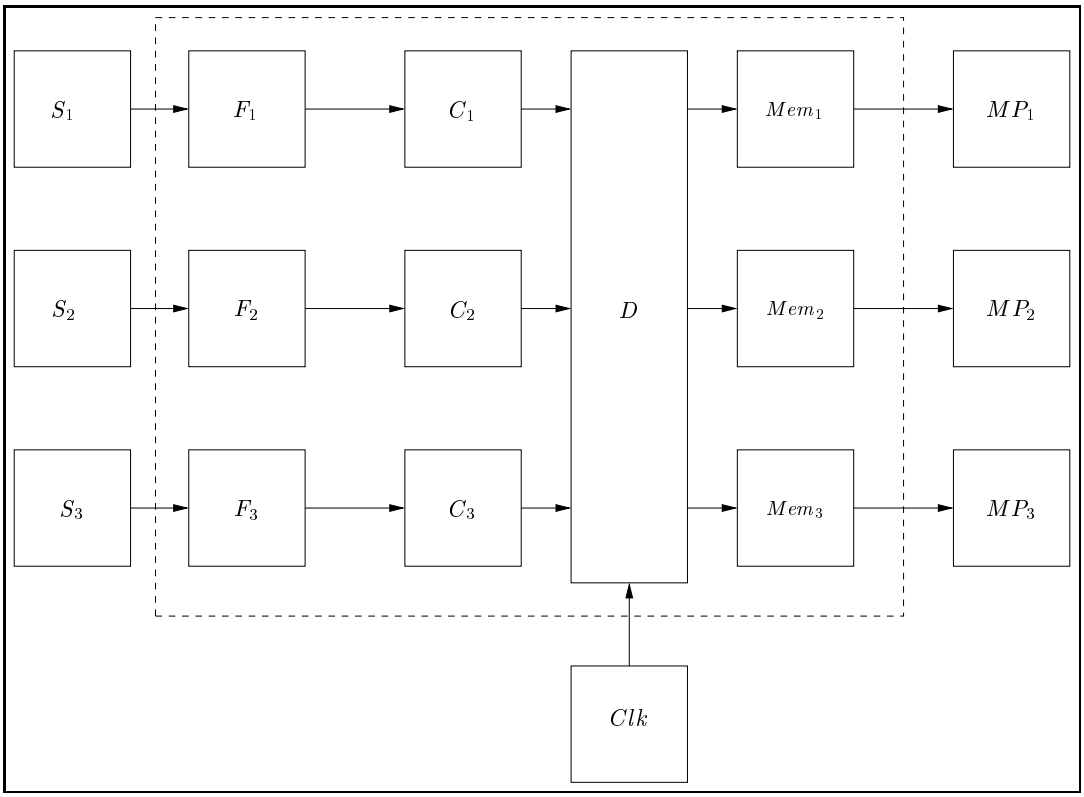

FIG. 9.7 – Modélisation de la carte TCBE au niveau carte.

Modèles fonctionnels des mémoires : le modèle fonctionnel de chaque mémoire est identique [6] au modèle fonctionnel de la mémoire Mem de la carte TCB représenté sur la figure 7.25.

Modèles fonctionnels des points de mesure : le modèle fonctionnel de chaque point de mesure est identique [7] au modèle fonctionnel du point de mesure MP de la carte TCB représenté sur la figure 7.20.

Modèles de test des filtres analogiques : le modèle de test de chaque filtre est identique [8] à celui du filtre F de la carte TCB représenté sur la figure 7.28.

Modèles de test des comparateurs : le modèle de test de chaque comparateur est identique [9] à celui du comparateur de la carte TCB représenté sur la figure 7.27.

[6] Il suffit de remplacer Adc par D, Mem par Mem_i et MP par MP_i avec $i = 1 \cdots 3$
[7] Il suffit de remplacer MP par MP_i et Mem par Mem_i avec $i = 1 \cdots 3$
[8] Il suffit de remplacer respectivement S par S_i et C par C_i avec $i = 1 \cdots 3$
[9] Il suffit de remplacer F par F_i avec $i = 1 \cdots 3$ et Adc par D

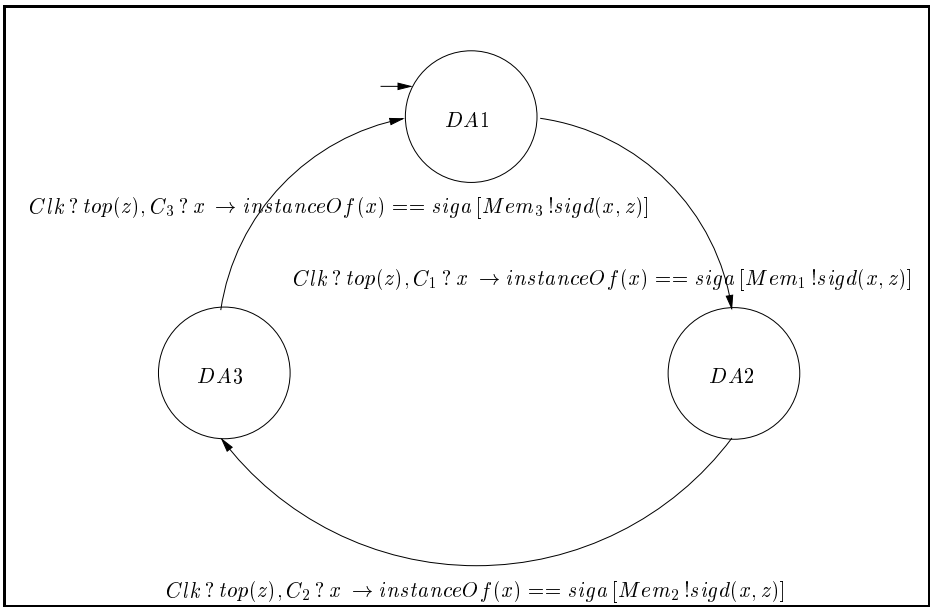

FIG. 9.8 – Modèle fonctionnel du contrôleur D

Modèle de test du contrôleur : le modèle de test du contrôleur est identique à son modèle fonctionnel (modèle de test par défaut).

Modèle de test des mémoires : le modèle de test de chaque mémoire est identique à son modèle fonctionnel (modèle de test par défaut).

Tactiques de test : deux tactiques de test distinctes ont été spécifiées pour la carte TCBE :

- la première tactique de test permet de rendre observables les réponses des filtres aux sorties primaires. Une telle tactique de test a déjà été présentée pour la carte TCB (voir figure 7.28 et les conditions (7.32) et (7.33)),
- la deuxième tactique de test permet de vérifier le fonctionnement avec l'hypothèse de synchronisation entre chaque voie analogique et le contrôleur. Une telle tactique de test a déjà été présentée pour la carte TCB (voir figure 7.29). La figure 9.9 montre les conditions ajoutées aux points de mesure afin que la synchronisation soit assurée.

9.2.3.2.3 Génération des données de test

Nous avons utilisé l'algorithme 4 décrit en section 8.3.1 et l'algorithme 5 décrit en section 8.3.1.1 afin de générer un jeu de données de test de la carte TCBE (stratégie de test globale utilisant la couverture des transitions des modèles de test). La donnée de test permettant de vérifier la synchronisation entre les voies analogiques et le contrôleur numérique a été générée en utilisant l'algorithme 9 décrit en section 8.3.2.1 (stratégie

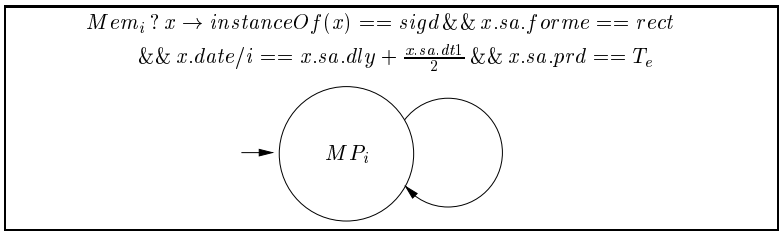

FIG. 9.9 – Tactique de test permettant le test de la synchronisation entre la voie analogique numéro i et le contrôleur

de test locale : test d'une transition). Les données de test générées et instanciées sont données ci-après. Elles ont été calculées avec les valeurs de paramètres suivantes :

- $S = 5\,V$ (seuil des comparateurs),
- $f_c = 1000\,Hz$ (fréquence de coupure des filtres),
- $T_e = 0.002\,s$ (période de l'horloge),
- $\delta_{11} = \delta_{12} = \delta_{21} = \delta_{22} = 0.1$ (tolérances du modèle de test des filtres),
- $\delta = 0.2$ (tolérance du modèle de test des comparateurs).

Le jeu de données de test obtenu pour la carte TCBE est le suivant :

$$\begin{aligned} TDS = \; & \{TDS_{filtre_1}, TDS_{filtre_2}, TDS_{filtre_3}, \\ & TDS_{comp_1}, TDS_{comp_2}, TDS_{comp_3}, \\ & TDS_{num}\} \end{aligned}$$

où

$$\begin{aligned} TDS_{filtre_i} &= \{TD1_{filtre_i}, TD2_{filtre_i}\} \\ TDS_{comp_i} &= \{TD1_{comp_i}, TD2_{comp_i}\} \\ TDS_{num} &= \{TD1_{num}\} \end{aligned}$$

avec $i = 1 \cdots 3$.

Une donnée de test (TD) est composée d'un quadruplet d'entrée et d'un triplet de sortie. Le quadruplet d'entrée est de la forme :

$$(S_1, S_2, S_3, Clk)$$

et le triplet de sortie est de la forme :

$$(MP_1, MP_2, MP_3)$$

L'élément S_i du quadruplet d'entrée de chaque donnée de test représente le signal analogique delivré par la source analogique S_i sur l'entrée primaire i. L'élément Clk représente la séquence de tops horodatés (entre crochets) envoyée par l'horloge Clk vers le contrôleur D (cf. le modèle fonctionnel de D représenté sur la figure 9.8). Cette séquence de tops permet d'échantillonner les sorties des comparateurs. L'élément MP_i du triplet de sortie de chaque donnée de test représente la valeur du signal numérique observé sur la sortie primaire i par le point de mesure MP_i à l'instant où la mesure est effective. Cet instant est précisé pour chaque donnée de test. Un élément est noté _ pour indiquer que le signal correspondant est quelconque.

Par souci de concision, nous nous limitons à décrire uniquement le jeu de test du filtre F_1 (TDS_{filtre_1}), du comparateur C_2 (TDS_{comp_2}) et de la partie numérique (TDS_{num}). Le jeu de test complet de la carte TCBE est donné en annexe E.

Le jeu de données de test TDS_{filtre_1} du filtre F_1 est composé des deux données de test suivantes :

$$\begin{array}{rll} TD1_{filtre_1} = & (In = (& siga(sinus, 5.5, 10000.0, 0.0), \_, \_, \\ & & [top(0.002)]), \\ & Out = (& sigd(siga(rect, 0.0000148, 0.0001, 0.0000159), 0.002), \_, \_)) \\ TD2_{filtre_1} = & (In = (& siga(sinus, 7.778, 1000.0, 0.0), \_, \_, \\ & & [top(0.002)]), \\ & Out = (& sigd(siga(rect, 0.000148, 0.001, 0.0000347), 0.002), \_, \_)) \end{array}$$

$TD1_{filtre_1}$ permet de tester le comportement du filtre F_1 dans sa bande passante et $TD2_{filtre_1}$ permet de tester son comportement à sa fréquence de coupure. L'instant de mesure est $t = 0.002s$. Ces données de test ont été générées en utilisant notre première tactique de test.

Le jeu de données de test TDS_{comp_2} du comparateur C_2 est composé des deux données de test suivantes :

$$\begin{array}{rll} TD1_{comp_2} = & (In = (& \_, siga(sinus, 10.762, 498.295, 0.0), \_, \\ & & [top(0.002), top(0.004)]), \\ & Out = (& \_, sigd(siga(DC, 0.0, \_, \_), 0.004), \_)) \\ TD2_{comp_2} = & (In = (& \_, siga(sinus, 14.158, 394.881, 0.0), \_, \\ & & [top(0.002), top(0.004)]), \\ & Out = (& \_, sigd(siga(rect, 0.000224, 0.00253, 0.0000394), 0.004), \_)) \end{array}$$

$TD1_{comp_2}$ permet de tester le comportement du filtre C_2 en dessous de son seuil d'entrée et $TD2_{comp_2}$ permet de tester son comportement au-dessus du seuil d'entrée. L'instant de mesure est $t = 0.004s$.

Le jeu de données de test TDS_{num} de la partie numérique de la carte TCBE (i.e. le contrôleur D et les mémoires Mem_i) est composé de l'unique donnée de test suivante :

$$\begin{array}{rll} TD1_{num} = & (In = (& siga(sinus, 12.298, 500.0, 5.819), \\ & & siga(sinus, 12.298, 500.0, 5.819), \\ & & siga(sinus, 12.298, 500.0, 5.819), \\ & & [top(0.002), top(0.004), top(0.006)]), \\ & Out = (& sigd(siga(rect, 0.000273, 0.002, 0.00186), 0.002), \\ & & sigd(siga(rect, 0.000273, 0.002, 0.00186), 0.004), \\ & & sigd(siga(rect, 0.000273, 0.002, 0.00186), 0.006))) \end{array}$$

$TD1_{num}$ permet de vérifier la synchronisation entre les voies analogiques et le contrôleur. Les instants de mesure sont $t = 0.002s$, $t = 0.004s$ et $t = 0.006s$. Cette donnée de

test a été générée en utilisant notre deuxième tactique de test. Elle donne les signaux analogiques délivrés par les trois sources analogiques aux entrées primaires, la séquence des trois tops horodatés fournie par l'horloge et les valeurs des signaux numériques aux sorties primaires de la carte pour les instants de mesure susmentionnés.

9.2.3.2.4 Modélisation de la carte TCBE avec Simulink

La modélisation hiérarchique de la carte TCBE est représentée sur les figures 9.10, 9.11, 9.12 et 9.13. La figure 9.10 représente le premier niveau de la modélisation où le rectangle central représente la carte TCBE. Les trois entrées primaires de la carte ($PI1$, $PI2$ et $PI3$) sont connectées chacune à un générateur de tension sinusoïdale. Les sorties primaires de la carte ($PO1$, $PO2$ et $PO3$) sont connectées respectivement aux blocs $po1$, $po2$ et $po3$ qui modélisent chacun un point de mesure pour les données écrites dans chaque mémoire de la carte. Ces données (et leur date d'acquisition) sont écrites dans l'espace de travail de MATLAB. Les oscilloscopes connectés aux sorties internes de la carte permettent d'observer les signaux internes de la carte TCBE. La valeur du seuil de chaque comparateur (5V sur la figure 9.10) peut être modifiée à ce niveau de modélisation.

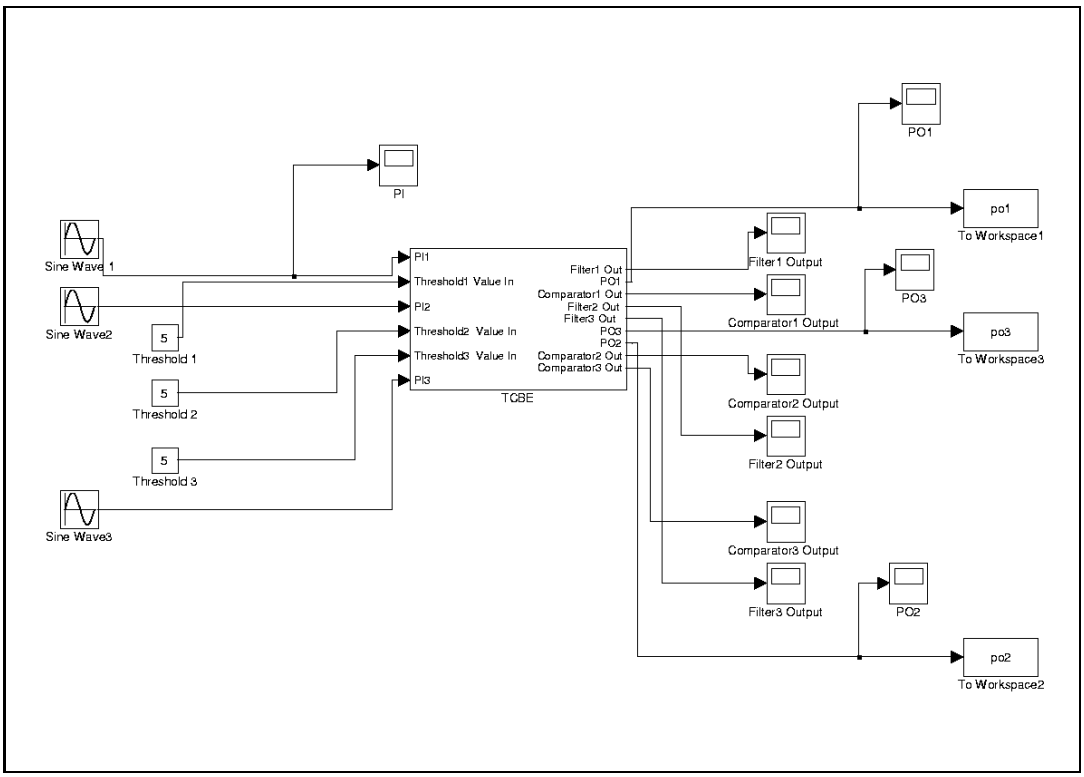

FIG. 9.10 – Modélisation de la carte TCBE avec Simulink : premier niveau

La figure 9.11 montre le deuxième niveau de modélisation. À ce niveau, la carte

est décomposée en une partie analogique et une partie numérique (*digital part*). La partie analogique est formée de trois voies (*analog channel*1, *analog channel*2 et *analog channel*3).

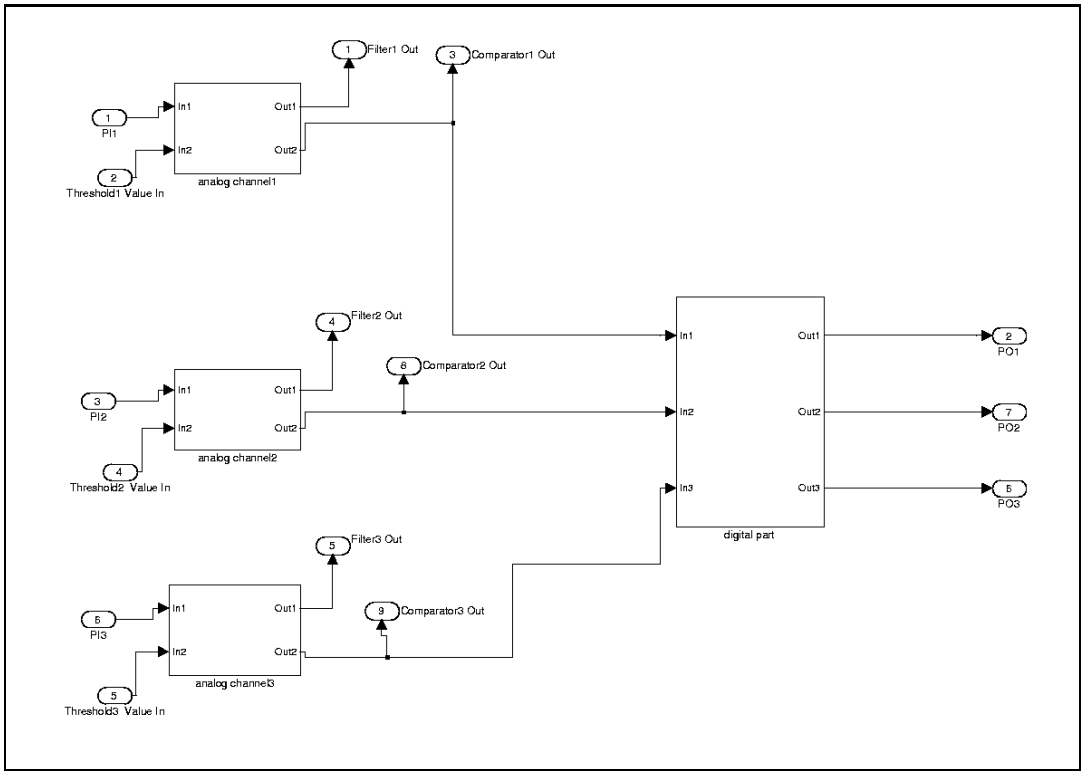

FIG. 9.11 – Modélisation de la carte TCBE avec Simulink : deuxième niveau

La figure 9.12 représente la modélisation d'une voie analogique en utilisant les diagramme blocs Simulink. Une voie analogique est constituée d'un filtre et d'un comparateur. Le filtre et le comparateur sont modélisés de façon similaire à ceux de la carte TCB.

La figure 9.13 représente la modélisation de la partie numérique de la carte TCBE en utilisant les diagramme blocs Simulink. Cette partie est composée du contrôleur et de trois mémoires :

- le contrôleur est modélisé par un bloc personnalisé (controller_SP) dont le comportement (balayage périodique des voies) a été décrit en utilisant l'API des S-Fonctions,
- chaque mémoire est modélisée de façon similaire à celle de la carte TCB.

9.2.3.2.5 Simulation

Nous avons simulé avec Simulink le comportement de la carte TCBE sur les données de test présentées en section 9.2.3.2.3. La simulation a été effectuée avec le solveur

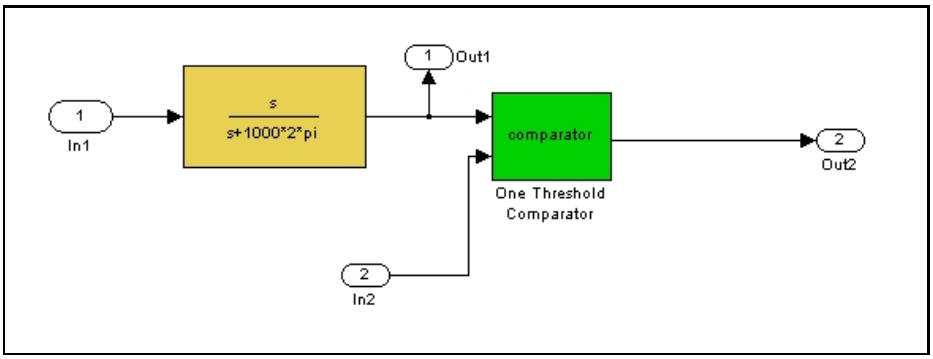

FIG. 9.12 – Modélisation d'une voie analogique de la carte TCBE avec Simulink : troisième niveau

MATLAB ode45 (Dormand-Prince), de l'instant 0 à l'instant 0.01s.

Nous examinons plus en détail la simulation avec la donnée de test $TD1_{num}$ qui permet de vérifier la synchronisation entre les voies analogiques et le contrôleur. Le signal obtenu par simulation en sortie de chaque comparateur est le même que celui obtenu en sortie du comparateur de la carte TCB représenté sur la figure 9.5. La figure 9.14 montre le signal numérique présent à chaque sortie primaire de la carte. Nous pouvons remarquer que les mémoires numéro 1, 2 et 3 prennent la valeur un respectivement aux instants $t_1 = 0.002s$, $t_2 = 0.004s$ et $t_3 = 0.006s$ et qu'elles gardent ensuite cette valeur jusqu'à la fin de la simulation. Chaque mémoire vaut zéro à l'instant $t = 0$ car c'est la valeur par défaut qu'elle prend lors de l'initialisation de la carte.

9.2.3.2.6 Bilan

Les résultats de la simulation sont en accord avec les sorties prédites par les données de test. Ces résultats nous font aboutir aux mêmes conclusions que celles énoncées en section 9.2.3.1.6 pour la carte TCB. L'utilisation de l'horloge est bien adaptée pour modéliser les fonctionnalités temporelles plus complexes de la carte TCBE. La couverture des modèles de test et l'utilisation des tactiques de test permettent d'obtenir un jeu de test pour la carte TCBE qui permet de vérifier son bon fonctionnement. On peut également noter le caractère « générique » de l'approche proposée et des formalismes sous-jacents qui facilitent la généralisation des solutions (passage d'un composant à plusieurs composants de même nature, $\cdots$).

9.2.3.3 La carte CS1

Dans cette section, nous présentons d'abord la carte CS1 ainsi que sa modélisation et la génération des données de test à l'aide de notre méthodologie. Nous présentons ensuite la modélisation de la carte CS1 avec Simulink. Nous abordons enfin la simulation du comportement de la carte avec ces données de test et les résultats que nous avons obtenus.

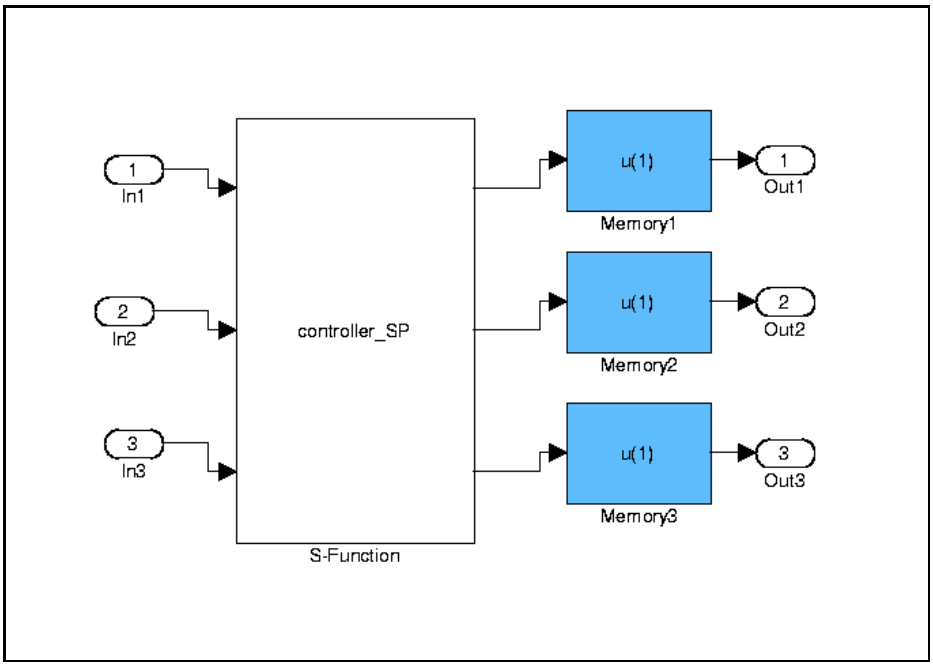

FIG. 9.13 – Modélisation de la partie numérique de la carte TCBE avec Simulink : troisième niveau

9.2.3.3.1 Description de la carte

La carte CS1 implante une chaîne d'acquisition numérique simple avec une restitution analogique. Le traitement numérique est réalisé par un filtre à réponse impulsionnelle finie d'ordre 2. Les composants analogiques et numériques de cette carte sont entrelacés. L'entrelacement de composants de nature différente apparaît souvent dans les cartes mixtes. Aussi est-il intéressant d'évaluer notre méthodologie sur la carte CS1.

9.2.3.3.2 Modélisation de la carte CS1 avec notre méthodologie

Nous présentons dans un premier temps la modélisation au niveau carte de la carte CS1. Nous présentons ensuite les modèles fonctionnels et les modèles de test des blocs de la carte et les tactiques de test.

La figure 9.15 montre la modélisation de la carte CS1 au niveau carte. Sur cette figure, le périmètre de la carte est délimité par le rectangle en pointillés et les blocs se trouvant dans ce périmètre sont des blocs fonctionnels. La carte est composée de deux blocs analogiques AAF et SF, de deux blocs mixtes ADC et DAC, et d'un bloc numérique FIR :

- le bloc AAF représente un filtre analogique passe-bas anti-repliement (*Anti-Aliasing Filter*) du premier ordre,
- le bloc SF représente un filtre analogique passe-bas de lissage (*Smoothing Filter*)

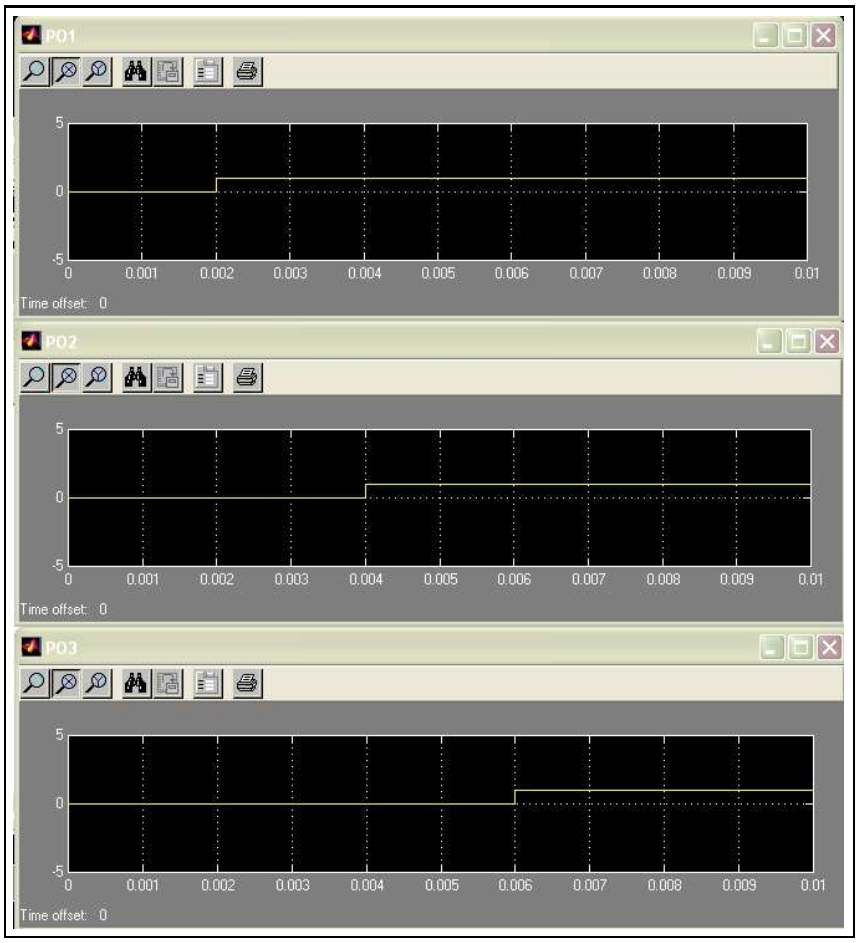

FIG. 9.14 – Signaux présents aux sorties primaires de la carte TCBE

du premier ordre,
- le bloc ADC représente un convertisseur analogique-numérique (*Analog-to-Digital Converter*),
- le bloc DAC représente un convertisseur numérique-analogique (*Digital-to-Analog Converter*),
- le bloc FIR représente un filtre à réponse impulsionnelle finie (*Finite Impulse Response*) d'ordre 2.

Les blocs externes à la carte sont les suivants :
- le bloc d'entrée S qui représente une source de tension analogique,
- le bloc de sortie MP qui représente un point de mesure (Measurement Point),
- le bloc CLK qui représente une horloge.

Nous présentons ci-après les modèles fonctionnels et les modèles de test des blocs susmentionnés.

Modèle fonctionnel de la source : la figure 9.16 montre le modèle fonctionnel de la

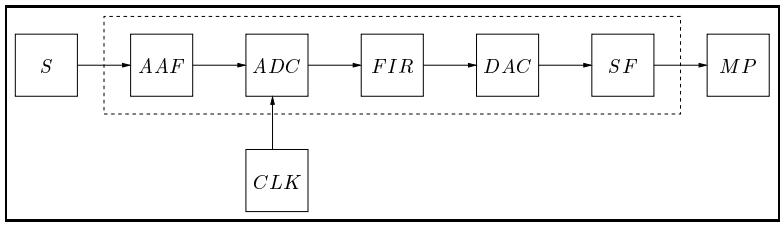

FIG. 9.15 – Modélisation de la carte CS1 au niveau carte

source S. Le CFSM S envoie le signal analogique x vers le CFSM AAF (la condition $instanceOf(x) == siga$ signifie que le signal x doit être un signal analogique).

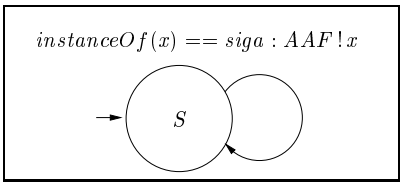

FIG. 9.16 – Modèle fonctionnel de la source S

Modèle fonctionnel du point de mesure : la figure 9.17 montre le modèle fonctionnel du point de mesure MP. Le CFSM MP attend le signal x émis par le CFSM SF.

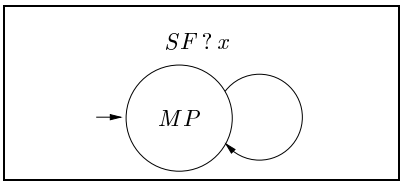

FIG. 9.17 – Modèle fonctionnel du point de mesure MP

Modèle fonctionnel de l'horloge : la figure 9.18 montre le modèle fonctionnel de l'horloge CLK. Le CFSM CLK envoie des tops périodiques aux instants y vers le CFSM ADC. La période entre les tops vaut T_e et représente la période d'échantillonnage de la chaîne d'acquisition.

Modèle fonctionnel du filtre anti-repliement : le modèle fonctionnel du filtre analogique passe-bas anti-repliement AAF est représenté par la figure 9.19. Le CFSM AAF décrit le comportement du filtre lorsque le signal appliqué à son entrée est un signal analogique sinusoïdal. Ainsi, le CFSM AAF attend la réception du signal x (analogique et sinusoïdal) émis par le CFSM S et envoie un signal analogique sinusoïdal vers le CFSM ADC en vérifiant la condition Γ_1 qui exprime les caractéristiques du signal de sortie calculées dans le domaine fréquentiel (atténuation, fréquence et déphasage). La

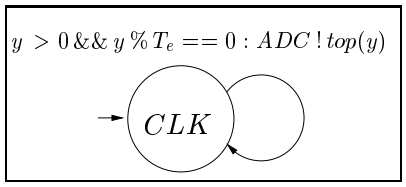

FIG. 9.18 – Modèle fonctionnel de l'horloge CLK

condition $\Gamma_1 = \gamma_{11}$ && γ_{12} && γ_{13} est définie par :

$$\Gamma_1 = \begin{cases} \gamma_{11} : \left(V_0 == \frac{x.ampl}{\sqrt{1+\frac{F_0^2}{f_{c_{AAF}}^2}}} \right) \\ \gamma_{12} : (F_0 == x.frq) \\ \gamma_{13} : \left(\phi_0 == x.phi + \arctan\left(\frac{F_0}{f_{c_{AAF}}}\right) \right) \end{cases} \tag{9.1}$$

où $f_{c_{AAF}}$ est la fréquence de coupure du filtre.

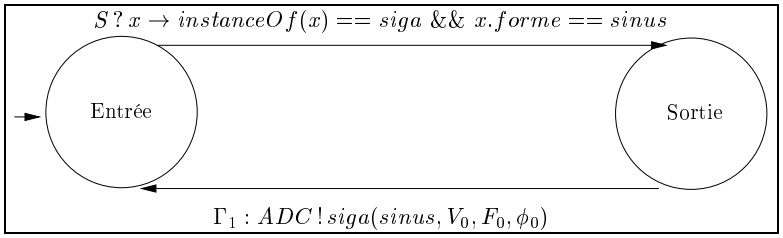

FIG. 9.19 – Modèle fonctionnel du filtre anti-repliement AAF

Modèle fonctionnel du filtre de lissage : la figure 9.20 montre le modèle fonctionnel du filtre analogique passe-bas de lissage SF. Le CFSM SF décrit le comportement du filtre lorsque le signal appliqué à son entrée est un signal analogique sinusoïdal. Ainsi, le CFSM SF attend la réception du signal x (analogique et sinusoïdal) émis par le CFSM DAC et envoie un signal analogique sinusoïdal vers le CFSM MP en vérifiant la condition Γ_2 qui exprime les caractéristiques du signal de sortie calculées dans le domaine fréquentiel. La condition $\Gamma_2 = \gamma_{21}$ && γ_{22} && γ_{23} est définie par :

$$\Gamma_2 = \begin{cases} \gamma_{21} : \left(V_0 == \frac{x.ampl}{\sqrt{1+\frac{F_0^2}{f_{c_{SF}}^2}}} \right) \\ \gamma_{22} : (F_0 == x.frq) \\ \gamma_{23} : \left(\phi_0 == x.phi + \arctan\left(\frac{F_0}{f_{c_{SF}}}\right) \right) \end{cases} \tag{9.2}$$

où $f_{c_{SF}}$ est la fréquence de coupure du filtre. Notons que la modélisation du filtre SF est similaire à celle du filtre AAF car tous les deux sont des filtres analogiques passe-bas

du premier ordre.

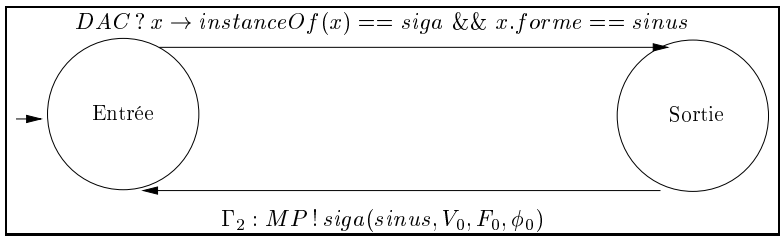

FIG. 9.20 – Modèle fonctionnel du filtre de lissage SF

Modèle fonctionnel du convertisseur analogique-numérique : la figure 9.21 montre le modèle fonctionnel du convertisseur analogique-numérique ADC. Le CFSM ADC décrit le comportement du convertisseur. Ainsi, le CFSM ADC attend un top d'horloge horodaté à la date z émis par le CFSM CLK. Lorsque le top est reçu, le CFSM ADC attend la réception du signal analogique x envoyé par le CFSM AAF. Puis, lorsque ce signal est reçu, il envoie la valeur du signal numérique (signal analogique numérisé) à la date z vers le CFSM FIR.

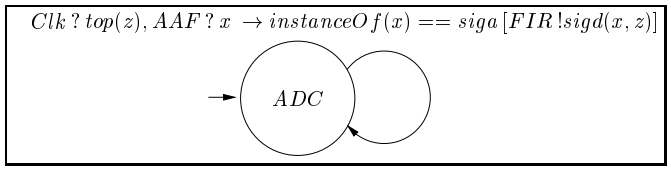

FIG. 9.21 – Modèle fonctionnel du convertisseur analogique-numérique ADC

Modèle fonctionnel du convertisseur numérique-analogique : la figure 9.22 montre le modèle fonctionnel du convertisseur numérique-analogique DAC. Le CFSM DAC décrit de manière très simplifiée le comportement du convertisseur. Ainsi, le CFSM DAC attend la réception du signal numérique x traité et envoyé par le CFSM FIR et envoie le signal analogique parfaitement interpolé (restitué) vers le CFSM SF.

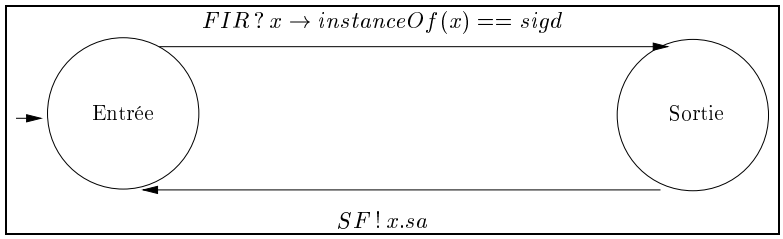

FIG. 9.22 – Modèle fonctionnel du convertisseur numérique-analogique DAC

Modèle fonctionnel du filtre numérique : le modèle fonctionnel du filtre numérique

FIR est représenté sur la figure 9.23. Le CFSM FIR décrit le comportement du filtre lorsque le signal numérique appliqué à son entrée est de forme sinusoïdale. Ainsi, le CFSM FIR attend la réception du signal numérique sinusoïdal x émis par le CFSM ADC puis envoie le signal numérique sinusoïdal filtré vers le CFSM DAC, en vérifiant la condition Γ_3 qui exprime les caractéristiques du signal de sortie calculées dans le domaine fréquentiel.

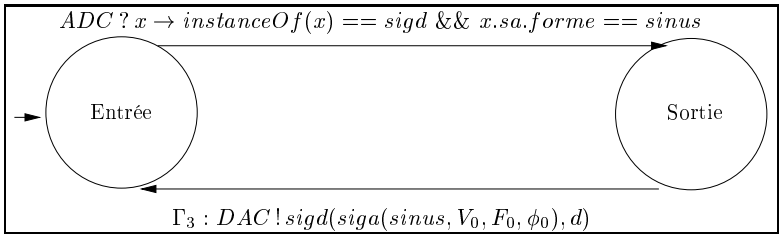

FIG. 9.23 – Modèle fonctionnel du filtre FIR

La condition Γ_3 est calculée à partir de l'équation aux différences du filtre. L'équation aux différences que nous considérons est donnée par :

$$y(n) = x(n) + x(n-1) + x(n-2) \tag{9.3}$$

Cette équation (9.3) montre que l'échantillon de sortie courant (présent) $y(n)$ du filtre ne dépend que de l'échantillon d'entrée courant $x(n)$ et des deux échantillons précédents $x(n-1)$ et $x(n-2)$. La période d'échantillonnage est τ.

La transformée en z de l'équation aux différences (9.3) permet d'obtenir la transmittance en z du filtre. Ainsi, la transmittance est donnée par l'expression :

$$H(z) = 1 + z^{-1} + z^{-2} \tag{9.4}$$

où z^{-1} représente l'opérateur de retard unitaire.

La fonction de transfert du filtre exprimée dans le domaine fréquentiel est alors obtenue en posant

$$z = e^{j\omega\tau}$$

où ω est la pulsation et τ la période d'échantillonnage. Ainsi, la fonction de transfert H du filtre est donnée par l'expression :

$$H(\omega) = H(e^{j\omega\tau}) = 1 + e^{-j\omega\tau} + e^{-2j\omega\tau} \tag{9.5}$$

Finalement, l'expression (9.5) peut être réécrite comme :

$$H(\omega) = H_0(w)\, e^{-j\Theta(\omega)} \tag{9.6}$$

où $H_0(\omega)$ et $\Theta(\omega)$ représentent respectivement le spectre d'amplitude (gain) et le spectre de phase du filtre. Ils sont définis par les expressions suivantes :

$$H_0(w) = 1 + 2\cos(\omega\tau) \tag{9.7}$$

$$\Theta(\omega) = \omega\tau \tag{9.8}$$

Les expressions (9.7) et (9.8) permettent alors de déduire directement les sous-conditions γ_{31} (atténuation) et γ_{33} (déphasage) de la condition Γ_3. $\Gamma_3 = \gamma_{31}$ && γ_{32} && γ_{33} && γ_{34} est finalement définie par :

$$\Gamma_3 = \begin{cases} \gamma_{31} : (V_0 == x.ampl\,(1 + 2\cos(2\pi F_0 T_e)) \\ \gamma_{32} : (F_0 == x.frq) \\ \gamma_{33} : (\phi_0 == x.phi + 2\pi F_0 T_e) \\ \gamma_{34} : (d == x.date) \end{cases} \tag{9.9}$$

Après avoir présenté les modèles fonctionnels de chacun des blocs de la carte, nous décrivons maintenant les modèles de test et les tactiques de test que nous proposons.

Modèle de test d'un filtre analogique passe-bas du premier ordre : la figure 9.24 montre le modèle de test générique d'un filtre analogique passe-bas du premier ordre (à instancier pour les filtres AAF et SF). Ce modèle de test définit deux données de test (une dans la bande passante et une à la fréquence de coupure f_c du filtre). Les conditions $\Gamma_1 = \gamma_{11}$ && γ_{12} && γ_{13} && γ_{14} && γ_{15} et $\Gamma_2 = \gamma_{21}$ && γ_{22} && γ_{23} && γ_{24} && γ_{25} décrivant les caractéristiques des deux signaux de sortie possibles sont respectivement définies par :

$$\Gamma_1 = \begin{cases} \gamma_{11} = (F_0 == x.frq) \\ \gamma_{12} = (V_0 \leq x.ampl + \delta_{11}) \\ \gamma_{13} = (V_0 \geq x.ampl - \delta_{11}) \\ \gamma_{14} = (\phi_0 \leq 0 + \delta_{12}) \\ \gamma_{15} = (\phi_0 \geq 0 - \delta_{12}) \end{cases} \tag{9.10}$$

et

$$\Gamma_2 = \begin{cases} \gamma_{21} = (F_0 == x.frq) \\ \gamma_{22} = (V_0 \leq x.ampl\frac{\sqrt{2}}{2} + \delta_{21}) \\ \gamma_{23} = (V_0 \geq x.ampl\frac{\sqrt{2}}{2} - \delta_{21}) \\ \gamma_{24} = (\phi_0 \leq +\frac{\pi}{4} + \delta_{22}) \\ \gamma_{25} = (\phi_0 \geq +\frac{\pi}{4} - \delta_{22}) \end{cases} \tag{9.11}$$

où δ_{11}, δ_{12}, δ_{21} et δ_{22} définissent des intervalles de tolérance. Ce modèle de test constitue le modèle de test des filtres analogiques passe-bas du premier ordre AAF et SF. Pour instancier le modèle de test du filtre AAF, il suffit de remplacer le CFSM *Amont* par le CFSM S et le CFSM *Aval* par le CFSM ADC. De même, pour instancier le modèle de test du filtre SF, on remplace le CFSM *Amont* par le CFSM DAC et le CFSM *Aval* par le CFSM MP.

Modèle de test du convertisseur analogique-numérique : pour des raisons de simplification, le modèle de test du convertisseur analogique-numérique ADC est identique à son modèle fonctionnel (modèle de test par défaut).

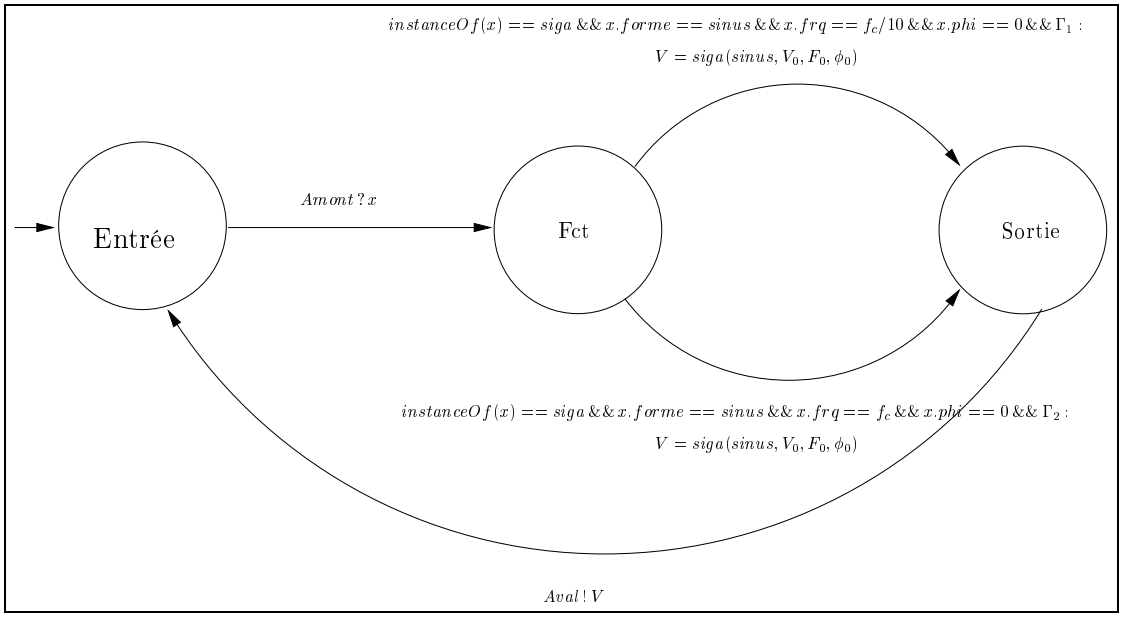

FIG. 9.24 – Modèle de test d'un filtre analogique passe-bas du premier ordre

Modèle de test du convertisseur numérique-analogique : pour des raisons de simplification, le modèle de test du convertisseur numérique-analogique DAC est identique à son modèle fonctionnel (modèle de test par défaut).

Modèle de test du filtre numérique : l'expression (9.7) donnant le gain du filtre FIR montre que celui-ci est nul pour une pulsation particulière ω_c telle que :

$$\cos(\omega_c \tau) = -\frac{1}{2} \tag{9.12}$$

En résolvant l'équation (9.12), on trouve la fréquence f_c éliminée par le filtre qui vaut :

$$f_c = \frac{f_e}{3} \tag{9.13}$$

où $f_e = \frac{1}{\tau}$ est la fréquence d'échantillonnage du signal numérique d'entrée et f_c est la fréquence de coupure du filtre (il est à noter que $f_c = \frac{\omega_c}{2\pi}$). Ainsi, nous proposons un modèle de test pour le filtre numérique qui permet de tester l'élimination d'une fréquence particulière qui est le tiers de la fréquence d'échantillonnage du signal numérique appliqué à son entrée.

Un premier modèle de test évident pour le filtre FIR est celui représenté sur la figure 9.25. Le CFSM FIR est en attente de réception d'un signal numérique sinusoïdal (en provenance du CFSM ADC) dont la fréquence est égale au tiers de la fréquence d'échantillonnage. Lorsque ce signal est reçu, le CFSM FIR envoie alors un signal numérique de type DC au CFSM DAC en vérifiant la condition $\Gamma_3 = \gamma_{31}$ && γ_{32} définie par :

$$\Gamma_3 = \begin{cases} \gamma_{31} : (V_0 == 0) \\ \gamma_{32} : (d == x.date) \end{cases} \tag{9.14}$$

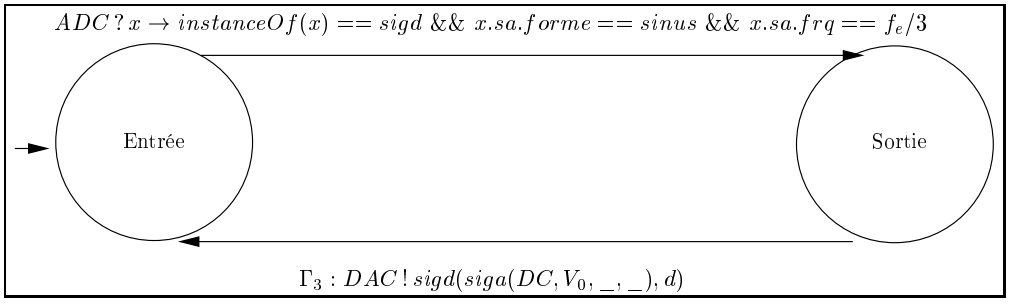

FIG. 9.25 – Premier modèle de test du filtre FIR

La condition γ_{31} impose que tous les échantillons du signal numérique de sortie soient nuls. La condition γ_{32} date l'échantillon d'entrée courant et l'échantillon de sortie courant au même instant. Cependant, il est important de remarquer que le modèle de test proposé ne permet pas la propagation de la séquence de test à travers le modèle fonctionnel du CFSM SF. En effet, la séquence de test pourra être propagée à travers le CFSM DAC (qui traite n'importe quel type de signal numérique d'entrée), mais pas à travers le CFSM SF qui ne traite que des signaux d'entrée analogiques sinusoïdaux. Afin de contourner ce problème, nous proposons un nouveau modèle de test du filtre numérique représenté sur la figure 9.26.

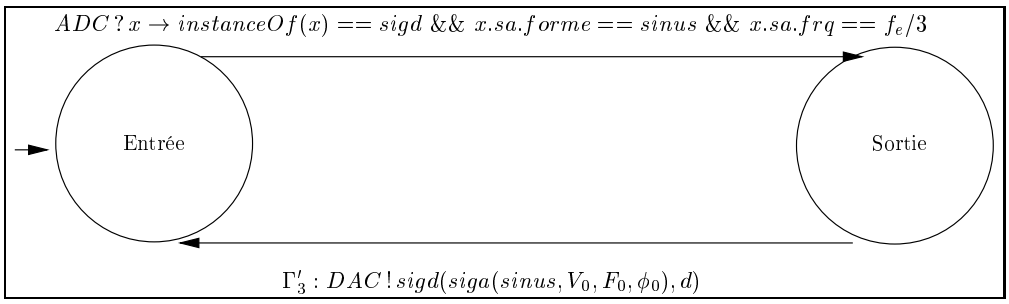

FIG. 9.26 – Deuxième modèle de test du filtre FIR

Maintenant, le signal de sortie est un signal sinusoïdal vérifiant la condition Γ'_3 définie par :

$$\Gamma'_3 = \begin{cases} \gamma'_{31} : (V_0 == 0) \\ \gamma'_{32} : (F_0 == x.frq) \\ \gamma'_{33} : (d == x.date) \end{cases} \tag{9.15}$$

qui exprime le signal numérique nul comme un signal sinusoïdal (forme nécessaire pour la propagation) d'amplitude nulle. Ce nouveau modèle de test permet alors la

propagation en aval qui était impossible avec le premier modèle de test.

Les tactiques de test : nous décrivons dans cette section une tactique de test qui consiste à ajouter des conditions sur le signal reçu au point de mesure. La figure 9.27 montre le modèle fonctionnel du point de mesure, enrichi de la tactique de test. Les conditions ajoutées dans cet exemple caractérisent entièrement le signal analogique (amplitude, fréquence et phase) présent à la sortie primaire de la carte CS1. La vérification de ces conditions et la propagation en amont vers les entrées primaires de la carte permettent de calculer le stimulus de test à appliquer à la carte afin d'obtenir le signal de sortie désiré (dans notre exemple, un signal sinusoïdal d'amplitude maximale égale à 15 V, de fréquence égale à 100 Hz et de phase égale à $\frac{\pi}{2}$). Cette tactique de test est donnée ici juste à titre d'exemple et n'est pas utilisée par le processus de génération des données de test de la carte CS1 décrit en section 9.2.3.3.3.

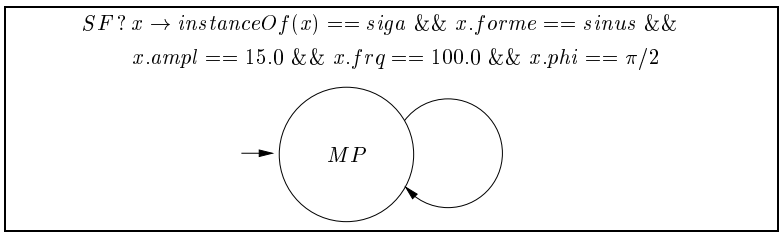

FIG. 9.27 – Tactique de test exprimée au point de mesure MP

9.2.3.3.3 Génération des données de test

Nous avons utilisé l'algorithme 4 décrit en section 8.3.1 et l'algorithme 5 décrit en section 8.3.1.1 afin de générer un jeu de données de test pour la carte CS1 (stratégie de test globale utilisant la couverture des transitions des modèles de test). Les données de test générées et instanciées sont données ci-après. Elles ont été calculées avec les valeurs de paramètres suivantes :

- $f_{c_{AAF}} = 1000\,Hz$ (fréquence de coupure du filtre anti-repliement),
- $f_{c_{SF}} = 1000\,Hz$ (fréquence de coupure du filtre de lissage),
- $T_e = 0.000454\,s$ (période de l'horloge correspondant à une fréquence d'échantillonnage de $2200\,Hz$),
- $\delta_{11} = \delta_{12} = \delta_{21} = \delta_{22} = 0.1$ (tolérances du modèle de test des deux filtres).

Le jeu de données de test obtenu pour la carte CS1 est le suivant :

$$TDS = \{TDS_{AAF}, TDS_{SF}, TDS_{FIR}\}$$

où

$$\begin{aligned} TDS_{AAF} &= \{TD1_{AAF}, TD2_{AAF}\} \\ TDS_{SF} &= \{TD1_{SF}, TD2_{SF}\} \\ TDS_{FIR} &= \{TD1_{FIR}\} \end{aligned}$$

Une donnée de test (TD) est composée d'un couple d'entrée et d'un singleton de sortie. Le couple d'entrée est de la forme (S, Clk) et le singleton de sortie est de la forme (MP) où S représente le signal de la source analogique, Clk celui de l'horloge et MP celui du point de mesure analogique.

Le jeu de données de test du filtre analogique anti-repliement est composé des deux données de test suivantes :

$$\begin{aligned} TD1_{AAF} = \quad & (In = (siga(sinus, 3.0, 100.0, 0.0), top(0.000454)), \\ & Out = siga(sinus, 8.710, 100.0, 0.385)) \end{aligned}$$

$$\begin{aligned} TD2_{AAF} = \quad & (In = (siga(sinus, 3.0, 1000.0, 0.0), top(0.000454)), \\ & Out = siga(sinus, 1.378, 1000.0, 1.288)) \end{aligned}$$

Le jeu de données de test du filtre analogique de lissage est composé des deux données de test suivantes :

$$\begin{aligned} TD1_{SF} = \quad & (In = (siga(sinus, 3.0, 100.0, -0.385), top(0.000454)), \\ & Out = siga(sinus, 8.713, 100.0, 0.0)) \end{aligned}$$

$$\begin{aligned} TD2_{SF} = \quad & (In = (siga(sinus, 3.0, 1000.0, -0.503), top(0.000454)), \\ & Out = siga(sinus, 1.378, 1000.0, 0.785)) \end{aligned}$$

Le jeu de données de test du filtre numérique est composé de la donnée de test suivante :

$$\begin{aligned} TD1_{FIR} = \quad & (In = (siga(sinus, 3.0, 733.33, 0.0), top(0.000454)), \\ & Out = siga(sinus, 0.0, 733.333, 1.265)) \end{aligned}$$

Le jeu de données de test TDS couvre les modèles fonctionnels du convertisseur analogique-numérique ADC et du convertisseur numérique-analogique DAC. Les modèles de test de ADC et de DAC étant identiques à leurs modèles fonctionnels, TDS permet de tester ADC et DAC.

9.2.3.3.4 Modélisation de la carte CS1 avec Simulink

La modélisation hiérarchique de la carte $CS1$ effectuée avec Simulink est représentée sur les figures 9.28 et 9.29. La figure 9.28 représente le premier niveau de modélisation où le rectangle central représente la carte CS1. L'entrée primaire de la carte (PI) est connectée à un générateur de tension sinusoïdale. La sortie primaire de la carte ainsi que les sorties internes sont reliées à des oscilloscopes, permettant ainsi l'observation des signaux présents sur ces sorties.

La figure 9.29 montre le deuxième niveau de modélisation. A ce niveau, la carte CS1 est modélisée par un ensemble de blocs simulink :

- le filtre anti-repliement est modélisé par le bloc prédéfini *fonction de transfert*. Ce bloc implante la fonction de transfert d'un filtre analogique passe-bas du premier ordre défini par l'expression :

$$H(s) = \frac{1000.\ 2\pi}{s + 1000\,.\,2\pi}$$

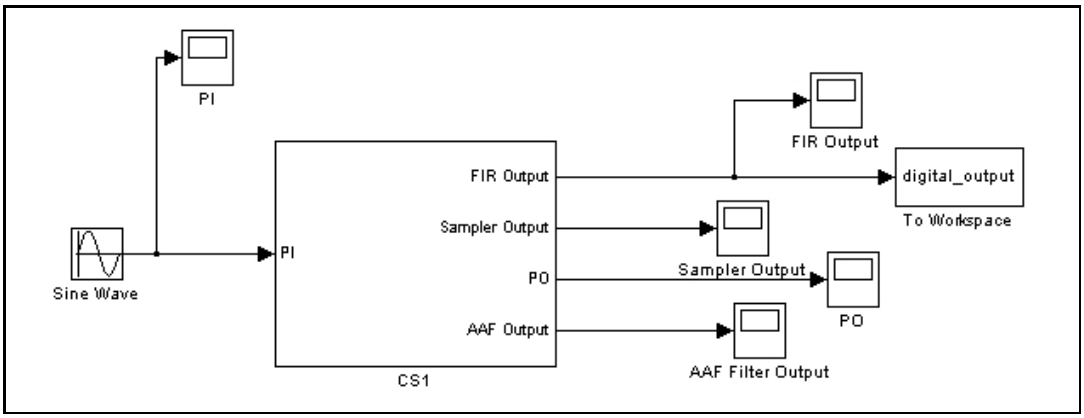

FIG. 9.28 – Modélisation de la carte CS1 avec Simulink : premier niveau

où 1000 (Hz) est la valeur de la fréquence de coupure du filtre, avec $s = j\omega$, où j est le nombre imaginaire de module unité et ω la pulsation,

- le convertisseur analogique-numérique est modélisé par le bloc prédéfini *échantillonneur bloqueur d'ordre zéro* avec une fréquence d'échantillonnage de 2200 Hz,
- le filtre numérique est modélisé par le bloc prédéfini *transmittance*. Ce bloc implante la transmittance en Z du filtre défini par l'expression :

$$H(z) = 1 + z^{-1} + z^{-2}$$

- le filtre de lissage est modélisé de la même manière que le filtre anti-repliement (les deux filtres sont de même type et possèdent la même fréquence de coupure),
- le convertisseur numérique-analogique qui se trouve entre le filtre numérique et le filtre de lissage est modélisé implicitement par Simulink.

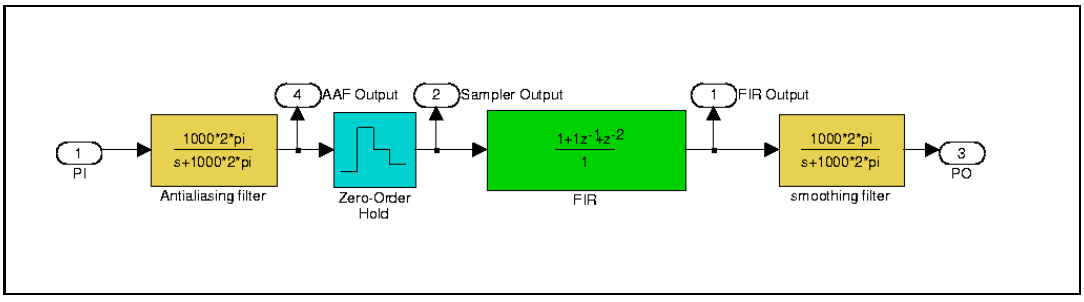

FIG. 9.29 – Modélisation de la carte CS1 avec Simulink : deuxième niveau

9.2.3.3.5 Simulation

Nous avons simulé avec Simulink le comportement de la carte CS1 sur les données de test présentées en section 9.2.3.3.3. La simulation a été effectuée avec le solveur MATLAB ode45 (Dormand-Prince), de l'instant 0 à l'instant 0.05s.

Les figures 9.30 et 9.31 montrent respectivement les signaux analogiques d'entrée et de sortie de la carte CS1 lorsque son comportement est simulé en utilisant la donnée de test $TD1_{AAF}$.

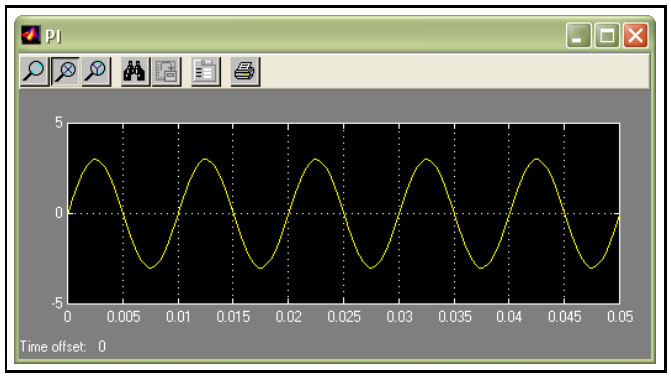

FIG. 9.30 – Signal analogique appliqué à l'entrée primaire pour la donnée de test $TD1_{AAF}$

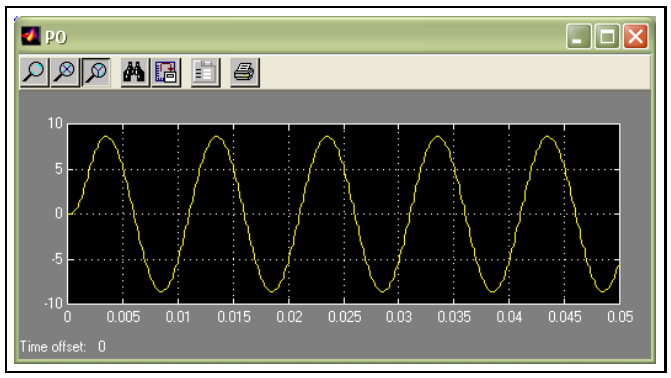

FIG. 9.31 – Signal analogique observé à la sortie primaire pour la donnée de test $TD1_{AAF}$

Les caractéristiques (amplitude, fréquence et phase) du signal analogique sinusoïdal de sortie mesurées sont cohérentes avec celles calculées par nos algorithmes de génération de données de test. Nous pouvons cependant constater que l'effet de lissage produit par la simulation n'est pas pris en compte dans notre modélisation de la carte CS1. Ceci est dû au modèle fonctionnel du convertisseur numérique-analogique (cf. figure 9.22). En effet, ce modèle fait l'hypothèse d'une interpolation parfaite. Il restitue ainsi un signal sinusoïdal parfait. Néanmoins, les signaux de sortie calculés et simulés restent très proches car, dans la simulation, le filtre de lissage lisse assez efficacement le signal de sortie. La raison en est que la fréquence du signal d'entrée (100 Hz) est petite devant la fréquence de Shannon (égale à la moitié de la fréquence d'échantillonnage, soit 1100 Hz). L'échantillonneur-bloqueur préserve alors la forme du signal analogique sinusoïdal appliqué à son entrée, comme le montre la figure 9.32. La même analyse s'applique à la donnée de test $TD1_{SF}$.

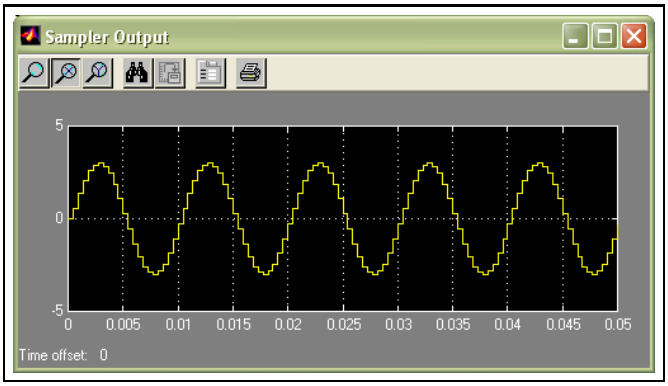

FIG. 9.32 – Signal de sortie de l'échantillonneur-bloqueur pour la donnée de test $TD1_{AAF}$

Les figures 9.33 et 9.34 montrent respectivement les signaux analogiques d'entrée et de sortie de la carte CS1 lorsque son comportement est simulé en utilisant la donnée de test $TD2_{AAF}$.

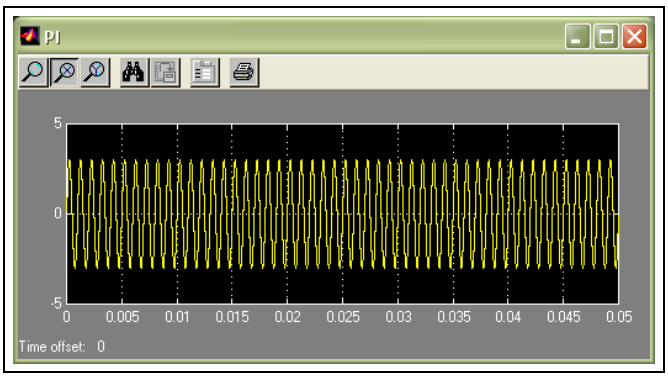

FIG. 9.33 – Signal analogique appliqué à l'entrée primaire pour la donnée de test $TD2_{AAF}$

Nous remarquons que le signal de sortie obtenu par simulation est très différent du signal calculé par les algorithmes de génération de données de test. Comme pour $TD1_{AAF}$, le modèle fonctionnel du convertisseur numérique-analogique produit un signal de sortie sinusoïdal. Mais, contrairement à $TD1_{AAF}$, la fréquence du signal sinusoïdal d'entrée de $TD2_{AAF}$ (1000 Hz) devient proche de la fréquence de Shannon. L'échantillonneur-bloqueur ne préserve alors pas la forme du signal analogique sinusoïdal appliqué à son entrée, comme le montre la figure 9.35 et le filtre de lissage utilisé n'est pas capable d'interpoler correctement le signal de sortie. La même analyse s'applique à la donnée de test $TD2_{SF}$.

La figure 9.36 montre le signal analogique de sortie de la carte CS1 lorsque son comportement est simulé en utilisant la donnée de test $TD1_{FIR}$. Alors que le signal de sortie calculé est toujours nul, le signal de sortie obtenu par simulation ne devient nul

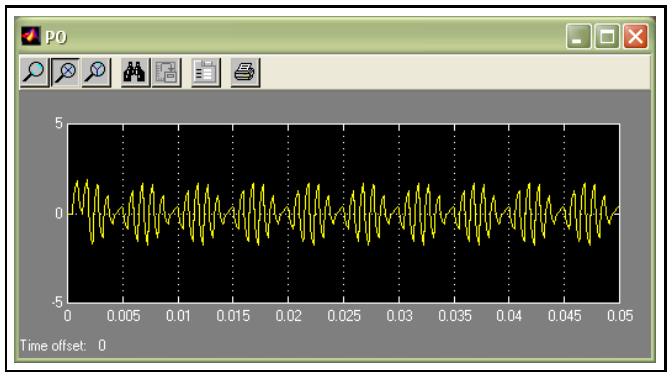

FIG. 9.34 – Signal analogique observé à la sortie primaire pour la donnée de test $TD2_{AAF}$

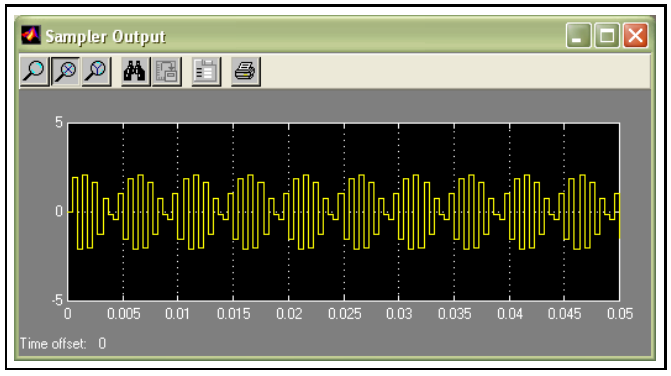

FIG. 9.35 – Signal de sortie de l'échantillonneur-bloqueur pour la donnée de test $TD2_{AAF}$

qu'après un certain temps. Dans la simulation, Simulink utilise l'équation aux différences du filtre FIR donnée par l'expression (9.3) avec des conditions initiales nulles :

$$\begin{aligned}
&y(0) = x(0) + x(-1) + x(-2) \; avec \; x(-1) = x(-2) = 0 \\
&y(1) = x(1) + x(0) + x(-1) \; avec \; x(-1) = 0 \\
&y(2) = x(2) + x(1) + x(0) \\
&\vdots \\
&y(k) = x(k) + x(k-1) + x(k-2)
\end{aligned}$$

Ainsi, les premiers échantillons en sortie du filtre FIR obtenus par simulation ne sont pas nuls, comme le montre la figure 9.37. Le modèle de test que nous proposons (cf. figure 9.26) est un modèle simplifié qui fait l'hypothèse que tous les échantillons en sortie du filtre FIR sont nuls.

9.2.3.3.6 Bilan

Les résultats obtenus montrent des écarts entre les sorties prédites par

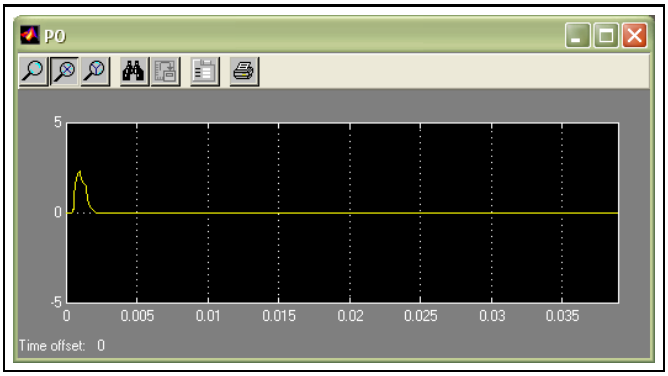

FIG. 9.36 – Signal présent à la sortie primaire avec la donnée de test $TD1_{FIR}$

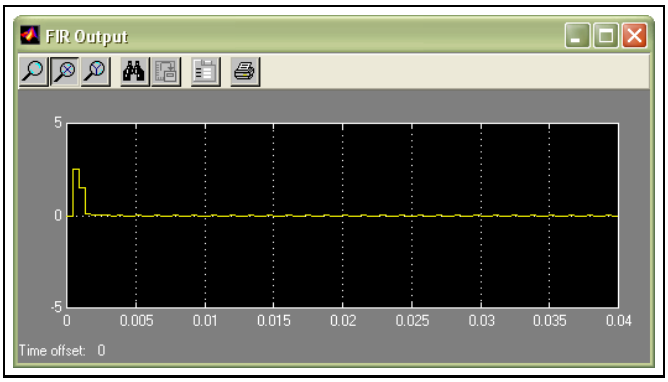

FIG. 9.37 – Signal de sortie du filtre FIR avec la donnée de test $TD1_{FIR}$

les données de test et celles obtenues par simulation. Ces écarts sont dûs au modèle fonctionnel du convertisseur numérique-analogique que nous avons proposé. Ce modèle, très simple, est basé sur l'hypothèse d'une reconstruction analogique parfaite rendue possible par le modèle des signaux numériques adopté dans notre méthodologie. Cette hypothèse n'est jamais vérifiée en pratique. Toutefois, lorsque la fréquence du stimulus de test sinusoïdal est faible par rapport à la fréquence de Shannon, le modèle reste acceptable.

9.3 Bilan de la validation

Nous avons présenté dans ce chapitre un protocole permettant de valider notre méthodologie de test. Les résultats de la simulation du comportement des cartes avec les données de test générées ont :

- confirmé la pertinence de celles-ci pour les cartes TCB et TCBE,
- montré les limites du modèle de signaux numériques dans la modélisation de la carte CS1.

Des travaux futurs seront nécessaires pour affiner la modélisation de la carte CS1 et évaluer la pertinence d'un nouveau jeu de test pour cette carte.

Nous rappelons que dans le protocole de validation, nous n'avons utilisé que la stratégie de test globale qui consiste à couvrir les transitions des modèles de test d'une carte et la stratégie de test locale qui permet de tester une transition. Le protocole de validation doit ainsi être étendu afin d'évaluer les données de test générées avec la deuxième stratégie de test globale (couverture des états des modèles de test d'une carte), ainsi que celles générées avec les autres stratégies de test locales (test d'un état, test d'un chemin).

Conclusion

Bilan Au cours de cette thèse, nous avons proposé une méthodologie de test de cartes mixtes en phase de maintenance. Les points forts de cette méthodologie de test sont les suivants :

- Le formalisme des automates à états finis communicants (CFSM - Communicating Finite State Machine) et la modélisation des signaux analogiques et numériques proposés permettent une modélisation uniforme des composants analogiques, numériques et mixtes d'une carte électronique, et de leurs interactions. Comme notre but est de tester des comportements de la carte, la modélisation des interactions entre les composants est particulièrement importante.
- Le formalisme est flexible par rapport au degré de connaissance initial concernant la carte à tester. Un même composant peut être ainsi modélisé par un CFSM ou un ensemble de CFSM plus ou moins complexe en terme de nombre d'états, de nombre de transitions et des conditions exprimées dans les étiquettes des transitions.
- L'expertise et le savoir-faire des ingénieurs de test (pratiques industrielles) est intégré dans la méthodologie sous la forme de modèles de test et de tactiques de test. Un modèle de test permet de définir la manière de tester efficacement un composant. Une tactique de test permet d'orienter plus finement la génération des données de test.
- Les stratégies de test globales permettent la génération de données de test ciblant la maintenance préventive.
- Les stratégies de test locales permettent la génération de données de test ciblant la maintenance corrective (aide à la localisation de panne et réparation).
- La génération des données de test s'effectuant en parcourant des chemins dans un ensemble de CFSM et en résolvant les conditions exprimées dans les transitions formant ces chemins, l'utilisation de la programmation logique par contraintes est particulièrement bien adaptée pour la mise en oeuvre de la méthodologie : la programmation logique apporte beaucoup de souplesse dans le parcours de chemin et la programmation par contraintes sa puissance et son efficacité pour la génération des données de test. Les données de test sont calculées sous forme d'intervalles que le testeur instancie au final à des valeurs spécifiques qui peuvent tenir compte de caractéristiques physiques particulières.

Nous avons mis en oeuvre un outil prototype basé sur la méthodologie proposée : l'outil *Copernicia*.

Notre approche a été appliquée à la génération de données de test pour un ensemble de cartes afin d'évaluer son adéquation réelle au test en maintenance de cartes mixtes. Les résultats obtenus sont satisfaisants. Ils ont confirmé les atouts susmentionnés et permis d'identifier quelques points méritant un approfondissement (par exemple, le modèle de signaux numériques nécessite d'être affiné pour permettre une meilleure modélisation de certaines cartes mixtes).

Perspectives Les prolongements de nos travaux sont les suivants :

- Une suite logique et immédiate de nos travaux est de poursuivre le protocole de validation présenté au chapitre 9 en :
  - mettant en oeuvre les stratégies de test qui n'ont pas été évaluées (couverture des états, test d'un chemin, ...),
  - enrichissant l'ensemble représentatif de cartes électroniques mixtes à tester. Il serait intéressant de continuer à valider notre méthodologie sur des cartes mixtes plus complexes que celles que nous avons considérées. Nous pourrions alors modéliser et tester des cartes mixtes qui possèdent un nombre de composants plus importants et/ou des fonctionnalités plus complexes.
- L'amélioration de l'implantation de la méthodologie. Actuellement, dans l'implantation, le modèle fonctionnel et le modèle de test d'un bloc fonctionnel sont constitués respectivement d'un unique CFSM, au lieu d'un ensemble de CFSM (cf. sections 7.3.2.2 et 7.3.2.3). Dans les exemples de cartes que nous avons traités, la représentation du modèle fonctionnel et du modèle de test d'un bloc simple par un seul CFSM s'est révélée suffisante. Cependant, l'utilisation d'un ensemble de CFSM au lieu d'un seul permettrait une granularité plus fine dans la modélisation d'un bloc plus complexe. Cette granularité plus fine serait utile à des fins de diagnostic.
- Les stratégies de test locales peuvent être enrichies. Nous rappelons que nous proposons de tester un chemin constitué d'une séquence de transitions. Il serait intéressant de tester un chemin constitué d'une séquence d'états, ce qui permettrait de tester de plus nombreux comportements de la carte, chaque comportement étant exprimé par une séquence de transitions.
- Le protocole de validation que nous avons utilisé ne s'appuie pas sur des cartes mixtes industrielles réelles. Un autre prolongement intéressant des travaux consisterait à appliquer notre méthodologie à de véritables cartes mixtes industrielles et à analyser son apport via l'utilisation de bancs de test industriels. Afin de mener correctement cette tâche, il serait nécessaire de traduire les données de test que nous générons pour une véritable carte mixte en un programme de test écrit dans le langage de programmation du testeur. L'exécution du programme de test sur le testeur permettrait, par rapport à la simulation, de mieux apprécier la pertinence de nos données de test.

Annexes

Annexe A

Grammaire des transitions (syntaxe)

```
T    ::= Rg [EAg] | Rg | EAg | G
Rg   ::= LR -> G | LR
EAg  ::= G : LEA | LEA
LR   ::= R, LR | R
LEA  ::= EA, LEA | EA
R    ::= IdAut ? S
S    ::= Signal | SStruct
SStruct ::= (LSignal)
LSignal ::= Signal, LSignal | Signal
Signal  ::= SignalId | Ssigd | Ssiga | Stop
EA   ::= IdAut ! S | A
A    ::= ident = Exp
Exp  ::= ExpArith | ExpBool | S
G    ::= ExpBool

IdAut    ::= ident
SignalId ::= ident
Ssigd  ::= sigd(Ssiga,Arg)
Ssiga  ::= siga(SForme,Arg,Arg,Arg)
Stop   ::= top(Arg)
Arg    ::= ident | reel | entier | Champ
SForme ::= sinus | rect | DC
Champ  ::= ident.SChamp | Champ.SChamp
SChamp ::= forme | freq | ampl | phi | dt1 | prd | dly | valeur | sa | date

ExpBool     ::= ExpBool || TermeBool | TermeBool
TermeBool   ::= TermeBool && FacteurBool | FacteurBool
FacteurBool ::= AtomeBool | neg AtomeBool
```

```
AtomeBool   ::= ident | (ExpBool) | EgalExp

EgalExp ::= EgalExp == ExpRel | EgalExp <> ExpRel | ExpRel
ExpRel  ::=   ExpRel < ExpArith
            | ExpRel > ExpArith
            | ExpRel <= ExpArith
            | ExpRel >= ExpArith
            | ExpArith

ExpArith  ::= ExpArith + Terme | ExpArith - Terme | Terme
Terme     ::= Terme * Facteur | Terme / Facteur | Facteur
Facteur   ::= ident | reel | entier | (ExpArith) | Champ | AppelFct
AppelFct  ::= ident(LArg)
LArg      ::= LArg, Arg | Arg

chiffre       [0-9]
entier        [+-]{chiffre}+
reel_sans_exp (({chiffre}+(\.{chiffre}*)?)|({chiffre}*\.{chiffre}+))
reel          {reel_sans_exp}([Ee][+-]?{chiffre}+)?
ident         [a-zA-Z_][a-zA-Z0-9_]*
```

Annexe B

Modélisation de la carte TCB en ECL^iPS^e (modèles fonctionnels)

```
:- lib(ic).

% nombre d'automates communicants
nombre_automates(7).

% liste des automates sources
sources([1]).

% liste des automates horloges
horloges([2]).

% liste des couples automate/transition observables en sortie
% (un seul couple par automate correspondant à la transition la plus lointaine)
sorties([(3,1)]).

% nombre d'états de chaque automate
% utilisé pour la couverture des états
nombre_etats(1,1). % S
nombre_etats(2,1). % CLK
nombre_etats(3,1). % MP
nombre_etats(4,2). % MEM
nombre_etats(5,3). % C
nombre_etats(6,3). % D
nombre_etats(7,2). % F

% nombre de transitions de chaque automate
% utilisé pour la couverture des transitions
nombre_transitions(1,1). % S
```

```
nombre_transitions(2,1). % CLK
nombre_transitions(3,1). % MP
nombre_transitions(4,2). % MEM
nombre_transitions(5,4). % C
nombre_transitions(6,3). % D
nombre_transitions(7,2). % F

% état initial de chaque automate
etat_initial(1,1). % S
etat_initial(2,1). % CLK
etat_initial(3,1). % MP
etat_initial(4,1). % MEM
etat_initial(5,1). % C
etat_initial(6,1). % D
etat_initial(7,1). % F

% S
existe_transition(1,1,1,1).
% CLK
existe_transition(2,1,1,1).
% MP
existe_transition(3,1,1,1).
% MEM
existe_transition(4,1,1,2).
existe_transition(4,2,2,1).
% C
existe_transition(5,1,1,2).
existe_transition(5,2,2,3).
existe_transition(5,3,2,3).
existe_transition(5,4,3,1).
% D
existe_transition(6,1,1,2).
existe_transition(6,2,2,3).
existe_transition(6,3,3,1).
% F
existe_transition(7,1,1,2).
existe_transition(7,2,2,1).

%%%------------------------------------------------ automate #1
transition(1,1,1,1,Env1,Env2,T) :-
        #>(T,0,Tok),
        ( Tok == 0 -> message_erreur_pas ; true),
        T #> 0,
        conforme_spec(1,1,V,_),
```

```
        ajout_env_elt(1,1,1,Env1,Env11),           % 1 = etat
        ajout_env_elt(2,1,1,Env11,Env12),          % 2 = transition
        ajout_env_elt(4,1,mag(1,V),Env12,Env13),   % 4 = mag_out si emission
        ajout_env_elt(5,_,emission(1,1,(7,1,V)),Env13,Env2).
%%%------------------------------------------------ automate #2
transition(2,1,1,1,Env1,Env2,T) :-
        #>(T,0,Tok),
        ( Tok == 0 -> message_erreur_pas ; true),
        T #> 0,
        conforme_spec(2,1,V,_),
        ajout_env_elt(1,2,1,Env1,Env11),
        ajout_env_elt(2,2,1,Env11,Env12),
        ajout_env_elt(4,2,mag(1,V),Env12,Env13),
        ajout_env_elt(5,_,emission(2,1,(6,1,V)),Env13,Env2).
%%%------------------------------------------------ automate #3
transition(3,1,1,1,Env1,Env2,T) :-
        #>(T,0,Tok),
        ( Tok == 0 -> message_erreur_pas ; true),
        T #> 0,
        T1 is T-1,
        conforme_spec(3,1,V,_),
        extraire_env_mag_out(Env1,4,2,V,Env10,T1),
        ajout_env_elt(1,3,1,Env10,Env11),
        ajout_env_elt(2,3,1,Env11,Env12),          % 2 = transition
        ajout_env_elt(3,3,mag(1,V),Env12,Env13),   % 3 = mag_in si reception
        ajout_env_elt(5,_,reception(3,1,(4,2,V)),Env13,Env2).
%%%------------------------------------------------ automate #4
transition(4,1,1,2,Env1,Env2,T) :-
        #>(T,0,Tok),
        ( Tok == 0 -> message_erreur_pas ; true),
        T #> 0,
        T1 is T-1,
        conforme_spec(4,1,V,_),
        extraire_env_mag_out(Env1,6,3,V,Env101,T1),
        ajout_env_elt(1,4,1,Env101,Env10),
        ajout_env_elt(1,4,2,Env10,Env11),
        ajout_env_elt(2,4,1,Env11,Env12),
        ajout_env_elt(3,4,mag(1,V),Env12,Env13),
        ajout_env_elt(5,_,reception(4,1,(6,3,V)),Env13,Env2).
transition(4,2,2,1,Env1,Env2,T) :-
        #>(T,0,Tok),
        ( Tok == 0 -> message_erreur_pas ; true),
        T #> 0,
        T1 is T-1,
```

```
        conforme_spec(4,2,V,[V1]),
        extraire_env_mag_in(Env1,4,1,V1,Env10,T1),
        ajout_env_elt(1,4,1,Env10,Env11),
        ajout_env_elt(2,4,2,Env11,Env12),
        ajout_env_elt(4,4,mag(2,V),Env12,Env13),
        ajout_env_elt(5,_,emission(4,2,(3,1,V)),Env13,Env2).
%%%------------------------------------------------ automate #5
transition(5,1,1,2,Env1,Env2,T) :-
        #>(T,0,Tok),
        ( Tok == 0 -> message_erreur_pas ; true),
        T #> 0,
        T1 is T-1,
        conforme_spec(5,1,V,_),
        extraire_env_mag_out(Env1,7,2,V,Env101,T1),
        ajout_env_elt(1,5,1,Env101,Env10),
        ajout_env_elt(1,5,2,Env10,Env11),
        ajout_env_elt(2,5,1,Env11,Env12),
        ajout_env_elt(3,5,mag(1,V),Env12,Env13), % reception
        ajout_env_elt(0,5,mag(2,V),Env13,Env14), %range dans magasin
                                                 % local etat 2
        ajout_env_elt(5,_,local(5,2,V),Env14,Env15),
        ajout_env_elt(5,_,reception(5,1,(7,2,V)),Env15,Env2).
transition(5,2,2,3,Env1,Env2,T) :-
        #>(T,0,Tok),
        ( Tok == 0 -> message_erreur_pas ; true),
        T #> 0,
        T1 is T-1,
        conforme_spec(5,2,V,[V1]),
        extraire_env_mag_local(Env1,5,2,V1,Env10,T1), % lit dans le
                                                      % magasin local de
                                                      % l'état de départ
                                                      % (état 2)
        ajout_env_elt(1,5,3,Env10,Env11),
        ajout_env_elt(2,5,2,Env11,Env12),
        ajout_env_elt(0,5,mag(3,V),Env12,Env13), % range dans magasin
                                                 % local etat 3
        ajout_env_elt(5,_,local(5,3,V),Env13,Env2).
transition(5,3,2,3,Env1,Env2,T) :-
        #>(T,0,Tok),
        ( Tok == 0 -> message_erreur_pas ; true),
        T #> 0,
        T1 is T-1,
        conforme_spec(5,3,V,[V1]),
        extraire_env_mag_local(Env1,5,2,V1,Env10,T1),
```

```
        ajout_env_elt(1,5,3,Env10,Env11),
        ajout_env_elt(2,5,3,Env11,Env12),
        ajout_env_elt(0,5,mag(3,V),Env12,Env13),
        ajout_env_elt(5,_,local(5,3,V),Env13,Env2).
transition(5,4,3,1,Env1,Env2,T) :-
        #>(T,0,Tok),
        ( Tok == 0 -> message_erreur_pas ; true),
        T #> 0,
        T1 is T-1,
        conforme_spec(5,4,V,[V1]),
        extraire_env_mag_local(Env1,5,3,V1,Env10,T1),
        ajout_env_elt(1,5,1,Env10,Env11),
        ajout_env_elt(2,5,4,Env11,Env12),
        ajout_env_elt(4,5,mag(4,V),Env12,Env13), % emission
        ajout_env_elt(5,_,emission(5,4,(4,1,V)),Env13,Env2).
%%%------------------------------------------------ automate #6
transition(6,1,1,2,Env1,Env2,T) :-
        #>(T,0,Tok),
        ( Tok == 0 -> message_erreur_pas ; true),
        T #> 0,
        T1 is T-1,
        conforme_spec(6,1,V,_),
        extraire_env_mag_out(Env1,2,1,V,Env101,T1),
        ajout_env_elt(1,6,1,Env101,Env10),
        ajout_env_elt(1,6,2,Env10,Env11),
        ajout_env_elt(2,6,1,Env11,Env12),
        ajout_env_elt(3,6,mag(1,V),Env12,Env13),
        ajout_env_elt(5,_,reception(6,1,(2,1,V)),Env13,Env2).
transition(6,2,2,3,Env1,Env2,T) :-
        #>(T,0,Tok),
        ( Tok == 0 -> message_erreur_pas ; true),
        T #> 0,
        T1 is T-1,
        conforme_spec(6,2,V,_),
        extraire_env_mag_out(Env1,5,4,V,Env10,T1),
        ajout_env_elt(1,6,3,Env10,Env11),
        ajout_env_elt(2,6,2,Env11,Env12),
        ajout_env_elt(3,6,mag(2,V),Env12,Env13),
        ajout_env_elt(5,_,reception(6,2,(5,4,V)),Env13,Env2).
transition(6,3,3,1,Env1,Env2,T) :-
        #>(T,0,Tok),
        ( Tok == 0 -> message_erreur_pas ; true),
        T #> 0,
        T1 is T-1,
```

```
        conforme_spec(6,3,V,[Z,X]),
        extraire_env_mag_in(Env1,6,1,Z,Env101,T1),
        extraire_env_mag_in(Env101,6,2,X,Env102,T1),
        ajout_env_elt(1,6,1,Env102,Env11),
        ajout_env_elt(2,6,3,Env11,Env12),
        ajout_env_elt(4,6,mag(3,V),Env12,Env13),
        ajout_env_elt(5,_,emission(6,3,(4,1,V)),Env13,Env2).
%%%------------------------------------------------- automate #7
transition(7,1,1,2,Env1,Env2,T) :-
        #>(T,0,Tok),
        ( Tok == 0 -> message_erreur_pas ; true),
        T #> 0,
        T1 is T-1,
        conforme_spec(7,1,V,_),
        extraire_env_mag_out(Env1,1,1,V,Env101,T1),
        ajout_env_elt(1,7,1,Env101,Env10),
        ajout_env_elt(1,7,2,Env10,Env11),
        ajout_env_elt(2,7,1,Env11,Env12),
        ajout_env_elt(3,7,mag(1,V),Env12,Env13),
        ajout_env_elt(5,_,reception(7,1,(1,1,V)),Env13,Env2).
transition(7,2,2,1,Env1,Env2,T) :-
        #>(T,0,Tok),
        ( Tok == 0 -> message_erreur_pas ; true),
        T #> 0,
        T1 is T-1,
        conforme_spec(7,2,V,[V1]),
        extraire_env_mag_in(Env1,7,1,V1,Env10,T1),
        ajout_env_elt(1,7,1,Env10,Env11),
        ajout_env_elt(2,7,2,Env11,Env12),
        ajout_env_elt(4,7,mag(2,V),Env12,Env13),
        ajout_env_elt(5,_,emission(7,2,(5,1,V)),Env13,Env2).


% caractéristiques des composants
caract_physiques(2, [0.002]).   % période de l'horloge (0.002s)
caract_physiques(5, [5.0]).     % seuil du comparateur (5V)
caract_physiques(7, [1000.0]).  % fréquence de coupure du filtre (1000 Hz)


arcsin(X,Y) :-
        Y $= atan(X / sqrt(1 - X^2)).

% S
% pas de contraintes
```

```
%=======================
conforme_spec(1,1,_,_).

% Clk
%=======================
conforme_spec(2,1,V,_) :-
     V = top(Arg1),
     caract_physiques(2, [CP]),
     integers(K),
     Arg1  $= K*CP.

% MP
% pas de contraintes
%=======================
conforme_spec(3,1,_,_).

% Mem (FM)
% pas de contraintes
%=======================
conforme_spec(4,1,_,_).
conforme_spec(4,2,V,[V|_]).

% C (FM)
%=======================
conforme_spec(5,1,_,_).
conforme_spec(5,2,V,Lv) :-
      Lv = [V1],
      V1 = siga(Forme,Arg1,_,_),
      caract_physiques(5,[S]),
      Forme $= 1,
      Arg1  $< S,
      V = siga(3,0,0,0).
conforme_spec(5,3,V,Lv) :-
      V  = siga(SeForme,SeArg1,SeArg2,SeArg3),
      Lv = [Xr],
      Xr = siga(Forme,XrArg1,XrArg2,XrArg3),
      caract_physiques(5,[S]),
      Forme $= 1,
      XrArg1  $>= S,
      SeForme $= 2,
      SeArg2  $= 1 / XrArg2,
      R       $= S / XrArg1,
      SeArg1  $= 1 / (2*XrArg2) - 1 / (pi*XrArg2) * arcsin(R),
      SeArg3  $= 1 / (2*pi*XrArg2) * arcsin(R) + XrArg3 / (2*pi*XrArg2),
```

```
      SeArg1 $> 0,
      SeArg2 $> 0,
      SeArg3 $> 0.
conforme_spec(5,4,V,[V]).

% D (FM)
%=======================
conforme_spec(6,1,_,_).
conforme_spec(6,2,_,_).
conforme_spec(6,3,V,Vp) :-
        Vp=[Z,X],
        V=sigd(X,Z).

% F (FM)
%=======================
conforme_spec(7,1,V,_) :-
        V = siga(Forme,_,_,_),
        Forme $= 1.
conforme_spec(7,2,V,Lv) :-
        Lv = [Xr|_],
        V = siga(SeForme,SeArg1,SeArg2,SeArg3),
        Xr = siga(_,XrArg1,XrArg2,XrArg3),
        caract_physiques(7, [Fc]),
        SeForme $= 1,
        SeArg2  $= XrArg2,
        SeArg1  $= XrArg1 / sqrt(1 + (Fc/XrArg2)^2),
        SeArg3  $= XrArg3 - atan(Fc/XrArg2).



% Pondération des états pour des parcours optimaux
poids(1,1,1,1,1).
poids(1,2,1,1,1).
poids(1,3,1,1,1).
poids(1,4,1,1,1).
poids(1,4,2,2,2).
poids(1,5,1,1,1).
poids(1,5,2,2,2).
poids(1,5,3,3,3).
poids(1,6,1,1,1).
poids(1,6,2,2,2).
poids(1,6,3,3,3).
poids(1,7,1,1,1).
poids(1,7,2,2,2).
```

```
% pondération des transitions pour des parcours optimaux
% On choisit de ponderer une transition par les poids inferieur et
% superieur de son etat initial.
% poids(2,A,transition,poids_inferieur,poids_superieur)
poids(2,A,N,P1,P2) :-
         existe_transition(A,N,X,_),
         poids(1,A,X,P1,P2).

% Pondération maximum des états (choix=1)
poids_max(1,1,1,1).
poids_max(1,2,1,1).
poids_max(1,3,1,1).
poids_max(1,4,2,2).
poids_max(1,5,3,3).
poids_max(1,6,3,3).
poids_max(1,7,2,2).

% Pondération maximum des transitions (choix=2)
poids_max(2,1,1,1).
poids_max(2,2,1,1).
poids_max(2,3,1,1).
poids_max(2,4,2,2).
poids_max(2,5,3,3).
poids_max(2,6,3,3).
poids_max(2,7,2,2).

% liste des états suivants pour chaque automate
lsuiv(1,1,[1]).
lsuiv(2,1,[1]).
lsuiv(3,1,[1]).
lsuiv(4,1,[2]).
lsuiv(4,2,[1]).
lsuiv(5,1,[2]).
lsuiv(5,2,[3]).
lsuiv(5,3,[1]).
lsuiv(6,1,[2]).
lsuiv(6,2,[3]).
lsuiv(6,3,[1]).
lsuiv(7,1,[2]).
lsuiv(7,2,[1]).

% liste des états précédents pour chaque automate
lprec(1,1,[1]).
```

```
lprec(2,1,[1]).
lprec(3,1,[1]).
lprec(4,1,[2]).
lprec(4,2,[1]).
lprec(5,1,[3]).
lprec(5,2,[1]).
lprec(5,3,[2]).
lprec(6,1,[3]).
lprec(6,2,[1]).
lprec(6,3,[2]).
lprec(7,1,[2]).
lprec(7,2,[1]).

%ltrans_suiv (automate, etat, liste des transitions sortantes)
ltrans_suiv(1,1,[1]).
ltrans_suiv(2,1,[1]).
ltrans_suiv(3,1,[1]).
ltrans_suiv(4,1,[1]).
ltrans_suiv(4,2,[2]).
ltrans_suiv(5,1,[1]).
ltrans_suiv(5,2,[2,3]).
ltrans_suiv(5,3,[4]).
ltrans_suiv(6,1,[1]).
ltrans_suiv(6,2,[2]).
ltrans_suiv(6,3,[3]).
ltrans_suiv(7,1,[1]).
ltrans_suiv(7,2,[2]).

%ltrans_prec (automate, etat, liste des transitions entrantes)
ltrans_prec(1,1,[1]).
ltrans_prec(2,1,[1]).
ltrans_prec(3,1,[1]).
ltrans_prec(4,1,[2]).
ltrans_prec(4,2,[1]).
ltrans_prec(5,1,[4]).
ltrans_prec(5,2,[1]).
ltrans_prec(5,3,[2,3]).
ltrans_prec(6,1,[3]).
ltrans_prec(6,2,[1]).
ltrans_prec(6,3,[2]).
ltrans_prec(7,1,[2]).
ltrans_prec(7,2,[1]).
```

Annexe C

Extrait des algorithmes de génération des données de test : algorithme de test d'une transition

```
:- lib(ic).
:- lib(ic_sets).
:- lib(lists).


%% Algo de test d'une transition : test_transition(Automate,Transition,Pas)
test_transition(A,N,T) :-
        T #> 0,
        poids(2,A,N,P,_),
        #>=(T,P,Tok),
        ( Tok == 0 -> message_erreur_pas ; true),
        T #>= P,
        T1 is T-1,
        init_automates(Envi),
        transition(A,N,_,_,Envi,Envm,T),
        propagation_sorties(Envm,Env,T1),
        Env=env(Lea,Lmess),
        printf("\n%s\n","***********************************************"
               "***** Affichage de toute l'information sur les"
               " domaines (sans instanciation) **********"),
        printf("%s\n","%% Numéro automate, numéro transition testée : "),
        write(A), printf("%s"," "), write(N),
        printf("\n%s\n","%% Messages : "),
        (foreach(M,Lmess) do
```

```
            write(M),
            printf("%s\n","")),
  nombre_automates(Nb),
  (for(I,1,Nb), foreach(Ea,Lea) do
         printf("\n%s\n","%% Environnement automate "),
                write(I),
                printf("%s\n",""),
         write(Ea)),
  printf("\n\n%s\n","***********************************************"
              "***** Affichage des sources, horloges, sorties ***************"),
  valuer_sources(Env),
  valuer_horloges(Env),
  valuer_sorties(Env),
  printf("\n%s\n","*********************************************"
              "******* Affichage total après instanciation ***************"),
  Env=env(Lea2,Lmess2),
  printf("%s\n","%% Numéro automate, numéro transition testée : "),
  write(A), printf("%s"," "), write(N),
  printf("\n%s\n","%% Messages : "),
  foreach(M2,Lmess2) do
            write(M2),
            printf("%s\n","")),
  nombre_automates(Nb),
  (for(I2,1,Nb), foreach(Ea2,Lea2) do
         printf("\n%s\n","%% Environnement automate "),
         write(I2),
         printf("%s\n",""),
         write(Ea2)),
  printf("%s\n","").


message_erreur_pas :-
        printf("\n%s\n","!! La transition ne peut être testée  : augmenter le pas").


%% init_automates : initialise le contexte global et les contextes de tous les automates
%%                  en fabricant la structure du contexte a partir de listes vides
%%                  Exemple pour 2 automates : env([enva([],[],[],[],[]),
%%                                                  enva([],[],[],[],[])],[])
init_automates(env(Env,[])) :-
nombre_automates(N),
init_automates2(N,Env).

init_automates2(1,[enva([],[],[],[],[])]).
```

```
init_automates2(N,[enva([],[],[],[],[])|Env]) :-
N #> 1,
N1 is N-1,
init_automates2(N1,Env).


%% propagation_sorties : trouve des chemins valides a partir des etats courants
%% (dans Env1) afin de propager les contraintes deja cumulees (dans Env1)
%% vers les automates de sortie. Le resultat est ensuite stocke dans Env2
propagation_sorties(Env1,Env2,T) :-
        sorties(Ls),
        propagation_sorties2(Ls,Env1,Env2,T).

propagation_sorties2([],Env,Env,_).
propagation_sorties2([(As,Ns)|Ls],Env1,Env2,T) :-
        extraire_env_vus(2,As,Env1,Ltvues),
        member(Ns,Ltvues), !,
        propagation_sorties2(Ls,Env1,Env2,T).
propagation_sorties2([(As,Ns)|Ls],Env1,Env3,T) :-
        transition(As,Ns,_,_,Env1,Env2,T), !,
        propagation_sorties2(Ls,Env2,Env3,T).

%% extraire_env_vus :
%% C=1 -> Lvus est la liste des etats vus du contexte de l'automate A
%% C=2 -> Lvus est la liste des transitions vues du contexte de l'automate A
extraire_env_vus(C,A,env(Env,_),Lvus) :-
        elt_liste(A,Env,X),
        (C == 1 -> X=enva(Lvus,_,_,_,_)
        ;
                   X=enva(_,Lvus,_,_,_)).

elt_liste(1,[X|_],X).
elt_liste(N,[_|L],X) :-
        N #> 1,
        N1 is N-1,
        elt_liste(N1,L,X).


%% valuer_sources
valuer_sources(Env) :-
printf("\n%s","%% ----------------> Sources : "),
sources(Ls),
(foreach(S,Ls), param(Env) do
extraire_env_mag(Env,S,_,M_out,_),
```

```
printf("\n%s","%% --> Domaines des données de test sources : "),
                (foreach(I,M_out), param(S) do
                     I=mag(N,V),
                     printf("\n%s\n","%% Numéro Automate Source "),
                     write(S),
                     printf("\n%s\n","%% Numéro Transition dans Automate Source"),
                     write(N),
                     printf("\n%s\n","%% Domaine Donnée de Test associée"),
                     write(V)),
printf("%s \n",""),
                printf("\n%s","%%--> Données de test (valeurs des sources après locate):"),
                (foreach(I,M_out), foreach(J,M_out2) do
                     I=mag(_,V),
                     (V=siga(Param1,Param2,Param3,Param4) ->
                                      Lv=[Param1,Param2,Param3,Param4]
                     ;
                       (V=sigd(siga(Param1,Param2,Param3,Param4),top(T)) ->
                                         Lv=[Param1,Param2,Param3,Param4,T]
                                 ;
                                       (V=top(T) -> Lv=[T]
                                                          ;
                                                              Lv=[]))),
                     locate(Lv,1e-7),
                     J=V),
(foreach(I,M_out), foreach(J,M_out2), param(S) do
                     I=mag(N,_),
                     printf("\n%s\n","%% Numéro Automate Source "),
                     write(S),
                     printf("\n%s\n","%% Numéro Transition dans Automate Source"),
                     write(N),
                     printf("\n%s\n","%% Donnée de Test associée après locate : "),
                     write(J)),
printf("%s \n","")).


%% valuer_horloges
valuer_horloges(Env) :-
printf("\n%s","%% ----------------> Horloges : "),
        horloges(Ls),
(foreach(S,Ls), param(Env) do
extraire_env_mag(Env,S,_,M_out,_),
printf("\n%s","%% --> Domaines des données de test horloges : "),
                (foreach(I,M_out), param(S) do
                     I=mag(N,top(V)),
```

```
                    printf("\n%s\n","%% Numéro Automate Horloge"),
                    write(S),
                    printf("\n%s\n","%% Numéro Transition dans Automate Horloge"),
                    write(N),
                    printf("\n%s\n","%% Domaine Donnée de Test associée"),
                    write(top(V))),
printf("%s \n",""),
               printf("\n%s","%% --> Données de test "),
               printf("\n%s","%%     (valeurs des horloges après locate) : "),
               (foreach(I,M_out), foreach(J,M_out2) do
                    I=mag(_,top(V)),
                    locate([V],1e-7),
                    J=top(V)),
               (foreach(I,M_out), foreach(J,M_out2), param(S) do
                    I=mag(N,_),
                    printf("\n%s\n","%% Numéro Automate Horloge"),
                    write(S),
                    printf("\n%s\n","%% Numéro Transition dans Automate Horloge"),
                    write(N),
                    printf("\n%s\n","%% Donnée de Test associée après locate : "),
                    write(J)),
                    printf("%s \n","")).


%% valuer_sorties
valuer_sorties(Env) :-
printf("\n%s","%% ----------------> Sorties : "),
sorties(Ls),
(foreach((S,_),Ls), param(Env) do
extraire_env_mag(Env,S,M_in,_,_),
printf("\n%s","%% --> Domaines des sorties : "),
               (foreach(I,M_in), param(S) do
                    I=mag(N,V),
                    printf("\n%s\n","%% Numéro Automate Sortie "),
                    write(S),
                    printf("\n%s\n","%% Numéro Transition dans Automate Sortie"),
                    write(N),
                    printf("\n%s\n","%% Domaine Sortie associée "),
                    write(V)),
                    printf("%s \n","")).


%% extraire_env_mag
%% extrait les magasins M_in, M_out et M_local du contexte de l'automate A
```

```
extraire_env_mag(env(Env,_),A,M_in,M_out,M_local) :-
        elt_liste(A,Env,X),
        X=enva(_,_,M_in,M_out,M_local).


%% ajout_env_elt (et modif_env) :
%% C=5 -> X est un message a ajouter au contexte global
%% C=0 -> X est une valeur mag(E,V) a ajouter a mag_local du contexte de l'automate A
%% C=1 -> X est un etat a ajouter a etats_vus du contexte de l'automate A
%% C=2 -> X est une transition a ajouter a trans_vues du contexte de l'automate A
%% C=3 -> X est une valeur mag(N,V) a ajouter a mag_in du contexte de l'automate A
%% C=4 -> X est une valeur mag(N,V) a ajouter a mag_out du contexte de l'automate A
ajout_env_elt(C,A,X,env(Env1,Env1_mess),env(Env2,Env2_mess)) :-
        (C == 5 -> (Env2_mess=[X|Env1_mess],
                    Env2=Env1)
        ;
                   (Env2_mess=Env1_mess,
                    modif_env(C,A,X,Env1,Env2))).

modif_env(C,1,X,[enva(Ea11,Ea12,Ea13,Ea14,Ea15)|Env1],[Ea2|Env1]) :-
        (C == 1 -> (ajout_liste([X],Ea11,Ea21),
                    Ea2=enva(Ea21,Ea12,Ea13,Ea14,Ea15))
        ;
           (C == 2 -> (ajout_liste([X],Ea12,Ea22),
                       Ea2=enva(Ea11,Ea22,Ea13,Ea14,Ea15))
           ;
             (C == 3 -> (ajout_liste_mag([X],Ea13,Ea23),
                         Ea2=enva(Ea11,Ea12,Ea23,Ea14,Ea15))
             ;
                (C == 4 -> (ajout_liste_mag([X],Ea14,Ea24),
                            Ea2=enva(Ea11,Ea12,Ea13,Ea24,Ea15))
                ;
                   (ajout_liste_mag([X],Ea15,Ea25),
                    Ea2=enva(Ea11,Ea12,Ea13,Ea14,Ea25)))))).

modif_env(C,A,X,[Ea1|Env1],[Ea1|Env2]) :-
        A #> 1,
        A1 is A-1,
        modif_env(C,A1,X,Env1,Env2).

ajout_liste_mag([],L,L).
ajout_liste_mag([mag(N,V)|L1],L2,L4) :-
        member(mag(N,_),L2), !,
        remp_mag(N,V,L2,L3),
```

```
        ajout_liste_mag(L1,L3,L4).
ajout_liste_mag([X|L1],L2,[X|L3]) :-
        ajout_liste_mag(L1,L2,L3).

remp_mag(N,V,[mag(N,_)|L],[mag(N,V)|L]) :- !.
remp_mag(N,V,[M|L1],[M|L2]) :-
        remp_mag(N,V,L1,L2).

%% ajout_liste(L1,L2,L3) : L3 est le resultat de l'adjonction des
%% elements de L1 dans L2, s'ils n'y sont pas deja.
ajout_liste([],L,L).
ajout_liste([X|L1],L2,L3) :-
        member(X,L2), !,
        ajout_liste(L1,L2,L3).
ajout_liste([X|L1],L2,[X|L3]) :-
        ajout_liste(L1,L2,L3).


%% extraire_env_mag_in
%% V = valeur associee a la transition N dans le mag_in de A
extraire_env_mag_in(Context1,A,N,V,Context2,T) :-
        Context1=env(Env1,_),
        elt_liste(A,Env1,X),
        X=enva(_,_,Lmag,_,_), !,
        extraire_mag_in(A,Lmag,N,V,Context1,Context2,T).

extraire_mag_in(A,[],N,V,Context1,Context2,T) :-
        transition(A,N,_,_,Context1,Context2,T),
        Context2=env(Env2,_),
        elt_liste(A,Env2,X),
        X=enva(_,_,Lmag,_,_),
        verif_mag(Lmag,N,V).
extraire_mag_in(_,[mag(N,V)|_],N,V,Context,Context,_).
extraire_mag_in(A,[mag(N1,_)|L],N,V,Context1,Context2,T) :-
        N1\==N,
        extraire_mag_in(A,L,N,V,Context1,Context2,T).

verif_mag([mag(N,V)|_],N,V).
verif_mag([mag(N1,_)|L],N,V) :-
        N1\==N,
        verif_mag(L,N,V).


%% extraire_env_mag_out
```

```
%% V = valeur associee a la transition N dans le mag_out de A
extraire_env_mag_out(Context1,A,N,V,Context2,T) :-
        Context1=env(Env1,_),
        elt_liste(A,Env1,X),
        X=enva(_,_,_,Lmag,_), !,
        extraire_mag_out(A,Lmag,N,V,Context1,Context2,T).

extraire_mag_out(A,[],N,V,Context1,Context2,T) :-
        transition(A,N,_,_,Context1,Context2,T),
        Context2=env(Env2,_),
        elt_liste(A,Env2,X),
        X=enva(_,_,_,Lmag,_),
        verif_mag(Lmag,N,V).
extraire_mag_out(_,[mag(N,V)|_],N,V,Context,Context,_).
extraire_mag_out(A,[mag(N1,_)|L],N,V,Context1,Context2,T) :-
        N1\==N,
        extraire_mag_out(A,L,N,V,Context1,Context2,T).


%% extraire_env_mag_local
%% V = valeur associee a l'etat E dans le mag_local de A
extraire_env_mag_local(Context1,A,E,V,Context2,T) :-
        Context1=env(Env1,_),
        elt_liste(A,Env1,X),
        X=enva(_,_,_,_,Lmag), !,
        extraire_mag_local(A,Lmag,E,V,Context1,Context2,T).

extraire_mag_local(A,[],E,V,Context1,Context2,T) :-
        transition(A,_,_,E,Context1,Context2,T),
        Context2=env(Env2,_),
        elt_liste(A,Env2,X),
        X=enva(_,_,_,_,Lmag),
        verif_mag(Lmag,E,V).
extraire_mag_local(_,[mag(E,V)|_],E,V,Context,Context,_).
extraire_mag_local(A,[mag(E1,_)|L],E,V,Context1,Context2,T) :-
        E1\==E,
        extraire_mag_local(A,L,E,V,Context1,Context2,T).
```

Annexe D

Rapport de test

Dans ce rapport de test, on trouve la donnée de test vérifiant la synchronisation analogique-numérique de la carte TCB. Cette donnée de test (voir la quatrième page de cette annexe) a été obtenue en appliquant la stratégie de test locale « test d'une transition » sur la transition No 1 du point de mesure (automate No 3) de la carte TCB.

```
***Affichage de toute l'information sur les domaines (sans instanciation) ****
%% Numéro automate, numéro transition testée :
3 1
%% Messages :
reception(3, 1, (4, 2,
sigd(siga(2, XrArg1{0.000273555303651743 .. 0.000856433706871294},
0.002__0.002, XrArg3{0.00157178314656435 .. 0.00186322234817413}),
top(0.002__0.002))))

emission(4, 2, (3, 1,
sigd(siga(2, XrArg1{0.000273555303651743 .. 0.000856433706871294},
0.002__0.002, XrArg3{0.00157178314656435 .. 0.00186322234817413}),
top(0.002__0.002))))

reception(4, 1, (6, 3,
sigd(siga(2, XrArg1{0.000273555303651743 .. 0.000856433706871294},
0.002__0.002, XrArg3{0.00157178314656435 .. 0.00186322234817413}),
 top(0.002__0.002))))

emission(6, 3, (4, 1,
sigd(siga(2, XrArg1{0.000273555303651743 .. 0.000856433706871294},
0.002__0.002, XrArg3{0.00157178314656435 .. 0.00186322234817413}),
top(0.002__0.002))))

reception(6, 2, (5, 4,
```

```
siga(2, XrArg1{0.000273555303651743 .. 0.00085643370687l294},
0.002__0.002, XrArg3{0.00157178314656435 .. 0.00186322234817413})))

emission(5, 4, (4, 1,
siga(2, XrArg1{0.000273555303651743 .. 0.00085643370687l294},
0.002__0.002, XrArg3{0.00157178314656435 .. 0.00186322234817413})))

local(5, 3,
siga(2, XrArg1{0.000273555303651743 .. 0.00085643370687l294},
0.002__0.002, XrArg3{0.00157178314656435 .. 0.00186322234817413}))

reception(5, 1, (7, 2,
siga(1, XrArg1{5.5 .. 22.3606797749979},
500.0__500.0, XrArg3{3.79680572563935 .. 5.62797223513004})))

local(5, 2, siga(1, XrArg1{5.5 .. 22.3606797749979},
500.0__500.0, XrArg3{3.79680572563935 .. 5.62797223513004}))

emission(7, 2, (5, 1,
siga(1, XrArg1{5.5 .. 22.3606797749979},
500.0__500.0, XrArg3{3.79680572563935 .. 5.62797223513004})))

reception(7, 1, (1, 1,
siga(1, XrArg1{12.2983738762488 .. 50.0},
500.0__500.0, XrArg3{4.90395444343343 .. 6.73512095292413})))

emission(1, 1, (7, 1,
siga(1, XrArg1{12.2983738762488 .. 50.0},
500.0__500.0, XrArg3{4.90395444343343 .. 6.73512095292413})))

reception(6, 1, (2, 1, top(0.002__0.002)))

emission(2, 1, (6, 1, top(0.002__0.002)))

%% Environnement automate
1
enva([1], [1], [],
[mag(1, siga(1, XrArg1{12.2983738762488 .. 50.0},
500.0__500.0, XrArg3{4.90395444343343 .. 6.73512095292413}))], [])
%% Environnement automate
2
enva([1], [1], [], [mag(1, top(0.002__0.002))], [])
%% Environnement automate
3
```

```
enva([1], [1],
[mag(1, sigd(siga(2, XrArg1{0.000273555303651743 .. 0.000856433706871294},
0.002__0.002, XrArg3{0.00157178314656435 .. 0.00186322234817413}),
top(0.002__0.002)))], [], [])
%% Environnement automate
4
enva([2, 1], [2, 1],
[mag(1, sigd(siga(2, XrArg1{0.000273555303651743 .. 0.000856433706871294},
0.002__0.002, XrArg3{0.00157178314656435 .. 0.00186322234817413}),
top(0.002__0.002)))],
[mag(2, sigd(siga(2, XrArg1, 0.002__0.002, XrArg3),
top(0.002__0.002)))], [])
%% Environnement automate
5
enva([3, 2, 1], [4, 3, 1],
[mag(1, siga(1, XrArg1{5.5 .. 22.3606797749979},
500.0__500.0, XrArg3{3.79680572563935 .. 5.62797223513004}))],
[mag(4, siga(2, XrArg1{0.000273555303651743 .. 0.000856433706871294},
0.002__0.002, XrArg3{0.00157178314656435 .. 0.00186322234817413}))],
[mag(3, siga(2, XrArg1, 0.002__0.002, XrArg3)),
mag(2, siga(1, XrArg1, 500.0__500.0, XrArg3))])
%% Environnement automate
6
enva([3, 2, 1], [3, 2, 1],
[mag(2, siga(2, XrArg1{0.000273555303651743 .. 0.000856433706871294},
0.002__0.002, XrArg3{0.00157178314656435 .. 0.00186322234817413})),
mag(1, top(0.002__0.002))],
[mag(3, sigd(siga(2, XrArg1, 0.002__0.002, XrArg3),
top(0.002__0.002)))], [])
%% Environnement automate
7
enva([2, 1], [2, 1], [mag(1, siga(1, XrArg1{12.2983738762488 .. 50.0},
500.0__500.0, XrArg3{4.90395444343343 .. 6.73512095292413}))],
[mag(2, siga(1, XrArg1{5.5 .. 22.3606797749979}, 500.0__500.0,
XrArg3{3.79680572563935 .. 5.62797223513004}))], [])

********** Affichage des sources, horloges, sorties **************

%% ----------------> Sources :
%% --> Domaines des données de test sources :
%% Numéro Automate Source
1
%% Numéro Transition dans Automate Source
1
```

```
%% Domaine Donnée de Test associée
siga(1, XrArg1{12.2983738762488 .. 50.0}, 500.0__500.0,
XrArg3{4.90395444343343 .. 6.73512095292413})

%% --> Données de test (valeurs des sources après locate) :
%% Numéro Automate Source
1
%% Numéro Transition dans Automate Source
1
%% Donnée de Test associée après locate :
siga(1, XrArg1{12.2983738762488 .. 12.2983779887692},
500.0__500.0, XrArg3{5.81953436873669 .. 5.81953526375851})

%% ----------------> Horloges :
%% --> Domaines des données de test horloges :
%% Numéro Automate Horloge
2
%% Numéro Transition dans Automate Horloge
1
%% Domaine Donnée de Test associée
top(0.002__0.002)

%% --> Données de test (valeurs des horloges après locate) :
%% Numéro Automate Horloge
2
%% Numéro Transition dans Automate Horloge
1
%% Donnée de Test associée après locate :
top(0.002__0.002)

%% ----------------> Sorties :
%% --> Domaines des sorties :
%% Numéro Automate Sortie
3
%% Numéro Transition dans Automate Sortie
1
%% Domaine Sortie associée
sigd(siga(2, XrArg1{0.000273555303651743 .. 0.000273556152122068},
0.002__0.002, XrArg3{0.00186322192393897 .. 0.00186322234817413}),
top(0.002__0.002))

****************************** Affichage total après instanciation ***************
%% Numéro automate, numéro transition testée :
3 1
```

```
%% Messages :
reception(3, 1, (4, 2,
sigd(siga(2, XrArg1{0.000273555303651743 .. 0.000273556152122068},
0.002__0.002, XrArg3{0.00186322192393897 .. 0.00186322234817413}),
top(0.002__0.002))))

emission(4, 2, (3, 1,
sigd(siga(2, XrArg1{0.000273555303651743 .. 0.000273556152122068},
0.002__0.002, XrArg3{0.00186322192393897 .. 0.00186322234817413}),
top(0.002__0.002))))

reception(4, 1, (6, 3,
sigd(siga(2, XrArg1{0.000273555303651743 .. 0.000273556152122068},
0.002__0.002, XrArg3{0.00186322192393897 .. 0.00186322234817413}),
top(0.002__0.002))))

emission(6, 3, (4, 1,
sigd(siga(2, XrArg1{0.000273555303651743 .. 0.000273556152122068},
0.002__0.002, XrArg3{0.00186322192393897 .. 0.00186322234817413}),
top(0.002__0.002))))

reception(6, 2, (5, 4,
siga(2, XrArg1{0.000273555303651743 .. 0.000273556152122068},
0.002__0.002, XrArg3{0.00186322192393897 .. 0.00186322234817413})))

emission(5, 4, (4, 1,
siga(2, XrArg1{0.000273555303651743 .. 0.000273556152122068},
0.002__0.002, XrArg3{0.00186322192393897 .. 0.00186322234817413})))

local(5, 3,
siga(2, XrArg1{0.000273555303651743 .. 0.000273556152122068},
0.002__0.002, XrArg3{0.00186322192393897 .. 0.00186322234817413}))

reception(5, 1, (7, 2,
siga(1, XrArg1{5.5 .. 5.500001839175},
500.0__500.0, XrArg3{4.7123856509426 .. 4.71238654596442})))

local(5, 2,
siga(1, XrArg1{5.5 .. 5.500001839175},
500.0__500.0, XrArg3{4.7123856509426 .. 4.71238654596442}))

emission(7, 2, (5, 1,
siga(1, XrArg1{5.5 .. 5.500001839175},
500.0__500.0, XrArg3{4.7123856509426 .. 4.71238654596442})))
```

```
reception(7, 1, (1, 1,
siga(1, XrArg1{12.2983738762488 .. 12.2983779887692},
500.0__500.0, XrArg3{5.81953436873669 .. 5.81953526375851})))

emission(1, 1, (7, 1,
siga(1, XrArg1{12.2983738762488 .. 12.2983779887692},
500.0__500.0, XrArg3{5.81953436873669 .. 5.81953526375851})))

reception(6, 1, (2, 1, top(0.002__0.002)))

emission(2, 1, (6, 1, top(0.002__0.002)))

%% Environnement automate
1
enva([1], [1], [],
[mag(1, siga(1, XrArg1{12.2983738762488 .. 12.2983779887692},
500.0__500.0, XrArg3{5.81953436873669 .. 5.81953526375851}))], [])
%% Environnement automate
2
enva([1], [1], [], [mag(1, top(0.002__0.002))], [])
%% Environnement automate
3
enva([1], [1],
[mag(1, sigd(siga(2, XrArg1{0.000273555303651743 .. 0.000273556152122068},
0.002__0.002, XrArg3{0.00186322192393897 .. 0.00186322234817413}),
top(0.002__0.002)))], [], [])
%% Environnement automate
4
enva([2, 1], [2, 1],
[mag(1, sigd(siga(2, XrArg1{0.000273555303651743 .. 0.000273556152122068},
0.002__0.002, XrArg3{0.00186322192393897 .. 0.00186322234817413}),
top(0.002__0.002)))],
[mag(2, sigd(siga(2, XrArg1, 0.002__0.002, XrArg3),
top(0.002__0.002)))], [])
%% Environnement automate
5
enva([3, 2, 1], [4, 3, 1],
[mag(1, siga(1, XrArg1{5.5 .. 5.500001839175},
500.0__500.0, XrArg3{4.7123856509426 .. 4.71238654596442}))],
[mag(4, siga(2, XrArg1{0.000273555303651743 .. 0.000273556152122068},
 0.002__0.002, XrArg3{0.00186322192393897 .. 0.00186322234817413}))],
[mag(3, siga(2, XrArg1, 0.002__0.002, XrArg3)),
mag(2, siga(1, XrArg1, 500.0__500.0, XrArg3))])
```

```
%% Environnement automate
6
enva([3, 2, 1], [3, 2, 1],
[mag(2, siga(2, XrArg1{0.000273555303651743 .. 0.000273556152122068},
0.002__0.002, XrArg3{0.00186322192393897 .. 0.00186322234817413})),
 mag(1, top(0.002__0.002))],
[mag(3, sigd(siga(2, XrArg1, 0.002__0.002, XrArg3),
top(0.002__0.002)))], [])
%% Environnement automate
7
enva([2, 1], [2, 1],
[mag(1, siga(1, XrArg1{12.2983738762488 .. 12.2983779887692},
500.0__500.0, XrArg3{5.81953436873669 .. 5.81953526375851}))],
[mag(2, siga(1, XrArg1{5.5 .. 5.500001839175}, 500.0__500.0,
XrArg3{4.7123856509426 .. 4.71238654596442}))], [])
```

Annexe E

Jeu de données de test de la carte TCBE

Ce jeu de test TDS_{carte} est composé d'un jeu de données de test TDS et d'un jeu de données de test TDS_{num}. TDS est extrait du rapport de test obtenu en appliquant la stratégie globale de couverture des transitions de la carte TCBE. TDS_{num} permet de tester la synchronisation analogique-numérique et est extrait du rapport de test obtenu en appliquant la stratégie locale « test d'une transition » à la transition No 1 de l'automate No 14 (troisième point de mesure).

$$
\begin{array}{ll}
TDS_{carte} = & TDS \bigcup TDS_{num} \\
TDS = & \{TDS_{filtre_1}, TDS_{filtre_2}, TDS_{filtre_3}, \\
 & TDS_{comp_1}, TDS_{comp_2}, TDS_{comp_3}\} \\
TDS_{filtre_i} & = \{TD1_{filtre_i}, TD2_{filtre_i}\} \\
TDS_{comp_i} & = \{TD1_{comp_i}, TD2_{comp_i}\} \\
TDS_{num} & = \{TD1_{num}\}
\end{array}
$$

$$
\begin{array}{lll}
TD1_{filtre_1} = & (In = (& siga(sinus, 5.5, 10000.0, 0.0), \_, \_, \\
 & & [top(0.002)]) \\
 & Out = (& sigd(siga(rect, 0.0000148, 0.0001, 0.0000159), 0.002), \_, \_)) \\
TD2_{filtre_1} = & (In = (& siga(sinus, 7.778, 1000.0, 0.0), \_, \_, \\
 & & [top(0.002)]) \\
 & Out = (& sigd(siga(rect, 0.000148, 0.001, 0.0000347), 0.002), \_, \_)) \\
TD1_{filtre_2} = & (In = (& \_, siga(sinus, 5.5, 10000.0, 0.0), \_, \\
 & & [top(0.002), top(0.004)]) \\
 & Out = (& \_, sigd(siga(rect, 0.0000148, 0.0001, 0.0000159), 0.004), \_)) \\
TD2_{filtre_2} = & (In = (& \_, siga(sinus, 7.778, 1000.0, 0.0), \_, \\
 & & [top(0.002), top(0.004)]) \\
 & Out = (& \_, sigd(siga(rect, 0.000148, 0.001, 0.0000347), 0.004), \_)) \\
TD1_{filtre_3} = & (In = (& \_, \_, siga(sinus, 5.5, 10000.0, 0.0), \\
 & & [top(0.002), top(0.004), top(0.006)]) \\
 & Out = (& \_, \_, sigd(siga(rect, 0.0000148, 0.0001, 0.0000159), 0.006)) \\
TD2_{filtre_3} = & (In = (& \_, \_, siga(sinus, 7.778, 1000.0, 0.0), \\
 & & [top(0.002), top(0.004), top(0.006)]) \\
 & Out = (& \_, \_, sigd(siga(rect, 0.000148, 0.001, 0.0000347), 0.006)) \\
TD1_{comp_1} = & (In = (& siga(sinus, 10.762, 498.295, 0.0), \_, \_, \\
 & & [top(0.002)]) \\
 & Out = (& sigd(siga(DC, 0.0, \_, \_), 0.002), \_, \_)) \\
TD2_{comp_1} = & (In = (& siga(sinus, 14.158, 394.881, 0.0), \_, \_, \\
 & & [top(0.002)]) \\
 & Out = (& sigd(siga(rect, 0.000224, 0.00253, 0.0000394), 0.002), \_, \_)) \\
TD1_{comp_2} = & (In = (& \_, siga(sinus, 10.762, 498.295, 0.0), \_, \\
 & & [top(0.002), top(0.004)]) \\
 & Out = (& \_, sigd(siga(DC, 0.0, \_, \_), 0.004), \_)) \\
TD2_{comp2} = & (In = (& \_, siga(sinus, 14.158, 394.881, 0.0), \_, \\
 & & [top(0.002), top(0.004)]) \\
 & Out = (& \_, sigd(siga(rect, 0.000224, 0.00253, 0.0000394), 0.004), \_)) \\
TD1_{comp_3} = & (In = (& \_, \_, sinus, 10.762, 498.295, 0.0), \\
 & & [top(0.002), top(0.004), top(0.006)]) \\
 & Out = (& \_, \_, sigd(siga(DC, 0.0, \_, \_), 0.006)) \\
TD2_{comp_3} = & (In = (& \_, \_, sinus, 14.158, 394.881, 0.0), \\
 & & [top(0.002), top(0.004), top(0.006)]) \\
 & Out = (& \_, \_, sigd(siga(rect, 0.000224, 0.00253, 0.0000394), 0.006)) \\
TD1_{num} = & (In = (& siga(sinus, 12.298, 500.0, 5.819), \\
 & & siga(sinus, 12.298, 500.0, 5.819), \\
 & & siga(sinus, 12.298, 500.0, 5.819), \\
 & & [top(0.002), top(0.004), top(0.006)]) \\
 & Out = (& sigd(siga(rect, 0.000273, 0.002, 0.00186), 0.002), \\
 & & sigd(siga(rect, 0.000273, 0.002, 0.00186), 0.004), \\
 & & sigd(siga(rect, 0.000273, 0.002, 0.00186), 0.006)))
\end{array}
$$

Bibliographie

[ABB+04] F. Azaïs, S. Bernard, Y. Bertrand, M.-L. Flottes, P. Girard, C. Landrault, L. Latorre, S. Pravossoudovitch, M. Renovell and B. Rouzeyre. *Test de circuits et de systèmes intégrés.* Hermes Science, 2004.

[ABF90] M. Abramovici, M. A. Breuer and A. D. Friedman. *Digital Systems Testing And Testable Design.* IEEE Press Wiley-Interscience, 1990.

[AHK95] B. Ayari, N. Ben Hamida and B. Kaminska. Automatic Test Vector Generation for Mixed-Signal Circuits. In *The European Design and Test Conference*, pages 458–463, 1995.

[AKS93] V. D. Agrawal, C. R. Kime and K. K. Saluja. A Tutorial on Built-In Self-Test. Part 1 : Principles. *IEEE Design & Test of Computers*, pages 73–82, March 1993.

[Arm72] D. B. Armstrong. On Deductive Method for Simulating Fault in Logic Circuits. *IEEE Transactions on Computers*, C-21(No. 5) :464–471, 1972.

[AW07] K. R. Apt and M. G. Wallace. *Constraint Logic Programming using ECLiPSe.* Cambridge University Press, 2007.

[BA00] M. L. Bushnell and V. D. Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits.* Springer, 2000.

[BR01] M. Burns and G. W. Roberts. *An Introduction to Mixed-Signal IC Test and Measurement.* Oxford University Press, 2001.

[But02] K. M. Butler. Is ITC Bored with Board Test ? In *International Test Conference*, 2002.

[BZ83] D. Brand and P. Zafiropulo. On Communicating Finite-State Machines. *Journal of Association for Computing Machinery*, 30(2) :323–342, April 1983.

[Cas76] G. R. Case. Analysis of Actual Fault Mechanisms in CMOS LogicGates. In *Proc. of the 13th Design Automation Conference*, pages 265–270, 1976.

[Che96] K.-T. Cheng. Tutorial and Survey Paper : Gate-Level Test Generation for Sequential Circuits. In *ACM Transactions on design Automation of Electronic Systems*, volume 1, pages 405–442, October 1996.

[CK93] K.-T. Cheng and A.S. Krishnakumar. Automatic Functional Test Generation Using The Extended Finite State Machine Model. In *30th ACM/IEEE Design Automation Conference*, pages 86–91, 1993.

[DO91] R.A. Demillo and A.J. Offutt. Constraint Based Automatic Test Data Generation. *IEEE Transactions on Software Engineering*, 17(9) :900–910, 1991.

[DR79] P. Duhamel and J.-C. Rault. Automatic Test Generation Techniques for Analog Circuits and Systems : A Review. *IEEE Transactions Circuits and Systems*, CAS26(7) :411–440, 1979.

[ea03] P. Brisset et al. *ECLiPSe Constraint Library Manual.* IC-Parc, Imperial College London, 2003.

[EBP02] B. Eklow, C.F Barnhart and K. Parker. IEEE P1149.6 : A Boundary-Scan Standard for Avanced Digital Networks. In *International Test Conference*, pages 1056–1065, 2002.

[EBRB03] B. Eklow, C.F Barnhart, M. Richetti and T. Borroz. IEEE 1149.6 : A Pratical Perspective. In *International Test Conference*, 2003.

[Ehr07] H. Ehrenberg. IEEE P1581 Can Solve Your Board Level Memory Cluster Test Problems. In *International Test Conference*, 2007.

[Ekl02] B. Eklow. Is Board Test Worth Talking About ? In *International Test Conference*, 2002.

[Ekl04] B. Eklow. What Do You Mean My Board Test Stinks ! ? In *International Test Conference*, 2004.

[ERT06] H. Ehrenberg, B. Russel and B. Van Treuren. IEEE P1581 - Getting More Board Test Out of Boundary Scan. In *International Test Conference*, 2006.

[FMP05] F. Fummi, C. Marconcini and G. Pravadelli. An EFSM-based Approach for Functional ATPG. In *GLSVLSI'05*, 2005.

[Fuj85] H. Fujiwara. *Logic Testing and Design for Testability.* The MIT Press, 1985.

[GBR98] A. Gotlieb, B. Botella and M. Rueher. Automatic test data generation using constraint solving techniques. *SIGSOFT Softw. Eng. Notes*, 23(2) :53–62, 1998.

[GFMP06] G. Di Guglielmo, F. Fummi, C. Marconcini and G. Pravadelli. EFSM Manipulation to Increase High-Level ATPG Effectiveness. In *International Symposium on Quality Electronic Design (ISQED'06)*, 2006.

[GKPvC85] F. Giannesini, H. Kanoui, R. Pasero and M. van Caneghem. *Prolog.* Inter-Editions, 1985.

[GNN07a] B. Gilles, L. Nana and V.-A. Nicolas. Implementing an Automatic Functional Test Pattern Generation for Mixed-Signal Boards in a Maintenance Context. In *Proc. of the 5th IEEE International East-West Design and Test Symposium (EWDTS'07)*, Yerevan, Armenia, September 2007.

[GNN07b] B. Gilles, L. Nana and V.-A. Nicolas. Modeling and Generation of Test Patterns for Mixed-Signal Boards : Dealing With Basic Signals. In *Proc. of the 6th IEEE International Board Test Workshop (BTW'07)*, Fort Collins, Colorado USA, September 2007.

[Goe81] P. Goel. An Implicit Enumeration Algorithm To Generate Test for Combinational Logic Circuits. *IEEE Transactions of Computers*, pages 215–222, March 1981.

[Gol82] S.W. Golomb. *Shift Register Sequences*. Aegean Park Press, 1982.

[Har03] I. G. Harris. Fault Models and Test Generation for Hardware-Software Covalidation. *IEEE Design & Test of Computers*, pages 40–47, July-August 2003.

[Hay85] J. P. Hayes. Fault Modeling. *IEEE Design & Test of Computers*, pages 88–95, 1985.

[Hie01] R. M. Hierons. Checking States and Transitions of a Set of Communicating Finite State Machine. *Microprocessors and Microsystems*, 24(9) :443–452, February 2001.

[HK93a] N. B. Hamida and B. Kaminska. Analog Circuit Testing Based on Sensitivity Analysis. In *International Test Conference*, pages 652–661, October 1993.

[HK93b] N. B. Hamida and B. Kaminska. Multiple Fault Analog Circuit Testing by Sensitivity Analysis. *Journal of Electronic Testing : Theory and Applications*, 4(4), 1993.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Therory, Languages and Computation*. Addison Wesley, 1979.

[IEE90] IEEE. *IEEE STANDARD 1149.1-1990, IEEE Standard Test Access Port and Boundary-Scan Architecture*, 1990.

[IEE93] IEEE. *IEEE STANDARD 1076-1993, IEEE Standard VHDL Language Reference Manual*, 1993.

[IEE94] IEEE. *IEEE STANDARD 1149.1-1994 (revision b), IEEE Standard Test Access Port and Boundary-Scan Architecture*, 1994.

[IEE99] IEEE. *IEEE STANDARD 1149.4-1999, IEEE Standard for a Mixed Signal Test Bus*, 1999.

[IEE01a] IEEE. *IEEE STANDARD 1364-2001, IEEE Standard Verilog Language Reference Manual*, 2001.

[IEE01b] IEEE. *IEEE STANDARD 1666-2005, IEEE Standard System C Language Reference Manual*, 2001.

[IEE03] IEEE. *IEEE STANDARD 1149.6-2003, IEEE Standard for Boundary-Scan Testing of Advanced Digital Networks*, 2003.

[ILO02] ILOG. *ILOG Views Fundation 5.0 User's Manual*, 2002.

[JBR+08] Y. Joannon, V. Beroulle, C. Robach, S. Tedjini and J.L. Carbonero. Decreasing Test Qualification Time in AMS and RF Systems. *IEEE Design & Test of Computers*, pages 29–37, 2008.

[KM98] T. Kirland and R. Mercer. Algorithms for Automatic Test Pattern Generation. *IEEE Design & Test of Computers*, pages 43–55, June 1998.

[Kor90] B. Korel. Automated Software Test Data Generation. *IEEE Transactions on Software Engineering*, 17(6) :591–603, 1990.

[Kun88] M. Kunt. *Traitement numérique des signaux.* Dunod, 1988.

[Lab] LabVIEW. http ://www.ni.com/labview.

[LY96] D. Lee and M. Yannakakis. Principles and Methods of Testing Finite State Machines–A Survey. In *IEEE Transactions on Computers*, volume 84, August 1996.

[MA97] A. K. Majhi and V. D. Agrawal. Tutorial : Delay Fault Models and Coverage. In *Proc. of 11th International Conference on VLSI Design : VLSI for Signal Processing*, pages 364–369, 1997.

[Mah87] M. V. Mahomey. *DSP-Based Testing of Analog and Mixed-Signal Circuits.* IEEE Computer Society Press, 1987.

[MAT] MATRIXx. http ://www.ni.com/matrixx.

[Mat03a] The MathWorks, Inc. *Using Simulink - Model-Based and System-Based Design*, 2003.

[Mat03b] The MathWorks, Inc. *Writing S-Functions - Model-Based and System-Based Design*, 2003.

[MJR86] Y. K. Malaiya, A. P. Jayasumana and R. Rajsuman. A Detailed Examination Of Bridging Faults. In *International Conference on Computer Design*, pages 78–81, 1986.

[Nel03] R. Nelson. Is functional test necessary ? *Test & Measurement World*, January 2003.

[NGN08] V.-A. Nicolas, B. Gilles and L. Nana. Validation of a Mixed-Signal Board ATPG Method. In *Proc. of the 6th IEEE International East-West Design and Test Symposium (EWDTS'08)*, Lvov, Ukraine, October 2008.

[NS97] R. Nikoukhah and S. Steer. SCICOS A Dynamic System Builder and Simulator. Technical Report 0207, INRIA, 1997.

[NSSS04] S. Novello, J. Schimpf, K. Shen and J. Singer. *ECLiPSe : Embedding and Interfacing Manual.* IC-Parc, Imperial College London, 2004.

[P1501] IEEE P158. IEEE P1581 Static Component Interconnection Test Protocol and Architecture http ://grouper.ieee.org/groups/1581/, 2001.

[Par01] K. P. Parker. Boundary-Scan Testing of AC Coupled Nets. In *International Test Conference*, page 1188, 2001.

[RB96] R. Ramadoss and M. L. Bushnell. Test Generation for Mixed-Signal Devices Using Signal Flow Graphs. In *9th International Conference on VLSI Design : VLSI in Mobile Communication*, pages 242–247, January 1996.

[RHB95] M. Renovell, P. Huc and Y. Bertrand. The Concept of Resistance Interval : A New Parametric Model for Realistic Resistive Briding Fault. In *Proc. of the 13th IEEE VLSI Test Symposium*, pages 184–189, 1995.

[Riv94] A. Rivat. *Logiciel de simulation analogique PSPICE 5.30.* Dunod Tech, 1994.

[RMBF92] R. Rodriguez-Montanes, E.M.J.G. Bruls and J. Figueras. Bridging Defects Resistance Measurements in a CMOS Process. In *Proc. of International Test Conference*, pages 892–899, 1992.

[Rob96] Gordon W. Roberts. Metrics, Techniques and Recent Developpements in Mixed-Signal Testing. In *International Conference on Computer-Aided Design*, pages 514–521, 1996.

[Rot66] J.P Roth. Diagnosis of automata failures : A calculus and a method. *IBM Journal Research and Development*, pages 278–290, July 1966.

[Ses65] S. Seshu. On an Improved Diagnosis Program. *IEEE Transactions on Electronic Computers*, EC-14 :76–79, 1965.

[Tui88] P.W. Tuinenga. *SPICE A Guide to Circuit Simulation & Analysis Using PSpice.* Prentice Hall, 1988.

[Vin98] B. Vinnakota, editor. *Analog and Mixed-Signal Test.* Prentice Hall, 1998.

[Wad78] R.L. Wadsack. Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits. *Bell System Technical Journal*, 57(5) :1449–1474, May-June 1978.

[Whe03] L. Whetsel. Adapting JTAG for AC Interconnect Testing. In *International Test Conference*, 2003.

[Wil02] D. J. Wilkins. The Bathtub Curve and Product Failure Behavior. Part One - The Bathtub Curve, Infant Mortality and Burn-in. *Reliability HotWire*, November 2002.

[WNS97] M. Wallace, S. Novello and J. Schimpf. Eclipse : A platform for constraint logic programming, 1997.

Résumé

Le problème abordé dans cette thèse concerne le test de cartes mixtes en phase de maintenance. Dans le domaine du test matériel, de nombreuses méthodes et outils de test existent, ciblant principalement le test de circuits en phase de conception et de production. Peu d'intérêt a été porté jusqu'à présent au test de cartes mixtes en phase de maintenance. Pourtant, certains systèmes comme par exemple les systèmes militaires et avioniques doivent rester opérationnels pendant plusieurs décennies. Il est alors important de s'assurer que les fonctionnalités des cartes électroniques composant ces systèmes ne se dégradent pas au cours du temps. D'autre part, lorsqu'une carte est en panne et qu'il est nécessaire de la réparer alors que les lots de rechanges sont épuisés, une aide au diagnostic s'avère précieuse.

Nous proposons une méthodologie de test fonctionnel adaptée au contexte de la maintenance. Cette méthodologie permet une modélisation fonctionnelle uniforme des composants analogiques, numériques et mixtes de la carte à tester tout en étant flexible vis-à-vis de la quantité d'informations disponibles sur la carte. La génération des données de test est pilotée par des stratégies de test globales (bien adaptées à la maintenance préventive) ou locales (plus appropriées dans le cas de la maintenance corrective). L'expertise et les pratiques industrielles des ingénieurs de test en maintenance, qui se révèlent indispensables, sont prises en compte par la méthodologie sous la forme de modèles de test et de tactiques de test qui précisent le processus de génération des données de test. La méthodologie proposée est implantée dans un outil prototype en utilisant la programmation logique par contraintes, et son application sur quelques exemples de cartes mixtes est discutée.

Mots clés

Test en maintenance, test fonctionnel, génération automatique de jeux de test, cartes mixtes, modélisation fonctionnelle.