ANALYSE DE SÛRETE DES CIRCUITS COMPLEXES DECRITS EN LANGAGE DE HAUT NIVEAU

A. Ammari

▶ **To cite this version:**

A. Ammari. ANALYSE DE SÛRETE DES CIRCUITS COMPLEXES DECRITS EN LANGAGE DE HAUT NIVEAU. Micro et nanotechnologies/Microélectronique. Institut National Polytechnique de Grenoble - INPG, 2006. Français. NNT : . tel-00101622

HAL Id: tel-00101622

https://theses.hal.science/tel-00101622

Submitted on 27 Sep 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
|__/__/__/__/__/__/__/__/__/__|

T H E S E

Pour obtenir le grade de

DOCTEUR DE l'INPG

Spécialité : Micro et Nano Electronique

Préparée au laboratoire **TIMA** dans le cadre de **l'Ecole Doctorale d'« Electronique, Electrotechnique, Automatique, Télécommunications, Signal »**

Présentée et soutenue publiquement

Par

Abdelaziz AMMARI

Le 31 août 2006

Titre :

ANALYSE DE SÛRETE DES CIRCUITS COMPLEXES DECRITS EN LANGAGE DE HAUT NIVEAU

Directeur de Thèse : Régis LEVEUGLE

JURY

M. Pierre GENTIL	, Président
M. Abbas DANDACHE	, Rapporteur
M. Michel RENOVELL	, Rapporteur
M. Régis LEVEUGLE	, Directeur de thèse
M. Jean-Pierre SCHOELLKOPF	, Examinateur

Dédicace

A ma mère, à mon père

Ce manuscrit de thèse représente le fruit de vos années d'investissement. Je suis fier de vous le dédier, veuillez y trouvez le témoignage de ma grande reconnaissance. Merci d'avoir contribué à ma réussite.

Je vous souhaite la bonne santé et que dieu vous garde.

A Aïcha, Raed et Raja.

Je vous souhaite un avenir radieux et plein de gloire. Je vous dédie ce travail en témoignage de ma grande affection.

Remerciements

Je souhaite m'adresser en premier lieu aux membres du jury de thèse. Je tiens à remercier Monsieur Abbas Dandache, Professeur à l'université de Metz, et Monsieur Michel Renovell, Directeur de recherche CNRS au Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM) d'avoir accepté de rapporter sur mon travail, pour l'intérêt qu'ils ont porté à ce projet ainsi que pour leurs commentaires et suggestions.

J'adresse aussi mes remerciements à Monsieur Pierre Gentil, Professeur à l'Institut National Polytechnique de Grenoble (INPG), pour m'avoir fait l'honneur de présider mon jury de thèse. Je remercie aussi Jean-Pierre Schoellkopf, chef du département ADT à STMicroelectronics Crolles, pour m'avoir fait l'honneur de faire partie de ce jury.

La présente thèse a été effectuée au laboratoire TIMA sur le site Viallet de l'INPG. Je remercie Monsieur Bernard Courtois, directeur de ce laboratoire, pour m'y avoir accueilli.

Je tiens tout particulièrement à remercier mon directeur de thèse Monsieur Régis Leveugle, Professeur à l'INPG, pour m'avoir encadré tout au long de ces travaux sur un sujet porteur et passionnant. Je lui dois des voies de recherches originales, ainsi qu'une formation scientifique et méthodologique de qualité. Toute ma reconnaissance pour son soutien et sa patience qu'il a su m'accorder pour mener à bien ce manuscrit.

Je tiens aussi à remercier Laurent Fesquet, Maître de conférences à l'INPG pour ces encouragements. Merci Laurent pour ces heures d'enseignement.

Je voudrais également remercier les administrateurs du CIME, Alexandre Chagoya, Bernard Bosc et Robin Rolland, et tout le personnel administratif du laboratoire TIMA. Tous mes remerciements à Kholdoun Torki pour son aide et à Jean-François Paillotin.

Je souhaite maintenant adresser toute ma sympathie à mes amis et collègues du laboratoire TIMA. C'est à eux que je dois des moments inoubliables, passés dans un cadre amical, encourageant et enrichissant. Un grand merci donc à Abdelmajid, Achraf, Ahcene, Aimen, Amel, Diana, Faiza, Karim, Lobna, Marius, Michele, Mohamed, Nacer, Pierre, Susi, Vincent et Wassim.

Un grand merci aux familles Ben Ahmed, Chérif et Zalama qui m'ont toujours encouragé et qui ont facilité mon séjour en France.

Que celles et ceux que j'aurais oubliés ici me pardonnent.

Enfin, je ne remercierais jamais assez mes chers parents, mon frère et mes deux soeurs pour leur soutien tout au long de ce travail de longue haleine. Ma pensée va également à l'ensemble de ma famille, petits et grands, ainsi qu'à tous mes proches.

Table des matières

Liste des figures

Liste des tableaux

Introduction

L'évolution actuelle des technologies CMOS (réduction des tailles des transistors, réduction des capacités des nœuds, diminution des tensions d'alimentation, augmentation de la fréquence d'horloge…) a, mis à part des bénéfices incontestables, un impact très important sur la fiabilité des circuits intégrés. En effet ceux-ci deviennent de plus en plus sensibles vis-à-vis de leur environnement. La probabilité d'apparition de fautes transitoires dues notamment aux particules chargées est de plus en plus grande. Ainsi les mémoires (SRAM ou DRAM) des nouvelles technologies sont plus sensibles aux erreurs de type SEU (inversion de bits occasionnée dans les points mémoires internes d'un circuit par des rayonnements ou des impacts de particules). Pour confirmer cette tendance, des mesures en accélérateur ont été réalisées sur des mémoires SRAM fabriquées en technologie 0,25µm et 0,18µm. Il apparaît que les particules alpha provoquent un taux d'erreurs 30 fois plus grand lorsqu'on passe de la technologie 0,25µm à la technologie 0,18µm.

Jusqu'à récemment les particules chargées n'avaient d'impact que sur les circuits évoluant dans des environnements hostiles (espace, centrale nucléaire…). Ainsi le premier satellite perdu à cause des particules chargées est Telstar [T.M. 89]. Celui-ci fut lancé le 10 juillet 1962, soit un jour après Starfish, qui avait pour but le test d'une arme nucléaire à haute altitude par les Etats-Unis d'Amérique. Cette explosion a produit des particules β (électrons), causant une radiation très intense qui a duré jusqu'au début des années 1970. Telstar et six autres satellites ont été perdus pendant les sept mois suivant ce test. De nos jours, et à cause de l'évolution des technologies, les incidents dus aux particules, surtout les neutrons atmosphériques ne cessent d'augmenter au niveau du sol. Ainsi, la société SUN Microsystems a admis en 2000 que plusieurs crashs de systèmes ont été observés sur les serveurs d'entreprise provenant des impacts cosmiques sur les mémoires caches [R.B. 02]. Cypress a annoncé en 2004 qu'une seule erreur transitoire et non destructrice (*Single Soft Error*) a amené une usine automobile d'un milliard de dollars à s'arrêter une fois par mois [J.Z. 04].

En plus des points mémoires, les parties combinatoires sont de plus en plus sensibles. Ceci est dû principalement aux temps de commutation des portes qui deviennent de plus en plus petits, et à l'augmentation de la fréquence d'horloge dans les systèmes synchrones. Ainsi une impulsion transitoire apparaissant dans un bloc combinatoire (appelés *Single Event Transients* ou SET) peut se propager, sous certaines conditions, et résulter en une ou plusieurs valeurs erronées sur les signaux de sortie d'un bloc lorsque des bascules échantillonnent ces signaux. Et comme la fréquence augmente, ceci rend la probabilité de mémorisation de telles erreurs de plus en plus grande.

Les circuits analogiques ne sont pas épargnés par ce phénomène, et pourtant très peu de travaux traitent du sujet bien que les systèmes destinés à l'environnement spatial, biomédical et aéronautique contiennent une partie analogique servant d'interface pour collecter les données.

Tenant compte de tous ces phénomènes, l'ITRS (*International Technology Roadmap for Semiconductors*) prévoit que les conséquences des SEUs et SETs représentent un défi important

pour les circuits VLSI avec des technologies en dessous de 90nm. Par conséquent, l'évaluation du comportement d'un circuit en présence de fautes et l'amélioration de la robustesse quand ceci est nécessaire constituent deux des principaux centres d'intérêt de l'industrie des semiconducteurs.

La phase d'évaluation est nécessaire pour savoir quelles parties du circuit doivent intégrer des mécanismes de détection ou de tolérance afin d'améliorer la robustesse. Habituellement, ces mécanismes se retrouvent dans les circuits évoluant dans des environnements hostiles (espace, centrale nucléaire…) et/ou dans des circuits pour des applications critiques (domaine militaire, aéronautique, biomédical…) où la conséquence d'un SEU peut être catastrophique. Dans un proche futur, et à cause de l'évolution de la technologie CMOS, les circuits grands publics (ordinateurs portables, téléphones portables, agenda électronique….), devront aussi intégrer des mécanismes de détection ou de tolérance. Il devient donc obligatoire d'analyser le plus tôt possible les conséquences des fautes afin (1) d'identifier les nœuds qui devraient être protégés, en minimisant le matériel dédié aux mécanismes de protection vis-à-vis du degré de protection visé, et (2) de valider l'efficacité de ces mécanismes. Une telle analyse doit être réalisée le plus tôt possible dans le processus de conception, et dans tous les cas avant la fabrication du premier circuit, afin de réduire le coût des itérations dans le processus de conception. Pour cela, il est possible d'utiliser une approche fondée sur l'injection de fautes dans des descriptions de haut niveau (classiquement, des descriptions VHDL au niveau RTL). Ces injections de fautes peuvent être réalisées soit par simulation soit par exécution sur un prototype matériel. L'analyse des résultats d'injection de fautes peut être de deux types : (1) une analyse classique appelée classification de fautes, identifiant les fautes menant à des évènements particuliers, ou (2) une analyse de propagation d'erreurs, montrant les configurations erronées internes activées par les fautes.

Dans ce contexte, deux aspects complémentaires sont considérés dans le cadre de la thèse : l'injection de fautes et l'analyse des résultats obtenus à l'issue de la campagne d'injection.

Le Chapitre 1 du manuscrit présente les notions de base relatives aux fautes, à leur modélisation et aux injections de fautes. Nous commencerons par introduire la terminologie relative à la sûreté de fonctionnement d'un circuit numérique. Ensuite nous présenterons les problèmes rencontrés dans les technologies sous-microniques. Parmi ces problèmes, nous détaillerons un peu plus l'environnement radiatif et ses effets sur les circuits intégrés. Par la suite, nous présenterons les différentes approches d'injections de fautes en mettant l'accent sur les injections dans les descriptions comportementales. Et pour finir nous détaillerons le flot d'analyse développé au sein du groupe, et présenterons la méthodologie adoptée pour effectuer les injections de fautes et l'analyse des résultats obtenus.

Le Chapitre 2 expose mes contributions sur le flot d'analyse présenté dans le Chapitre 1. Nous commencerons par indiquer les limitations de ce flot, puis les solutions apportées pour pallier ces limitations. Après quoi nous nous intéresserons à la modélisation d'un circuit dans

son environnement. Ceci a pour but d'analyser avec plus de précision les conséquences des fautes injectées dans un circuit numérique faisant partie d'un système complexe.

Dans le Chapitre 3, des injections de fautes dans un microprocesseur 8051 sont analysées. Nous considérons des descriptions de circuits et des modèles de fautes multi niveaux. La première partie sera consacrée à la comparaison entre les niveaux portes et RTL. A l'issue de cette comparaison, notre choix s'orientera vers le niveau RTL pour la suite des analyses. La seconde partie traite de l'intérêt de combiner la classification et l'analyse de propagation d'erreur au niveau RTL. La dernière partie évalue par injection de fautes une technique de détection d'erreurs logicielle.

Le Chapitre 4 propose enfin un flot d'analyse unifié pour circuits numériques, analogiques et mixtes. Un exemple d'étude décrit à haut niveau d'abstraction sera présenté puis exploité pour y injecter des fautes, en se basant sur le flot développé. Les résultats des injections seront par la suite comparés à ceux obtenus au niveau transistor afin de valider ce flot.

Chapitre 1.

Fautes, modélisation et injection de fautes

1.1. Terminologie

Les notions présentées ici sont extraites de [J.L. 88] et [A.A. 04]. Elles permettront au lecteur de bien identifier la terminologie employée dans cette thèse.

Le service délivré par un système est défini comme étant son comportement tel qu'il est perçu par son, ou ses, utilisateurs (humains ou physiques) interagissant avec lui. Deux cas de figures sont possibles : le service est soit correct, soit défaillant. Le service correct correspond au cas où le service implémente la fonction du système telle qu'elle est décrite par la spécification fonctionnelle en termes de fonctionnalité et de performance. La défaillance correspond au cas où le service délivré dévie des conditions établies dans la spécification.

Puisqu'un service est une série d'états externes du système, une défaillance de service signifie qu'au moins un état externe du système dévie de l'état du service correct. Cette déviation s'appelle erreur. Une faute est la cause supposée ou adjugée d'une erreur. Dans la plupart des cas, une faute cause d'abord une erreur dans l'état d'un composant interne du système et l'état externe n'est pas immédiatement affecté. Pour cette raison, la définition d'une erreur est un état, par rapport au processus de traitement, susceptible de conduire à une défaillance. Il est important de noter que beaucoup d'erreurs ne conduisent pas à des défaillances. Une faute est active lorsqu'elle cause une erreur, autrement elle est dormante.

Une faute peut être classée suivant huit classes élémentaires (Figure 1-1) :

- La phase de vie du système pendant laquelle la faute se produit :
  - Fautes de développement qui se produisent pendant (1) le développement du système, (2) la maintenance pendant la phase d'utilisation, (3) la génération de procédures pour actionner ou faire la maintenance du système.
  - Fautes de fonctionnement qui se produisent durant la délivrance du service.
- L'emplacement de la faute par rapport aux limites du système :
  - Fautes internes qui naissent à l'intérieur des limites du système.
  - Fautes externes qui naissent à l'extérieur du système et propagent des erreurs à l'intérieur du système par interaction ou propagation.
- La cause de la faute :
  - Fautes naturelles qui sont causés par des phénomènes naturels sans participation de l'homme.
  - Fautes fabriquées par l'homme qui résultent des actions faites par l'homme.
- La dimension dans laquelle la faute se produit :
  - Fautes de type matériel (physique) qui se produisent dans la partie matérielle du système.
  - Fautes de type logiciel (information) qui affectent la partie logicielle du système c'est-à-dire les programmes ou données.

- Le but d'introduction des fautes :
  - Fautes malveillantes qui sont introduites avec le but de nuire au système.
  - Fautes non malveillantes.
- L'intention de l'homme qui cause la faute :
  - Fautes intentionnelles qui résultent d'une décision nuisible.
  - Fautes non intentionnelles qui sont introduites sans en prendre conscience.
- La capacité de l'homme qui introduit la faute :
  - Fautes accidentelles qui sont introduites par inadvertance.
  - Fautes d'incompétence qui résultent d'une incompétence ou d'un manque d'organisation.
- La persistance temporelle de la faute :
  - Fautes permanentes dont la présence est continue dans le temps.
  - Fautes transitoires dont la présence est limitée dans le temps.

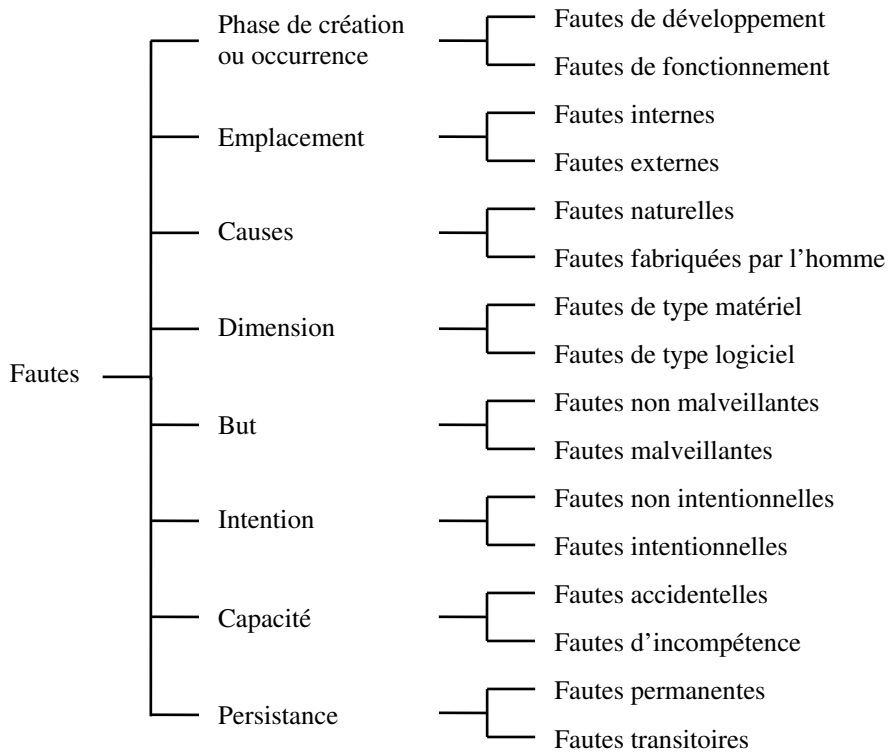


Figure 1-1: Les classes de fautes élémentaires [A.A. 04]

La sûreté de fonctionnement d'un système est la propriété qui permet à ses utilisateurs de placer une confiance justifiée dans le service qu'il délivre. Cette définition souligne le besoin de la justification de confiance. Une définition alternative fournit le critère pour décider si le service est sûr : la sûreté de fonctionnement d'un système est la capacité d'éviter les erreurs de service qui sont plus fréquentes et plus sévères que celles acceptables pour l'utilisateur.

La sûreté de fonctionnement englobe les attributs suivants :

- Disponibilité : mesure de la délivrance d'un service approprié par rapport à l'alternance service approprié - service inapproprié.
- Fiabilité : mesure de la délivrance continue d'un service approprié, ou, ce qui est équivalent, mesure du temps jusqu'à défaillance.
- Sûreté : absence de conséquences catastrophiques pour l'utilisateur et l'environnement.
- Confidentialité : absence de divulgation non autorisée d'informations.
- Intégrité : absence d'altérations du système.
- Maintenabilité : habilité à subir des modifications ou des réparations.

La sécurité est la présence concurrente de (1) la disponibilité pour les personnes autorisées uniquement, (2) la confidentialité et (3) l'intégrité.

Plusieurs méthodes ont été développées pour garantir la sûreté de fonctionnement. Celles-ci peuvent être classées en quatre grandes catégories :

- Prévention des fautes : comment empêcher, par construction, l'occurrence ou l'introduction de fautes.
- Tolérance aux fautes : comment fournir, par redondance, un service conforme à la spécification en dépit des fautes.
- Elimination des fautes : comment minimiser, par vérification, la présence de fautes.
- Prévision des fautes : comment estimer la présence, la création, et les conséquences de fautes.

Prévention des fautes et élimination des fautes peuvent être vues comme constituant l'évitement des fautes : comment tendre vers un système exempt de fautes. Evitement des fautes et tolérance aux fautes peuvent être vus comme constituant l'obtention de la sûreté de fonctionnement : comment procurer au système l'aptitude à délivrer un service conforme à la spécification. Élimination des fautes et prévision des fautes peuvent être regroupées dans la validation de la sûreté de fonctionnement : comment avoir confiance dans l'aptitude du système à délivrer un service conforme au service spécifié.

Le schéma complet de la taxonomie de la sûreté de fonctionnement est donné à la Figure 1-2.

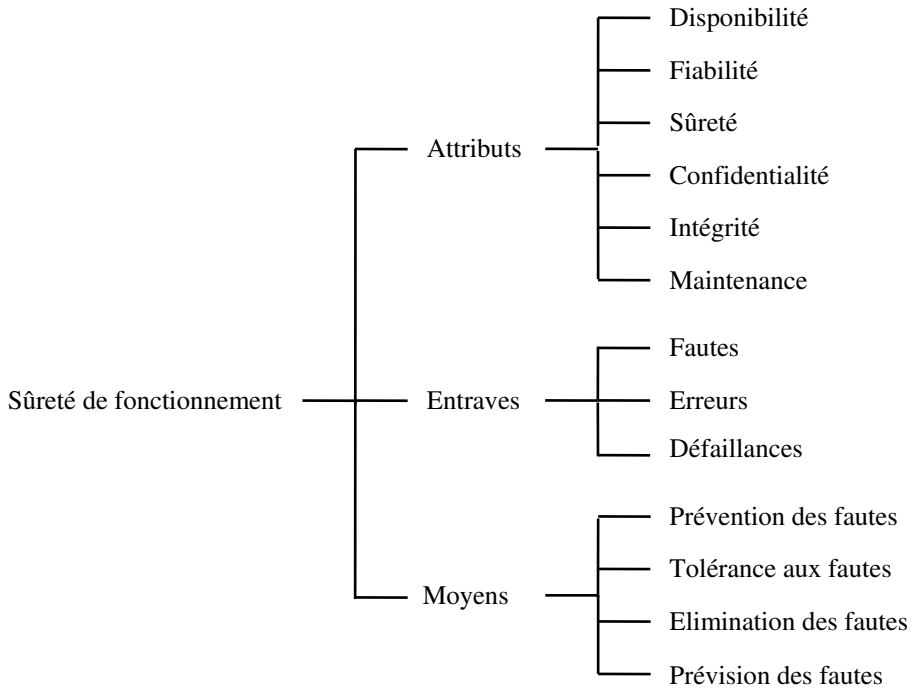


Figure 1-2: Taxonomie de la sûreté de fonctionnement [A.A. 04]

1.2. Les problèmes rencontrés dans les technologies sous-microniques

L'évolution des technologies sous-microniques nécessite la réduction de la tension d'alimentation (V_{DD}) et de la tension de seuil (V_T), en permettant l'augmentation de la vitesse de fonctionnement et du nombre de transistors dans une même puce (ces transistors étant de plus en plus petits). En contrepartie cette évolution rend les circuits intégrés de plus en plus sensibles à toute source de bruit : phénomènes de couplage capacitif (en anglais *cross-talks*), influence électromagnétique, bruit sur les lignes d'alimentation (*ground-bounce*), ainsi qu'aux phénomènes radiatifs (particules alpha, neutrons atmosphériques…). Ce sont ces derniers phénomènes qui nous intéressent particulièrement dans le cadre de cette thèse. Initialement, ces phénomènes n'avaient de l'influence sur les circuits intégrés que lorsque ceux-ci étaient destinés à des applications spatiales ou nucléaires. Mais avec ces nouvelles technologies, les circuits sont de plus en plus sensibles même au niveau du sol [E.N. b].

Cette sensibilité accrue s'explique par le fait qu'en diminuant la tension d'alimentation et la taille des transistors, les capacités des nœuds du circuit diminuent aussi et la charge électrique stockée dans un noeud est considérablement réduite. La charge déposée par une particule chargée lors de son impact sur le silicium peut donc plus facilement renverser la valeur logique associée au noeud. Ainsi si la particule affecte un nœud d'une cellule mémoire, son état logique peut basculer, résultant en une erreur transitoire appelée SEU pour *Single Event Upset* (1.4.2.2). Par contre si la particule affecte un nœud d'un bloc combinatoire, l'impulsion ainsi créée et dénommée SET (*Single Event Transient* voir 1.4.2.3) n'est plus filtrée par les portes logiques et se propage jusqu'à sa capture probable par des éléments mémoires. Cette probabilité est d'autant plus grande que la fréquence d'horloge des systèmes actuels est de plus en plus grande.

Les grandes compagnies du domaine des semi-conducteurs se préoccupent de plus en plus de tous les aspects liés à ces problèmes. Selon Intel, les erreurs transitoires dues aux particules cosmiques deviennent le deuxième grand problème après l'augmentation des courants de fuites dans le domaine sous-micronique [A.C. 99]. La plupart des compagnies observent une augmentation importante des erreurs transitoires à partir de la technologie 0,25 µm. Ces erreurs correspondent à l'ensemble des cas conduisant à des bits erronés dans le circuit sans effet destructeur.

1.3. L'environnement radiatif

Cette section présente les deux principales sources de rayonnement susceptibles de produire des erreurs dans le circuit intégré : les rayons cosmiques et les particules alpha.

1.3.1. Les rayons cosmiques

Les notions présentées ici sont extraites de [R.M. 96], [C.R. a] et [C.R. b].

Les rayons cosmiques sont des flots de particules chargées extrêmement énergétiques (certains ions peuvent atteindre 10^{21} eV) se déplaçant à peu près à la vitesse de la lumière et bombardant la terre au-delà de son atmosphère. Ces particules (dites primaires) sont produites par différentes sources comme le soleil, les étoiles… Ces rayons cosmiques ont été découverts par V. Hess en 1912 grâce à des mesures effectuées à partir de ballons sondes, et c'est ce qui lui a valu le prix Nobel en 1936. Ces rayons recouvrent tous les éléments du tableau périodique et sont principalement constitués de protons (entre 85 et 90%), de noyaux d'hélium ou hélions (de 9 à 14%), le reste étant constitué d'électrons, de différents nucléons (noyaux d'atomes) et autres particules. Les ions lourds (numéro atomique $Z \geq 3$) ne représentent que 1% du flux total.

Il y a trois types de rayons cosmiques :

- Les rayons cosmiques galactiques (*Galactic Cosmic Rays*) : Ils proviennent de l'extérieur du système solaire. La plupart ont une énergie comprise entre 100MeV et 10GeV. Bien que leur origine soit inconnue, il est probable qu'ils proviennent d'explosions de supernova et/ou de processus de fusion stellaires.
- Les rayons cosmiques solaires (*Solar Cosmic Rays*) : Les SCRs sont des particules relativement peu énergétiques dont le flux est modulé par l'activité solaire. Elles sont principalement associées aux éruptions solaires (explosions énormes de courte durée sur la surface du soleil). Les éjections de masses coronales (énormes bulles de gaz éjectées depuis le soleil) produisent aussi des particules énergétiques.
- Les rayons cosmiques anormaux (*Anomalous Cosmic Rays*) : Ils sont constitués d'éléments difficiles à ioniser comme He, N, O, Ne, et Ar. Les ACRs sont dus aux particules interstellaires neutres pénétrant le système solaire. Ils ne sont pas affectés par le champ magnétique du vent solaire (plasma de particules chargées provenant du soleil). Ces particules interstellaires sont ionisées, et puis accélérées au niveau de l'onde

de choc provoquée par le ralentissement soudain du vent solaire approchant l'héliopause (situé à une distance de 75 à 100 fois la distance terre-soleil).

Certaines particules (les plus énergétiques) sont peu pénétrantes dans les orbites basses terrestres : le champ magnétique terrestre et les ceintures de radiations (appelées ceintures de Van Allen et constituées de particules légères piégées) constituent un bouclier naturel. D'autres particules (1%) atteignent la terre sans interaction. Les rayons cosmiques secondaires se produisent lorsqu'un rayon cosmique entre dans l'atmosphère supérieure, qu'il se heurte à une particule, habituellement un azote ou une molécule d'oxygène, produisant une douche de particules d'énergie inférieures comme les neutrons atmosphériques. Plusieurs de ces particules sont absorbées par l'atmosphère terrestre, tandis que d'autres atteignent la surface de la terre. Les particules prédominantes sont les muons, pions, neutrons et protons. En raison de leur flux et stabilité élevés, les neutrons d'énergie élevée (> 1MeV) sont la source principale des erreurs non destructrices dus aux rayons cosmiques [O.A. 84].

1.3.2. Les particules alpha

Découvertes en 1899 par Ernest Rutherford, les particules alpha ou rayons alpha sont une forme de rayonnement émis par des particules hautement ionisées et peu pénétrantes. Elles sont constituées de deux protons et deux neutrons combinés en une particule identique au noyau d'hélium; elles peuvent donc s'écrire He^{2+}.

Les particules alpha sont émises par des noyaux radioactifs par l'intermédiaire d'un processus nommé désintégration alpha. Ces noyaux radioactifs comme l'Uranium-238, le Radium-226 et le Radon-222, apparaissent de manière naturelle, en quantités minuscules, dans l'environnement. Si les émetteurs de particules alpha sont utilisés dans certains domaines, comme le traitement du cancer, les détecteurs de fumées et la réduction de l'électricité statique dans les usines à papier, les particules alpha peuvent provoquer des erreurs *soft* à l'intérieur d'un circuit intégré (voir section 1.4.2).

1.4. Les effets des radiations sur les circuits intégrés

Les effets des radiations sur les circuits intégrés peuvent être séparés en deux grandes catégories : la dose cumulée et les SEEs (*Single Event Effect*) ou évènements à effets singuliers [S.E]. Il est important de préciser que plusieurs facteurs contribuent à l'apparition des SEEs. Ces facteurs dépendent aussi bien de la particule (nature, énergie, angle d'incidence, …) que du circuit (technologie…).

1.4.1. Dose cumulée

La dose cumulée (en anglais *Total Ionising Dose*) est une dégradation cumulative du circuit à long terme, quand il est exposé à des radiations ionisantes (essentiellement les protons et puis les électrons). Ce phénomène se manifeste dans les oxydes et à l'interface

oxyde/semi-conducteur où les charges cumulées sont prises au piège et conduisent dans les transistors MOS entre autres, au décalage de la tension de seuil.

De nos jours, compte tenu des progrès importants réalisés dans la fabrication des oxydes de plus en plus fins, les conséquences néfastes du TID tendent à se réduire.

1.4.2. SEE

Cette abréviation de l'anglais (*single event effect*) désigne n'importe quel évènement qui apparaît quand une particule chargée dépose assez d'énergie et ionise le milieu dans lequel elle passe en causant un effet sur le circuit.

Les SEEs se divisent en deux types : les erreurs transitoires et non destructrices (dites *soft*) et les erreurs pouvant entraîner une destruction (dites *hard*) et leurs effets sont acceptables ou pas en fonction de l'application. Les erreurs *soft* sont transitoires en ce sens qu'il suffit d'une réécriture (pour les points mémoires) ou d'une remise à zéro pour que le circuit reprenne son fonctionnement normal. Ces erreurs correspondent par exemple au basculement d'état au niveau d'un bit mémoire (SEU) ou bien résultent d'une impulsion transitoire qui se propage de porte en porte jusqu'à ce qu'elle soit potentiellement captée par un registre (SET). Les erreurs *hard* peuvent conduire à la destruction d'un élément du circuit, et leurs effets sont alors permanents.

En environnement spatial, il y a deux causes principales aux SEEs : les rayons cosmiques à travers leurs ions lourds et les protons. Les ions lourds causent une ionisation directe de la matière à travers laquelle elles passent laissant derrière elles des paires d'électrons-trous qui en se recombinant donnent naissance à un SEE (Figure 1-3 (a)). Les protons quant à eux ne causent d'ionisation directe que dans les zones très sensibles du circuit. Cependant, généralement ils créent une réaction nucléaire qui peut produire un SEE par ionisation indirecte (Figure 1-3 (b)). En environnement terrestre et de haute altitude, les particules alpha provoquent des SEEs de façon directe.

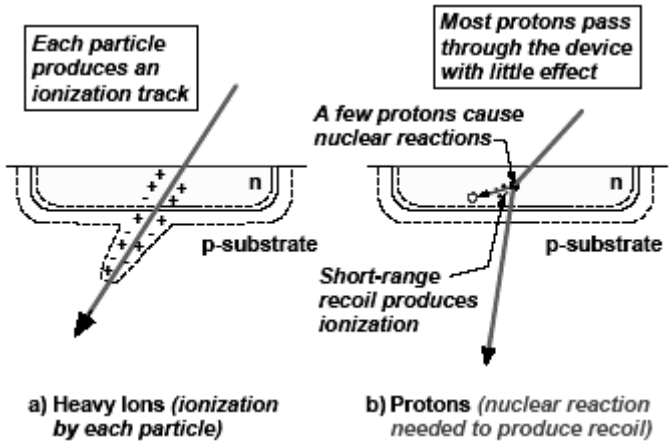


Figure 1-3: Ionisation du matériau [S.K]

L'impact sur un circuit d'un SEE dépend de son type, de sa localisation et aussi bien du circuit. Par exemple, une erreur peut avoir des effets se propageant aux éléments critiques d'un

système provoquant une erreur de commande. Il y a aussi des cas où les SEEs n'ont pas d'effets observables au niveau système. En effet, il y a des zones du système qui sont plus au moins sensibles face aux effets des radiations. La donnée enregistrée en RAM, par exemple, peut comporter des codes correcteurs, qui, même en présence de l'erreur la rend transparente au niveau système.

Par la suite on va décrire quelques exemples de SEE.

1.4.2.1. Latch-up

Le SEL (*Single event latchup*) est un évènement qui conduit à la perte de la fonctionnalité du circuit suite à une surcharge de courant au-delà des spécifications. Kolasinski et al est le premier à avoir observé le SEL en 1979 [W.K. 79]. Les SELs sont des erreurs hard, et sont potentiellement destructrices [A.D. 81]. Ils résultent du déclenchement de structures thyristor parasites (Figure 1-4) donnant naissance à un fort courant entre l'alimentation et la masse à travers la zone où s'est produit le SEL. Ceci peut conduire à la destruction du circuit, ou de l'alimentation. Le SEL disparaît dès qu'on coupe (rapidement) l'alimentation. Il est très dépendant de la température : plus celle-ci est forte et plus le SEL a de chance de se produire [I.M. 94]. Les SELs sont causés par des ions lourds, des protons ou des neutrons.

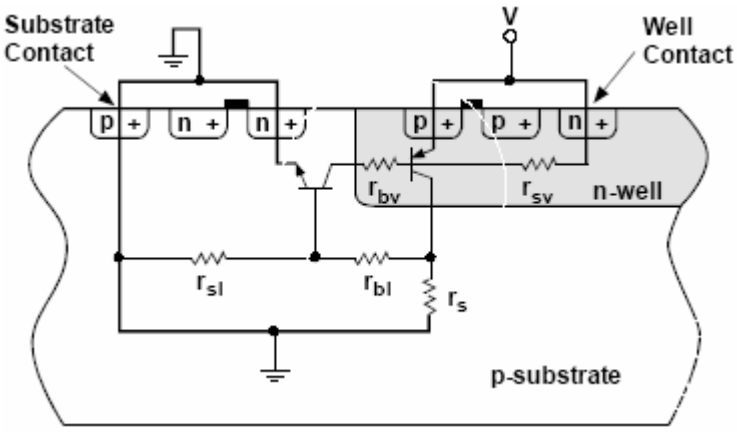


Figure 1-4: Mise en évidence de la structure parasite NPNP

1.4.2.2. SEU

Le SEU (*Single Event Upset*) est un SEE de type erreur *soft*. Le SEU résulte du passage d'une particule chargée créant un *upset* (ou *bit-flip*), qui correspond à l'inversion du bit mémorisé dans un point mémoire. La particule ionise donc le matériau à travers lequel elle passe et laisse des paires d'électrons-trous qui en se recombinant produisent un pic de courant. Ce pic de courant change le potentiel du nœud, et si ce dernier est suffisant, un basculement logique se produit et se maintient jusqu'à ce qu'on écrive de nouveau dans le point mémoire. Tout composant électronique possédant des points de mémorisation peut être sensible au phénomène d'*upset* (mémoires dynamiques ou statiques, microprocesseurs et périphériques de microprocesseurs, bascules, verrous…).

Le SEU n'est pas un phénomène nouveau. En effet, la possibilité du SEU a été d'abord postulée par Wallmark et Marcus en 1962 [J.W. 62]. Le premier à l'avoir découvert est J. Ziegler en 1979 [J.Z. 79] ; ses travaux portaient sur les rayons cosmiques. La même année, May et Woods, ont travaillé sur les particules alpha qui induisent des SEU [T.M. 79]. Dans leurs travaux, les particules alpha provenaient de la désintégration naturelle des traces d'uranium et de thorium présents dans les boîtiers des circuits intégrés. Le SEU n'est plus réservé à l'environnement spatial mais aussi bien aux hautes altitudes [E.N. a] qu'au niveau de la mer [E.N. b].

1.4.2.3. SET

Les SETs (*Single Event Transient*) sont des SEEs de type erreurs *soft*. Ils résultent d'une impulsion transitoire créée par le passage d'une particule chargée dans la logique combinatoire d'un circuit numérique. Un cas typique de circuit logique affecté par une impulsion transitoire est présenté dans la Figure 1-5.

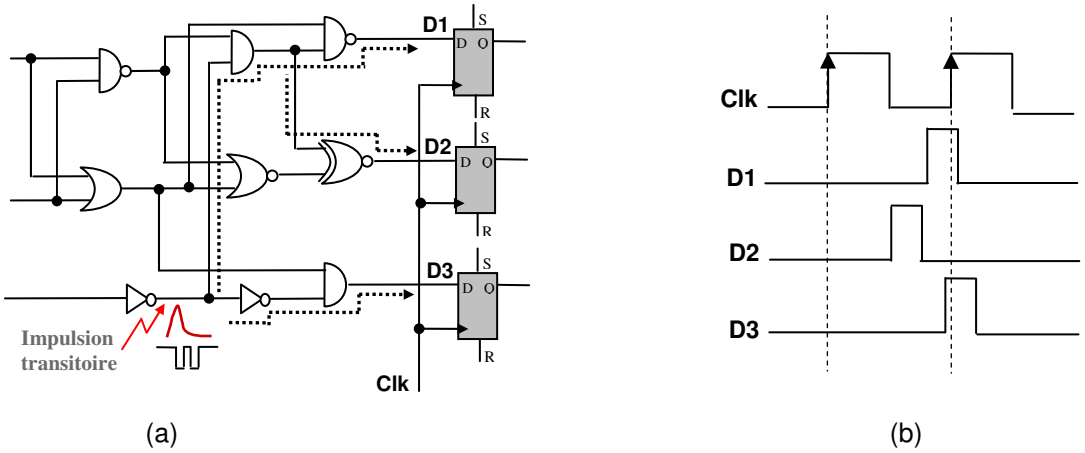


Figure 1-5: Scénario de propagation d'une impulsion transitoire dans un circuit combinatoire (a) et les formes d'onde des entrées de bascules (b)

Les trois chemins de propagation montrés dans la Figure 1-5 (a) sont supposés sensibilisés par un vecteur d'entrée adéquat. Ainsi, l'impulsion transitoire peut arriver au niveau des entrées D1, D2 et D3 des bascules en fonction de la longueur du chemin de propagation correspondant. Si elle arrive à l'entrée d'une bascule au moment du front actif de l'horloge du système, elle sera capturée et mémorisée comme une valeur erronée (cas des entrées D1 et D3 dans la Figure 1-5 (b)).

Dans d'autres cas, l'impulsion transitoire peut être masquée logiquement ou atténuée avant son arrivée aux entrées des bascules. Si sa durée est plus grande que les temps de transition des portes logiques, elle peut se propager aux entrées des bascules sans atténuation [M.B. 97]. Comme les portes logiques deviennent très rapides dans les nouvelles technologies, avec des temps de propagation très courts, les impulsions transitoires ne seront plus atténuées, même pour des particules d'énergie réduite [M.N. 99]. Néanmoins, la durée de l'impulsion finale pourrait être réduite ou augmentée selon les portes traversées. Les SETs peuvent aussi perturber

l'horloge elle-même ou le reset des éléments mémoires ce qui provoquera plusieurs inversions de bits à la fois.

1.4.3. Fautes et erreurs multiples

Un seul impact de particules peut dans certains cas causer directement des erreurs multiples. Les MBUs (*Multiple-Bit Upset*) apparaissent du fait des charges qui diffusent dans le substrat, pouvant ainsi être collectées par plusieurs éléments du circuit. Cela concerne surtout les mémoires DRAMs et SRAMs où plusieurs cellules du plan mémoire peuvent changer d'état logique simultanément.

1.5. Modélisation de plus haut niveau

Différents niveaux de modélisation des erreurs *soft* sont disponibles, allant du niveau physique jusqu'au niveau comportemental en passant par les niveaux transistor, portes ou RTL.

Dans le cadre de cette thèse, nous nous sommes focalisés sur la modélisation des fautes au niveau RTL sur des descriptions VHDL. A ce niveau, les bits mémoires sont facilement identifiables. Ces bits mémoires englobent toutes sortes de registres (données, contrôle, états) ainsi que les mémoires. Donc pour modéliser les fautes associées aux phénomènes SEU et MBU, il suffit d'inverser les bits mémoires concernés. Par contre, il est souvent impossible de modéliser avec précision le phénomène SET car, à ce niveau de description, les nœuds d'une logique combinatoire ne sont pas identifiables et aucune donnée temporelle n'est disponible. Il est donc impossible d'évaluer avec précision les configurations d'erreur dues aux SETs qui peuvent éventuellement se produire. Cependant, supposer que seulement des inversions de bits uniques peuvent se produire est évidemment très optimiste. En conséquence, l'utilisation du modèle classique d'inversion de bit unique n'est pas suffisante. Des modes de défaillance critiques peuvent ne pas être identifiés pendant l'analyse haut niveau et devraient donc être traités beaucoup plus tard dans le processus de conception. Une solution à cette limitation consiste à étendre le modèle de fautes habituel à un modèle que nous appelons MBF (multiple *bit-flips*). En effet, un SET qui se propage à travers des portes logiques sera finalement capturé par des éléments mémoires ou bien n'aura pas d'effets. Cela correspond donc à une inversion de bits potentiellement multiple au niveau des points mémoires.

Les modèles SEU, MBU ou MBF présentés précédemment supposent donc que des bits mémoires soient clairement spécifiés lors de la description du circuit. Si cela est généralement vrai pour les registres de données, ce n'est pas toujours le cas pour les registres d'états qui peuvent être définis par des types énumérés utilisant des noms symboliques. La spécification au niveau bit de ces registres d'états n'est disponible qu'après synthèse. Pourtant ces registres sont aussi sensibles aux erreurs *soft*. Pour pouvoir modéliser l'effet de ces erreurs, un modèle appelé transitions erronées (1.7.3.5) a été développé au sein du groupe. Ce modèle correspond à un modèle plus abstrait où l'occurrence d'un SEU ou d'un MBF dans un registre assigné avec des noms symboliques est assimilée à une transition inattendue d'une valeur symbolique à une autre.

Puisque les codes binaires ne sont pas encore connus, il n'est pas possible de limiter les transitions erronées à des cas correspondant à des inversions de bits uniques.

1.6. Injections de fautes

Pour identifier et comprendre les défaillances potentielles d'un système et garantir le niveau requis de sûreté de fonctionnement, nous avons besoin d'une approche expérimentale. Une telle approche peut être appliquée durant la phase de conception, mais aussi durant la phase de prototypage et la phase de qualification. Cette approche, appelée injection de fautes, consiste en l'introduction intentionnelle de fautes dans un système afin d'en mesurer la fiabilité et/ou déterminer sa tolérance aux fautes. Pour réussir une campagne d'injections de fautes, il faut bien sûr comprendre l'architecture, la structure et le comportement du système. Il est également fondamental d'utiliser un modèle de fautes cohérent avec les évènements pouvant réellement survenir dans l'environnement applicatif.

La Figure 1-6 montre la composition générale d'un environnement d'injection de fautes, qui inclut un système cible, un injecteur de fautes associé à une bibliothèque de fautes, un générateur de *stimulis* associé éventuellement à une bibliothèque, un contrôleur, un moniteur, un collecteur de données et un analyseur de données [M.H].

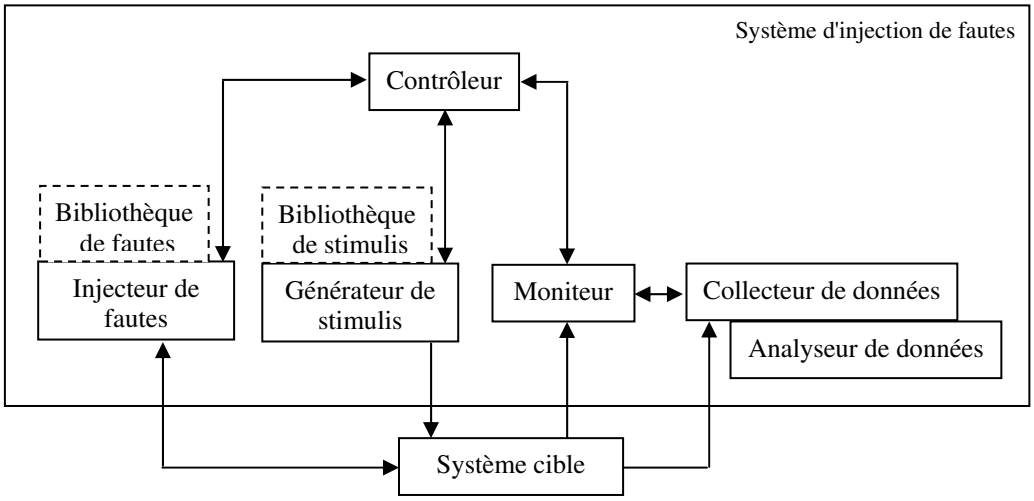


Figure 1-6: Les composants de base d'un environnement d'injection de fautes

L'injecteur de fautes active des fautes dans le système cible pendant que celui-ci exécute des commandes du générateur de *stimulis* (applications, benchmarks, ou *stimulis* synthétiques). Le moniteur suit l'exécution des commandes et initie l'enregistrement des données lorsque nécessaire. Le collecteur de données effectue l'enregistrement des données en ligne, tandis que l'analyseur de données effectue le traitement de données hors ligne. Le contrôleur se charge de commander l'ensemble des expériences d'injection.

Dans la pratique, le contrôleur est un programme qui peut s'exécuter soit sur le système cible soit séparément sur un ordinateur. L'injecteur de fautes peut correspondre à une implantation matérielle ou logicielle. Il peut supporter différents types de fautes (avec une

sémantique matérielle ou une structure logicielle appropriée), injectées à des emplacements multiples et à des instants différents. Ces valeurs sont extraites de la bibliothèque de fautes. La bibliothèque de fautes, le générateur de *stimulis*, le moniteur ainsi que les autres composants permettent, lorsqu'ils sont indépendants, d'obtenir une grande flexibilité et portabilité.

Concernant l'analyseur de données, deux types d'analyses peuvent être effectuées : classification de fautes en identifiant les évènements menant à des évènements particuliers, ou analyse de propagation d'erreurs, montrant les configurations erronées internes activées par les fautes (1.7.5).

Un grand nombre de techniques d'injection de fautes ont été développées, applicables à des stades différents du développement d'un système. Ainsi on retrouve des techniques d'injection au niveau broche, corruption de la mémoire [T.M. 94], injection d'ions lourds [U.G. 89], perturbations de l'alimentation [J.K. 91], injection de fautes par laser [J.S. 97]…. Dans la suite, la présentation sera focalisée sur les techniques permettant d'analyser la sûreté d'un circuit tôt dans le flot de conception, et plus particulièrement dès la description comportementale.

1.6.1. Injections de fautes dans les descriptions comportementales

Plusieurs auteurs ont proposé d'injecter des fautes très tôt dans le flot de conception. L'approche principale consiste à injecter des fautes dans des descriptions de haut niveau (généralement des modèles VHDL) d'un circuit ou système. Delong [T.D. 96] décrit par exemple l'injection de fautes dans des descriptions VHDL comportementales de systèmes à base de microprocesseur. Cette approche utilise des modèles au niveau instruction (*Instruction Set Architecture*) et permet d'injecter des fautes permanentes dans la mémoire ou dans les registres du processeur, durant la simulation, à travers un contrôleur d'injection de fautes qui est un *process* rajouté à la description initiale du circuit. Ce *process* d'injection exploite des types de données spécifiques et une fonction de résolution ; ce qui fait que cette approche soit difficilement généralisable. L'approche présentée en [E.J. 94], [S.S. 97] ou [J.B. 98], est plus générale et gère différents types de fautes dans le modèle VHDL d'un circuit à différents niveaux d'abstraction et en utilisant différentes techniques basées sur la modification de la description initiale du circuit ou l'utilisation de primitives de simulation. Cette modification est souvent appelée "instrumentation" (voir 1.6.2). De telles techniques ont été développées dans différents travaux et les modifications dans la description VHDL initiale peuvent être structurelles en rajoutant des blocs "saboteurs" (voir 1.6.2.1), comportementales en générant des "mutants" (voir 1.6.2.2), ou par *process* [G.C. 02].

Comme mentionné dans [R.L. 99], la principale limitation des simulations est le temps requis pour les expérimentations quand de nombreuses fautes sont injectées dans un circuit complexe. D'autres inconvénients sont donnés à la section 1.6.3. Cela dit, des techniques ont été développées pour diminuer ce temps, comme dans [B.P. 00, L.B. 02] en supprimant les fautes qui n'aboutissent pas à des erreurs. Pour s'abstraire de cette limitation imposée par la simulation, il a été aussi proposé de tenir compte des avantages du prototypage, en utilisant un

émulateur à base de circuits FPGA [R.L. 98, L.A. 00, C.O. 05]. Le gain en temps dépend de l'organisation des expérimentations [R.L. 01], mais un autre avantage de l'émulation est la possibilité d'étudier le comportement dans son environnement, en tenant compte des interactions en temps réel des composants logiciels et matériels [E.B. 99]. L'analyse est ainsi plus complète et précise que celle basée sur la simulation.

Quand un émulateur est utilisé, la description VHDL initiale doit être synthétisable. Dans certains cas, les approches développées pour l'évaluation d'un taux de couverture par émulation de fautes (exemples [R.W. 95, K.C. 99]) peuvent aussi être utilisées pour injecter des fautes. Mais de telles approches sont classiquement limitées aux fautes permanentes. Dans le cas de l'émulateur SimExpress, les fautes pouvaient être injectées dans le prototype en utilisant des mécanismes incorporés à l'émulateur [J.A. 98], mais de tels mécanismes n'existent pas dans les émulateurs disponibles aujourd'hui dans le commerce. Différentes équipes ont donc récemment considéré plusieurs approches spécifiques [L.A. 00, L.A. 02, P.C. 01-a, P.C. 01-b, P.C. 01-c, R.L. 01, J.T. 04, C.O. 05]. Certaines de ces approches permettent d'instrumenter le circuit comme dans [R.L. 00, P.C. 01-b]. Elles sont basées sur une reconfiguration statique (*Compile-Time Reconfiguration*, CTR) du FPGA. D'autres comme [L.A. 00] évitent toute instrumentation et permettent d'injecter des fautes permanentes ou transitoires dans les parties combinatoires du circuit en reconfigurant les *Look-Up Tables* (LUT) du FPGA. Cette approche est basée sur une reconfiguration dynamique (*Run-Time Reconfiguration*, RTR) lors de l'injection de chaque faute. Cette approche permet un gain de temps par rapport à l'instrumentation dans la mesure où il n'y a pas de temps supplémentaire nécessaire à la préparation de la description instrumentée. De plus, il n'y a pas de ressources matérielles à rajouter dédiées à l'injection, ce qui permet d'avoir un prototype plus efficace en termes de surface et de fréquence d'horloge maximale. Toutefois, le temps nécessaire pour la reconfiguration dynamique peut conduire à augmenter fortement la durée des expériences, selon les caractéristiques du circuit et de celle de l'application étudiée.

1.6.2. Méthodes d'instrumentation

L'instrumentation consiste à transformer la description initiale du circuit pour permettre l'injection de fautes. Ainsi des signaux de contrôle additionnels et des éléments spécifiques sont introduits en modifiant soit la description au niveau RTL soit celle au niveau portes. Deux approches différentes peuvent être utilisées : les saboteurs ou les mutants. Dans la suite de ce document, nous considérons uniquement les instrumentations au niveau RTL, compatibles avec des campagnes réalisées sur la base de simulations comportementales.

1.6.2.1. Les saboteurs

Les saboteurs [E.J. 94, J.G. 01] sont des blocs supplémentaires qui, lorsqu'ils sont activés, permettent l'injection de fautes en altérant la valeur ou les caractéristiques temporelles d'un ou de plusieurs signaux. Ils sont insérés entre les composants existants dans la description du

circuit. Ce genre de modification est facile à réaliser et nécessite surtout la modification de certaines interconnexions dans la description initiale. Le saboteur peut injecter différents types de fautes dans les interconnexions modifiées. Cependant, et de manière générale, il est presque impossible d'injecter des fautes comme les SEU ou les transitions erronées dans certains éléments des blocs existants.

1.6.2.2. Les mutants

Un mutant correspond à la modification d'un ou de plusieurs composants existants dans la description du circuit. Lorsque l'injection est inactive, le mutant se comporte exactement comme le composant remplacé et lorsque l'injection est active, il imite un comportement fautif. Grâce aux mutants, des modèles complexes de fautes comme les transitions erronées peuvent être injectés. Les modifications à apporter sont plus difficiles que pour les saboteurs mais sont beaucoup plus puissantes.

Une technique d'injection de fautes dans des descriptions VHDL en utilisant les mutants est par exemple proposée dans [R.L. 00]. La génération des mutants est optimisée de telle manière que des comportements erronés significatifs puissent être forcés pendant les expériences d'injection de fautes. En outre, ces descriptions mutées sont optimisées pour la synthèse matérielle, de sorte que l'on puisse utiliser la simulation et l'émulation à partir de la même description modifiée. Les fautes sont injectées dans les machines à états synthétisables. Le modèle de fautes correspond aux inversions de bits transitoires, le modèle d'erreur traité représentant l'effet des impacts de particules (SEU).

1.6.3. Injections à base de simulations

La simulation permet d'analyser un système à tous les niveaux, du plus haut (niveau fonctionnel) au plus bas (le comportement élémentaire, transistor…). La simulation utilise aujourd'hui beaucoup les langages de description de haut niveau tels que VHDL et VERILOG. Ces langages permettent la description de systèmes très complexes aussi bien au niveau comportemental (algorithmes, équations booléennes…) qu'au niveau structurel (niveau portes). De plus, avec ces langages, une description multi niveaux est possible. A un niveau plus élevé, nous pouvons citer les techniques dites de co-simulation. Elles consistent à simuler conjointement la partie matérielle et la partie logicielle du système. Dans un tel environnement, il est possible d'effectuer une co-simulation d'un ensemble de modèles hétérogènes. Par exemple, dans un environnement de co-design dédié à la conception de circuits pour les commandes de processus, la co-simulation consistera à faire communiquer un simulateur de langage HDL, un exécutable provenant d'un programme C et un simulateur du processus électrique (exemple Matlab). A l'autre extrémité, les caractéristiques électriques du circuit pourront être analysées avec une description SPICE. Dans la suite de ce document, nous nous intéresserons plus particulièrement à l'utilisation des langages de haut niveau.

Les approches d'injection de fautes basées sur la simulation offrent plusieurs possibilités, et les principaux avantages sont :

- C'est une technique qui permet d'analyser la sensibilité d'un circuit digital complexe très tôt dans le processus de conception.
- L'observabilité et la contrôlabilité ne sont pas limitées.
- La modélisation est multi-niveau : du système au transistor.
- Les techniques employées sont flexibles car une description de circuit peut facilement être modifiée pour injecter des fautes à différents endroits. De plus, une même description peut être utilisée sur différentes plateformes si elles intègrent le HDL correspondant.
- Elles n'exigent aucun matériel particulier d'où un faible coût d'implantation.

Par ailleurs, les principaux inconvénients sont :

- Les expériences d'injection de fautes nécessitent un temps considérable de simulation.
- Les expériences ne sont pas réalisées sur un circuit réel, ce qui conduit à un manque de précision dans (1) l'estimation de la couverture et de la latence, et (2) l'analyse des chemins de propagation d'erreur.
- La construction de modèles fidèles pose problème, notamment pour la représentation de l'environnement opérationnel.

1.6.4. Injections à base d'émulations

Dans le cas de l'émulation, le circuit doit être synthétisé sur des composants programmables. Donc la description initiale du circuit et la description modifiée pour l'injection doivent être synthétisables. Par ailleurs, le nombre de ressources utilisées dans l'émulateur et la fréquence maximale de l'horloge du prototype dépendent fortement de la qualité de l'écriture de la description.

Les approches d'injection de fautes basées sur le prototypage matériel offrent plusieurs possibilités, et les principaux avantages sont :

- C'est une technique qui permet d'analyser la sensibilité d'un circuit digital complexe très tôt dans le processus de conception.
- Les expériences d'injection de fautes sont rapides et peuvent être exécutées à fréquence nominale ou proche, ce qui permet d'exécuter un grand nombre d'expériences d'injection de fautes.
- Il est possible de connecter le prototype du circuit dans son environnement opérationnel.
- Les fautes peuvent être injectées à différents endroits.

Par ailleurs, les principaux inconvénients sont :

- Certaines contraintes matérielles de l'émulateur peuvent limiter l'observabilité et la contrôlabilité par rapport à une simulation.
- Dans certains cas, le temps nécessaire pour la synthèse et le placement routage sur l'émulateur peut pénaliser les expériences d'injection.
- Le modèle cible doit être synthétisable.

1.7. Vue d'ensemble du flot d'analyse développé au sein du groupe

1.7.1. Introduction

L'approche proposée a été principalement développée dans le cadre de la thèse de Karim Hadjiat [K.H. 05]. Elle se base sur l'injection de fautes dans des descriptions de haut niveau (description VHDL au niveau RTL) et sur la génération d'un modèle représentant les comportements consécutifs à la faute. Le travail réalisé pendant la thèse de Karim Hadjiat a donc consisté à concevoir et à implémenter un environnement de CAO automatisé permettant d'analyser, très tôt dans le flot de conception, les effets potentiels d'une faute sur une application exécutée par un circuit intégré numérique.

L'un des centres d'intérêt principal des analyses est la propagation des erreurs provenant du phénomène SEU. Les types de fautes visés sont donc dans un premier temps les inversions de bits uniques dans des éléments mémoires et les transitions erronées dans des registres d'états.

1.7.2. Description du flot existant

L'environnement de CAO développé a pour but d'aider le concepteur à identifier et à quantifier les modes de défaillance d'un circuit numérique. Le résultat d'analyse doit montrer les comportements erronés et les propagations d'erreur réellement observées pour un ensemble donné de fautes et pour une application donnée. L'ensemble de solutions alternatives qui ont été considérées est illustré dans la Figure 1-7 [R.L. 03].

La description initiale du circuit peut être instrumentée ou utilisée sans aucune modification. L'instrumentation permet l'injection de fautes en utilisant des saboteurs pour des modèles simples de faute, ou des mutants pour injecter des modèles de faute plus complexes comme les SEUs. La campagne d'injection peut alors être exécutée en utilisant la simulation ou l'émulation tout en offrant une compatibilité entre les deux. Dans le cas d'une campagne basée sur la simulation, le but est de pouvoir utiliser différents simulateurs commerciaux. Par conséquent, les approches d'injection utilisant des primitives de simulation ou un simulateur dédié n'ont pas été implémentées. Ceci implique que la description du circuit doit être modifiée.

Dans le cas d'une campagne basée sur l'émulation, nous considérons deux approches. La première, fondée sur une reconfiguration statique (CTR), sera utilisée lorsque la description du circuit est instrumentée. Dans ce cas, l'architecture du circuit est chargée sur l'émulateur et la

configuration matérielle reste statique jusqu'à la fin de l'exécution. La deuxième approche, basée sur la reconfiguration dynamique (RTR), permet de réaliser des injections sans modifier la description initiale du circuit. Les fautes sont injectées directement dans le circuit pendant l'exécution sur l'émulateur.

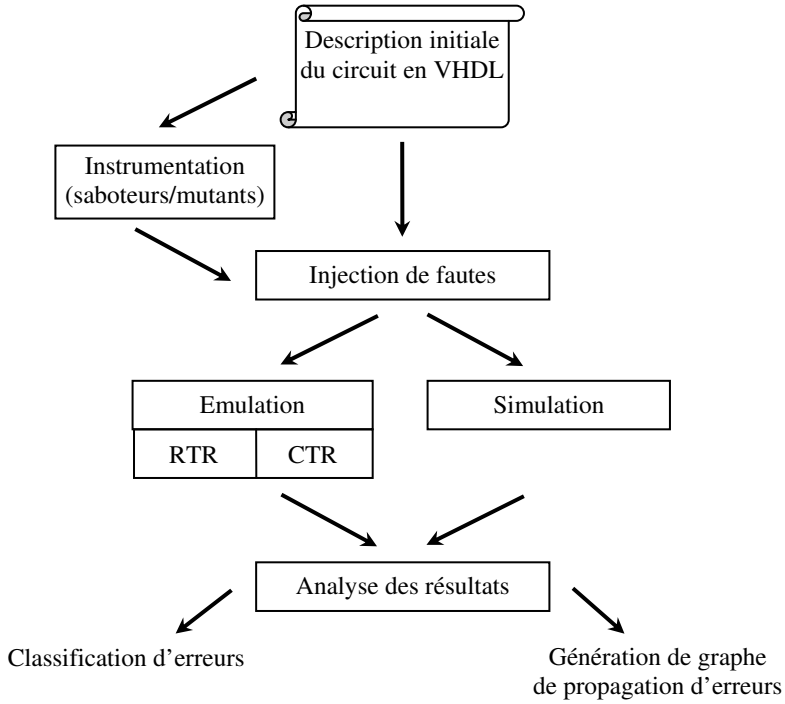


Figure 1-7: Vue globale pour injection hétérogène

Le flot global d'analyse considéré dans cette thèse sera limité aux cas où le circuit est instrumenté. Il est détaillé dans la Figure 1-8. Pendant la définition de la campagne, le concepteur fournit les informations nécessaires pour l'injection de fautes et l'analyse des résultats ainsi que le(s) modèle(s) de fautes, les cibles d'injection et les nœuds à observer. Les expériences d'injection sont réalisées soit par simulation soit par émulation matérielle.

Dans le cas de la simulation, la description VHDL initiale est modifiée de façon à rendre possible l'injection de toutes les fautes choisies aux endroits sélectionnés. Un "macro-script" contrôle le processus d'injection en activant les fautes aux instants spécifiés. Ce script se base sur des distributions spatiale et temporelle des fautes (déterministes ou aléatoires) spécifiées par l'utilisateur. Les traces de simulation (ou les résultats obtenus par émulation matérielle) sont sauvegardées dans des fichiers.

Dans le cas de l'émulation, les contraintes matérielles de l'émulateur peuvent exiger plus de modifications et exclure l'exécution d'une seule description modifiée. Les raisons seront détaillées dans la section 1.7.3.3. Dans ce cas, la campagne d'injection doit être divisée en plusieurs sous-campagnes. Pour chacune d'entre elles, des modifications spécifiques seront apportées à la description initiale. Chaque description modifiée doit être synthétisée, placée et routée sur l'émulateur. Selon les objectifs de la campagne d'injection, les exécutions peuvent impliquer l'émulation du circuit dans le système opérationnel.

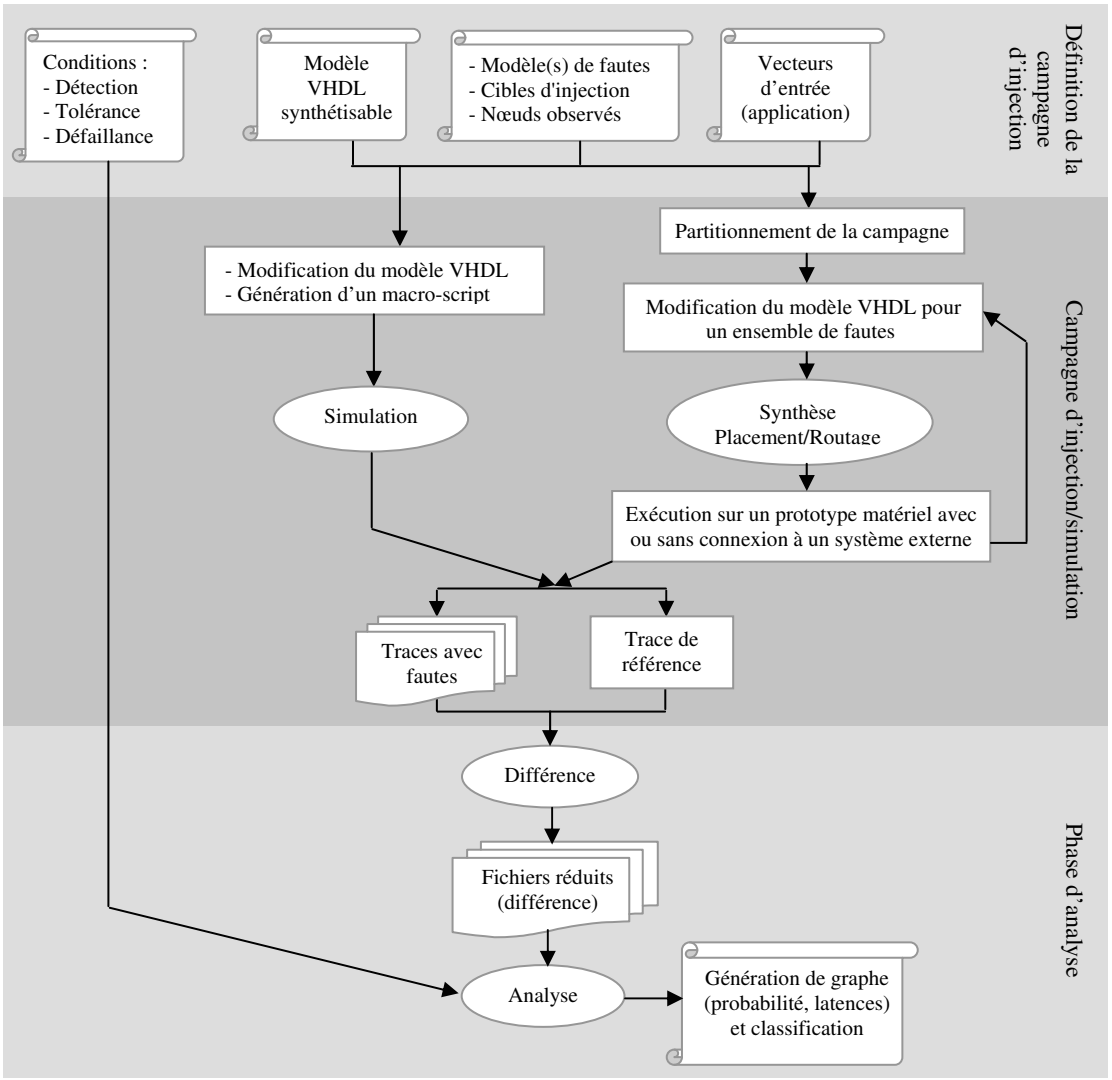


Figure 1-8: Flot global d'analyse pour une description modifiée du circuit [K.H. 05]

Les résultats issus de la campagne d'injection correspondent à des traces sauvegardées dans des fichiers. Ils montrent l'évolution des signaux observés et sont indépendants de l'approche utilisée pour exécuter les expériences (simulation ou émulation, avec ou sans modification) puisque seulement une analyse fonctionnelle du circuit est réalisée (aucune donnée temporelle n'est sauvegardée, seulement l'état des signaux échantillonnés à chaque cycle d'horloge). Ces traces sont alors utilisées pour effectuer une analyse comportementale du circuit en présence de fautes. Cela consiste à comparer le comportement de référence du circuit avec le comportement obtenu en présence de chaque faute de l'ensemble de fautes prédéterminé. Cela permet de détecter l'apparition d'évènements redoutés et de fournir au concepteur des résultats synthétiques (qualitatifs et quantitatifs). Les conditions significatives associées aux évènements critiques (défaillance, détection…) sont spécifiées par le concepteur. Les résultats quantitatifs incluent la probabilité de transition vers un état erroné donné et la latence observée pour passer d'un état à un autre.

Les sections 1.7.3, 1.7.4 et 1.7.5 détaillent chaque étape du flot.

1.7.3. Modification des fichiers VHDL en vue de l'injection

Les modifications de la description du circuit sont réalisées de façon automatique en fonction des cibles et des modèles de fautes à injecter. Ainsi on ne modifie pas de la même manière un registre concerné par l'injection et un autre qui ne l'est pas. De même les modifications ne sont pas les mêmes selon que l'on vise l'injection de SEUs ou de transitions erronées.

1.7.3.1. Injections asynchrones

Par injections asynchrones, on désigne des injections dont les instants d'activation sont indépendants de l'horloge. Ceci permet au concepteur de réaliser des injections qui se rapprochent de la réalité opérationnelle du circuit cible. Pour y parvenir il faut rajouter un signal "Asyn_inj" afin d'obtenir un contrôle externe du chargement des points mémoires concernés par l'injection, c'est-à-dire que l'horloge du circuit d'origine "Clk" doit être modifiée. Cette modification permet de rajouter un front d'horloge en plus au moment de l'injection qui n'aura d'effets que sur les points mémoires concernés par l'injection. Un autre signal de contrôle "En_inj" permet d'autoriser l'injection en sélectionnant les entrées de certains éléments mémoires du circuit, ciblés par l'injection, de façon à modifier le traitement par rapport à celui réalisé en mode de fonctionnement normal. Cela dépend non seulement du signal "En_inj" mais aussi de l'adresse de la cible. Les signaux de contrôle "En_inj" et "Asyn_inj" ajoutés comme entrées primaires permettent de réaliser des injections asynchrones, représentant l'effet des SEUs. Pendant une expérience de simulation ou d'émulation, le programme de test, qui se charge d'appliquer les entrées fonctionnelles au circuit, va aussi agir sur les signaux de contrôle externes afin d'exécuter l'injection. Pour le bon déroulement d'une injection asynchrone, il est important de respecter les deux contraintes suivantes :

- Les deux signaux de contrôle "Asyn_inj" et "En_inj" sont remis à '0' avant le front d'horloge qui suit l'injection.
- Le signal "En_inj" est actif (au niveau haut) uniquement au front d'horloge relatif à l'injection.

1.7.3.2. Injections synchrones

Par injections synchrones, on désigne des injections dont les instants d'activation sont dépendants des fronts de l'horloge. Donc contrairement aux injections asynchrones l'horloge n'a pas besoin d'être modifiée. Cela dit, vu que le phénomène SEU est indépendant de l'horloge, on préfère utiliser des injections asynchrones même si les modifications à apporter au circuit sont un peu plus complexes.

Les injections synchrones ne sont employées que lors d'injections de transitions erronées dans les registres d'états (voir 1.7.3.5).

1.7.3.3. Modification de l'horloge

La solution proposée pour la modification de l'horloge consiste à ajouter un processus, illustré par la Figure 1-9, afin d'obtenir un contrôle externe des points mémoires par le signal "Asyn_inj". L'implantation du processus correspond à une porte XOR. Le signal "Asyn_inj" constitue l'entrée primaire qui nous permet de contrôler l'horloge "Clk". Le nouveau signal d'horloge "Clk_inj" est généré en sortie et sera utilisé comme signal d'horloge à la place du signal d'origine Clk.

```
seq: process(Clk, Asyn_inj)
 begin
     if(Asyn_inj='1') then
          Clk_inj <=  not(Clk);
     else
          Clk_inj <= Clk;
     end if;
 end process ;
```

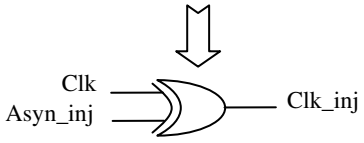


Figure 1-9: Génération du signal Clk_inj

La porte XOR de la Figure 1-9 rend possible la génération d'un nouveau front d'horloge à n'importe quel instant situé entre les deux fronts d'horloge d'un cycle donné. En effet, le passage du signal "Asyn_inj" de '0' à '1' implique l'inversion de la valeur du signal "Clk_inj", ce qui correspond, dans la Figure 1-10, à l'instant d'injection.

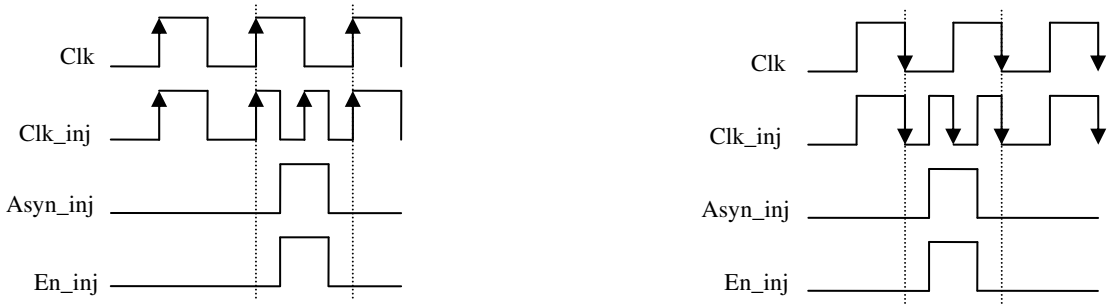


Figure 1-10: Protocole d'injection (injection active sur front montant ou descendant)

Dans le cas d'une campagne d'injection basée sur la simulation post-synthèse avec délais ou l'émulation, les délais T1 et T2 (illustrés dans la Figure 1-11) doivent être suffisamment longs pour que les signaux à l'entrée des registres puissent se stabiliser avant l'impulsion d'injection et aussi avant la montée suivante du front d'horloge.

Par conséquent, la fréquence de l'horloge de la simulation niveau portes ou du prototype doit être choisie de telle sorte que le délai T_2 soit supérieur au chemin critique. Par ailleurs, si le concepteur souhaite réduire au minimum le temps nécessaire pour exécuter les expériences d'injection alors la fréquence d'horloge maximum sera obtenue en effectuant des injections de

fautes au milieu de la période et dans ce cas, la fréquence d'horloge de la simulation ou du prototype peut être égale à (fréquence nominale)/2.

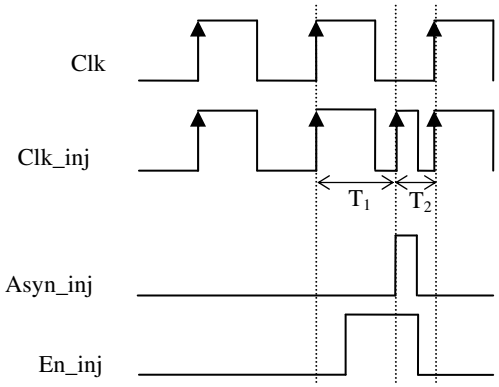


Figure 1-11: Injection au milieu de la 2ème moitié de la période

1.7.3.4. Injection d'inversions de bits

La Figure 1-12 (a) donne un exemple d'une machine à états dont le registre state sera concerné par des injections de SEUs. Les registres "R_i" sont des registres de données non concerné par l'injection. Les différentes modifications et insertions apportées à la description initiale sont illustrées en gras en Figure 1-12 (b).

```
signal state, nstate: std_logic_vector (2 downto 0) ;

seq: process(clk, Reset)
begin
        if Reset='1' then
                state <= waiting ;
        elsif (clk'event and clk='1') then
                state <= nstate ;
                R_i <= …;
                …;
        end if ;
end process ;
```

(a)

```
signal state, nstate: std_logic_vector (2 downto 0) ;

seq: process(Clk_inj, Reset)
begin
        if Reset='1' then
                state <= waiting ;
        elsif (Clk_inj'event and Clk_inj='1') then
                if(En_inj='0') then
                        state <= nstate ;
                        R_i <= … ;
                else
                        case Sel_bit is
                                when "01" =>state(0) <=not state(0) ;
                                when "10" =>state(1) <=not state(1) ;
                                when "11" =>state(2) <=not state(2) ;
                                when others => null;
                        end case;
                end if ;
        end if ;
end process ;
```

(b)

Figure 1-12: Application du modèle des inversions de bits uniques sur le registre "state"

Premièrement l'horloge est changée conformément à ce qui a été dit à la section précédente. Ensuite un bloc d'instruction "case" est inséré dans la partie séquentielle pour nous permettre d'inverser un des points mémoires du registre "state". Le bit à inverser est sélectionné par un signal d'entrée externe "Sel_bit" ajouté à l'entité. Le même type de modification peut être utilisé pour effectuer des inversions de bits dans des registres de données. Les registres "R_i" non concernés par l'injection sont retranscrits dans la partie conditionnelle où le signal "En_inj" est inactif ('0'), ainsi lorsqu'il y a injection seul le registre state sera modifié.

L'exemple présenté précédemment ne gère l'injection que dans un seul registre. Si plusieurs sont concernés par l'injection ou si la cible est une RAM, le principe reste le même. Pour une RAM le code change pour en produire un autre beaucoup plus compact. Une RAM est fonctionnellement équivalente à un groupement de "n" registres de taille "m". Chaque registre possède "m" bascules avec une entrée de validation pour chacune d'elle. Lorsqu'il y a écriture ou lecture d'un mot, c'est l'ensemble des bascules du registre qui sont activées. En VHDL RTL, une RAM ou un banc de registres est généralement défini par un tableau de vecteurs mémorisés. La Figure 1-13 (a) représente un exemple VHDL d'une mémoire de 4 mots de deux bits chacun.

```
entity RamMemory is
        port (data : in std_logic_vector(1 downto 0);
                address : in std_logic_vector(1 downto 0);
                we, Clk : in std_logic;
                q : out std_logic_vector(1 downto 0)
        );
end RamMemory;

architecture example of RamMemory is
        type MemType is array(3 downto 0) of
        std_logic_vector (1 downto 0);
signal mem : MemType;

begin

process (Clk, we, address)
begin
        if (Clk='1' and Clk'event)then
                if we='1' then
                        mem(conv_integer(address)) <= data;
                end if;
        end if;
end process;

process (Clk, address)
begin
        if (Clk='1' and Clk'event) then
                q <= mem(conv_integer(address));
        end if;
end process;
end example;
```

(a)

```
entity RamMemory is
        port (data : in std_logic_vector(1 downto 0);
                address : in std_logic_vector(1 downto 0);
                we, Clk : in std_logic;
                q : out std_logic_vector(1 downto 0);
                En_inj : in std_logic;
                En_asyn : in std_logic;

                -- Selection du mot cible
                Sel_mot : in integer range 0 to 3;
                -- Numero du bit a inverser
                Sel_bit : in integer range 0 to 1
        );
end RamMemory;

architecture example of RamMemory is
        type MemType is array(3 downto 0) of std_logic_vector (1
        downto 0);
        signal mem : MemType;
        signal Clk_inj : std_logic; -- memory clock
begin

process (Clk_inj, we, address, Sel_mot, Sel_bit)
        variable Mot_inj : std_logic_vector(1 downto 0));
begin
        if (Clk_inj='1' and Clk_inj'event)then
                if En_inj='1' then
                        Mot_inj := mem(Sel_mot);
                        mot_inj(Sel_bit):= not Mot_inj(Sel_bit);
                        mem(Sel_mot) <= Mot_inj;
                elsif we='1' then
                        mem(conv_integer(address)) <= data;
                end if;
        end if;
end process;

process (Clk_inj, address)
begin
        if (Clk_inj='1' and Clk_inj'event) then
                q <= mem(conv_integer(address));
        end if;
end process;
process(En_asyn, Clk)
begin
        if( En_asyn='1') then
                Clk_inj <= not(Clk);
        else
                Clk_inj <= Clk;
        end if;
end process;
end example;
```

(b)

Figure 1-13: Modification de la RAM pour des inversions de bit uniques

Les modifications VHDL apportées à la description initiale concernent particulièrement le processus séquentiel modélisant l'écriture de données dans le banc de registres cible (Figure 1-13 (b)). Nous définissons la cible d'injection par le biais des signaux de sélection "Sel_mot" et "Sel_bit" insérés dans la définition de l'entité. Le mot cible est déterminé par le sélecteur "Sel_mot" et le numéro du bit cible est défini par l'entrée de sélection "Sel_bit". Des instructions sont insérées dans la partie conditionnelle qui exprime le front d'horloge. L'activation des signaux "En_inj" et "En_asyn" permet d'exécuter l'instruction d'affectation de la valeur obtenue après inversion de bit. La variable "Mot_inj", insérée dans la description initiale, sera utilisée comme variable intermédiaire afin de permettre l'inversion de bit. Ceci s'effectue après exécution de l'instruction "mem(Sel_mot) <= Mot_inj".

Le modèle de fautes "inversion de bits" peut être appliqué uniquement sur des éléments mémoires de type binaire. Les cibles d'injection qui sont déclarées initialement de type non binaire (de type entier par exemple) doivent être converties pour être prises en compte. Par conséquent, nous devons effectuer une conversion explicite en utilisant une fonction de conversion adaptée au type vers lequel on souhaite convertir (par exemple la fonction *conv_std_logic_vector* permet de convertir vers le type *std_logic_vector*).

1.7.3.5. Injections de transitions erronées

Dans des descriptions niveau RTL, les registres d'état sont un cas particulier pour le processus d'injection. La conséquence d'une inversion de bit dans un tel registre peut être évaluée seulement si le codage des états est connu, c'est-à-dire dans la plupart des cas après la synthèse du circuit et donc assez tard dans le processus de conception. Il est alors nécessaire de modéliser à un niveau plus élevé l'effet d'un SEU qui survient dans un registre d'états. La solution proposée est d'injecter des transitions erronées asynchrones entre les états définis par leurs noms symboliques.

Quatre approches ont été développées pour modifier la description VHDL initiale, afin que des transitions erronées puissent être injectées dans une machine à états. La description initiale est modifiée de façon à rendre possible l'injection de toutes les transitions erronées entre les états de la description initiale. L'injection est autorisée par le signal d'entrée externe "En_inj" ajouté à l'entité ; il est alors possible de définir de manière directe la transition erronée à injecter, ou de la définir de manière indirecte. Dans le premier cas, l'état initial et final d'une transition ("I_state" et "F_state") correspondent à des signaux d'entrée insérés dans la définition de l'entité. Dans le deuxième cas, on indique l'état initial "I_state" et une valeur de déplacement "inj_N". L'état final de la transition "F_state" est déterminé comme étant l'état qui est à une distance "inj_N" de l'état "I_state" dans la liste des états énumérés spécifiée dans la description initiale. Ce type de définition de l'injection simplifie la commande externe lorsque toutes les transitions erronées doivent être injectées pendant une campagne. Dans les deux cas, il est possible soit de spécifier les états par des codes avec un nombre minimal de bits ou par les noms symboliques d'états. Si l'on choisi une spécification par codes, ces derniers seront assignés en fonction de l'ordre des états dans la définition du type énuméré ou suivant un codage introduit

par l'utilisateur avant de lancer le processus de génération de mutants. En effet, dans certains cas il est possible d'utiliser le codage réel des états (s'il est disponible) et cela à des fins d'optimisation. Nous utiliserons les acronymes suivants afin d'identifier les quatre possibilités (alternatives) :

- DFS ("*Direct definition of Final State*") : définition directe de l'état final.
- IFS ("*Indirect definition of Final State*") : définition indirecte de l'état final.
- SE ("*State Encoding*") : codage des états.
- SN ("*State symbolic Name*") : noms symboliques des états.

Par la suite je vais donner des détails sur les approches DFS-SN et DFS-SE car elles sont exclusivement utilisées lors des injections dans le Chapitre 3.

DFS-SE

Dans le cas d'une combinaison DFS-SE, les entrées requises pour l'injection sont insérées à la fin de l'entité FSM comme décrit en Figure 1-14. La taille des vecteurs init_state_code et final_state_code est calculée en fonction du nombre total des états.

```
entity FSM is
port
(-- inputs
        Clk, Reset : in std_logic;          -- clock and reset signals
        i1, i2 : in std_logic;              -- primary inputs
-- outputs
        o1, o2 : out std_logic;             -- primary outputs
        En_inj: in std_logic;               -- insertion
        init_state_code:   in std_logic_vector(1 downto 0);     -- insertion
        final_state_code: in std_logic_vector(1 downto 0)       -- insertion
) ;
end FSM;
```

Figure 1-14: Insertion des trois signaux d'entrées dans une entité FSM

Les codes spécifiés en signaux d'entrée avec un type classique (std_logic ou std_ulogic) doivent être convertis en noms symboliques. Ceci est réalisé par la fonction "std_to_state" représentée en Figure 1-15.

```
function std_to_state (signal S1:in std_logic_vector(1 downto 0)) return STATETYPE is
begin
        case S1 is
                when "00" =>return(idlestate);
                when "01" => return(init);
                when "10" => return(doit);
                when others => null;
        end case;
end std_to_state;
```

Figure 1-15: Exemple de fonction "std_to_state"

La solution retenue pour l'injection est l'utilisation d'une fonction "affect" Figure 1-16. Son rôle est d'injecter la transition erronée si le signal "en_inj" est mis à 1 et si l'état courant

correspond à l'état initial de la transition erronée. La fonction "affect" appelle à son tour la fonction "std_to_state".

```
function affect(state_code: std_logic_vector(1 downto 0);
        nstate_name:        STATETYPE;
        signal En_inj:      std_ulogic;
        signal init_state_code:   std_ulogic_vector(1 downto 0);
        signal final_state_code: std_ulogic_vector(1 downto 0))
return STATETYPE is

begin
        if(En_inj='1' and init_state_code=state_code)then
                return(std_to_state(final_state_code));
        else
                return(nstate_name);
        end if;
end affect;
```

Figure 1-16: Fonction "affect" pour l'injection

DFS-SN

La spécification de l'état initial et de l'état final d'une transition erronée en utilisant directement des noms symboliques a l'avantage d'éviter l'utilisation de la fonction de conversion et par conséquent d'obtenir un gain considérable en terme de logique combinatoire.

Cette fois, les entrées init_state_code et final_state_code sont déclarées de type STATETYPE comme illustré en Figure 1-17.

```
entity FSM is
port
( -- inputs
        Clk, Reset: in std_logic;   -- clock and reset signals
        i1, i2: in std_logic;       -- primary inputs
        o1, o2: out std_logic;      -- primary outputs
  -- outputs
        en_inj: in std_logic;       -- insertion
        init_state_code:   in STATETYPE;     -- insertion
        final_state_code: in STATETYPE);     -- insertion
end FSM;
```

Figure 1-17: Insertion des trois signaux d'entrée dans l'entité FSM

Cependant, dans ce cas le type énuméré des états doit être défini dans un paquetage externe. Par conséquent, la définition de type est extraite de la description initiale et mise dans un nouveau paquetage. Par la suite une variable "state_tmp" est créée et toutes les affectations de "nstate" seront remplacées par des affectations de "state_tmp". L'injection est réalisée par une instruction "if-then-else" à la fin du processus ; cette instruction est illustrée en Figure 1-18.

```
If (En_inj='1' and init_state_code=state) then
        nstate<=final_state_code;
else
        nstate<=state_tmp;
end if;
```

Figure 1-18: Exemple d'injection dans l'approche DFS-SN

1.7.4. Simulation

Après modification des fichiers, il reste à générer les scripts d'injection nécessaires à la simulation. Pour cela deux approches ont été développées : la première permet de configurer un script écrit en TCL (*Tool Command Language*) et la deuxième utilise un test bench écrit en VHDL. Le script TCL permet de piloter le logiciel de simulation (en l'occurrence *MODELSIM* de *Mentor Graphics*) en utilisant certaines de ses commandes ce qui limite la portabilité de ce script. La deuxième est totalement portable puisqu'il s'agit d'un code VHDL.

Après avoir choisi l'approche à utiliser, il faut choisir les cycles et les cibles d'injection dans le cas d'un tirage aléatoire (indépendamment de l'approche). Pour cela, un programme en C a été développé pour générer ces nombres pseudo-aléatoires en utilisant plusieurs fonctions de répartition (uniforme, normale,…). Il ne reste à l'utilisateur qu'à définir la loi de répartition, le cycle de départ et de fin, la cible de départ et de fin et le nombre d'injections.

Après quoi la campagne d'injection peut commencer. Les résultats issus de cette campagne correspondent à des traces sauvegardées dans des fichiers. Ils contiennent l'évolution des signaux observés échantillonnés à intervalle régulier (généralement le cycle d'horloge pour les systèmes synchrone). Ces traces sont utilisées par la suite pour générer des fichiers réduits qui n'enregistrent que les différences des signaux observés avec ceux de la simulation sans fautes et ceci pour chaque fichier trace (si c'est un fichier trace est vide, les signaux observés n'ont aucune différence par rapport à ceux de la simulation sans injection). Après quoi l'analyse des résultats peut commencer.

Comme montré à la Figure 1-8, les fichiers traces sont indépendants de la méthode de génération, c'est-à-dire la simulation ou l'émulation.

1.7.5. Analyse des résultats

Après génération des fichiers réduits, la phase d'analyse peut débuter. Elle comporte deux possibilités : une analyse classique consistant en la classification des résultats et une beaucoup plus détaillée consistant à générer un graphe de propagation d'erreurs.

1.7.5.1. Classification des résultats

La plupart des outils décrits dans la littérature visent à classifier les fautes, c'est-à-dire identifier les ensembles de fautes associés aux différents modes de défaillance ou de détection pouvant être activés. Ceci correspond à la génération d'un modèle simple (Figure 1-19) représentant le comportement en présence de fautes. Une telle classification permet surtout à un concepteur d'évaluer l'efficacité des mécanismes de détection d'erreurs et le taux de défaillance critique du circuit. L'outil développé réalise ce type d'analyse classique.

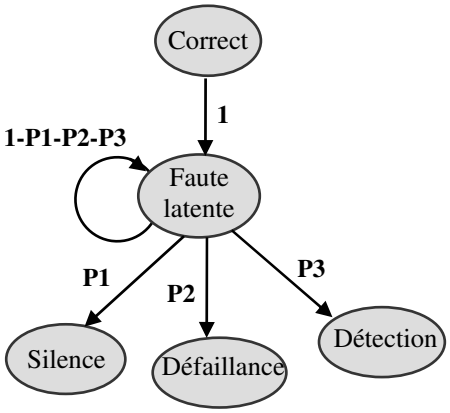


Figure 1-19: Exemple de modèle obtenu en classifiant l'effet des fautes

1.7.5.2. Graphe de propagation d'erreurs

La Figure 1-20 illustre la structure générale du modèle comportemental visé par l'outil d'analyse, correspondant à modèle fonctionnel de haut niveau du comportement du circuit en présence de fautes. Un tel modèle est complètement et automatiquement obtenu depuis les traces de simulation ou d'émulation obtenues durant la campagne d'injection. Le résultat est un graphe dont les sommets correspondent aux états atteints par le circuit durant les expérimentations. Chaque état est défini par une liste de signaux erronés parmi ceux observés durant les expériences. Chaque état est aussi étiqueté avec un type fonctionnel :

- Correct : Avant l'introduction de la faute.
- Latent : la faute injectée n'est pas encore active et donc elle n'a pas encore produit d'erreurs.
- Erreur : Une faute a été activée et au moins un des signaux observés est erroné.
- Sans erreur : Une faute a été activée mais il n'y a pas de différence parmi les signaux observés.
- Dégradé : Le circuit est opérationnel mais en mode dégradé. Cet état est spécifié par l'utilisateur.
- Détection : Un mécanisme de détection d'erreurs a été activé. Cet état est spécifié par l'utilisateur.
- Tolérance : La faute activée est tolérée par des mécanismes internes. Cet état est spécifié par l'utilisateur.
- Défaillance : Une erreur inacceptable s'est produite. Cet état est spécifié par l'utilisateur.

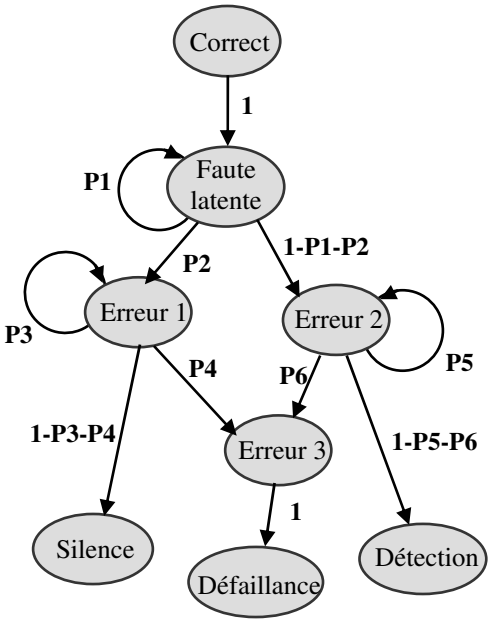


Figure 1-20: Exemple de modèle obtenu en analysant les chemins de propagation d'erreurs

Les transitions entre les états sont également enregistrées. Le graphe de propagation d'erreurs commence par l'état "Latent" et se termine par une configuration d'erreur ou bien un état de "Détection" par exemple. Chaque transition est associée à une probabilité de passage de l'état "Si" à l'état "Sj" calculée en fonction de l'activité du système au moment des expériences. Cette probabilité est obtenue à partir du pourcentage de cas où l'état "Sj" est atteint juste après l'état "Si", durant l'ensemble de la campagne d'injection.

L'analyse des résultats d'une expérience donnée peut s'interrompre sous certaines conditions optionnelles définissant des états terminaux (généralement liés à une détection ou à une défaillance). Les conditions sont spécifiées par l'utilisateur et peuvent être très précises, incluant l'état exact de certains signaux ou des propriétés d'évolution temporelles.

1.8. Conclusion

Les techniques d'injection de fautes basées sur la modification d'une description de haut niveau permettent, lorsque l'on dispose uniquement du modèle d'un système, d'obtenir une évaluation "très tôt" de la sûreté de fonctionnement de ce système. Par ailleurs, elles sont très flexibles en terme de modèle de faute puisque la plupart des modèles de fautes peuvent potentiellement être supportés et les fautes peuvent être injectées dans n'importe quel module du système. Leur principale limitation, à savoir le temps de simulation requis, peut être réduite en utilisant une approche à base de prototypage.

D'autres limitations existent toutefois dans le flot présenté. Elles seront analysées dans le chapitre suivant et des solutions seront proposées.

Chapitre 2.

Contributions sur le flot d'analyse pour un circuit numérique

2.1. Introduction

Dans le Chapitre 1, j'ai présenté le flot d'analyse développé au sein du groupe (section 1.7.2). Au moment où j'ai eu à l'utiliser plusieurs problèmes sont apparus. Le premier but de ce chapitre est de proposer des solutions aux problèmes identifiés. Un autre aspect de ma contribution sur le flot numérique consiste en l'analyse des résultats d'injection en tenant compte de façon plus précise de l'environnement du circuit. On aura ainsi une analyse beaucoup plus fine et complète que ne pourrait l'apporter une analyse des résultats avec une modélisation simplifiée de l'environnement.

2.2. Limitations du flot existant

Au moment où j'ai eu à travailler avec le flot, deux limitations principales subsistaient :

- Les modifications réalisées au niveau VHDL n'étaient valables que si la description comportait uniquement des process séquentiels cibles d'injection mais n'étant pas tous concernés par l'injection. Dans le cas de deux *process* séquentiels (ou plus) s'échangeant des données, l'instrumentation était faite de telle sorte à ne modifier que l'horloge des blocs ou du *process* concernés par l'injection. Du coup avec ces modifications et comme montré sur la Figure 2-1, les horloges Clk (du *process* séquentiel PS_1) et Clk_inj (du *process* séquentiel PS_2, celui concerné par l'injection), n'étaient pas les mêmes et faisaient apparaître des problèmes de synchronisation au niveau de PS_1 et PS_2 lors d'échanges de données. En effet, même sans injection, et lors d'une simulation au niveau RT, les horloges Clk et Clk_inj bien qu'elles changent en même temps n'ont pas le même intervalle de temps virtuel appelé delta [H.S]. Le delta dont la durée est nulle, est utilisé dans les simulateurs HDL pour ordonner les évènements "simultanés", c'est-à-dire déterminer lequel a provoqué l'autre. Dans le cas de la Figure 2-1, le Clk a provoqué le Clk_inj et c'est ce qui fait qu'au niveau du PS_2, ses registres ne récupèrent pas les données actuelles (D1) en provenance du PS_1 mais celles futures, ce qui est faux d'un point de vue synchronisation des *process*. Ce problème a été remarqué sur des exemples concrets et la solution pour le corriger est décrite à la section 2.3.1.
- Une simulation au niveau RT peut s'arrêter sans être terminée, à cause de l'occurrence d'un état d'erreur que le simulateur ne peut pas (ou n'est pas censé) résoudre. Par exemple, une faute peut faire prendre à une variable déclarée en entier une valeur hors de l'intervalle spécifié, et une telle erreur conduit à une interruption de la simulation ; il n'est par conséquent pas possible d'analyser la propagation de l'erreur après cet évènement.

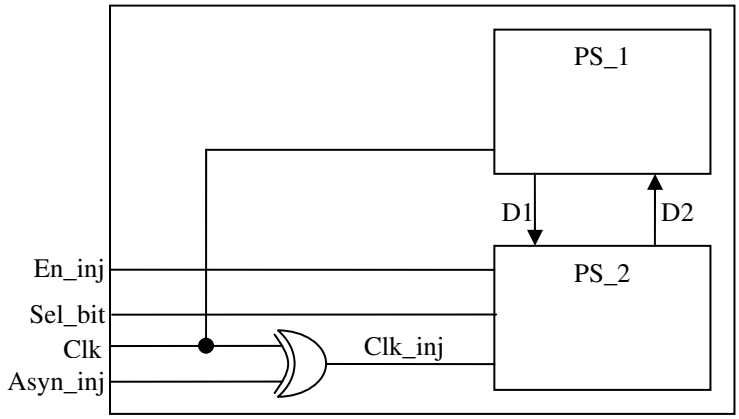


Figure 2-1: Exemple de deux process communicants dont l'un est concerné par l'injection (ancienne modification)

2.3. Contributions spécifiques sur les différentes phases du flot

Mes contributions sur les phases du flot se sont principalement limitées à la correction des problèmes présentés à la section ci-dessus, même si plusieurs scripts ont aussi été développés pour pallier certaines insuffisances (lors de l'analyse).

2.3.1. Au niveau de la modification des fichiers VHDL

Pour résoudre le problème lié à la modification des fichiers VHDL il faut que :

- Tous les *process* continuent à utiliser la même horloge après modification. C'est pourquoi le signal d'horloge Clk_inj, dans la Figure 2-2, est connecté aux deux blocs PS_1 et PS_2 même si il n'y en a qu'un seul qui est concerné par l'injection.

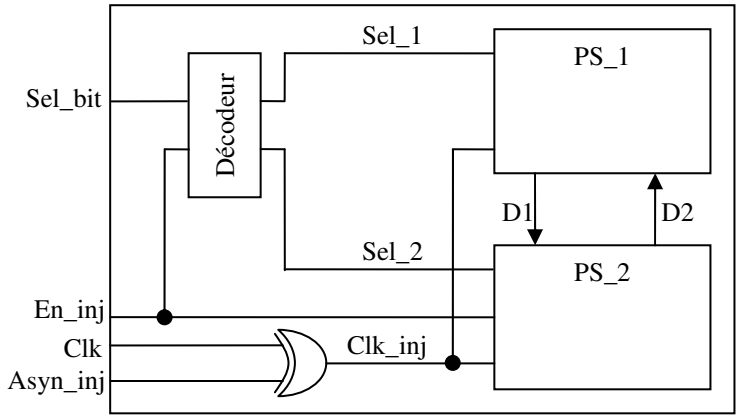


Figure 2-2: Exemple de deux process communicants dont l'un est concerné par l'injection (nouvelle modification)

- Rajouter un décodeur qui permet en fonction des cibles d'injection (Sel_bit) et du signal En_inj de piloter les signaux de sélection (Sel_1, Sel_2 dans le cas de la Figure 2-2) de telle sorte à activer ou désactiver les blocs ou les *process* concernés par l'injection. Le décodeur met à '1' les signaux de sélection (Sel_1 et Sel_2) lorsqu'il n'y a pas d'injection (En_inj et Asyn_inj à '0'). Pour une injection, il met à '1' le bloc concerné par l'injection et désactive les autres. Dans le cas de la Figure 2-2, on ne désire injecter des

fautes que dans PS_2, donc le signal Sel_1 sera toujours à '0' au moment de l'injection. Le signal PS_2 restera quant à lui toujours à '1' dans ce cas particulier, mais changerait de valeur si d'autres *process* étaient la cible d'injections.

Cette modification est plus puissante et plus générale que l'ancienne mais est plus difficile à automatiser car même les blocs dans lesquels on ne désire pas injecter doivent être modifiés afin qu'ils ne tiennent pas compte du front d'horloge supplémentaire (sur le signal Clk_inj) rajouté à l'activation du signal Asyn_inj.

La Figure 2-3 donne un exemple de modification à apporter dans le *process* où il n'y a pas d'injection. Seul le signal Sel_1 est rajouté en plus de la modification du signal clk (qui devient Clk_inj).

```
signal state, nstate: std_logic_vector (2 downto 0) ;

seq: process (Clk, Reset)
begin
        if Reset='1' then
                state <= waiting ;
        elsif (Clk'event and Clk='1') then
                state <= nstate ;
                R_i <= …;
                …;
        end if ;
end process ;
```

(a)

```
signal state, nstate: std_logic_vector (2 downto 0) ;

seq: process (Clk_inj, Reset)
begin
        if Reset='1' then
                state <= waiting ;
        elsif (Clk_inj'event and Clk_inj='1') then
                if Sel_1='1' then
                        state <= nstate ;
                        R_i <= …;
                        …;
                end if;
        end if ;
end process ;
```

(b)

Figure 2-3: Modifications à apporter pour un process dans lequel il n'y a pas d'injection : description initiale (a) description modifiée (b)

La Figure 2-4 donne un exemple de modification lorsqu'il y a injection dans un bloc. Les modifications rajoutées par rapport aux anciennes (Figure 1-12) sont données en gras.

```
signal state, nstate: std_logic_vector (2 downto 0) ;

seq: process (Clk_inj, Reset)
begin
        if Reset='1' then
                state <= waiting ;
        elsif (Clk_inj'event and Clk_inj='1') then
                if (En_inj = '0') and (Sel_2 = '1') then
                        state <= nstate ;
                        R_i <= … ;
                elsif Sel_2 = '1' then
                        case Sel_bit is
                                when "01" =>state(0) <=not state(0) ;
                                when "10" =>state(1) <=not state(1) ;
                                when "11" =>state(2) <=not state(2) ;
                                when others => null;
                        end case;
                end if ;
        end if ;
end process ;
```

Figure 2-4: Modifications pour l'injection des inversions de bits uniques sur le registre "state" dans le cas de l'exemple de la Figure 2-3

2.3.2. Au niveau de l'analyse des résultats

Pour résoudre le deuxième problème présenté à la section 2.2, un état "crash" peut être ajouté dans les graphes générés à partir de résultats de simulation au niveau RTL. Il s'agit d'un état terminal au même titre que les états "détection" ou "défaillance". Cet état "crash" correspond donc à un état terminal dans lequel l'analyse de la propagation d'erreur a été prématurément stoppée. D'un point de vue fiabilité, ceci veut dire que l'analyse n'arrive pas à prédire ce qui pourrait se produire dans le circuit. D'autres aspects des limitations de la simulation comportementale sont présentés dans le Chapitre 3. Ces limitations sont liées à certains manques d'informations, par rapport aux simulations au niveau portes : notamment, manque d'informations temporelles (en général, un modèle sans retard est employé) et méconnaissance des valeurs finales qui seront prises par les signaux phi-Booléens.

2.4. Modélisation d'un environnement complexe

Cette section représente un résumé d'une étude beaucoup plus complète réalisée dans le cadre d'un stage d'ingénieur. Des informations supplémentaires sont donc disponibles dans [D.C. 04].

2.4.1. Introduction

Identifier les défaillances d'un système en ne considérant que les comportements erronés au niveau circuit n'est pas facile. En fait, il est souvent insuffisant d'identifier les cas dans lesquels la réponse du circuit dévie de sa valeur nominale. L'amplitude ou la durée de la déviation ne sont pas non plus significatives dans bon nombre de cas. Pour distinguer les cas réels de défaillance des erreurs non critiques, les concepteurs doivent être capables de réaliser des injections de fautes dans un environnement incluant un modèle précis de l'ensemble du système. Les environnements d'émulation peuvent être vus comme une première réponse à ce besoin, mais l'investissement est important et l'approche ne peut pas toujours être utilisée pour modéliser les systèmes complexes ou hétérogènes. Par exemple il n'est pas possible dans beaucoup de cas d'avoir un prototype complet d'une voiture lors de l'analyse des effets des fautes sur un circuit conçu pour l'environnement automobile. Les outils de modélisation au niveau système sont, dans ces cas, les seuls capables de répondre à ces besoins.

De tels outils, comme *Matlab* de *The Mathworks*, ont été utilisé auparavant pour des analyses de sûreté [F.C. 03]. L'avantage principal est la possibilité de modéliser les systèmes complexes avec par exemple des parties mécaniques ou hydrauliques. Un autre avantage est la réutilisation possible, pour de telles expériences, de modèles développés dans la phase préliminaire de conception du système. La limitation principale est que les fonctions électroniques implémentées dans le circuit (par exemple, les fonctions de contrôle implémentées en circuit numérique) sont modélisées au même niveau que l'environnement du circuit (par exemple le moteur ou la suspension de la voiture). Une telle modélisation très abstraite ne

permet pas au concepteur d'analyser l'effet de fautes réelles qui pourraient potentiellement se produire dans le circuit, telles que les inversions de bits dans les registres internes. Pour réaliser de telles analyses, les injections doivent être réalisées au niveau RT (ou un niveau plus bas). Ceci nous a conduit à l'utilisation d'un modèle mixte pour connecter la description RTL du circuit avec la modélisation de l'environnement au niveau système représentant l'application.

2.4.2. Outils utilisés pour une analyse de sûreté mixte

Le but de l'étude est d'évaluer l'intérêt d'une modélisation mixte, incluant celle au niveau RT d'un circuit numérique et celle au niveau système de l'environnement.

Les descriptions VHDL généralement développées au sein du groupe sont validées en utilisant le simulateur *ModelSim* ; cet outil est donc utilisé dans le cadre de cette étude. Cependant les principales conclusions sur les bénéfices potentiels de l'approche proposée ne sont pas directement dépendantes du choix du simulateur.

Plusieurs outils sont utilisés pour la modélisation des systèmes. Il est important de choisir un outil qui soit largement utilisé dans l'industrie et qui permette d'utiliser des modèles de systèmes disponibles. En fait, le modèle au niveau système de l'environnement du circuit devrait être un modèle préalablement développé lors de la conception préliminaire du système. Cette réutilisation est très importante d'un point de vue pratique et il n'est pas intéressant par conséquent de choisir un langage de modélisation ou un outil qui n'est pas largement utilisé dans l'industrie. De plus, avoir la capacité de représenter graphiquement le déroulement de l'application est un avantage pour visualiser les comportements nominaux et erronés. Les candidats potentiels sont *Labview* de *National Instruments*, *Matlab/Simulink* de *The Mathworks*, et d'autres environnements similaires (*LabWindows/CVI* ou *Scilab* de l'INRIA).

Labview a été récemment présenté comme un outil de modélisation en plus de ses capacités pour l'instrumentation. Cependant, les fonctions de modélisation ne sont pas vraiment supportées dans la version qui était disponible au moment de l'étude. De plus, la connexion entre un modèle *Labview* et un modèle VHDL nécessite l'utilisation de *Multisim* d'*Electronics Workbench* [R.M] ou bien de rajouter *Simulink* comme interface.

LabWindows/CVI offre des capacités semblables avec des outils de développement en C, mais est limité à l'environnement *Windows*. Comme pour *Labview*, la connexion avec un modèle HDL n'est pas simple.

Matlab/Simulink est très adapté à la modélisation de systèmes et est classiquement utilisé dans des environnements industriels, par exemple pour le développement d'équipements automobiles. Des systèmes complexes peuvent être facilement décrits en utilisant un langage simple appelé M. De plus, *The Mathworks* a développé un outil spécial [L.M] pour connecter des modèles de haut niveau dans ce langage à des modèles HDL simulés avec *ModelSim* en utilisant une interface de co-simulation bidirectionnelle (Figure 2-5). Cet outil peut aussi être facilement connecté à *Simulink* à travers la même interface.

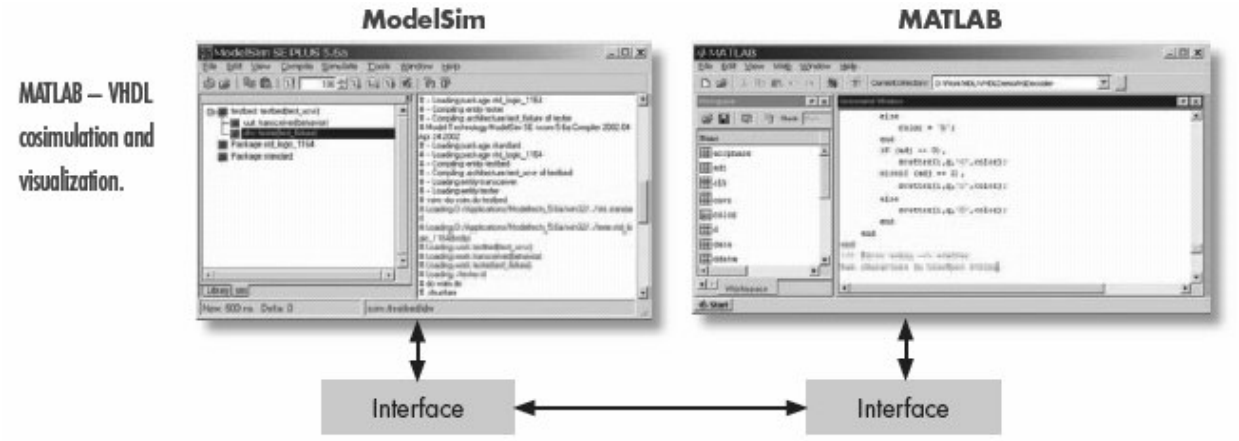


Figure 2-5: Lien entre Modelsim et Matlab/Simulink [L.M]

Scilab/Scicos est distribué gratuitement et est très semblable à *Matlab/Simulink*. Mais il n'y a pas de lien implémenté avec les simulateurs HDL.

En conséquence, nous avons choisi lors de cette étude de modéliser l'environnement du circuit avec *Matlab/Simulink* et de profiter du lien avec *ModelSim*.

2.4.3. Etude de cas

2.4.3.1. Choix de l'exemple

Afin de démontrer les bénéfices potentiels de la modélisation mixte proposée, plusieurs critères sont définis lors du choix du cas d'étude :

- L'exemple doit évidemment inclure des fonctions qui seront implémentées avec un circuit intégré numérique,
- L'environnement du circuit doit inclure des éléments complexes qui ne sont pas facilement modélisables en HDL,
- Lors des injections, les conditions de défaillance au niveau de l'application doivent être faciles à définir.

Le modèle choisi fait partie des fichiers de démonstration de *Simulink*, et est appelé "*Automatic Transmission Control*". La Figure 2-6 illustre les éléments de base du modèle initial.

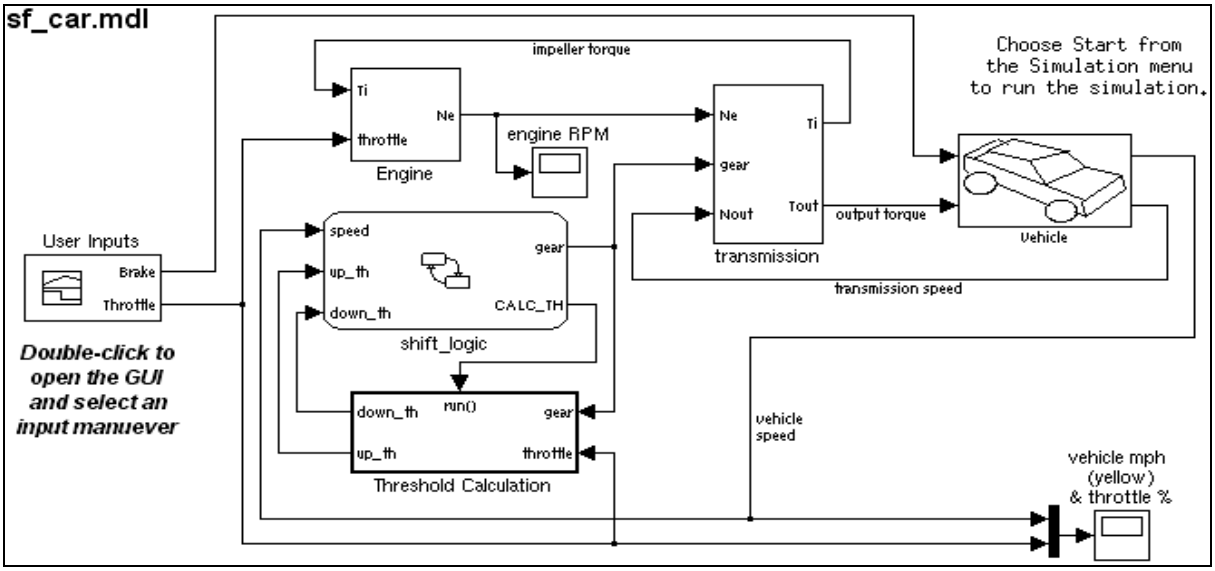


Figure 2-6: Cas d'étude choisi (version originale dans les fichiers de démonstration de Simulink)

Les contrôles externes du modèle sont le frein (*brake*) et l'accélérateur (*throttle*) et les sorties principales relatives à l'application sont la vitesse du véhicule et la vitesse de rotation du moteur. Les blocs "moteur", "véhicule" et "transmission" (respectivement "*Engine*", "*Vehicle*" et "*Transmission*" dans la Figure 2-6) ont été utilisés sans modification. Les blocs "*shift_logic*" et "*threshold calculation*" ont été sélectionnés pour être implémentés dans un circuit numérique et ont été remplacés par un bloc décrit en VHDL (Figure 2-7). Ce bloc correspond au calcul de la sélection automatique de la boîte de vitesse. La description hiérarchique inclut une machine à états finis (*transmission_control*) et deux diviseurs séquentiels entiers de 11 bits chacun utilisés pour déterminer les limites des quatre rapports de la boîte en fonction de la vitesse du véhicule et de l'accélération.

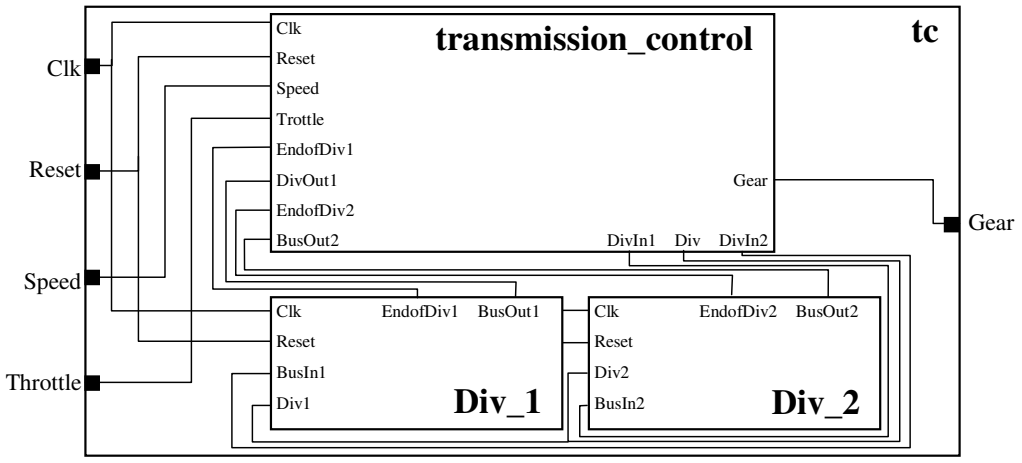


Figure 2-7: Architecture du contrôleur

Le comportement d'un tel système serait très difficilement représentable par un *testbench* VHDL, en particulier à cause des fonctions d'intégration nécessaires pour modéliser avec précision le comportement du véhicule. Cependant, les fonctions logiques sont représentées au niveau RT, nous autorisant par exemple d'injecter des fautes dans les registres d'états du bloc de

contrôle correspondant au bloc "*shift_logic*". Ceci est la principale différence avec l'étude présentée dans [F.C. 03], où le système en entier a été modélisé uniquement en utilisant *Matlab*. Ceci a évidemment quelques conséquences sur la modélisation du système, en particulier en termes de types utilisés pour quelques données. Pour l'exemple étudié, il a été nécessaire, par exemple, de transformer les valeurs réelles en valeurs entières (en rajoutant des fonctions d'arrondi), afin d'éviter l'implantation d'opérateurs complexes à virgule flottante dans le circuit. Nous avons vérifié que ceci ne change pas le comportement global du véhicule modélisé. Ce travail peut être évité si les fautes sont directement injectées dans le modèle *Matlab* comme dans [F.C. 03], mais aurait été nécessaire en pratique dans un contexte industriel réel avant l'implantation d'un circuit.

2.4.3.2. Scénarios d'application

Les simulations du modèle ont été réalisées pour quatre scénarios d'application différents : manœuvre de dépassement, accélération progressive, freinage d'urgence et avance en roue libre. Par exemple, le freinage d'urgence est illustré à la Figure 2-8, avec une évolution brusque du frein et de l'accélération.

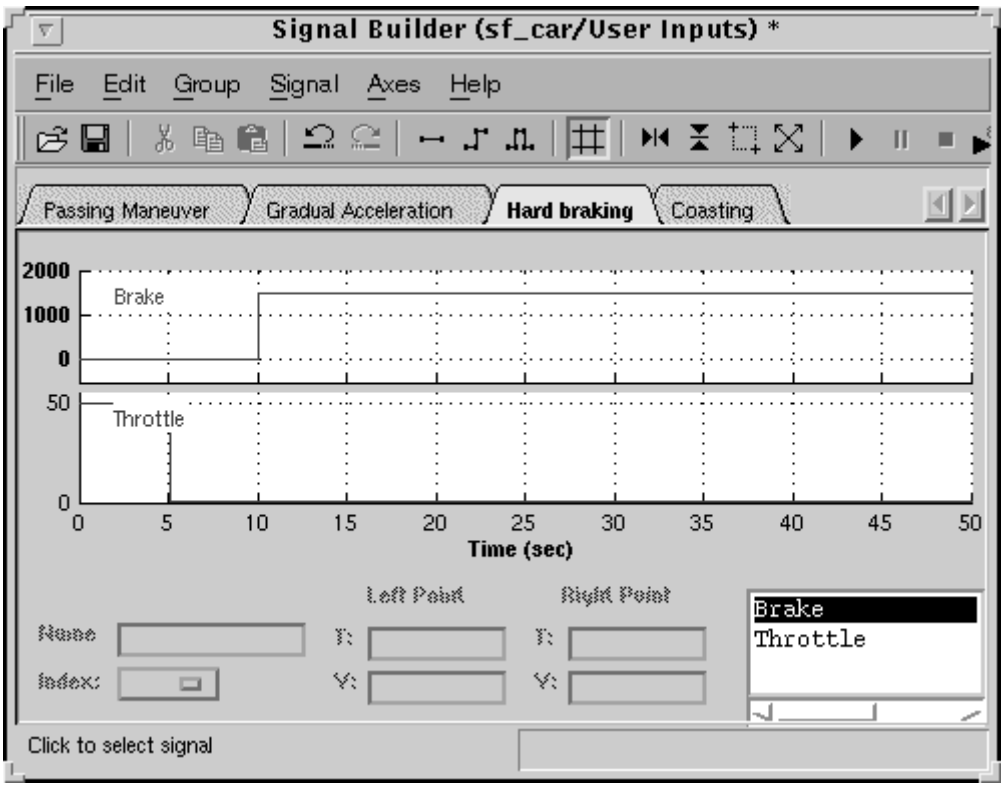


Figure 2-8: Définition des entrées du modèle pour un scénario de freinage d'urgence

2.4.3.3. Critères de défaillance au niveau système

Afin d'évaluer les conséquences réelles d'une faute sur l'application, des critères de défaillance doivent être définis au niveau système. Certains critères sont relatifs aux caractéristiques de base du modèle ; d'autres à un scénario d'application particulier.

Par exemple, pour la première catégorie, la vitesse de rotation maximale autorisée du moteur est de 5500tr/mn. Pour la deuxième catégorie deux conditions supplémentaires ont été définies pour le scénario de freinage d'urgence, correspondant à :

- L'accélération du véhicule doit être négative, après que le frein ait été activé,
- La distance parcourue avant l'arrêt complet ne doit pas être supérieure à la distance d'arrêt d'urgence, définie par $(Vs/10)^2$, où Vs est la vitesse du véhicule.

Dans le cas d'une manœuvre de dépassement, les deux conditions deviennent :

- La distance parcourue pendant la manœuvre par le véhicule qui dépasse correspond au moins à la distance parcourue par le véhicule dépassé (supposé à la même vitesse que le véhicule qui dépasse au début de la manœuvre) incrémentée par les distances de sécurité et par la longueur du véhicule dépassée (distance supplémentaire évaluée à 7m dans notre cas).
- La vitesse de rotation du moteur ne doit pas être inférieure à 605tr/mn définie comme étant la limite en dessous de laquelle le moteur pourrait caler.

2.4.3.4. Fautes injectées

L'ensemble de la campagne d'injection regroupe les fautes permanentes et transitoires dans la partie VHDL du modèle, dans le cas des quatre scénarios précédemment mentionnés. En particulier, les fautes sont injectées dans les registres d'état, conduisant à des transitions erronées.

Le but de ces expérimentations n'est pas d'étudier le comportement de l'exemple choisi pour un ensemble particulier de fautes, mais d'évaluer la possibilité de raffiner l'analyse en tenant compte d'une modélisation précise de l'environnement du circuit dans l'application. Les résultats présentés dans la prochaine section sont donnés comme exemples de cas d'injection dans lesquels le comportement observé lors des injections diffère sensiblement du comportement nominal, mais sans conséquence sur l'application, si l'on se réfère aux critères de défaillance définis dans la section 2.4.3.3.

2.4.4. Résultats et discussion

2.4.4.1. Exemple d'un freinage d'urgence en présence de fautes

La Figure 2-9 illustre un exemple d'injection de fautes durant un freinage d'urgence.

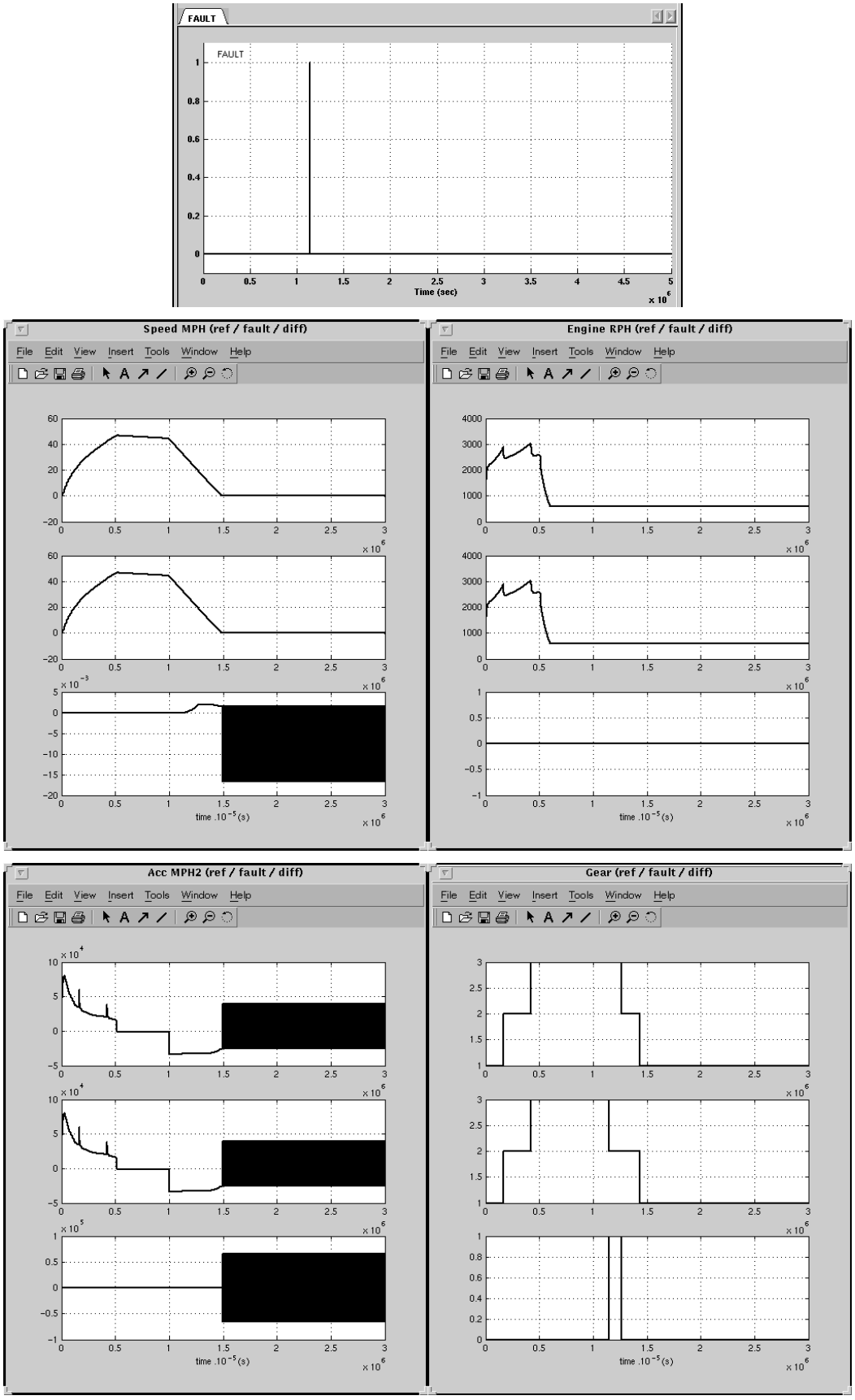


Figure 2-9: Exemple d'injection de fautes pendant le scénario "freinage d'urgence". La fenêtre supérieure montre l'instant d'injection. Chacune des autres parties de la figure montre de haut en bas pour une caractéristique donnée (vitesse du véhicule, accélération du véhicule, vitesse de rotation du moteur et rapport de vitesse sélectionné), le comportement nominal (ref), le comportement erroné (fault) et la différence entre les deux comportements (diff) en fonction du temps

Dans ce cas, une transition erronée se produit dans la machine à états du sélectionneur de rapports de vitesse, forçant le deuxième rapport au lieu du troisième. La faute est injectée à l'instant 11,44301s. Dans cet exemple, le rapport de vitesse sélectionné reste erroné pendant 1,20008s, avant que le contrôleur ne le corrige. Ceci correspond à une réponse incorrecte du circuit pendant plus de 60000 cycles d'horloge, ce qui serait probablement considéré comme une défaillance dans des évaluations classiques de sûreté. En regardant les données du système, la vitesse du véhicule et son accélération sont incorrects pendant une durée supérieure à 18s (et 3s avant le temps auquel le véhicule devrait s'arrêter). La différence moyenne est petite mais la différence dans l'accélération peut être aussi grande que 18miles/h². Cependant, aucun des critères de défaillance au niveau système n'est observé et par conséquent cette faute peut être considérée comme n'ayant pas d'effets critiques sur l'application, même si son effet est assez long.

2.4.4.2. Exemple d'une manœuvre de dépassement en présence de fautes

La Figure 2-10 montre un autre exemple pendant une manœuvre de dépassement. Dans ce cas, une autre machine à états du circuit est perturbée, ce qui conduit à omettre à l'instant 18,00124s une opération dans l'un des diviseurs et par conséquent à supprimer un calcul d'interpolation. Ainsi l'évaluation des limites entre les rapports de boîte est fausse ainsi que leur sélection. Le quatrième rapport est sélectionné au lieu du troisième pendant 2,0008s. Ceci correspond à une réponse incorrecte du circuit pendant plus de 100000 cycles d'horloge, ce qui serait probablement considéré, au même titre que l'exemple précédent, comme une défaillance dans une approche classique d'évaluation. En regardant les données du système, la vitesse du véhicule, son accélération et la vitesse de rotation du moteur sont tous incorrects pendant près de 12s, avec des différences moyennes qui sont plus grandes que dans le cas précédent. Cependant, encore une fois, aucun des critères de défaillance au niveau système n'est observé et par conséquent cette faute peut être considérée comme n'ayant pas d'effets critiques sur l'application, malgré son effet assez long.

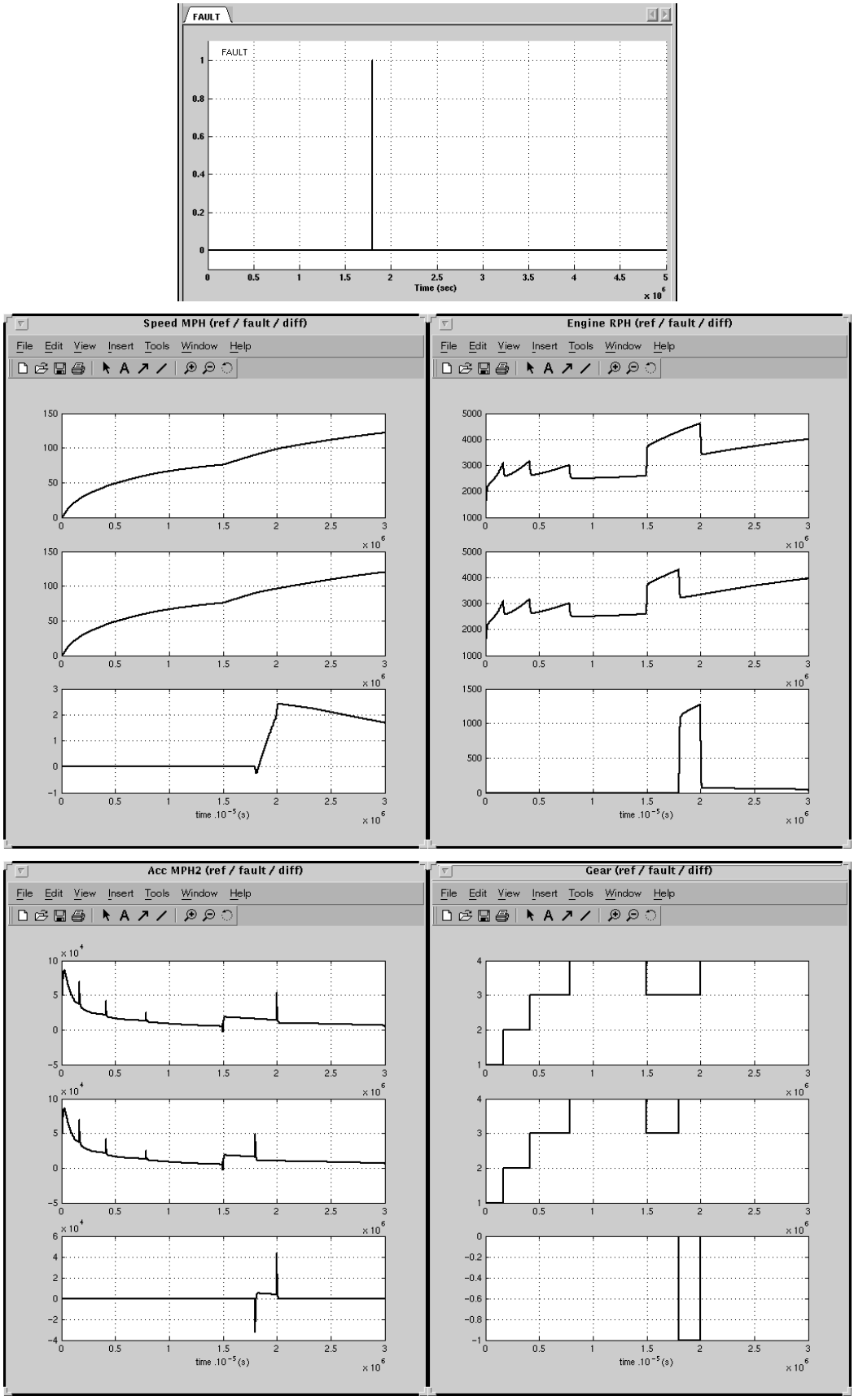


Figure 2-10: Exemple d'injection de fautes pendant une manœuvre de dépassement. La fenêtre supérieure montre l'instant d'injection. Chacune des autres parties de la figure montre de haut en bas pour une caractéristique donnée (vitesse du véhicule, accélération du véhicule, vitesse de rotation du moteur et rapport de vitesse sélectionné), le comportement nominal (ref), le comportement erroné (fault) et la différence entre les deux comportements (diff) en fonction du temps

2.5. Conclusion

Les premières contributions sur le flot d'analyse numérique ont permis de pallier certaines insuffisances. La contribution principale a cependant concerné la modélisation de l'environnement applicatif du circuit.

L'intérêt de tenir compte d'un modèle précis du système global lors d'une évaluation du comportement erroné d'un circuit a été démontré. L'approche proposée permet à un concepteur d'évaluer les conséquences réelles des fautes sur l'application avec une bonne précision. Des exemples significatifs ont été fournis pour différents scénarios de l'exemple choisi.

Les exemples d'injection donnés dans la section 2.4.4 correspondent à des fautes survenant aux moments où elles peuvent mener à une valeur fausse du rapport de boîte pendant une durée relativement longue. Ceci est possible parce que plusieurs rapports sont valides pour certaines valeurs du couple (vitesse, accélération). Les résultats expérimentaux permettent au concepteur d'évaluer la criticité des fautes injectées, et potentiellement de modifier les limites des différents rapports.

Chapitre 3.

Analyse au niveau RTL : avantages et limitations

3.1. Introduction

Comme indiqué précédemment, deux types d'analyses peuvent être réalisées : une classification des fautes, en identifiant celles conduisant à des évènements particuliers, ou une analyse de propagation d'erreurs, montrant les configurations erronées internes activées par les fautes, et aidant le concepteur à mieux identifier les points critiques dans le circuit. Ces deux types d'analyse peuvent être réalisées à partir d'une description de haut niveau ou au niveau portes c'est-à-dire après synthèse.

Ce chapitre présente une étude de cas utilisant une description VHDL du microcontrôleur 8051 (3.2). L'intérêt principal n'est pas de discuter du comportement du microcontrôleur 8051 quand les fautes se produisent, comme c'est le cas par exemple dans [G.C. 02], mais plutôt d'utiliser cet exemple pour discuter les avantages et les limitations des analyses réalisées très tôt dans le flot de conception, c'est-à-dire avant la synthèse. Les résultats reportés dans ce chapitre commencent par une comparaison détaillée entre les deux niveaux d'analyse (3.3) en présentant leurs avantages et inconvénients en terme de durée d'expérimentation, de taille de fichiers générés et de qualité pour la classification ou l'analyse de propagation d'erreurs. Par la suite nous nous intéresserons à la combinaison des deux types d'analyse (3.4), en explorant la dépendance entre l'application et les configurations critiques (c'est-à-dire les configurations d'erreur internes menant aux défaillances) qui peuvent être identifiées au niveau RT. De plus, nous discuterons le nombre d'expériences d'injection requis pour avoir un bon compromis entre la durée des expériences et la qualité de l'analyse. Ce point est très important pour éviter de perdre trop de temps pendant les injections préliminaires, exécutées pour identifier les parties principales du circuit qui ont besoin de protections. Enfin, nous montrerons l'utilisation possible de la même approche pour l'analyse d'une méthode de protection au niveau logiciel.

3.2. Etude de cas basée sur le microcontrôleur I8051

Nous avons choisi un modèle du microcontrôleur 8051 d'Intel comme véhicule d'expérimentation. En effet sa disponibilité ainsi que son architecture simple et rapidement compréhensible le rendent très attractifs.

3.2.1. Modèle comportemental

Les expériences sont basées sur la description VHDL synthétisable disponible dans sa version 2.4 depuis [M.V]. Il est compatible avec le jeu d'instructions au niveau binaire, bien qu'il ne soit pas complètement compatible avec la version d'Intel d'un point de vue temps de cycles. Cette description est divisée en cinq blocs (Figure 3-1) :

- Le décodeur (I8051_DEC) : Modèle du bloc qui transforme les instructions 8051 non uniformes en représentations uniformes, c'est-à-dire en codes énumérés. Ce modèle est décrit comme un flot de données implémentant un bloc logique combinatoire.

- Le contrôleur (I8051_CTR) : Modèle du noyau processeur du 8051. Il définit les états du CPU et l'exécution des instructions décodées. Ce modèle est décrit de manière comportementale comme un bloc logique séquentiel.
- L'ALU (I8051_ALU) : Modèle de l'Unité Arithmétique et Logique (ALU) qui exécute les calculs spécifiques du 8051. Ce modèle est décrit de manière comportementale comme un bloc logique combinatoire.
- La RAM (I8051_RAM) : Modèle de 128 octets de RAM, spécifique au 8051, c'est-à-dire adressable par bit. Il contient aussi les registres de fonction spéciaux (SFR) correspondant aux registres internes du processeur. Ce modèle est décrit de manière comportementale comme un bloc logique séquentiel.
- La ROM (I8051_ROM) : Modèle d'une ROM atteignant jusqu'à 4Ko. Ce modèle est généré automatiquement et de manière comportementale, comme un bloc logique séquentiel.

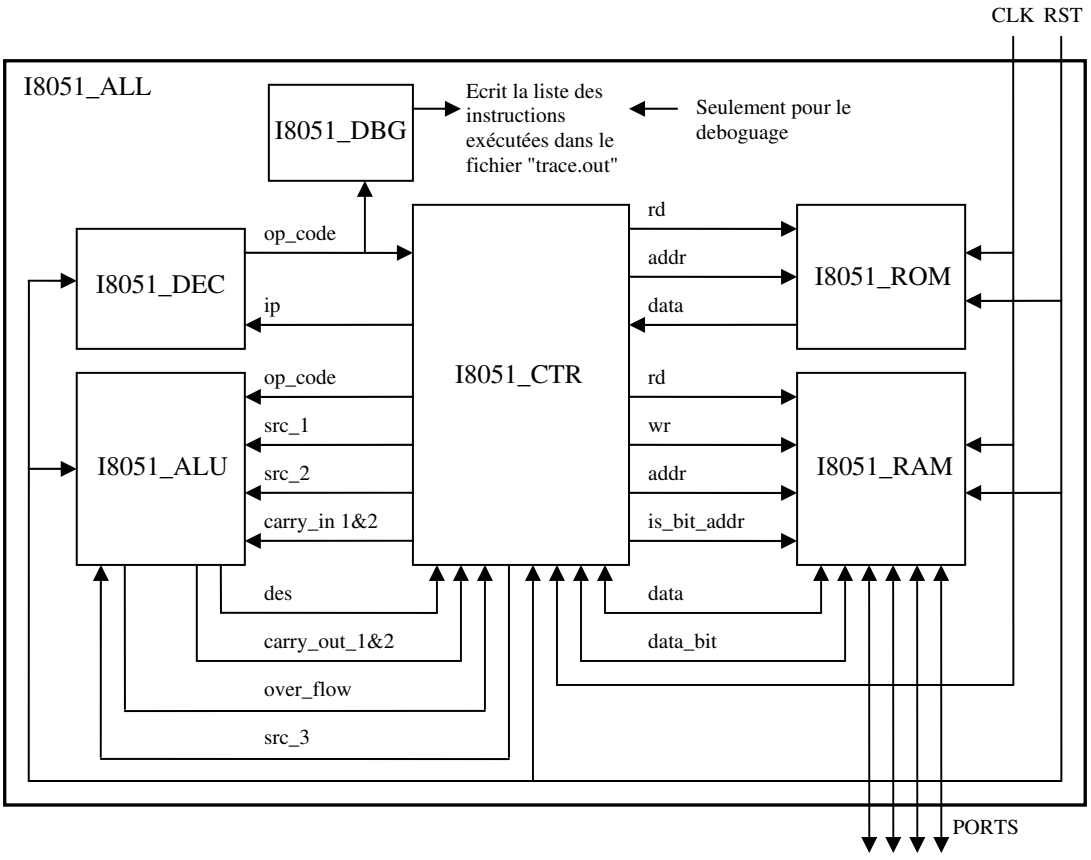


Figure 3-1: Diagramme bloc du 8051

Le bloc contrôleur contient le compteur de programme (Reg_pc), un registre d'instruction (Reg_op1), deux registres d'opérande (Reg_op2, Reg_op3), l'accumulateur (Reg_acc), 71 points mémoires enregistrant les signaux de contrôle intermédiaires et deux registres d'états, CS et EXE. Le registre CS définit un des quatre états possibles du CPU. Le registre EXE est utilisé pour définir jusqu'à 8 étapes de base d'exécution pour chacun des états du CPU.

3.2.2. Programmes exécutés

Les analyses détaillées par la suite ont été réalisées en exécutant 4 applications différentes qui correspondent à : (1) une multiplication de matrice 2x2 ("Matrix"), (2) un calcul du résultat d'un filtre à réponse impulsionnelle à 17 coefficients pour un vecteur d'entrée de 4 éléments ("FIR"), (3) un calcul de nombre premier avec l'algorithme du crible d'Eratosthenes ("Sieve") et un tri d'un vecteur à 10 éléments ("Bubble"). Ces programmes sont écrits en C et on utilise le compilateur Keil [K.E] (avec les mêmes options pour tous les programmes) pour les convertir au format "hex". Ce format est utilisé pour produire des fichiers VHDL représentant la ROM.

Le nombre de cycles nécessaires pour calculer les résultats est 13741, 51516, 52145 et 155199 respectivement pour "Matrix", "Fir", "Sieve" et "Bubble". A la fin de l'exécution, les résultats sont disponibles en RAM. Des données statistiques concernant ces programmes seront fournies à la section 3.4.2.

Des versions durcies du programme "Matrix" seront présentés dans la section 3.6.3.

3.2.3. Cibles d'injection

Les fautes injectées durant ces expériences représentent le phénomène SEU. Des inversions de bits sont donc injectées dans les points mémoires et les registres de données. Dans le cas de variables d'états spécifiées en utilisant un type énuméré, les inversions de bits peuvent uniquement être injectées après codage binaire des états ; le modèle des transitions erronées est donc utilisé avant cette affectation.

Les campagnes d'injection sont spécifiées de telle sorte à avoir un bon compromis entre la précision des résultats et la durée des expériences. Les injections ont été réalisées durant l'exécution des différents programmes, après les procédures d'initialisation. Elles sont donc aléatoirement distribuées entre 7547, 43554, 46342 et 147662 cycles respectivement pour les programmes "Matrix", "Fir", "Sieve" et "Bubble". En raison des différents nombres de cycles, le nombre de fautes injectées pour chaque campagne (c'est-à-dire pour chaque cible dans le processeur) n'est pas le même pour tous les programmes. 2000 fautes ont été injectées dans le cas de "Matrix" parce que ceci correspond déjà à un grand pourcentage de toutes les fautes possibles. 6000 fautes ont été injectées dans le cas de "Fir" et "Sieve". Les injections dans le cas de "Bubble" sont limitées à 2000 fautes à cause de la lenteur de chaque expérience (chaque simulation prend à peu près 13 s sur une station de travail SunBlade 100). La précision des résultats obtenus sera discutée en détails dans la section 3.5.

La classification des fautes est réalisée en se basant sur la justesse des résultats au cycle correspondant à la fin de l'application exécutée en l'absence de fautes. Trois classes de résultats ont donc été générées : résultats corrects, résultats incorrects et situation de "crash".

La propagation d'une erreur n'implique pas toujours que les résultats soient faux du point de vue de l'application. Dans plusieurs cas, la faute peut mener à une exécution différente du programme, en changeant le comportement des signaux observés durant un certain nombre de

cycles d'horloge, mais en ne changeant pas le résultat final du calcul. Le comportement exact du système durant un calcul n'est pas vraiment significatif pour le type d'application étudié, pourvu que le résultat correct soit finalement disponible dans la RAM, aux adresses correctes, et au temps auquel ces résultats sont attendus. La corrélation entre la justesse des résultats et la propagation d'erreurs est reportée en détail dans la section 3.4.

Les cycles d'injection et les cibles (les bits des registres) sont générés aléatoirement excepté pour CS et EXE lors de l'exécution du programme "Matrix" où l'injection est exhaustive. On considère deux groupes de cibles :

- CS (transitions erronées et inversion de bits) et EXE (transitions erronées). Sept signaux ou bus sont observés durant les expériences, qui correspondent à un maximum de 129 états possibles et 16642 transitions possibles dans le graphe de propagation d'erreurs, après que l'état "Latent" soit atteint (l'état "Crash" est inclus).
- Reg_pc, Reg_op1, Reg_op2, Reg_op3, RAM_UTIL (partie de la mémoire RAM utilisée par une application donnée), SFR, Reg_acc. Huit signaux ou bus sont observés durant les expériences, qui correspondent à un maximum de 257 états possibles et 66050 transitions possibles dans le graphe de propagation d'erreurs.

3.3. Comparaison entre les niveaux RTL et portes

Cette comparaison tient compte des critères suivants : (1) durée des expérimentations, (2) taille des fichiers, (3) qualité de la classification et (4) l'analyse de propagation d'erreurs.

3.3.1. Durée des expérimentations

Lors de la comparaison entre les analyses RTL et niveau portes, le premier critère est la durée des expériences. Une campagne complète d'injection de fautes requiert un nombre très grand de simulations et le temps requis pour chaque expérience est souvent une limitation d'ordre pratique. Le Tableau 3-1 résume, pour deux types de stations de travail, la perte moyenne relative de vitesse en utilisant la simulation au niveau portes afin d'évaluer l'effet d'injections de fautes dans le registre CS. Ces mesures concernent uniquement la génération de fichiers traces, avant tout traitement pour générer les graphes. Dans cette première partie de la campagne, le temps requis peut être réduit par 2 ordres de grandeur en utilisant la simulation RTL (par exemple sur la station Sun Blade, la simulation au niveau portes est 121 fois moins rapide que la simulation comportementale pour cette campagne particulière). Par contre, comme on peut le voir sur le Tableau 3-2, il n'y a pratiquement aucune différence à utiliser la Sun Blade 100 ou la Sun Ultra 10 lors de la génération du graphe de propagation d'erreurs.

Tableau 3-1: Temps relatif nécessaire pour chaque expérience (génération de fichiers trace)

Génération de fichiers traces	Temps moyen pour chaque expérience	
	Sun Blade 100	Sun Ultra 10
Simulation comportementale	1	1
Simulation au niveau portes	121	71

Tableau 3-2: Temps d'analyse relatif nécessaire pour chaque campagne (génération du graphe de propagation d'erreurs)

Génération de fichiers traces	Temps moyen par campagne	
	Sun Blade 100	Sun Ultra 10
Simulation comportementale	1	1
Simulation au niveau portes	0,95	0,95

3.3.2. Taille des fichiers

Le second critère est l'espace disque requis par les expériences. Le Tableau 3-3 montre qu'ici aussi la simulation comportementale est plus efficace.

Tableau 3-3: Espace disque relatif nécessaire pour chaque expérience (génération de fichiers trace)

	Espace disque moyen pour chaque expérience	
	Fichiers trace de simulation	Fichiers intermédiaires durant le traitement
Simulation comportementale	1	1
Simulation au niveau portes	2,76	1,54

3.3.3. Qualité de la classification des résultats au niveau RTL

Comme montré dans les sections précédentes, la simulation au niveau comportemental a plusieurs avantages par rapport à celle au niveau portes. Cependant, elle a aussi un inconvénient important, lié au plus faible niveau de détails disponible. Un tel inconvénient est analysé dans [R.L. 02] du point de vue de l'architecture et il a été montré que des informations significatives peuvent être obtenues depuis une description de haut niveau du circuit, même si l'architecture décrite diffère de l'architecture finale. La comparaison dans cette section pointe sur un autre aspect, celui des limitations de la simulation comportementale, pour une description donnée du circuit. Ces limitations ont essentiellement deux raisons. La première est le modèle de temporisation (en général, le modèle zéro-délai) et la seconde est liée aux valeurs logiques spécifiques (phi-Booléens) utilisées durant les simulations au niveau RT, qui peuvent mener à quelques contradictions avec de futures simulations au niveau portes, après synthèse.

Les résultats de classification sont résumés dans le Tableau 3-4 et 3-5, pour des injections de fautes dans les deux registres de contrôle qui conduisent au plus grand nombre de "crash" durant la campagne (Reg_pc et CS). Les mêmes fautes ont été injectées durant les simulations au niveau RT et portes, respectivement en utilisant la description comportementale et celle synthétisée du circuit. La classification des fautes a été réalisée en se basant sur la justesse des résultats à la fin de la multiplication des matrices (c'est-à-dire au cycle correspondant à la fin de la multiplication quand il n'y a pas de fautes). Les cycles d'injection et les cibles (les bits des registres) ont été générés aléatoirement.

Tableau 3-4: Résultats de classification de fautes après injection des mêmes 2000 fautes dans le compteur de programme Reg_pc au niveau RT et portes

Niveau RT	Niveau portes		
	Correct	Incorrect	Total
Correct	873 (68,47%)	0 (0%)	873
Incorrect	31 (2,43%)	490 (67,59%)	521
Crash	371 (29,1%)	235 (32,41%)	606
Total	1275	725	2000

Tableau 3-5: Résultats de classification de fautes après injection des mêmes 2000 fautes dans le registre CS au niveau RT et portes

Niveau RT	Niveau portes		
	Correct	Incorrect	Total
Correct	1125 (95,74%)	3 (0,36%)	1128
Incorrect	37 (3,15%)	478 (57,94%)	515
Crash	13 (1,11%)	344 (41,70%)	357
Total	1175	825	2000

Au niveau RT, trois classes de fautes sont générées, incluant le "crash". Au niveau portes, seulement deux classes de fautes restent (celles correspondant aux résultats corrects et incorrects). Comme on peut le voir dans les tableaux (spécialement pour le Tableau 3-5), un grand pourcentage des fautes conduisant à un arrêt de la simulation conduit à des résultats faux au niveau portes. Une hypothèse de "pire cas" revient donc à considérer qu'une telle faute est critique. Une analyse plus détaillée montre que, pour les injections dans le registre CS, les 13 cas dans lesquels le "crash" correspond finalement à des résultats corrects concernent des injections réalisées après la fin du programme "Matrix", juste après l'enregistrement du résultat final. Toutes les injections conduisant à un "crash" avant cet instant conduisent à des résultats incorrects.

Un deuxième point intéressant est la très bonne corrélation observée entre les classifications aux deux niveaux. Seulement un pourcentage réduit de fautes conduisant à des résultats incorrects au niveau RT n'a finalement aucun impact fonctionnel au niveau portes (à cause du raffinement de la conception du circuit). Par ailleurs, très peu de fautes mènent à une situation critique où les résultats sont corrects à haut niveau et incorrects à bas niveau. Cette dernière situation devrait être évitée puisqu'elle correspond à un cas dans lequel le concepteur ne sera pas capable de détecter assez tôt le problème, exigeant ainsi potentiellement des modifications de la description du circuit assez tard dans le processus de conception. Les trois cas observés durant les injections dans CS ont été analysés, et il apparaît qu'ils correspondent tous à une situation unique (même type de fautes dans CS durant la même étape d'exécution d'une instruction donnée, à trois moments différents lors de l'exécution du programme "Matrix"). Dans cette situation particulière, la différence entre les deux analyses est causée par les phi-Booléens utilisés dans la description comportementale, qui sont remplacés par des zéros après la synthèse. Ces zéros ont un impact sur l'instruction de saut exécutée juste après l'injection. Les différences pourraient donc être évitées en supprimant les phi-Booléens dans la description de haut niveau.

La haute précision des résultats obtenue avec des analyses au niveau portes (quand les phi-Booléens sont utilisés) est seulement significative quand la technologie cible durant la synthèse est définitivement fixée ; une synthèse réalisée avec une autre technologie (par exemple pour un prototypage) peut utiliser des affectations différentes des phi-Booléens et dans ce cas l'analyse n'est pas plus proche de la réalité (bien que réalisée au niveau portes) que l'analyse réalisée directement au niveau RT avant les affectations. Réaliser les expériences par prototypage peut bien sûr mener à une accélération des expériences, mais pas nécessairement à une analyse plus exacte que la simulation au niveau RT.

3.3.4. Analyse de propagation d'erreurs aux niveaux RTL et portes

Les expériences réalisées sur le microcontrôleur 8051 confirment une des conclusions présentées dans [R.L. 02] ; le nombre d'états et de transitions dans les graphes de propagation d'erreur générés après les expériences est très petit comparé avec le nombre total de configurations erronées potentielles des signaux observés. Ces graphes aident remarquablement le concepteur à identifier les configurations erronées qui devraient être prises en compte pendant le durcissement.

Un autre aspect intéressant des analyses au niveau RT est, dans le cas présenté, la génération de graphes de propagation d'erreur plus simples que ceux obtenus au niveau portes. Cela peut donc simplifier la tâche du concepteur. Mais cette caractéristique intéressante semble directement liée à l'utilisation des phi-Booléens dans la description comportementale. Il pourrait ainsi y avoir un certain compromis à définir, tenant compte de la discussion dans la section précédente.

3.3.5. Synthèse comparative

Le Tableau 3-6 résume les différences entre les deux niveaux RT et portes. Premièrement que ce soit en termes de vitesse d'exécution ou de taille de fichiers générés, le niveau RT est beaucoup plus adéquat surtout pour la vitesse. En ce qui concerne la classification d'erreurs, plus les détails de l'implantation finale sont disponibles, meilleure sera la qualité de l'analyse. Donc le niveau portes est mieux, cela dit compte tenu des autres critères et au vu des résultats obtenus, le niveau RT reste une très bonne alternative. Finalement, en termes de graphes de propagation d'erreurs, le niveau RT semble préférable dans certains cas. Donc le choix du niveau d'analyse par la suite sera le niveau RT.

Tableau 3-6: Synthèse comparative entre les niveaux RT et portes

Niveau	Vitesse d'exécution	Taille des fichiers	Qualité de la classification	Propagation d'erreurs
RT	+++	++	++	++
Portes	+	+	+++	+

3.4. Combinaison de la classification et de l'analyse de propagation d'erreur au niveau RTL

Nous présentons dans cette section la combinaison de la classification et de l'analyse de propagation d'erreur au niveau RT. Tout d'abord, nous présentons les résultats d'injection puis l'impact du sous-ensemble d'instructions utilisé par l'application et l'impact de la durée de la campagne d'injection de fautes.

3.4.1. Résultats d'injection

Cette section discute les résultats détaillés donnés dans les Tableaux 3-7 à 3-19.

En regardant les graphes de propagation d'erreurs générés pour chaque classe de fautes, il apparaît clairement que certaines configurations d'erreurs sont spécifiques à une classe donnée. Le nombre de telles erreurs spécifiques est reporté dans les trois premières colonnes des Tableaux 3-7, 3-10, 3-13, 3-15, 3-16, 3-18 et 3-19. Ces tableaux montrent aussi le nombre d'états communs aux deux ou trois classes, et le nombre total d'états dans les graphes de propagation d'erreurs générés pour la totalité des expérimentations (colonne "Total"). Les pourcentages sont calculés par rapport au nombre total de configurations d'erreurs actuellement observées durant les expérimentations. Le nombre reporté dans la colonne "Spécifique Crash" inclut l'état "Crash" lui-même, donc un zéro dans cette colonne signifie qu'il n'y a pas de crash durant la campagne d'injection.

Les Tableaux 3-8, 3-9, 3-11, 3-12, 3-14 et 3-17 comparent les états spécifiques à certaines classes, observées pour les différents programmes d'application.

3.4.1.1. Injection dans les registres de contrôle

Comme illustré dans les Tableaux 3-7, 3-10, 3-13, 3-15 et 3-16, la répartition des états dans les différentes classes est presque la même pour les différents benchmarks. L'analyse détaillée des résultats conduit à ces remarques :

- Un pourcentage réduit des configurations erronées possibles est effectivement observé. De tels pourcentages réduits ont été obtenus durant d'autres campagnes d'injection de fautes avec différents circuits et semble donc être un cas fréquent.
- Bien que le pourcentage d'erreurs observées soit petit, les états erronés recensés lors de l'injection dans une cible donnée sont presque les mêmes pour les différents benchmarks. Par exemple seulement 26% des configurations erronées possibles sont observées lors de l'injection d'inversion de bits dans CS. Toutefois, les configurations obtenues en exécutant "Bubble" sont toutes incluses dans celles obtenues pour les mêmes injections en exécutant "Matrix", malgré les différences notables entre les deux programmes. De même, seulement 18% des configurations possibles sont observées en

injectant des inversions de bits dans Reg_op1 mais les configurations obtenues sont presque les mêmes en exécutant "Matrix" ou "Sieve".

- La similitude en termes de configurations erronées observées n'est pas reflétée dans la classification des résultats. L'effet d'une configuration d'erreur donnée, et par suite la classe à laquelle chaque état appartient, apparaît comme très dépendant du programme.

Tableau 3-7: Injection d'inversion de bits dans le registre CS

	Spécifique			Commun				Total
Bench	Correct	Incorrect	Crash	Correct/ Incorrect	Correct/ Crash	Incorrect/ Crash	Tout	
Matrix	0 (0%)	4 (12%)	8 (24,2%)	2 (6,1%)	1 (3%)	0 (0%)	18 (54,5%)	33
Fir	0 (0%)	2 (6,4%)	7 (22,6%)	3 (9,7%)	0 (0%)	4 (12,9%)	15 (48,4%)	31
Sieve	0 (0%)	1 (3,1%)	9 (28,1%)	0 (0%)	0 (0%)	3 (9,4%)	19 (59,4%)	32
Bubble	0 (0%)	0 (0%)	7 (25,9%)	4 (14,8%)	1 (3,7%)	0 (0%)	15 (55,6%)	27

Les injections d'inversion de bits dans CS sont exhaustives pour "Matrix". Pour "Fir", "Sieve" et "Bubble", respectivement 6,88%, 6,48% et 0,68% des fautes possibles (sélectionnées aléatoirement) sont injectées. Un nombre réduit d'états a été observé dans le cas de "Bubble", qui peut être expliqué par le pourcentage réduit d'injections réalisé. Dans le cas de "Sieve", la classe "Crash" contient plus d'états que dans les autres cas ; en conséquence, il y a plus d'états communs à toutes les classes. Pour trois benchmarks, quelques configurations erronées sont observées seulement quand les résultats finaux sont incorrects. Pourtant, le Tableau 3-8 montre que ces états dépendent principalement de l'application. Seulement les états Cs_bit-flips_incorrect_2 et Cs_bit-flips_incorrect_3 sont communs à "Matrix" et "Fir". De tels états sont recensés dans un nombre réduit d'expériences en regard à l'ensemble de la campagne, et correspondent aux étapes critiques du calcul. Par exemple, pour la multiplication de matrices, Cs_bit- flips_incorrect_2 et Cs_bit-flips_incorrect_3 correspondent à l'écriture dans la zone de données de la mémoire, qui est perturbée par l'injection. Ces états particuliers peuvent permettre à un concepteur d'identifier les étapes critiques ou localisations, pour une application particulière exécutée sur le processeur. Le Tableau 3-9 montre que les états suivis systématiquement par un arrêt de la simulation sont plus indépendants de l'application utilisée durant les expériences.

Tableau 3-8: Répartition des états spécifiques à la classe "Incorrect" lors de l'injection d'inversion de bits dans le registre CS

Bench	Cs_bit-flips_Incorrect1	Cs_bit-flips_Incorrect2	Cs_bit-flips_Incorrect3	Cs_bit-flips_Incorrect4	Cs_bit-flips_Incorrect5
Matrix	+	+	+	+	
Fir		+	+		
Sieve					+

Tableau 3-9: Répartition des états spécifiques à la classe "Crash" lors de l'injection d'inversion de bits dans le registre CS

Bench	Cs_bit-flips Crash1	Cs_bit-flips Crash2	Cs_bit-flips Crash3	Cs_bit-flips Crash4	Cs_bit-flips Crash5	Cs_bit-flips Crash6	Cs_bit-flips Crash7	Cs_bit-flips Crash8	Cs_bit-flips Crash9	Cs_bit-flips Crash10
Matrix	+	+	+	+	+			+	+	
Fir	+	+	+	+	+	+				
Sieve	+	+		+	+	+	+	+		+
Bubble	+		+	+	+	+	+			

Le Tableau 3-10 montre les résultats d'injection de transitions erronées dans le registre CS. Le modèle englobe les inversions de bit uniques et multiples. Comme attendu, un peu plus de configurations erronées sont observées, bien que le nombre total reste très petit vis-à-vis de toutes les configurations possibles. Les injections sont exhaustives pour "Matrix" et sont de 4,59%, 4,31% et 0,45% des fautes possibles respectivement pour "Fir", "Sieve" et "Bubble". Ce pourcentage réduit d'injections pour "Bubble" explique qu'un état spécifique pour la classe "Correct" apparaît dans les résultats. Moins d'états spécifiques sont trouvés pour les autres classes, comparés avec les résultats d'injection d'inversion de bits, et la comparaison de configurations déclenchées lors des différentes applications (Tableaux 3-11 et 3-12) confirme les commentaires précédents. Il est à noter que les états spécifiques communs à "Matrix" et "Fir" sont aussi observés avec l'injection d'inversion de bits ; quelques configurations critiques sont d'ailleurs identifiées pour différentes applications et différents modèles. Tous les états spécifiques relatifs à la classe "Incorrect" et "Crash" pour les transitions erronées sont trouvées lors de l'injection d'inversion de bits.

Tableau 3-10: Injection de transitions erronées dans le registre CS

	Spécifique			Commun				Total
Bench	Correct	Incorrect	Crash	Correct/ Incorrect	Correct/ Crash	Incorrect/ Crash	Tout	
Matrix	0 (0%)	3 (8,1%)	6 (16,2%)	6 (16,2%)	0 (0%)	0 (0%)	22 (59,5%)	37
Fir	0 (0%)	2 (5,7%)	4 (11,4%)	5 (14,3%)	0 (0%)	2 (5,7%)	22 (62,9%)	35
Sieve	0 (0%)	0 (0%)	3 (8,82%)	1 (2,94%)	0 (0%)	3 (8,82%)	27 (79,4%)	34
Bubble	1 (3%)	0 (0%)	5 (15,1%)	7 (21,2%)	0 (0%)	0 (0%)	20 (60,6%)	33

Tableau 3-11: Répartition des états spécifiques à la classe "Incorrect" lors de l'injection de transitions erronées dans le registre CS

Bench	Cs_tr_err_Incorrect_1	Cs_tr_err_Incorrect_2	Cs_tr_err_Incorrect_3
Matrix	+	+	+
Fir		+	+

Tableau 3-12: Répartition des états spécifiques à la classe "Crash" lors de l'injection de transitions erronées dans le registre CS

Bench	Cs_tr_err Crash_1	Cs_tr_err Crash_2	Cs_tr_err Crash_3	Cs_tr_err Crash_4	Cs_tr_err Crash_5	Cs_tr_err Crash_6
Matrix	+	+	+	+	+	
Fir	+			+	+	
Sieve	+				+	
Bubble	+			+	+	+

Le Tableau 3-13 montre les résultats obtenus lors de l'injection de transitions erronées dans le registre EXE. Le modèle d'inversion de bits n'a pas été utilisé car, par défaut, ce registre est encodé en 1 parmi N, et l'injection d'une inversion de bit arrête le calcul définitivement. L'injection est exhaustive pour "Matrix" et limitée à 1,97%, 1,85% et 0,19% des cas possibles pour respectivement "Fir", "Sieve" et "Bubble". Plus d'états d'erreurs ont été obtenus pour cette cible d'injection, mais moins d'états sont spécifiques à une classe donnée. Les états spécifiques à "Crash" sont comparés dans le Tableau 3-14.

Tableau 3-13: Injection de transitions erronées dans le registre EXE

	Spécifique			Commun				Total
Bench	Correct	Incorrect	Crash	Correct/ Incorrect	Correct/ Crash	Incorrect/ Crash	Tout	
Matrix	0 (0%)	0 (0%)	3 (7,1%)	10 (23,8%)	2 (4,8%)	0 (0%)	27 (64,3%)	42
Fir	0 (0%)	1 (2,6%)	4 (10,3%)	7 (17,9%)	0 (0%)	1 (2,6%)	26 (66,7%)	39
Sieve	0 (0%)	0 (0%)	5 (12,5%)	1 (2,5%)	0 (0%)	2 (5%)	32 (80%)	40
Bubble	1 (2,8%)	0 (0%)	3 (8,6%)	7 (20%)	2 (5,7%)	1 (2,8%)	21 (60%)	35

Tableau 3-14: Répartition des états spécifiques à la classe "Crash" lors de l'injection de transitions erronées dans le registre EXE

Bench	Exe_tr_err Crash_1	Exe_tr_err Crash_2	Exe_tr_err Crash_3	Exe_tr_err Crash_4
Matrix	+	+		
Fir	+	+	+	
Sieve	+	+	+	+
Bubble		+	+	

Dans le cas d'injections dans le compteur de programme Reg_pc, la plupart des erreurs internes sont communes aux trois classes (Tableau 3-15). Les résultats ont été obtenus pour des injections de 2,21%, 1,15%, 1,08% et 0,11% des inversions de bits possibles respectivement pour "Matrix", "Fir", "Sieve" et "Bubble". Ils montrent que l'impact d'inversions de bits dans ce registre ne dépend pas réellement de l'application exécutée.

Tableau 3-15: Injection d'inversion de bits dans le registre Reg_pc

	Spécifique			Commun				Total
Bench	Correct	Incorrect	Crash	Correct/ Incorrect	Correct/ Crash	Incorrect/ Crash	Tout	
Matrix	0 (0%)	0 (0%)	1 (2,7%)	2 (5,4%)	0 (0%)	2 (5,4%)	32 (86,5%)	37
Fir	0 (0%)	1 (2,6%)	1 (2,6%)	1 (2,6%)	0 (0%)	2 (5,3%)	33 (86,8%)	38
Sieve	0 (0%)	0 (0%)	2 (5,3%)	2 (5,3%)	0 (0%)	0 (0%)	34 (89,5%)	38
Bubble	0 (0%)	0 (0%)	2 (5,3%)	2 (5,3%)	0 (0%)	0 (0%)	34 (89,5%)	38

Le dernier registre qui peut être considéré comme un registre de contrôle est Reg_op1, utilisé pour sauvegarder le premier mot du code d'instruction. Le Tableau 3-16 montre les résultats d'injection de 3,31%, 1,72%, 1,62% et 0,17% des inversions de bits possibles pour respectivement "Matrix", "Fir", "Sieve" et "Bubble". Comme illustré dans le Tableau 3-17, les configurations erronées qui conduisent finalement à des résultats corrects sont très dépendantes de l'application. Comme exemple, on a analysé l'état Reg_op1_Correct1 : il correspond à un état qui apparaît après corruption du code d'instruction qui a pour conséquence de modifier le

contenu de l'adresse 00H de la RAM et de l'accumulateur. Les valeurs corrompues ne sont pas utilisées par le calcul suivant, et par suite n'ont aucune conséquence sur les résultats. Pour la classe "incorrect", les états spécifiques observés pour "Matrix" et "Sieve" sont différents. D'un autre côté l'état spécifique à "Crash" pour "Sieve" est aussi spécifique à "Crash" pour "Fir".

Tableau 3-16: Injection d'inversion de bits dans le registre Reg_op1

	Spécifique			Commun				Total
Bench	Correct	Incorrect	Crash	Correct/ Incorrect	Correct/ Crash	Incorrect/ Crash	Tout	
Matrix	2 (4,8%)	1 (2,3%)	1 (2,3%)	5 (11,9%)	0 (0%)	1 (2,4%)	32 (76,1%)	42
Fir	1 (2,3%)	1 (2,3%)	3 (7%)	4 (9,3%)	0 (0%)	1 (2,3%)	33 (76,7%)	43
Sieve	2 (4,4%)	0 (0%)	2 (4,4%)	1 (2,2%)	1 (2,2%)	1 (2,2%)	38 (84,4%)	45
Bubble	3 (7,3%)	0 (0%)	1 (2,4%)	4 (9,7%)	0 (0%)	1 (2,4%)	32 (78%)	41

Tableau 3-17: Répartition des états spécifiques à la classe "Correct" lors de l'injection d'inversion de bits dans le registre Reg_op1

Bench	Reg_op1 Correct_1	Reg_op1 Correct_2	Reg_op1 Correct_3	Reg_op1 Correct_4	Reg_op1 Correct_5	Reg_op1 Correct_6	Reg_op1 Correct_7
Matrix	+	+					
Fir			+				
Sieve				+	+		
Bubble			+			+	+

Les injections ont été aussi réalisées dans le registre Reg_op2, avec le même pourcentage que pour Reg_op1. Le Tableau 3-18 résume les résultats. Le registre enregistre l'opérande principale des instructions et une grande différence est observée entre les différentes applications, bien que le nombre total d'états enregistrés est presque le même.

Tableau 3-18: Injection d'inversion de bits dans le registre Reg_op2

	Spécifique			Commun				Total
Bench	Correct	Incorrect	Crash	Correct/ Incorrect	Correct/ Crash	Incorrect/ Crash	Tout	
Matrix	0 (0%)	2 (5,9%)	2 (5,9%)	10 (29,4%)	0 (0%)	2 (5,9%)	18 (52,9%)	34
Fir	3 (8,8%)	2 (5,9%)	1 (2,9%)	27 (79,4%)	0 (0%)	0 (0%)	1 (2,9%)	34
Sieve	0 (0%)	1 (2,7%)	1 (2,7%)	7 (18,9%)	0 (0%)	1 (2,7%)	27 (73%)	37

3.4.1.2. Injection dans les registres de données et la mémoire

Les pourcentages de fautes injectées pour Reg_op3 et Reg_acc sont les mêmes que pour Reg_op1. RAM_UTIL représente les mémoires utilisées pour chaque application. Les injections dans cette mémoire correspondent à 0,17%, 0,03% et 0,03% des fautes possibles respectivement pour "Matrix", "Fir" et "Sieve". Pour les registres SFR, le pourcentage d'injections est respectivement de 0,19%, 0,1% et 0,1%. Le Tableau 3-19 montre une grande différence enregistrée entre les applications dans le cas d'injections dans ces registres de données. Par exemple, la situation de "Crash" observée pour "Sieve" est due à certaines instructions qui ne sont pas utilisées dans les autres programmes.

Tableau 3-19: Injection d'inversion de bits dans les registres de données et les points mémoires

Bench	Spécifique			Commun				Total
	Correct	Incorrect	Crash	Correct/ Incorrect	Correct/ Crash	Incorrect/ Crash	Tout	
Injection d'inversion de bits dans le registre Reg_op3								
Matrix	0 (0%)	27 (100%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	27
Fir	0 (0%)	17 (89,5%)	0 (0%)	2 (10,5%)	0 (0%)	0 (0%)	0 (0%)	19
Sieve	1 (3%)	0 (0%)	1 (3%)	30 (90,9%)	0 (0%)	0 (0%)	1 (3%)	33
Injection d'inversion de bits dans RAM_UTIL								
Matrix	0 (0%)	5 (16,1%)	0 (0%)	26 (83,9%)	0 (0%)	0 (0%)	0 (0%)	31
Fir	0 (0%)	2 (6,9%)	0 (0%)	27 (93,1%)	0 (0%)	0 (0%)	0 (0%)	29
Sieve	0 (0%)	0 (0%)	1 (1%)	14 (40%)	0 (0%)	0 (0%)	20 (57,1%)	35
Injection d'inversion de bits dans SFR								
Matrix	0 (0%)	12 (38,7%)	0 (0%)	19 (61,3%)	0 (0%)	0 (0%)	0 (0%)	31
Fir	0 (0%)	0 (0%)	0 (0%)	7 (100%)	0 (0%)	0 (0%)	0 (0%)	7
Sieve	0 (0%)	4 (11,1%)	1 (2,8%)	27 (75%)	0 (0%)	0 (0%)	4 (11,1%)	36
Injection d'inversion de bits dans le registre Reg_acc								
Matrix	0 (0%)	24 (80%)	0 (0%)	6 (20%)	0 (0%)	0 (0%)	0 (0%)	30
Fir	0 (0%)	1 (3,7%)	0 (0%)	26 (96,3%)	0 (0%)	0 (0%)	0 (0%)	27
Sieve	0 (0%)	0 (0%)	0 (0%)	33 (100%)	0 (0%)	0 (0%)	0 (0%)	33

3.4.2. Impact du sous-ensemble d'instructions utilisé par l'application

La remarque de la section précédente nous conduit à faire une analyse détaillée des instructions utilisées dans les quatre programmes d'application, aussi bien en terme de listes d'instructions utilisées (analyse statique) qu'en terme de pourcentage d'instructions exécutées pour chaque type (analyse dynamique). Un résumé des statistiques est présenté dans le Tableau 3-20.

Tableau 3-20: Analyse des instructions utilisées dans les quatre programmes (partie dans laquelle les fautes sont injectées)

Benchmark	Matrix	Fir	Sieve	Bubble
Nombre de types d'instructions	19	21	25	18
Analyse statique				
Instructions Arithmétiques/logiques	37%	26%	29%	32%
Instructions de transfert	53%	68%	56%	61%
Instructions de contrôle	10%	6%	15%	7%
Nombre total d'instructions	62	65	93	66
Analyse dynamique				
Instructions Arithmétiques/logiques	37%	27%	31%	35%
Instructions de transfert	55%	68%	56%	58%
Instructions de contrôle	8%	5%	13%	7%
Nombre total d'instructions	444	2562	2726	8686

Certaines différences n'apparaissent pas dans un résumé aussi bref (par exemple l'emploi de routines uniquement dans "Sieve" ou bien les différences en terme de modes d'adressage). Il apparaît cependant que la liste des instructions utilisées dans les différents programmes (en tenant compte des différents modes d'adressage) ne peut pas être directement corrélée avec les différences ou similitudes observées dans la liste des configurations d'erreurs critiques ou avec leur répartition durant la classification. Ceci est vrai aussi bien pour la répartition statique que pour la répartition dynamique des instructions. La combinaison des instructions pour un programme donné est en fait plus significative que la liste des instructions exécutées en ce qui concerne l'impact des fautes transitoires.

3.4.3. Impact sur la durée de la campagne d'injection de fautes

Comme mentionné précédemment, l'approche proposée en couplant les deux types d'analyse est indépendante des techniques utilisées pour réaliser les expériences (émulation ou simulation, avec ou sans optimisations comme la diminution de fautes ou l'accélération du simulateur). Le point le plus important est par conséquent le temps supplémentaire nécessaire pour combiner les deux parties des analyses au lieu d'en faire une seule. Si le but de la campagne est d'identifier les parties du circuit qui doivent être durcies, la classification à elle seule n'est pas suffisante. On suppose par conséquent que l'analyse de la propagation d'erreurs est requise, et est réalisée soit toute seule, soit en combinaison avec la classification. Le temps supplémentaire nécessaire durant la campagne pour réaliser les deux analyses a été évalué autour de 15% dans le cas de notre environnement actuel, dans lequel les deux types d'analyse doivent être réalisés séquentiellement, et puis combinées. Quelques modifications de nos programmes pourraient facilement permettre d'exécuter l'analyse combinée en une seule étape, réduisant sensiblement ce surcoût qui pourrait devenir de quelques pourcents de la durée de la campagne.

3.5. Pourcentage d'injections requis

Les résultats présentés dans les sections précédentes sont souvent basés sur l'injection d'un pourcentage très réduit de fautes, par rapport aux fautes possibles qui peuvent être injectées dans une cible donnée durant l'exécution d'une application donnée. Ce pourcentage réduit, qui correspond déjà à un grand nombre d'expériences et par suite à un long temps d'expérimentation, peut justifier quelques questions concernant la qualité des analyses réalisées. On va montrer dans cette section que ces faibles pourcentages peuvent être suffisants pour la première campagne d'injection, réalisée le plus tôt possible durant le processus de conception de façon à identifier les parties critiques à protéger dans le circuit. Une analyse le plus tôt possible et une première phase de durcissement peuvent réduire le coût des itérations dans le processus de conception. On doit cependant voir cette étape comme une première phase de caractérisation, qui doit être raffinée sur des descriptions de niveau inférieur, une fois la conception achevée et le circuit prêt pour la fabrication. Le but de l'analyse au niveau RTL ne peut pas par conséquent être une analyse exhaustive ; dans tous les cas, certaines informations concernant le circuit final

ne sont pas encore disponibles. Le but devrait plutôt être d'obtenir un bon compromis entre le temps expérimental et la qualité de l'analyse, de manière à minimiser globalement le *time-to-market*.

La Figure 3-2 illustre pour deux campagnes différentes l'évolution au fur et à mesure des expériences, des configurations d'erreurs identifiées (c'est-à-dire les états et les transitions dans le graphe généré à la fin de l'analyse). Comme on peut le voir dans cette figure, les premières expériences d'injection identifient rapidement un grand pourcentage de configurations erronées qui peuvent apparaître durant la campagne d'injection ; quelques configurations erronées additionnelles sont ensuite identifiées par certaines expériences d'injection de fautes réalisées tard lors de la campagne. Ce type d'évolution est tout à fait semblable à celui observé pour la couverture de test pseudo aléatoire.

Dans le cas d'injections dans le registre CS (Figure 3-2 (a)), la plupart des configurations erronées sont obtenues avec un nombre réduit d'expériences. Les transitions possibles entre elles (c'est-à-dire les séquences d'erreurs correspondant aux propagations d'erreurs) requiert un plus grand nombre d'expériences pour atteindre le même pourcentage d'identification. La même remarque est valable lors des injections dans le registre Reg_op1 (Figure 3-2 (b)), mais l'identification de toutes les configurations erronées est moins rapide que dans le cas de CS.

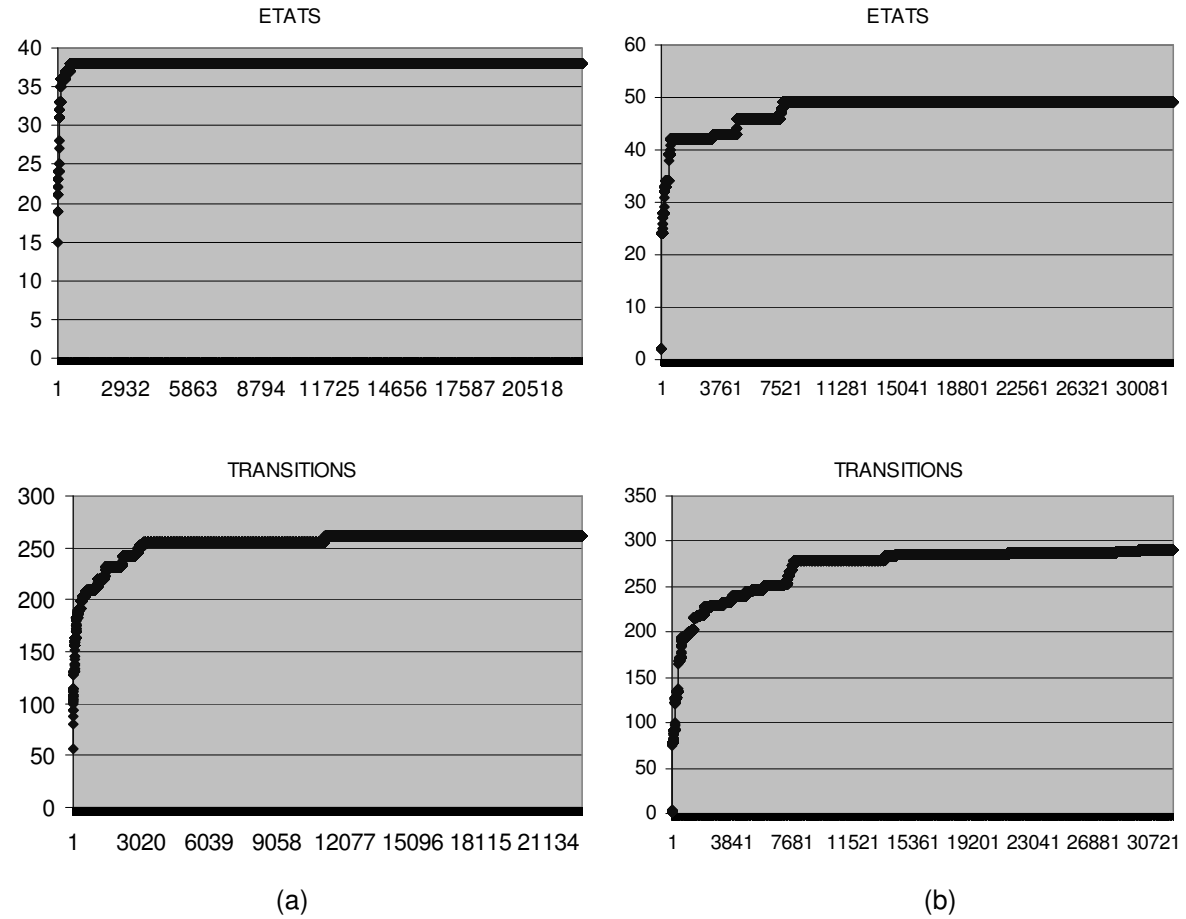


Figure 3-2: Evolution des configurations d'erreurs observées en fonction du nombre d'expériences, en exécutant le programme "Matrix" pour des injections exhaustives d'inversion de bits dans le registre CS (a) et Reg_op1 (b)

Le graphe de la Figure 3-2 a été obtenu avec des injections exhaustives, réalisées "dans l'ordre" (c'est-à-dire, injection dans chaque bit du registre au 1er cycle, ensuite injection dans chaque bit du registre au 2ème cycle, et ainsi de suite jusqu'au dernier cycle du programme exécuté). Quand les expériences d'injection sont définies aléatoirement, un tel ordre n'est généralement pas respecté. Nous avons donc fait les mêmes graphiques d'évolution pour plusieurs ordres aléatoires des expériences. La forme générale des graphes est similaire dans tous les cas ; la seule différence est le nombre d'expériences requis pour atteindre un pourcentage donné d'états ou de transitions identifiés. Ceci est illustré dans les Tableaux 3-21 et 3-22, dans le cas d'injections dans quatre registres différents. Dans la plupart des cas, on remarque que 85% (ou plus) des états d'erreurs possibles sont identifiés après injection de 1% à 2% des fautes possibles. Une autre remarque est qu'une campagne d'injection "ordonnée" est souvent plus efficace, spécialement dans le cas d'injections dans les registres de contrôle (par exemple CS ou EXE, en opposition avec le registre Reg_op3 qui est un peu plus orienté données).

Tableau 3-21: Pourcentage d'injection requis pour atteindre 85% des configurations erronées qui peuvent se produire lors de l'injection de fautes dans les différents registres pendant l'exécution du programme "Matrix"

Cible d'injection	Ordre	Désordre 1	Désordre 2	Désordre 3	Moyenne
CS (trans. err.)	0,34	1,77	2,16	1,97	1,56
EXE (trans. err.)	0,12	1,09	1,57	1,01	0,95
Reg_op1 (bit-flips)	0,96	1,29	0,71	4	1,74
Reg_op3 (bit-flips)	1,45	0,02	0,97	0,85	0,82

Tableau 3-22: Pourcentage d'injection requis pour atteindre 90% des configurations erronées qui peuvent se produire lors de l'injection de fautes dans les différents registres pendant l'exécution du programme "Matrix"

Cible d'injection	Ordre	Désordre 1	Désordre 2	Désordre 3	Moyenne
CS (trans. err.)	0,52	3,90	18,73	4,15	6,82
EXE (trans. err.)	0,55	2,90	5,58	1,65	2,67
Reg_op1 (bit-flips)	7,80	2,27	0,89	4,83	3,94
Reg_op3 (bit-flips)	1,45	0,02	0,97	0,85	0,82

Deux directives importantes peuvent être dérivées à partir de ces résultats. Premièrement, afin d'atteindre un bon compromis entre la précision et la longueur des expériences, l'injection en tout et pour tout de 1% des fautes peut être suffisante. Deuxièmement, une meilleure efficacité peut être attendue si les fautes dans les registres de contrôle sont injectées dans la première partie du programme exécuté, tandis que les fautes dans les registres de données devraient être injectées plus uniformément pendant la totalité de l'exécution. Ces directives devraient bien sûr être confirmées par d'autres études, mais les résultats obtenus avec ce cas d'étude appuient fortement l'intérêt de disposer de différents types de fonctions aléatoires dans un environnement d'injection de fautes.

Pour conclure cette section, on fournit le Tableau 3-23 qui compare les résultats obtenus pour des injections dans les parties utilisées de la RAM lors de l'exécution du programme

"Matrix". Dans la section 3.4.1.2, les résultats sont donnés après injection de seulement 0,17% des fautes possibles, ce qui représente 2000 fautes. Exploitant les avantages offerts par des stations de travail plus puissantes, la campagne a été de nouveau réalisée avec 20000 injections c'est-à-dire 1,74% des fautes possibles. En utilisant la même station de travail, la seconde campagne aurait été dix fois plus longue que la première. Le Tableau 3-23 montre que les résultats sont presque les mêmes, excepté pour la classification d'un état qui n'est plus spécifique à la classe "incorrect". En fait, les résultats précédents correspondent à un bon compromis entre la précision et la durée des expériences.

Tableau 3-23: Injection d'inversion de bits lors de l'exécution du programme "Matrix"

	Spécifique			Commun				Total
Pourcentage de fautes injectées	Correct	Incorrect	Crash	Correct/ Incorrect	Correct/ Crash	Incorrect/ Crash	Tout	
0,17	0 (0%)	5 (16.1%)	0 (0%)	26 (83.9%)	0 (0%)	0 (0%)	0 (0%)	31
1,74	0 (0%)	4 (12.9%)	0 (0%)	27 (87.1%)	0 (0%)	0 (0%)	0 (0%)	31

3.6. Evaluation par injection de fautes d'une technique de détection d'erreurs au niveau logiciel

3.6.1. Introduction

Plusieurs approches ont été proposées pour augmenter la robustesse d'un système. Beaucoup sont basées sur des modifications au niveau matériel, par exemple en travaillant au niveau micro-architecture. Mais des approches ont aussi été proposées pour des systèmes à base de processeurs standards. Habituellement les injections RTL sont utilisées pour évaluer le matériel, tandis que d'autres techniques d'injection (notamment à base de simulateur de jeu d'instructions ou ISS) sont utilisées pour évaluer la robustesse du logiciel. L'objectif de cette partie du document est de montrer l'intérêt des techniques d'injection de fautes au niveau RTL pour la validation des techniques de protection appliquées au niveau logiciel, sans intervention sur le matériel. On évaluera ici l'efficacité d'une technique de détection d'erreurs, qui a été à la base conçue pour détecter des fautes qui altèrent les registres accessibles à travers le jeu d'instructions du processeur. La technique de détection d'erreurs étudiée combine des redondances logicielles et une analyse de signature [B.N. 04]. Cette technique a été précédemment évaluée au moyen d'une campagne d'injection de fautes à base de simulateur ISS, comme c'est souvent le cas pour les approches de durcissement logicielles, et il a été prouvé qu'elle est très efficace contre les inversions de bits simples. Les évaluations décrites par la suite sont réalisées en injectant des fautes dans une version instrumentée du processeur 8051. Les cibles sont les mêmes que dans la section 3.2.3. Le but de cette étude est notamment d'évaluer la robustesse du système durci vis-à-vis des fautes dont on ne tient pas compte lors de la définition du mécanisme de protection.

3.6.2. Technique de détection d'erreur étudiée

Pour la technique considérée, les fautes corrompant les données sont détectées en comparant les résultats d'une même opération sur deux copies des variables. En plus, des instructions de neutralisation sont introduites entre l'opération d'origine et sa copie (Figure 3-3). Leur rôle est d'effacer le contenu résiduel des registres utilisés par l'instruction d'origine avant que sa copie ne commence son exécution.

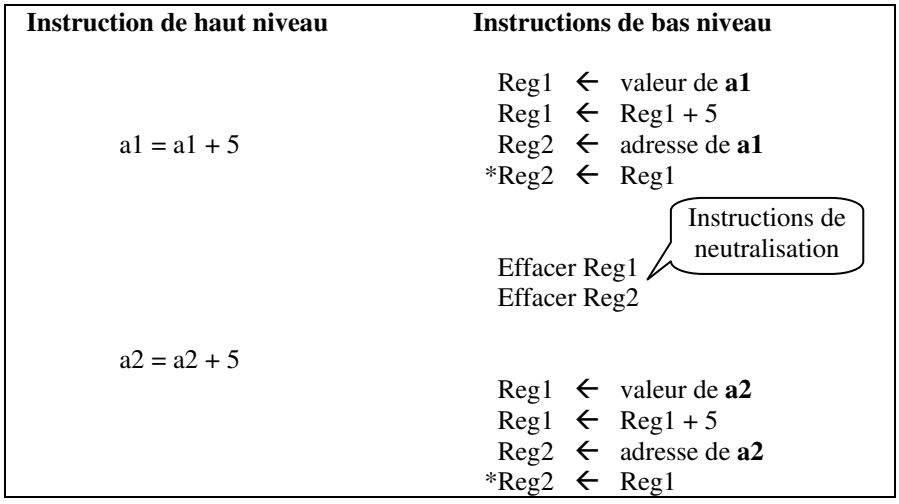


Figure 3-3: Redondance logicielle avec instructions de neutralisation

Pour les erreurs altérant le flot d'exécution, la technique de détection étudiée implémente une technique dynamique originale d'analyse de signature. A la différence d'approches classiques, la technique de contrôle dynamique associe une signature à chaque transition inter-bloc. La valeur de la signature à un instant donné ("signature en ligne") dépend de la dernière transition inter-bloc exécutée, et de quelques signatures locales associées au bloc en cours d'exécution. Le rôle des signatures cumulatives locales est de s'assurer que le bloc est entièrement exécuté, y compris la vérification de signature. A la fin de chaque bloc, les signatures cumulatives locales sont rajoutées à la signature en ligne. La Figure 3-4 montre les emplacements où les signatures cumulatives locales sont combinées avec la signature en ligne.

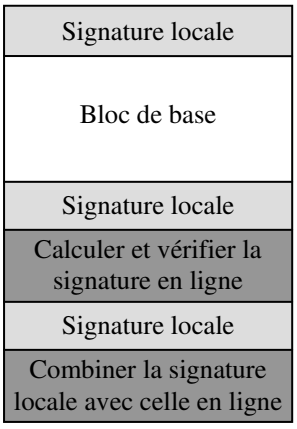


Figure 3-4: Un bloc avec signature cumulative locale de vérification d'instructions

Une autre différence majeure avec les techniques traditionnelles consiste dans la surveillance du flot de contrôle par prédiction de branchement. Chaque bloc connaît d'avance l'identité du bloc qui sera exécuté. Quand le contrôle est transféré à un bloc de destination, la signature en ligne est calculée et comparée avec la signature de référence correspondant à la dernière transition inter-bloc. Pour prédire quel bloc de base sera exécuté, trois types de transitions inter-bloc sont identifiés : certain, conditionnel et dépendant de l'état. Des détails sur l'algorithme et comment le contrôle du flot est vérifié peuvent être trouvés dans [B.N. 04].

Il est important de noter qu'à cause de limitations matérielles (RAM du 8051 insuffisante), certains mécanismes de détection de la technique d'origine ne sont pas implémentés. Ces mécanismes sont :

- La séparation physique au niveau de la mémoire de l'instruction d'origine et de sa copie à une distance telle qu'aucune inversion de bit ne peut résulter en un saut de l'un vers l'autre.
- L'insertion de blocs dédiés au début et à la fin du programme.

3.6.3. Programmes exécutés

Les analyses ont été réalisées en exécutant trois versions du programme "Matrix" présenté à la section 3.2.2. La première version est la même que lors de la section 3.2.2 et donc n'intègre aucun mécanisme de protection, les deux autres utilisent les techniques de détection d'erreurs présentées dans la section 3.6.2 avec des implantations de protection différentes. La taille des programmes (nombre d'octets dans la ROM) est de 265, 2804 et 2690 respectivement pour la version non durcie, la première version durcie et la deuxième version durcie. Le nombre de cycles nécessaire pour le calcul des résultats est respectivement de 13.741, 86.860 et 100.578 respectivement. A la fin de l'exécution, les résultats se situent dans la RAM.

La différence entre les deux versions durcies se situe principalement dans la définition du DCB (*Data Computing Blocks*). Un DCB peut être associé à un ACU (*atomic computation unit*) ou à un MACU (multiple ACUs). Dans le premier cas, la redondance est insérée pour chaque instruction, l'instruction copiée est exécutée juste après l'originale ; ceci est le choix adopté pour la première version durcie. Dans le second cas, la redondance est insérée pour un groupement d'instructions dans le programme et des points de synchronisation sont utilisés pour transférer le contrôle du programme entre les MACUs originaux et ceux copiés. Ceci est illustré à la Figure 3-5, où les flèches montrent le flot d'exécution.

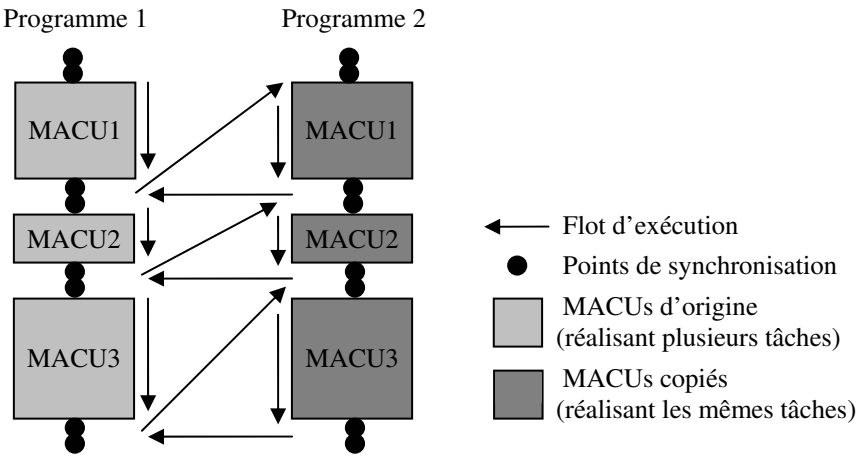


Figure 3-5: Redondance à base de MACU

3.6.4. Fautes injectées

La classification des fautes est réalisée en fonction de la justesse des résultats au cycle correspondant à la fin de l'exécution du programme quand il n'y a pas de fautes injectées. Trois classes de résultats sont considérées pour le programme de référence : résultats corrects, incorrects, ou "crash". Pour les versions durcies, deux classes supplémentaires sont rajoutées (Détection1 et Détection2), qui correspondent à deux techniques différentes de détection (détection directement durant l'exécution du programme, ou détection supposée par le système à cause de l'incohérence des résultats avec leurs copies). La classe "Détection" est la somme de Détection1 et Détection2.

2000 fautes ont été injectées pour chaque cible avec la version initiale du programme. Le nombre d'injections passe à 5000 pour chaque cible avec les programmes durcis (10000 injections dans RAM_UTIL et SFR) pour tenir compte de l'augmentation du nombre de cycles d'exécution.

3.6.5. Résultats d'injection

Le Tableau 3-24 montre les résultats obtenus durant les expérimentations d'injection de fautes, en exécutant le programme initial et sa première version durcie. En comparant le pourcentage des résultats incorrects pour chaque version, on observe que dans le cas de la version durcie celui-ci diminue de manière significative mais reste non négligeable pour certaines cibles d'injection (les cibles d'injection sont définies à la section 3.2.3). De même, certains types de fautes n'ont pas été détectés pendant les simulations alors qu'ils ont été injectés dans des éléments qui a priori ont été protégés par l'approche étudiée. En fait les conséquences d'une faute dépendent de la nature de l'information perturbée et la raison de non détection n'est pas toujours la même. Les situations où les détections ont failli ont été analysées en détail.

Certaines de ces situations sont dues à la description VHDL du processeur. Par exemple, dans le cas de la cible Reg_op1, certaines fautes transforment l'instruction courante en une autre

qui n'est pas implémentée dans le modèle du processeur. Cette classe d'erreur est la principale cause des non détections, et c'est ce qui explique le pourcentage élevé de résultats incorrects dans le cas des injections d'inversions de bits ou de transitions erronées dans le registre CS et EXE (lignes CS_bitflip, CS_tr_err et Exe_tr_err du Tableau 3-24). Dans un processeur réel, les conséquences exactes de ces fautes dépendraient des optimisations choisies durant la synthèse de la description VHDL.

Tableau 3-24: Résultats de classification pour le programme initial et la première version durcie (% des expériences d'injection par cible)

Cible	Pas de mécanisme de protection			Avec mécanismes de protection (version 1)					
	Correct	Incorrect	Crash	Correct	Incorrect	Crash	Détection1	Détection2	Détection
Cs_bitflip	57,78	24,95	17,27	51,74	14,36	0	32,58	1,32	33,9
Cs_tr_err	60,09	28,27	11,63	49,28	16,12	0	32,84	1,76	34,6
Exe_tr_err	65,91	27,47	6,61	71,6	9,54	0,12	17,94	0,8	18,74
Reg_pc	43,65	26,05	30,3	52,9	3,32	6,58	36,52	0,68	37,2
Reg_op1	72,5	24,55	2,95	77,72	3,72	0,86	17,38	0,32	17,7
Reg_op2	95,15	4,7	0,15	95,12	0,02	0	4,76	0,1	4,86
Reg_op3	99,95	0,05	0	99,58	0,02	0	0,4	0	0,4
Sfr	97,35	2,65	0	90,24	0,34	0	9,08	0,34	9,42
Ram_util	36,15	63,85	0	49,02	0	0,01	44,35	6,62	50,97
Reg_acc	89,65	10,35	0	89,24	0	0	10,76	0	10,76
Moyenne	71,82	21,29	6,89	72,64	4,74	0,76	20,67	1,19	21,86

D'autres raisons pour des non détections sont présentées par la suite. Certaines fautes conduisent à un redémarrage du programme, c'est-à-dire relancent le programme de nouveau depuis le début (adresse 000H dans la ROM), mais il n'y a alors pas assez de temps pour calculer tous les résultats avant le temps de fin normal (en l'absence de fautes). Dans le cas de contraintes temps réel, ceci conduirait à une défaillance. Sans contraintes temporelles pour le calcul, les résultats corrects seraient obtenus, mais plus tard. Certaines fautes dans le SFR illustrent cette classe d'erreurs. Des fautes modifient le pointeur de pile, ce qui conduit le processeur à lire à une adresse erronée lors de l'exécution des instructions ACALL, LCALL ou RET. Quand la donnée lue est 00H, le programme pointe vers son instruction de départ.

Dans le cas des injections dans Reg_op1, on trouve aussi ce type d'erreurs mais la cause est différente. On observe par exemple qu'une instruction MOV_8 corrompue par la faute injectée peut être transformée en une instruction LCALL laquelle lit alors une adresse contenant 00H. la probabilité de tels évènements peut seulement être évitée, ou au moins réduit, si il est possible de choisir astucieusement les codes binaires des instructions, par exemple en concevant un processeur spécifique à une application.

Un autre type de situation peut être trouvé, par exemple, quand des fautes se produisent dans Reg_pc. Une instruction corrompue peut faire un saut dans une zone mémoire stockant les instructions d'initialisation du programme. Ces instructions sont générées par le compilateur (utilisé avec des options de base dans nos expérimentations) et donc n'intègrent pas de mécanismes de détection. En conséquence, le processeur ne calcule pas les résultats corrects mais aucune détection n'est activée. En conclusion pour cette situation, il est nécessaire de faire

attention aux parties du programme rajoutées par le compilateur quand des mécanismes de détection d'erreurs sont rajoutés dans un programme décrit dans un langage de haut niveau (C dans notre cas).

Dans d'autres cas, l'injection résulte en un saut dans une zone du programme (en dehors du code d'initialisation) mais aucune détection n'est activée alors que les résultats sont incorrects dans les adresses d'origine et leurs copies. Ceci se produit spécialement quand les fautes sont injectées dans Reg_pc, mais aussi dans Reg_op1. A partir de ces remarques, on peut dire qu'apparemment la version durcie du programme ne réagit pas assez bien pour des sauts erronés.

Le pourcentage de "crash" a diminué pour la version durcie du programme, sauf lors des injections dans RAM_UTIL. C'est en raison de la plus grande taille du programme qui réduit la probabilité d'activer des conditions menant au "crash" durant la campagne d'injection.

Les résultats pour la seconde version durcie sont résumés dans le Tableau 3-25.

Tableau 3-25: Résultats de classification pour la seconde version durcie (% des experiences d'injection par cible)

Cible	Avec mécanismes de protection (version 2)					
	Correct	Incorrect	Crash	Détection1	Détection2	Détection
Cs_bitflip	46,18	12,96	1,44	36,6	2,82	39,42
Cs_tr_err	47,22	14,4	1,64	33,4	3,34	36,74
Exe_tr_err	68	7,68	0,36	21,98	1,98	23,96
Reg_pc	52,52	1,64	8,26	36,08	1,5	37,58
Reg_op1	72,7	2,92	0,82	22,4	1,16	23,56
Reg_op2	93,18	0	0,26	6,4	0,16	6,56
Reg_op3	98,68	0,02	0,16	1,08	0,06	1,14
Sfr	88,96	1,56	0,3	8,74	0,44	9,18
Ram_util	52,07	0,23	0,56	41,43	5,71	47,14
Reg_acc	90,12	0,02	0,08	9,74	0,04	9,78
Moyenne	70,96	4,14	1,39	21,79	1,72	23,51

Les campagnes d'injection de fautes précédentes [B.N. 04] montraient que l'approche basée sur les MACUs est plus robuste que celle utilisant les ACUs. Ces expériences confirment ceci pour certaines cibles d'injections. Mais les résultats obtenus par exemple avec les fautes injectées dans SFR ou RAM_UTIL montrent le contraire. Ceci peut être expliqué par deux causes différentes. Premièrement, comme mentionné dans la section 3.6.2, deux mécanismes du schéma de protection n'ont pas été implémentés à cause d'un manque de mémoire RAM. En particulier, les différents MACUs ne sont pas physiquement séparés dans la mémoire. La mémoire disponible a donc un impact direct sur la probabilité de détection. La deuxième cause est liée à l'implantation des points de synchronisation qui dépend étroitement du jeu d'instructions du processeur. Dans les expériences précédentes, utilisant un processeur SPARC (Leon) ou un DSP32, la synchronisation était facile à implémenter puisque le compteur de programme pouvait être changé en utilisant seulement deux instructions (Figure 3-6 (a) et (b)). Dans le cas du 8051, la procédure de synchronisation est plus complexe, comme montré à la Figure 3-6 (c), et utilise un troisième registre (DPTR), qui est un registre dédié utilisé dans des

opérations de lecture/écriture depuis la mémoire. D'ailleurs, des variables additionnelles sont utilisées pour garder l'information sur la valeur actuelle du compteur de programme. En analysant ces résultats, il apparaît que la façon dont sont implémentés les points de synchronisation est réellement un point faible. Pour certaines valeurs corrompues de ces variables, le programme entre dans une boucle infinie. Ceci a été observé lors des injections dans RAM_UTIL, mais aussi dans SFR ou Reg_acc. L'efficacité du schéma de protection utilisant des MACUs dépend donc du processeur cible et de son jeu d'instructions.

a) LEON:
```
asm ("jmpl %14,%13");
asm ("add %13,8,%14");
```

b) DSP32C:
```
rl = PC
PC=0x2000
```

c) 8051:
```
getPC();
DPH = PCH;
DPL = PCL;
PCH = PCH_current;
PCL = PCL_current;
PCH_current = DPH;
PCL_current = DPL;
#pragma asm
clr A
jmp @A+DPTR
#pragma endasm
```

Figure 3-6: Exemples d'implantation de la synchronisation MACU pour différents processeurs

3.7. Conclusion

La principale conclusion des résultats présentés dans ce chapitre est que l'analyse de sûreté réalisée au niveau RT donne des résultats fiables malgré le manque de détails d'implantation. En particulier, les arrêts de la simulation liés au "crash" n'empêchent pas d'avoir des résultats intéressants, qui peuvent être utilisés très tôt dans le flot de conception pour guider un durcissement matériel (ou logiciel) du circuit. De plus, les résultats montrent que dans la plupart des cas, la combinaison de la classification de fautes et de l'analyse de propagation d'erreurs permet au concepteur d'identifier des configurations erronées de signaux internes qui sont spécifiques à une classe donnée. Cette information peut par la suite être utilisée pour durcir uniquement les parties critiques dans le circuit, menant à un coût maîtrisé des modifications. Mais les états spécifiques dépendent sensiblement du programme exécuté. Il vaut donc mieux tenir compte de l'application réelle au moment du durcissement d'un circuit à but général comme un microprocesseur, pour atteindre le meilleur compromis entre le niveau de robustesse de l'application et les surcoûts d'implantation.

Une autre contribution de cette étude est la discussion de la qualité de l'analyse en fonction du pourcentage de fautes injectées durant les expériences. Quelques directives sont proposées pour améliorer l'efficacité de la campagne, en limitant la durée des expériences.

Ces injections de fautes ont été par la suite utilisées pour valider une technique de protection au niveau logiciel, bien qu'elles soient habituellement utilisées pour valider des mécanismes de protection implémentés au niveau matériel. Ces injections sont plus générales que celles basées sur les ISS, et c'est ce qui nous a permis de trouver des situations critiques que

les mécanismes implémentés n'ont pas su détecter, et qui n'auraient pas pu être identifiés avec d'autres types d'injections de fautes. Les erreurs non détectées ont été analysées et quelques pistes d'amélioration de la technique de protection ont été suggérées.

Chapitre 4.

Extension du flot d'analyse aux blocs analogiques

4.1. Introduction

De nos jours, les circuits analogiques et mixtes (AMS pour *Analog Mixed Signal*) sont de plus en plus requis dans les applications liées à l'automobile, les systèmes de contrôle à temps réel, les communications… Les méthodes automatisées de conception et de test analogique sont actuellement loin derrière leurs homologues numériques. Ces blocs analogiques sont donc souvent le goulot d'étranglement en concevant un circuit AMS. Parallèlement, l'analyse de fiabilité d'un circuit est seulement réalisée en injectant des fautes dans les parties numériques. Cette analyse est insuffisante dans certains cas, vu que les parties analogiques sont elles-mêmes sujettes aux fautes transitoires [T.T. 96]. Pour pallier cette insuffisance, une nouvelle approche a été développée. Elle permet à un concepteur d'évaluer globalement, tôt dans le flot de conception, la réponse d'un circuit AMS quand des fautes de type SET se produisent. Cette approche a été testée sur un cas d'étude et les résultats obtenus sont montrés ainsi que la comparaison avec des résultats obtenus au niveau transistor.

4.2. Injections de fautes dans les circuits analogiques

Avant de proposer l'approche développée, on se propose de présenter brièvement les techniques d'injections de fautes dans les circuits analogiques. Ces injections sont de trois types : au niveau transistor, au niveau comportemental et sous radiations.

4.2.1. Injections au niveau transistor

Les articles [M.S. 01, M.S. 03] présentent des injections de fautes transitoires réalisées au niveau transistor. Ces fautes sont modélisées en injectant des courants dont l'équation est une double exponentielle. Ces expérimentations font appel à des simulateurs électriques (SPICE ou SPECTRE). De telles injections peuvent être considérées comme équivalentes aux injections de fautes au niveau portes dans les parties numériques, et ne peuvent être réalisées que très tard dans le flot de conception (juste avant le dessin des masques). De plus, l'impact global du comportement erroné du circuit sur le système complet est difficilement évaluable.

4.2.2. Injections au niveau comportemental

Les articles [C.P. 97, P.W. 02] proposent des injections de fautes dans des descriptions comportementales de blocs analogiques. De telles descriptions de haut niveau peuvent être faites dans des langages comme VHDL-AMS [Y.H] ou Verilog-A et peuvent permettre des analyses au niveau système. Par contre, l'injection de fautes au niveau comportemental est réalisée en modifiant les valeurs numériques des paramètres décrivant le comportement, c'est-à-dire en injectant des fautes paramétriques. De telles fautes peuvent être représentatives aussi bien des variations dans le processus que du vieillissement du circuit, mais peuvent difficilement modéliser l'effet des fautes transitoires dues à l'impact de particules chargées.

4.2.3. Injections sous radiations

De nombreux travaux [R.E. 94, K.L. 97, S.B. 00] présentent des expérimentations d'injection de fautes transitoires réalisées sous radiations. Le principe reste rigoureusement le même que pour les circuits numériques. La seule chose qui change est l'analyse des résultats qui est plus complexe vu la nature du signal analogique. Ce type d'injection n'est toutefois réalisable qu'après fabrication du circuit et sort donc du cadre de cette thèse.

4.3. Extension du flot d'analyse des circuits numériques

Le but ici est de permettre des injections de fautes transitoires dans les blocs analogiques ou mixtes, tout en effectuant le moins de modifications possibles par rapport au flot d'analyse des circuits numériques présenté à la section 1.7.2. Pour commencer, on suppose qu'une description haut niveau du circuit en entier est disponible. Dans notre cas, VHDL-AMS [Y.H] est choisi comme langage de description et les blocs analogiques sont décrits en combinant des descriptions structurelles et comportementales : l'architecture interne du bloc est spécifiée par une description hiérarchique structurelle, et chaque sous bloc est spécifié au niveau comportemental.

La première modification du flot précédent est bien sûr celle concernant le remplacement du simulateur VHDL (ou Verilog) par un simulateur mixte. L'émulation n'est pas prise en considération dans cette étude. Mais cette alternative peut devenir intéressante dans un futur proche, grâce aux nouveaux réseaux programmables permettant d'implémenter des blocs analogiques, comme ceux présentés récemment par Lattice Semiconductor.

Pour garantir la compatibilité avec le flot numérique, les injections de fautes dans les blocs analogiques peuvent utiliser des saboteurs ou des mutants. Le haut niveau de description des sous blocs analogiques est basé sur des équations qui ne sont pas facilement modifiables pour tenir compte d'une façon précise des fautes transitoires. Ceci nous oblige à utiliser exclusivement les saboteurs pour injecter des fautes de type SET, contrairement aux blocs numériques où les mutants sont mieux adaptés. Utiliser les saboteurs offre une grande flexibilité pour la modélisation de pics de courants. Par contre cette injection est limitée aux interconnexions entre les blocs ; le nombre de nœuds d'injection dépend donc directement de la décomposition architecturale du bloc analogique. Bien sûr les injections de fautes paramétriques peuvent être réalisées, quand cela est significatif, dans les sous blocs décrits au niveau comportemental.

Etant donné que les saboteurs sont disponibles dans une bibliothèque, l'instrumentation d'un bloc analogique est très facile. Par contre, le concepteur doit spécifier :

- Les paramètres du pic de courant (section 4.4),
- Les temps d'injection.

La spécification de ces temps d'injection est plus difficile pour les blocs analogiques que pour leurs homologues numériques, car l'instant exact d'injection (et non pas le cycle d'injection pour l'horloge du système) peut avoir un impact significatif sur l'effet de la faute, même quand une simulation comportementale de la partie numérique du circuit est réalisée.

L'analyse des résultats peut utiliser le module disponible dans le flot numérique si uniquement des nœuds numériques sont observés durant les expérimentations. Au cas où des nœuds analogiques seraient également observés, il serait nécessaire de définir une tolérance additionnelle sur les valeurs, de telle sorte à éviter l'identification d'erreurs non significatives.

Le flot qui en résulte est donné en Figure 4-1.

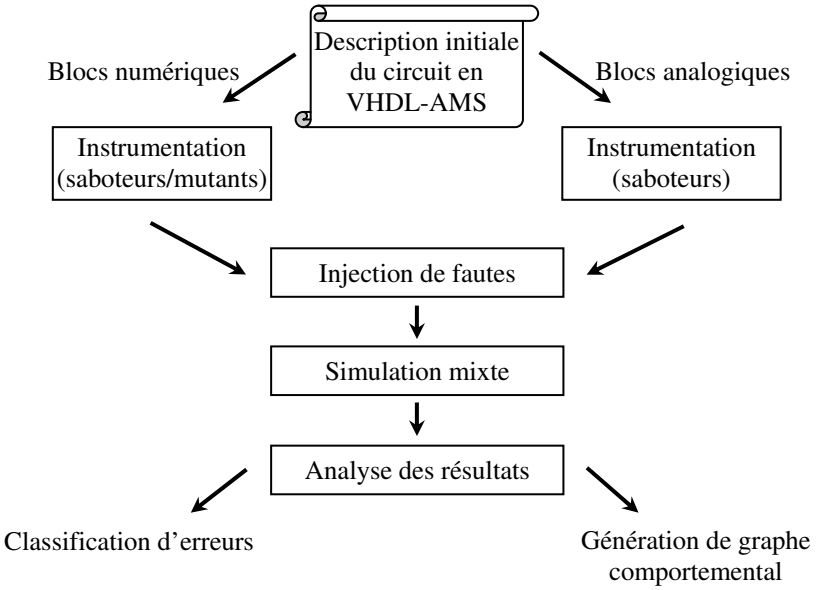


Figure 4-1: Principales étapes dans le flot d'analyse proposé

4.4. Modèles des fautes injectées

Le but de cette section est de donner un modèle des fautes se produisant dans les circuits analogiques c'est-à-dire modéliser un SET par des équations.

Au niveau électrique, un SET correspond à un pic de courant provoqué par ionisation de la matière par exemple après l'impact d'une particule. Ce pic de courant ne peut pas être modélisé à haut niveau par des inversions de bits, comme en numérique. Par contre, il est nécessaire, d'un point de vue pratique, de limiter la complexité du saboteur de telle sorte à simplifier les simulations et réduire la durée des campagnes d'injection. Un modèle classique à double exponentielle est largement employé dans le cas des fautes induites par les particules alpha [G.M. 82]. Un autre modèle basé sur l'approximation du précédent a été aussi développé et les deux vont être présentés aux sections suivantes.

4.4.1. Modèle à double exponentielle

Le courant d'injection dû aux particules alpha heurtant un nœud du circuit, dénoté Iinj, est donné par l'Equation 4-1 où τ_1 est la constante de temps de collection de la jonction et τ_2 est le temps d'établissement de la trace initiale créée par la particule. Ces constantes de temps dépendent de la technologie. Pour la technologie ST 0,18µm, $\tau_1 = 2E^{-10}$s et $\tau_2 = 5E^{-11}$s [S.S. 05]. I_0 est le courant maximum et il peut être calculé avec l'Equation 4-2 où Qinj est la charge injectée en Coulomb et dépend de l'angle d'incidence de la particule alpha. I_0 peut être positif ou négatif suivant que la particule heurte le drain d'un NMOS ou d'un PMOS [H.C. 96].

Equation 4-1: $$Iinj = I_0\left(e^{-t/\tau 1} - e^{-t/\tau 2}\right)$$

Equation 4-2: $$I_0 = \frac{Qinj}{\tau_1 - \tau_2}$$

La Figure 4-2 donne la forme du courant pour différentes valeurs de Qinj.

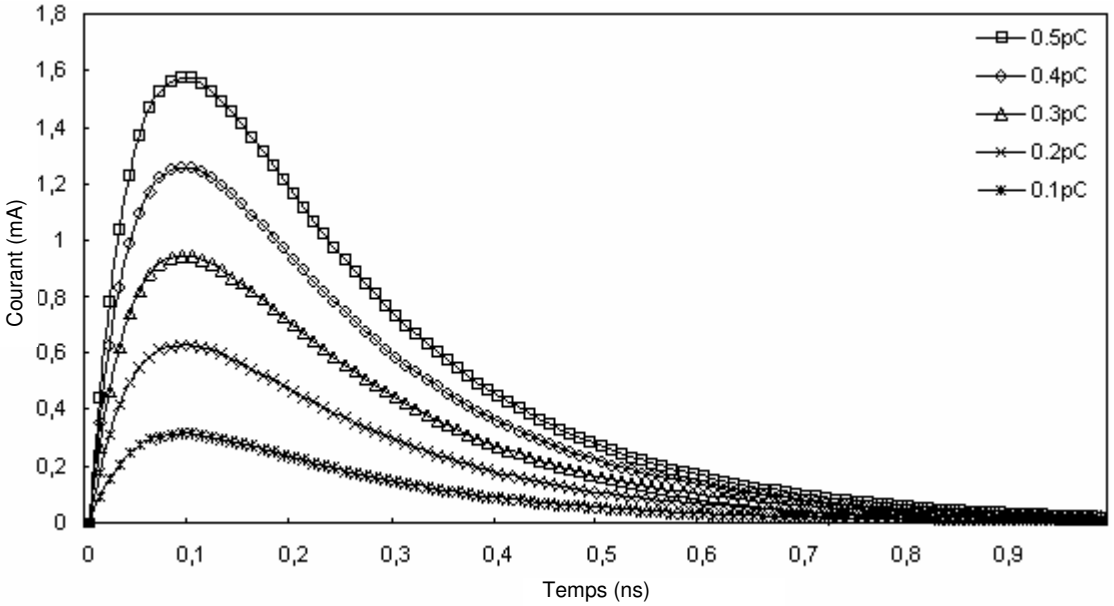


Figure 4-2: Courant d'injection pour différentes valeurs de Qinj

La programmation en VHDL-AMS d'un saboteur conforme à ce modèle a posé problème. Vu la durée très faible du phénomène, deux choix s'offraient à nous pour pouvoir le modéliser correctement :

- Augmenter la précision de la simulation et/ou diminuer le pas de simulation de telle sorte que ce phénomène soit bien représenté. Cette solution n'est pas du tout pratique car elle prend un temps excessivement long, d'autant plus que le surplus de précision ne sert qu'à la représentation du pic de courant.
- Piloter un signal déclaré comme réel de telle sorte à obliger le simulateur à calculer les points à chaque changement de ce signal. C'est cette solution qui a été retenue et la Figure 4-3 donne le code utilisé.

```
library ieee,disciplines;
use disciplines.electromagnetic_system.all;
use ieee.math_real.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity inj_current_exp is
        generic (Ta : real;
                Tb : real;
                Qinj : real);
        port (  terminal p1 : electrical;
                signal inj : in std_logic);
end entity inj_current_exp;

architecture bhv of inj_current_exp is
signal intI, temps : real := 0.0;
signal inj_int : std_logic;
quantity set_courant through electrical_ground to p1;

begin
        break on inj, intI, temps, inj_int;

        process(inj)
        begin
                if inj'event and inj = '1' then
                        intI <= now;
                end if;
        end process;

        process (temps, intI, inj_int)
        begin
                if (intI = now) then
                        temps <= now + 1.0e-12 after 1.0 ps;
                elsif inj_int = '1' then
                        temps <= temps + 1.0e-12 after 1.0 ps;
                end if;
        end process;

        process (inj)
        begin
                if inj = '1' then
                        inj_int <= '1';
                else
                        inj_int <= '0';
                end if;
        end process;

        if intI = 0.0 or inj_int = '0' use
                set_courant == 0.0;
        else
                set_courant == (Qinj/(Ta - Tb)) * (exp(-(temps-intI)/Ta) - exp(-(temps-intI)/Tb));
        end use;
end architecture bhv;
```

Figure 4-3: Modèle de saboteur à double exponentielle

Pour pouvoir utiliser ce modèle, il suffit de spécifier les valeurs de Ta(τ_1), Tb(τ_2), Qinj et la durée pendant laquelle le signal inj reste à '1'. Cette durée doit être au minimum de 1ns. En effet d'après la Figure 4-2, on peut considérer qu'après 1ns, le courant d'injection est nul.

Les pics de courant sont injectés aux nœuds spécifiés comme étant des "quantités" en utilisant une sommation de courant au niveau du nœud.

4.4.2. Modèle à rampes

Afin de simplifier le saboteur, un modèle à rampes a été développé pour approximer la double exponentielle. La Figure 4-4 représente le modèle et montre ses paramètres : temps d'injection, amplitude du pic (PA), temps de montée (RT), temps de descente (FT) et durée du pic (PW).

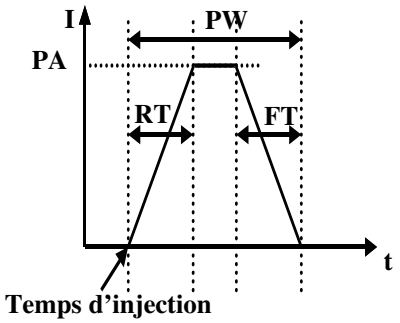


Figure 4-4: Modèle à rampes

Les valeurs de ces paramètres peuvent être calculés à partir du modèle classique à double exponentielle (Figure 4-5). On peut aussi les faire varier pour étudier la sensibilité du circuit en fonction de ces paramètres.

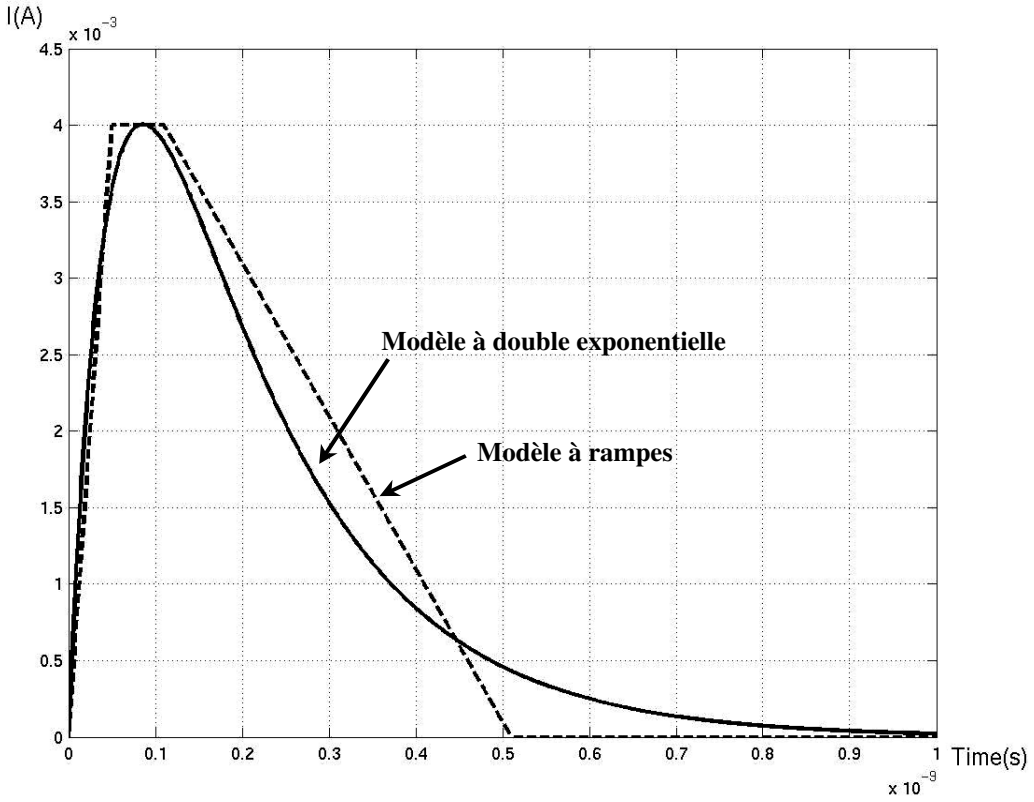


Figure 4-5: Approximation du modèle à double exponentielle par un modèle à rampes

La Figure 4-6 donne le code utilisé pour générer ce saboteur.

```
library Disciplines, IEEE;
use Disciplines.ELECTROMAGNETIC_SYSTEM.all;
use IEEE.math_real.all;

entity GenCur is
        generic (RT: real;
                FT: real;
                PA: real );
        port ( terminal out_cur : electrical;
                signal inj: in bit);
end entity GenCur;

architecture GenArch of GenCur is
quantity out_current through out_cur;
signal intI: real:=0.0;

begin
        p: process(inj)
        begin
                intI <= 0.0;
                if inj='1' then
                        intI <= PA;
                end if;
        end process;
        out_current == intI'ramp(RT,FT);
end GenArch;
```

Figure 4-6: Modèle du saboteur à rampes

4.5. Etude de cas : PLL à pompe de charge décrite en langage de haut niveau

Avant de présenter des résultats d'injection, je vais commencer par décrire la PLL ayant été choisie comme étude de cas. Les blocs la constituant sont décrits à l'Annexe A. Le but ici n'est pas de les décrire en profondeur, mais de donner quelques notions nécessaires à la compréhension de ce chapitre pour ceux qui ne sont pas familiarisés avec les PLL. Pour de plus amples renseignements sur la PLL, je renvoie le lecteur vers le lien [D.B].

Les blocs modélisés en VHDL-AMS sont pris dans la bibliothèque COMM_LIB de ADVANCE_MS [A.M]. Pour les utiliser, il suffit de les connecter judicieusement de telle sorte à former la PLL et de spécifier certains paramètres nécessaires à l'établissement de chaque modèle. Ces paramètres, même s'ils sont définis à haut niveau d'abstraction, sont quand même rattachés à la technologie ST 0,18µm pour certains aspects comme la tension au niveau haut qui est de 1,8V. Ceci vient du fait que cette technologie a été utilisée pour implémenter cette PLL (section 4.7) afin de vérifier la précision des résultats obtenus à haut niveau.

4.5.1. Description fonctionnelle de la PLL

Le schéma de principe d'une PLL à pompe de charge est donné en Figure 4-7. Il s'agit d'un système asservi qui fait en sorte d'avoir un signal Fdivout en sortie du diviseur (rapport de division N) de même fréquence et en phase avec un signal d'entrée Fcomp. Pour ce faire un détecteur de phase-fréquence (PFD) compare en permanence les phases (ou les fréquences) de Fdivout et Fcomp. Cette différence, donnée par des niveaux numériques, est convertie en niveaux analogiques sous forme de courant par la pompe de charge (CP). Le filtre de boucle

(LF) est associé à la pompe de charge et a pour rôle de filtrer les composantes parasites. La tension en sortie du filtre est destinée à piloter l'oscillateur commandé en tension (VCO) soit en diminuant soit en augmentant Fout (et par suite Fdivout) suivant les variations en entrée du PFD.

Grâce à cette configuration, on obtient une fréquence Fout=N*Fcomp. Donc en modifiant N, on obtient un synthétiseur de fréquences.

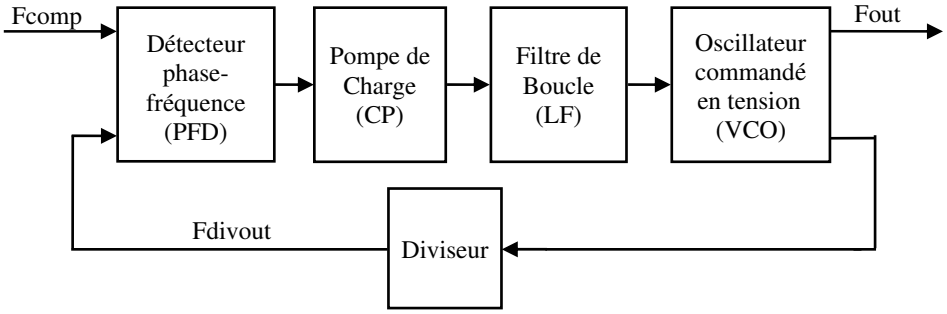


Figure 4-7: Schéma de principe d'une PLL

4.5.2. Spécifications générales de la PLL

Ici sont présentées les spécifications générales de la PLL :

- Dans notre cas la fréquence d'entrée (ou de comparaison Fcomp) est de 25MHz, celle de sortie Fout est de 200MHz d'où un rapport de division N de 8.
- Suivant les corrections à apporter au signal Fdivout, la pompe de charge fournit un courant positif ou négatif de valeur maximum 100µA : c'est le gain du CP noté KΦ.
- Certains effets indésirables apparaissent à Fout±Fcomp (appelés en anglais *reference spur* (voir A.5)). Ceux-ci devront être atténués d'au moins 50dB par le LF. Cette valeur influe sur la topologie de LF ainsi que sur les valeurs des différents éléments le constituant.
- Le VCO a une caractéristique linéaire (du moins au niveau comportemental). L'Equation 4-3 décrit son fonctionnement. Vvcoin est la tension en entrée du VCO qui varie de 0 à 1V et Kvco son gain valant 225 (MHz).

Equation 4-3: $Fout = -Kvco \times Vvcoin + 275$

4.5.3. Terminologie

Je définis ici quelques termes utilisés par la suite :

- **Temps de verrouillage ou d'accrochage de la PLL:** Il s'agit du temps au bout duquel on considère que les signaux Fdivout et Fcomp sont de même fréquence et en phase avec une certaine tolérance. A partir de cet instant Fout est N fois égale à Fcomp.
- **Jitter[D.S, N.S]:** Pour évaluer le niveau de perturbation apportée lors de l'injection, nous nous intéresserons à la qualité de l'horloge générée par la PLL après verrouillage.

Celle-ci est spécifiée par son *jitter* ou par son bruit de phase. Par la suite on ne s'intéressera qu'au *jitter*. Le *jitter* correspond à la déviation (positive ou négative) des transitions de l'horloge réelle par rapport aux transitions (dans notre cas montantes) du signal attendu. Cette déviation est causée par le bruit au niveau du circuit. Pour la mesurer, on utilisera la définition du *jitter* de période [D.S] qui est la déviation maximale entre une période réelle et celle spécifiée (dans notre cas 5ns). A cause de sa nature aléatoire, cette valeur est mesurée soit pic-à-pic ce qui correspond à la différence entre la valeur maximum et la valeur minimum de la période (Figure 4-8), soit en moyenne quadratique (rms : *root of mean square*), ce qui représente l'écart-type (ou la déviation standard) de la distribution normale des déviations [L.N]. On utilisera par la suite la mesure pic-à-pic.

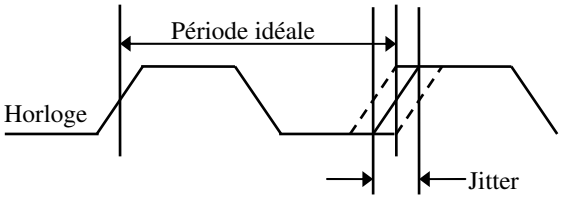


Figure 4-8: Définition du jitter de période

4.5.4. Description et paramétrage des blocs de la PLL

Afin de respecter les spécifications générales de la PLL, les différents blocs doivent être configurés en conséquence. Pour des raisons de clarté, la description détaillée des différents blocs de la PLL et leurs paramétrages sont présentés à l'Annexe A.

4.5.5. Simulation de la PLL sans injections

La Figure 4-9 donne la forme du signal d'entrée ou de comparaison Fcomp, du signal en sortie du diviseur Fdivout, des signaux (UP) et (DN) en entrée de la pompe de charge, du courant en sortie de la pompe de charge (Ipump) et de la tension à l'entrée du VCO (Vvcoin). Cette figure servira ultérieurement de référence pour l'analyse des effets des injections de fautes.

La tension à l'entrée du VCO permet de vérifier si la bonne tension (333,33mV d'après l'Equation 4-3), permettant d'obtenir une sortie de 200MHz, a été appliquée en entrée du VCO. La PLL est considérée comme verrouillée à partir de 3µs. Malgré cela, la PLL apporte de petites corrections dues aux imperfections volontaires du modèle (résistance d'entrée du VCO non infinie, *dead zone*..). Le courant Ipump a été donné, car si, après injection, il y a perturbation, la pompe de charge devra le corriger donc sa forme changera.

La Figure 4-10 donne un agrandissement de la zone après verrouillage. Comme on peut le voir, la tension à l'entrée du VCO n'est pas tout à fait continue et elle évolue dans une plage allant de 333,3285mV à 333,33mV. Le courant n'est pas nul mais a une valeur maximale de 12µA bien en dessous de ce que pourrait fournir la pompe de charge (100µA). Ces valeurs

évoluent dans des plages assez restreintes, bien que le modèle de la PLL prenne en compte certaines imperfections. Mais il ne faut pas oublier que la description est faite à un niveau d'abstraction élevé, avec notamment une modélisation du VCO selon l'Equation 4-3 et en l'absence de bruit.

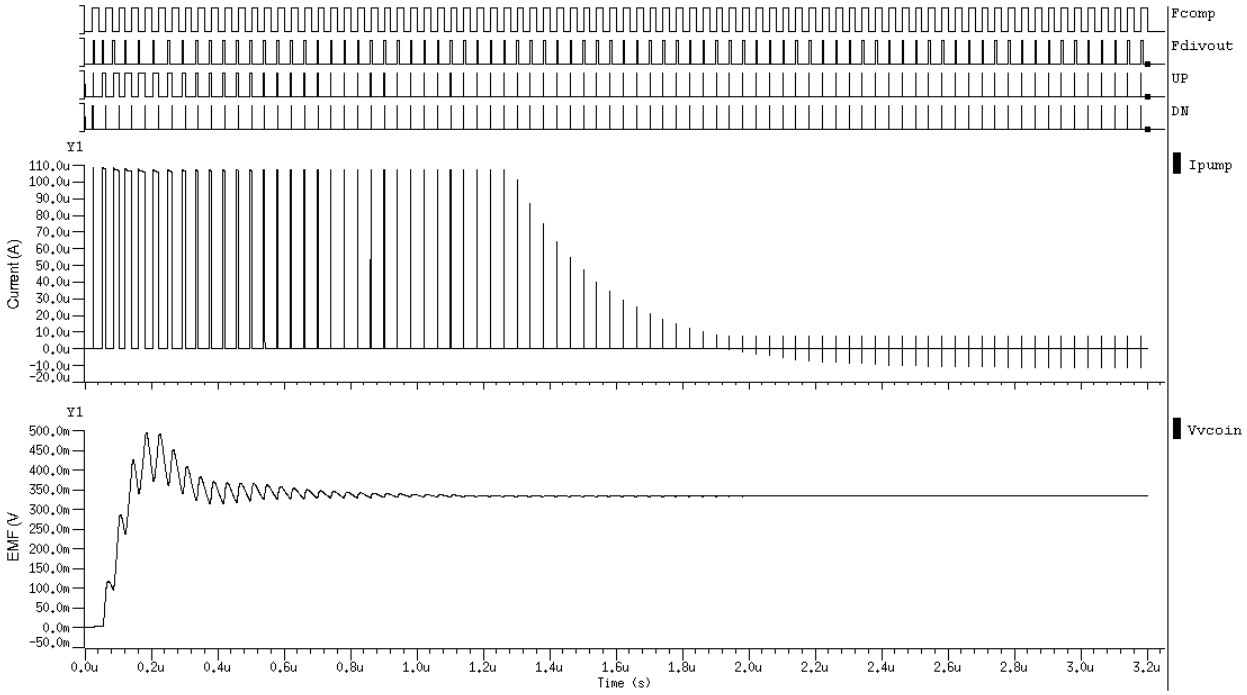


Figure 4-9: Simulation de la PLL sans injections

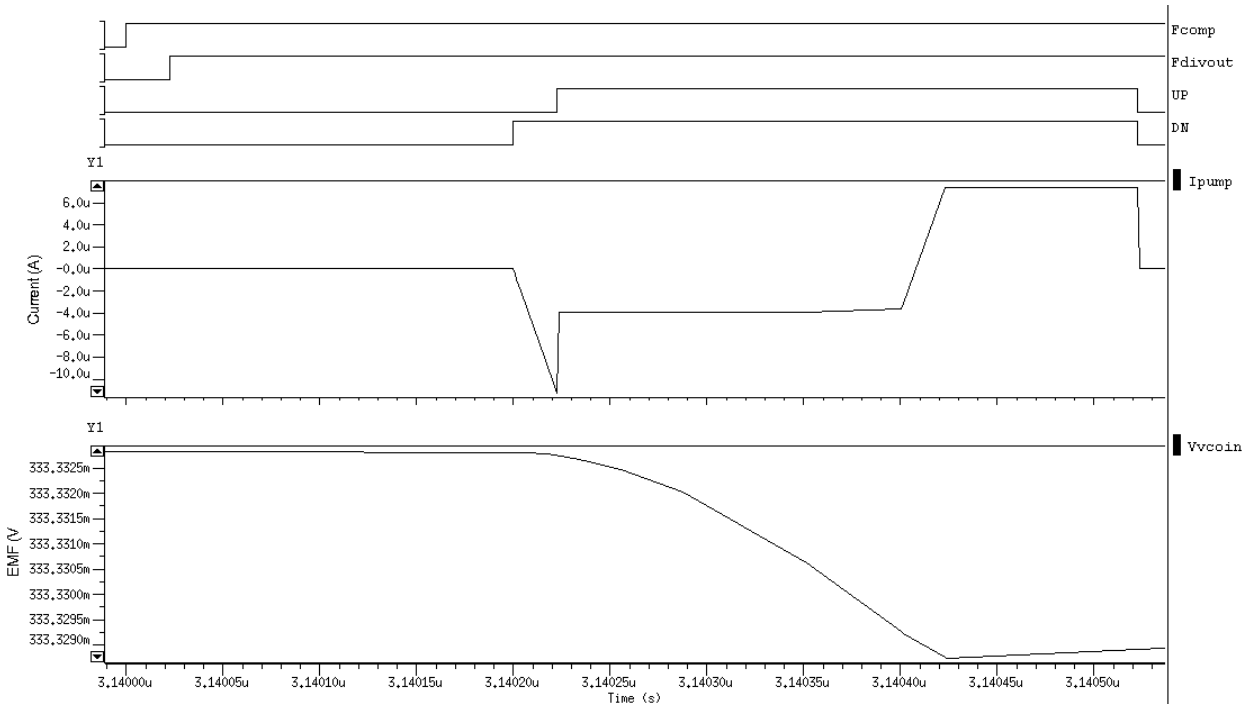


Figure 4-10: Corrections après verrouillage de la PLL

Le *jitter* mesuré sur quelques périodes est égal à 2ps. Cette valeur est sous-estimée pour les mêmes raisons que précédemment. L'article [D.S. 02] considère qu'un *jitter* de 15ps rms (pour

les mêmes plages de fréquence que dans notre cas) est une valeur agressive pour le système en aval qui est un convertisseur analogique numérique. Faute de pouvoir calculer précisément ce *jitter* (une étude de bruit est nécessaire pour cela), j'ai converti cette valeur en pic-à-pic en ne considérant que 95,40% des mesures possibles [M.P] ce qui donne un *jitter* maximum de 60ps pic-à-pic. Ceci donne au niveau des périodes générées une plage de tolérance située entre 5±0,030ns. Toute valeur de période en dehors de celle-ci n'est pas bonne.

4.5.6. Injections dans la PLL décrite en VHDL-AMS

Dans ce qui suit, les résultats de quelques injections sont présentés afin de montrer l'impact global sur la PLL. Une étude beaucoup plus détaillée sera présentée à partir de la section 4.5.7. Les injections présentées ont été réalisées après le verrouillage de la PLL, afin d'étudier l'influence sur la fréquence générée.

4.5.6.1. Application du flot

L'approche présentée à la section 4.3 (injections sur une description de haut niveau) n'est pas applicable n'importe où dans un bloc analogique. Il faut que la cible d'injection soit un nœud contenant une capacité et connectant deux sous blocs dans la hiérarchie, et le choix dépend du degré de raffinement des blocs concernés par l'injection. Ainsi un VCO décrit par l'Equation 4-3 ne peut pas directement faire l'objet d'une injection de courant sur sa sortie car cette dernière est un signal numérique. Par contre, le noeud entre la pompe de charge et le filtre peut par exemple faire l'objet d'une telle injection. Dans notre cas, la pompe de charge est décrite avec des transistors NMOS et PMOS fonctionnant dans l'une des trois régions. Le filtre est décrit avec des composants discrets. Donc un courant circule dans ce nœud et on pourra le sommer avec celui de l'injection.

4.5.6.2. Réactions à l'injection d'un pic de courant

La Figure 4-11 montre le nœud concerné par l'injection situé à l'entrée du filtre.

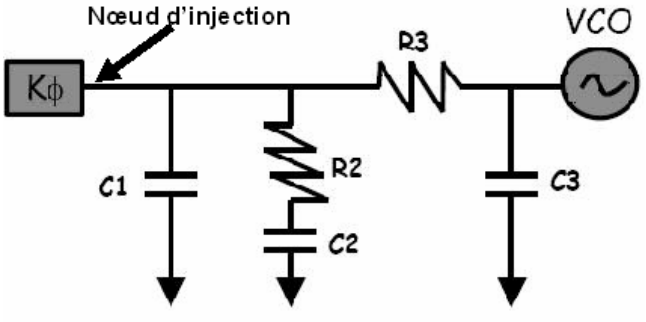


Figure 4-11: Emplacement de l'injection

Le pic de courant injecté a différentes configurations permettant de couvrir tous les cas de figures lors du fonctionnement normal. Ainsi, on peut injecter lorsque la pompe de charge se trouve dans son état de haute impédance, ou bien lorsqu'elle fournit ou reçoit du courant. Si

correction il y a, celle-ci se fera tous les 40ns et pendant un temps très court. Donc l'état de haute impédance est le plus probable et toutes les injections présentées ici l'ont été dans cet état, cela dit d'autres injections ont été réalisées lors des deux autres états sans mettre en évidence de différences notables par rapport à l'injection à l'état haute impédance.

Pour bien voir le comportement de la PLL après injection, toutes ces injections se produiront lorsque la PLL est verrouillée, c'est-à-dire plus de 3µs après le début de la simulation réalisée par ailleurs dans les mêmes conditions que précédemment (Figure 4-9).

La Figure 4-12 donne la forme du courant injecté (set_courant). Il s'agit d'un pic positif décrit par l'Equation 4-1 avec les mêmes paramètres que dans la section 4.4.1 et Qinj = 0,2pC qui donne un courant maximum relativement faible. L'instant d'injection correspond à 3,1µs. Cette injection est contrôlée par le signal numérique inj. Celui-ci reste à '1' pendant 5ns.

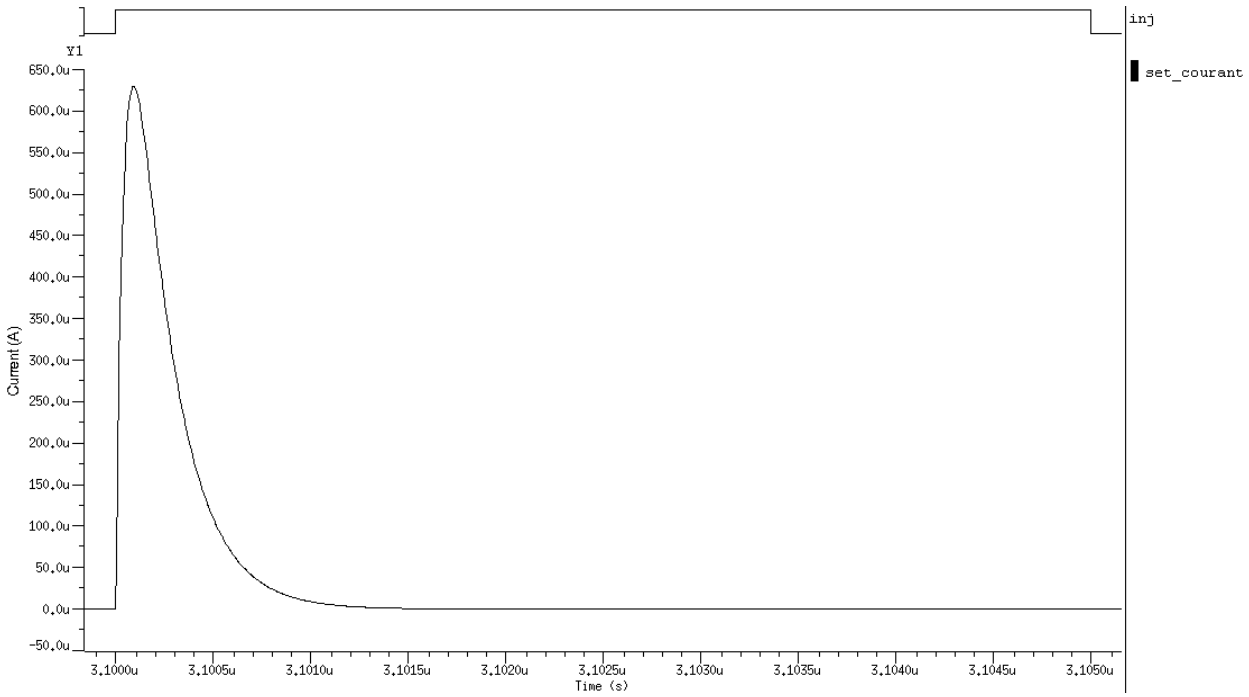


Figure 4-12: Forme du courant injecté

La Figure 4-13 correspond à l'injection du pic de courant précédemment cité entre 3,1µs et 3,105µs. En comparant cette figure à la Figure 4-9, on voit clairement par exemple qu'aux instants 3,14µs et 3,18µs, deux pics de courants plus grands apparaissent en sortie de la pompe de charge. Ceci vient du fait que le déphasage entre les signaux Fcomp et Fdivout à l'entrée du PFD est plus important. La raison de ce changement vient de l'augmentation de la tension Vvcoin. Pour compenser le courant injecté, la PLL fait en sorte de ramener la tension Vvcoin à son niveau d'avant injection. La forme du courant donné (Ipump) est celui en sortie de la pompe de charge uniquement. Le courant fourni au filtre est la sommation entre le courant injecté (set_courant) et Ipump.

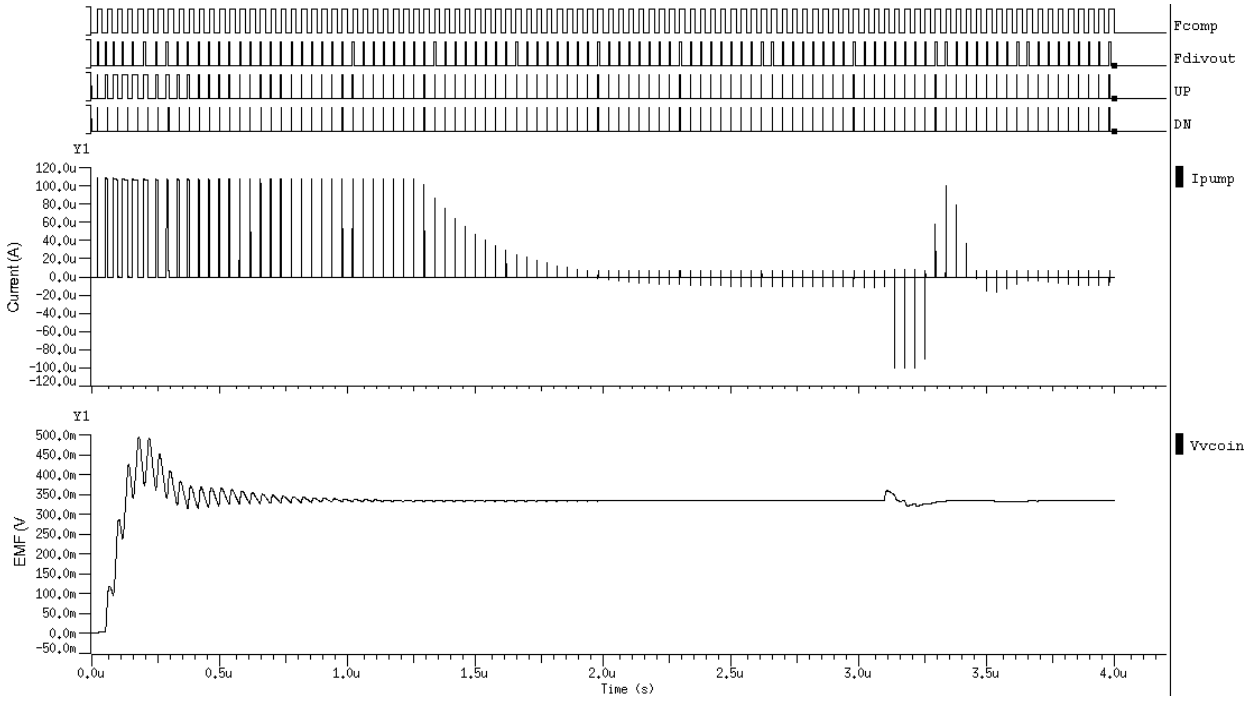


Figure 4-13: Résultat de l'injection d'un pic de courant positif

4.5.6.3. Forme de la tension au niveau du VCO

La Figure 4-14 montre la forme de la tension à l'entrée du VCO avant, pendant et après l'injection du pic de courant positif. On voit sur cette figure que la tension augmente, ce qui est normal vu qu'un pic positif signifie que la pompe de charge fournit du courant au filtre. A 3,1µs, cette tension augmente donc et cherche à atteindre au fur et à mesure du temps écoulé environ 333,33mV qui donne une fréquence de 200MHz.

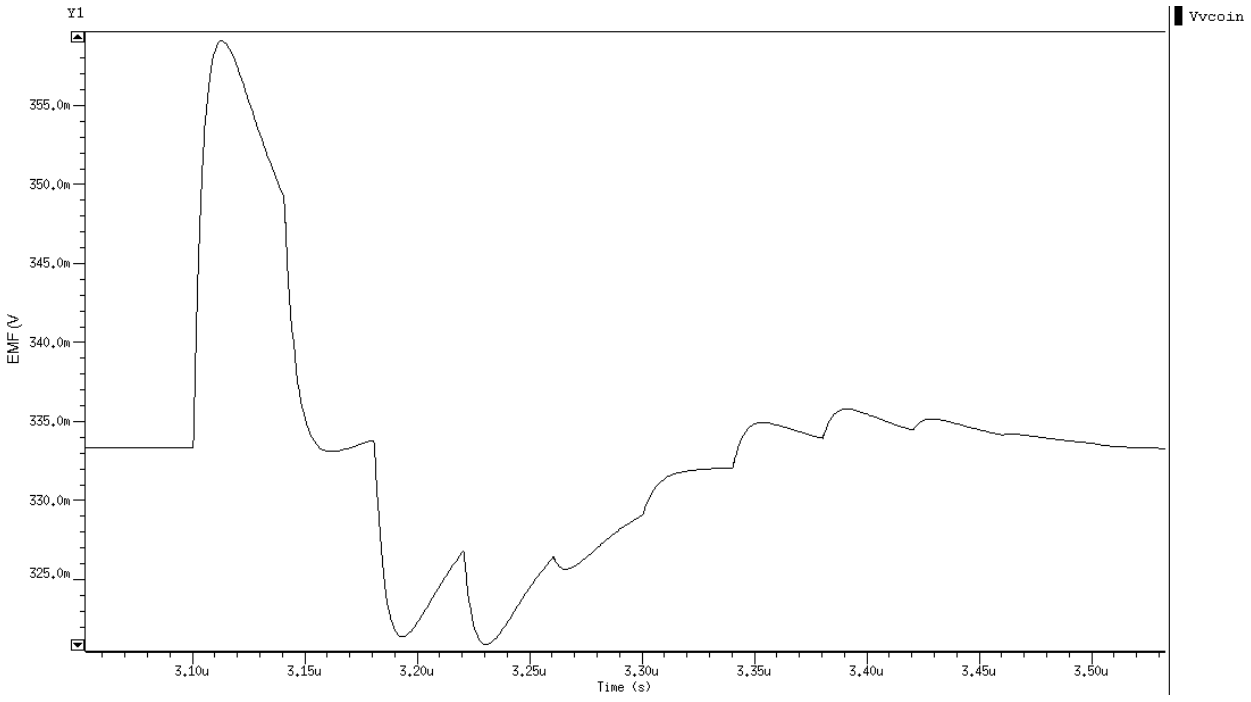


Figure 4-14: Forme de la tension à l'entrée du VCO avant, pendant et juste après l'injection du pic

4.5.7. Etude détaillée des injections situées à l'entrée du filtre

4.5.7.1. Impact de la modélisation des fautes

Par la suite on s'intéressera à l'étude des injections en faisant varier l'amplitude du pic et sa durée. Pour ne changer que l'amplitude ou la durée, le modèle à double exponentielle n'est pas vraiment adapté car il faut jouer sur les trois paramètres τ_1, τ_2 et I_0. Par contre le modèle à rampes, de part sa définition, peut être utilisé tout en gardant ou l'amplitude ou la durée constante. A partir de la Figure 4-15 on obtient les paramètres suivants pour la rampe (section 4.4.2): RT=60ps, FT=380ps, PW=500ps et PA=0,63mA, les paramètres pour la double exponentielle étant les mêmes que dans la section 4.4.1.

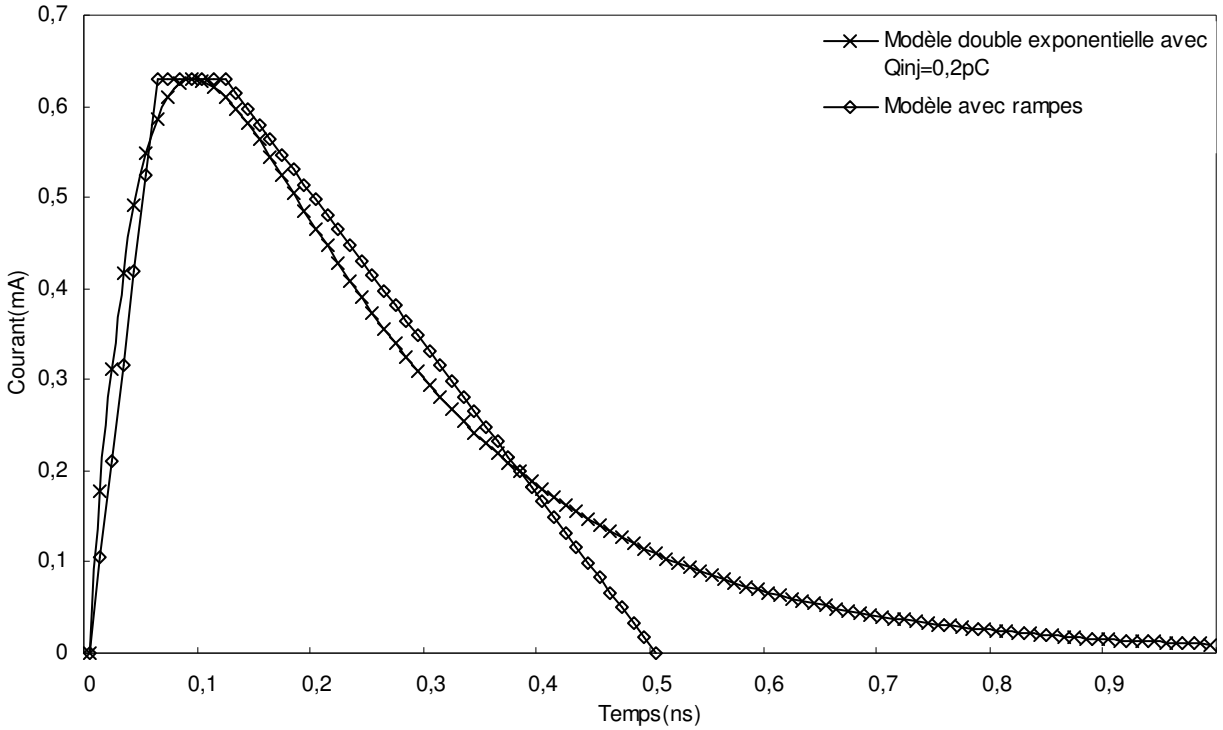


Figure 4-15: Approximation du modèle à double exponentielle par celui à rampes pour Qinj=0,2pC

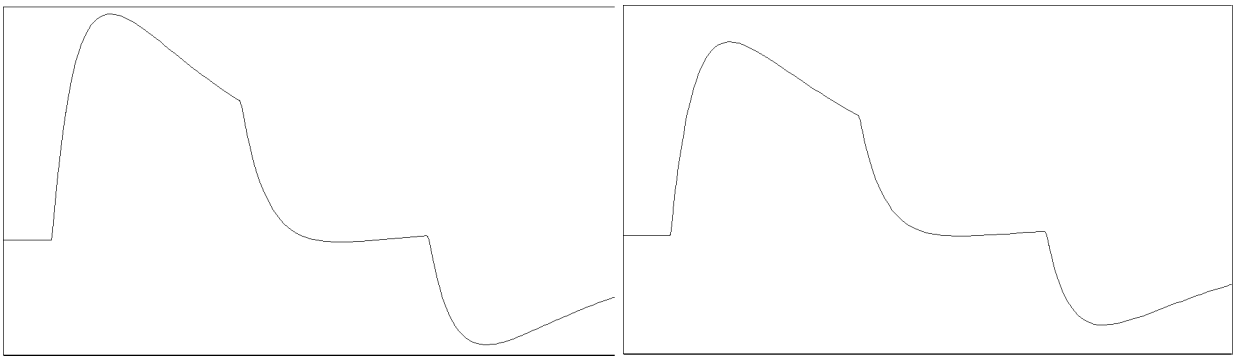


Figure 4-16: Comparaison de l'évolution de la tension d'entrée du VCO après injection d'un même pic de courant, modélisé par une double exponentielle (a) et un modèle à rampes (b) – les échelles sont identiques dans les deux graphes

Le résultat de l'injection dans le même nœud à l'entrée du filtre (Figure 4-11), avec les deux pics est résumé dans la Figure 4-16 (les deux échelles sont identiques). Les deux pics injectés produisent pratiquement le même effet à 10% près ; cette tolérance est calculée par rapport à l'amplitude crête à crête du pic généré. Même si le modèle à double exponentielle est plus précis, on peut considérer que le modèle à rampes reste exploitable d'autant plus que les simulations avec ce pic sont un peu plus rapides. Donc, dans les prochaines sections seules des injections utilisant le modèle à rampes seront analysées. Cela dit, rien n'empêche d'utiliser le modèle à double exponentielle notamment lors des injections au niveau transistor.

4.5.7.2. Résultats en fonction de l'amplitude du pic

La Figure 4-17 donne les résultats d'injection d'un pic à l'entrée du filtre pour différentes valeurs de l'amplitude. Seuls Ipump et Vvcoin sont donnés. Les pics sont injectés à partir de 3,1µs et ont la même forme c'est-à-dire que RT, FT et PW sont les mêmes et égaux respectivement à 60ps, 380ps et 500ps. Ainsi la Figure 4-17 (a) est le résultat de l'injection d'un pic de 1,6mA, (b) de 1,3mA, (c) de 1,0mA, (d) de 0,7mA, (e) de 0,4mA et (f) de 0,1mA. Tous les graphes reportés ici ont les mêmes échelles pour les deux axes (temps en µs et amplitude en V). Au vu de la Figure 4-17, on peut dire que bien que la durée du pic soit très faible, celui-ci a un impact sur la PLL pendant une durée beaucoup plus longue durant laquelle les fréquences produites ne sont plus "correctes". Pour bien comprendre ce qui se passe, voyons plus en détail le résultat de l'injection d'un pic de 1,6mA.

On remarque qu'au niveau de l'instant d'injection (3,1µs), la tension augmente rapidement de près de 60mV ce qui fait diminuer la fréquence à la sortie du VCO. Ainsi juste avant l'injection, on avait une période de 5,00000ns et juste après l'injection, la période du signal est de 5,28334ns. Cette dernière valeur sort de la plage de tolérance précédemment fixée (5±0,030ns). Pour corriger la fréquence la PLL doit faire en sorte de diminuer la tension à l'entrée du VCO en récupérant du courant depuis la pompe de charge d'où l'apparition de pics négatifs à la première correction se produisant après l'injection. Juste avant la 1ère correction de la pompe de charge (à environ 3,140µs) la période de Fout est de 5,24346ns qui ne fait toujours pas partie de la plage de tolérance.

Entre les moments où l'injection et la première correction se sont produites, on dénombre 7 périodes d'horloges "erronées". Grâce à un effet cumulatif, ceci permet de retarder le front montant du signal en sortie du diviseur (Fdivout), ce qui oblige la pompe de charge à fonctionner pendant un temps beaucoup plus important. En fait, avant injection, on avait un déphasage entre Fcomp (signal de comparaison dont la fréquence est de 25MHz) et Fdivout de 22,5ps (en théorie ce déphasage devrait être nul, mais sa valeur résulte des imperfections introduites dans le modèle). Au moment de la 1ère correction (3,140µs), ce décalage est de 2,09ns ce qui explique que la pompe de charge fonctionne plus longtemps (le pic est plus important à l'instant 3,140µs). Juste après la 1ère correction, la période est de 5,036ns et ne cesse de s'améliorer, pourtant les corrections ne cessent pas. Ceci vient du fait que le déphasage entre les signaux Fcomp et Fdivout n'a pas repris sa valeur initiale, d'ailleurs au moment de la 2ème

correction (3,18µs) ce déphasage est de 2,248ns. Donc même si la période reste dans la plage de tolérance fixée, les corrections se poursuivent pour faire en sorte que les signaux à l'entrée du PFD soient en phase (dans notre cas à quelques 22ps près). Globalement dans ce cas le pic injecté a un effet sur la PLL pendant 900ns qui correspondent à 180 cycles d'horloges. Parmi ces 180 cycles, il y a ceux qui sortent de la plage de tolérance et le reste correspond à un déphasage trop important (par rapport à celui obtenu après verrouillage) entre les deux signaux Fdivout et Fcomp à l'entrée du PFD. La PLL retrouve son état normal lorsque ces deux derniers signaux sont de même fréquence et en phase.

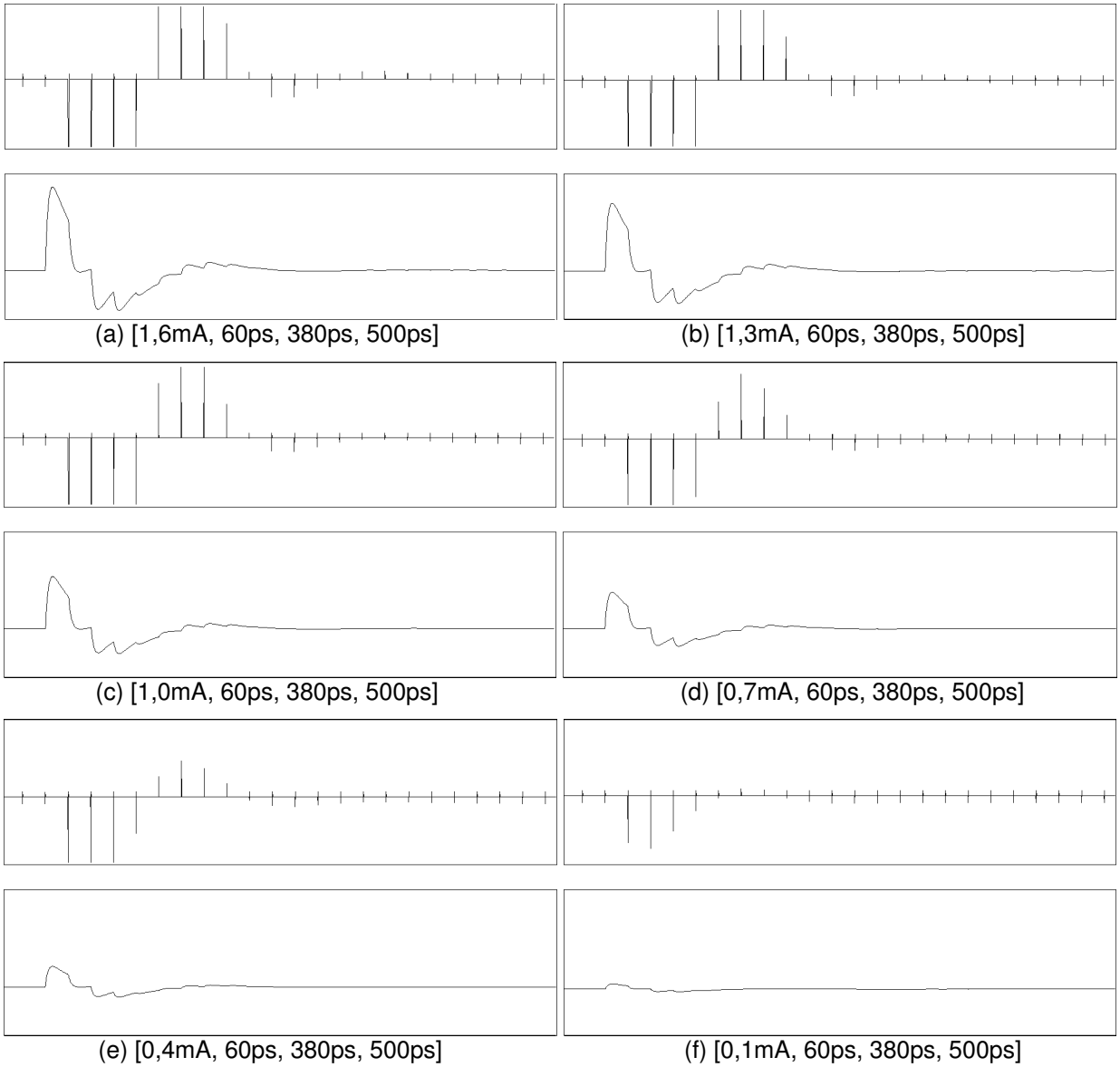


Figure 4-17: Résultat de l'injection d'un pic positif [PA, RT, FT, PW] à l'entrée du filtre. Seule l'amplitude maximale (PA) change – Les échelles sont identiques pour les différents graphes

Pour les autres injections, on remarque que plus la valeur maximum du pic diminue, moins la PLL est perturbée. Si on continue à baisser la valeur maximum du pic, on atteint une valeur en dessous de laquelle le pic injecté n'a plus d'effet sur la PLL c'est-à-dire que la période reste dans la plage de tolérance fixée. Dans notre cas cette limite s'établit autour d'un pic de 0,1mA.

4.5.7.3. Résultats en fonction de la durée du pic

La Figure 4-18 montre l'injection d'un pic de courant d'amplitude maximum 1,6mA à l'entrée du filtre, pour différentes valeurs du temps de montée, du temps de descente et de la durée totale. Tous les graphes reportés ici ont les mêmes échelles pour les deux axes (temps en µs et amplitude en mV).

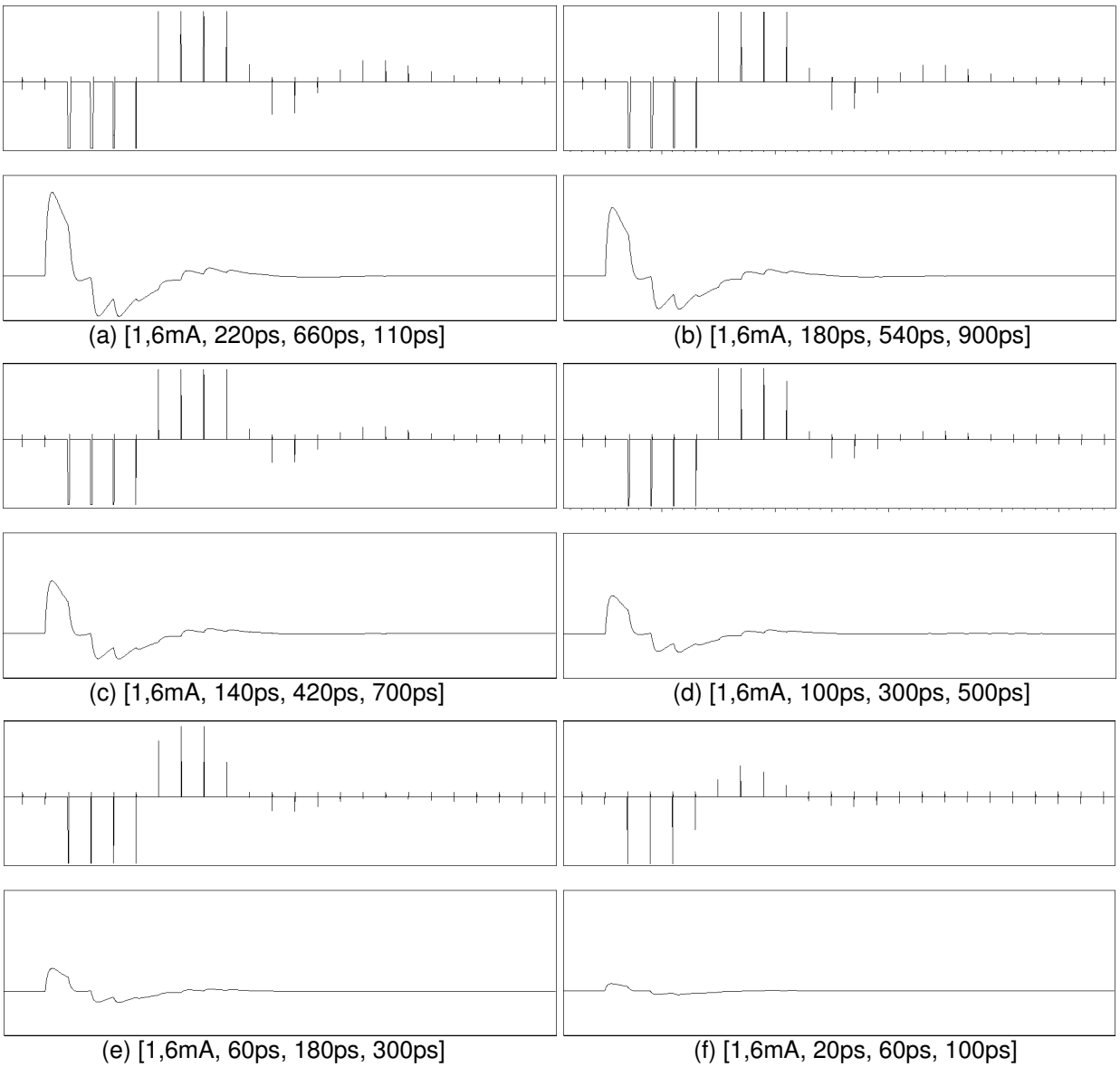


Figure 4-18: Résultat de l'injection d'un pic positif [PA, RT, FT, PW] à l'entrée du filtre dont l'amplitude maximale (PA) est fixée à 1,6mA – Les échelles sont identiques pour les différents graphes

Ces résultats montrent que plus la durée des pics diminue, moins la PLL est perturbée. Mais les mêmes remarques que précédemment sont valables à savoir que la fréquence Fout change sur une durée très longue par rapport à la durée de l'injection. En plus le déphasage généré entre les signaux Fcomp et Fdivout augmente avec l'augmentation de la durée des pics.

4.6. Conclusions sur les injections à haut niveau

Le cas d'étude a démontré la faisabilité de fournir un flot unifié pour des blocs numériques et analogiques ainsi que l'intérêt de l'approche en analysant tôt l'effet de fautes transitoires sur le comportement du circuit AMS. La PLL a été choisie comme démonstrateur, mais l'intérêt de l'approche est de permettre l'analyse de l'impact des fautes dans les blocs fonctionnels aussi bien numériques qu'analogiques. Il serait ainsi possible d'observer directement l'impact du *jitter* de l'horloge généré sur la mémorisation de données erronées dans les blocs séquentiels numériques.

On a vu que lors des injections dans la PLL, un courant de faible amplitude produit un effet sur une durée très longue par rapport à la durée de l'injection. Durant cette durée certaines fréquences générées font que le *jitter* de la PLL sorte de la plage de tolérance fixée.

Des analyses au niveau transistor vont être maintenant présentées pour valider les résultats obtenus à haut niveau.

4.7. Injections de fautes dans la PLL décrite au niveau transistor

4.7.1. Introduction

Le but de cette section est de vérifier les résultats précédemment obtenus. Pour cela des injections seront réalisés aussi bien à l'entrée du filtre qu'à l'intérieur des autres blocs non visibles lorsque la PLL est décrite en langage de haut niveau. Une PLL au niveau transistor a donc été réalisée et la description des différents blocs est résumée à l'Annexe B. Cette annexe décrit l'architecture utilisée pour respecter les caractéristiques résumées dans les Tableaux A-1 à A-6 de l'Annexe A. En fait, tous les paramètres spécifiés dans ces tableaux ne sont pas directement reproductibles, comme les temps de propagation qui sont donnés a priori mais sont forcément à affiner lors de l'implantation effective.

Comme mentionné dans la section 4.5, la technologie ST 0,18µm a été choisie. Les simulations ont été réalisées avec le logiciel virtuoso spectre sous cadence [V.S].

4.7.2. Simulation de la PLL sans injections au niveau transistor

La Figure 4-19 montre l'évolution de la tension d'entrée du VCO en fonction du temps. Comme on peut le remarquer, la tension finale (après accrochage qui apparaît au bout de 3µs) est d'à peu près 500mV, ce qui permet de donner 200MHz en sortie du VCO (Figure B-9). La tension est donc différente de celle modélisée à haut niveau, mais la fréquence générée est la même.

Cette figure servira au moment des injections. En effet si la PLL est perturbée, elle cherchera à corriger l'erreur produite en réajustant la valeur de Vvcoin de telle sorte à ce qu'elle repasse à peu près à 500mV.

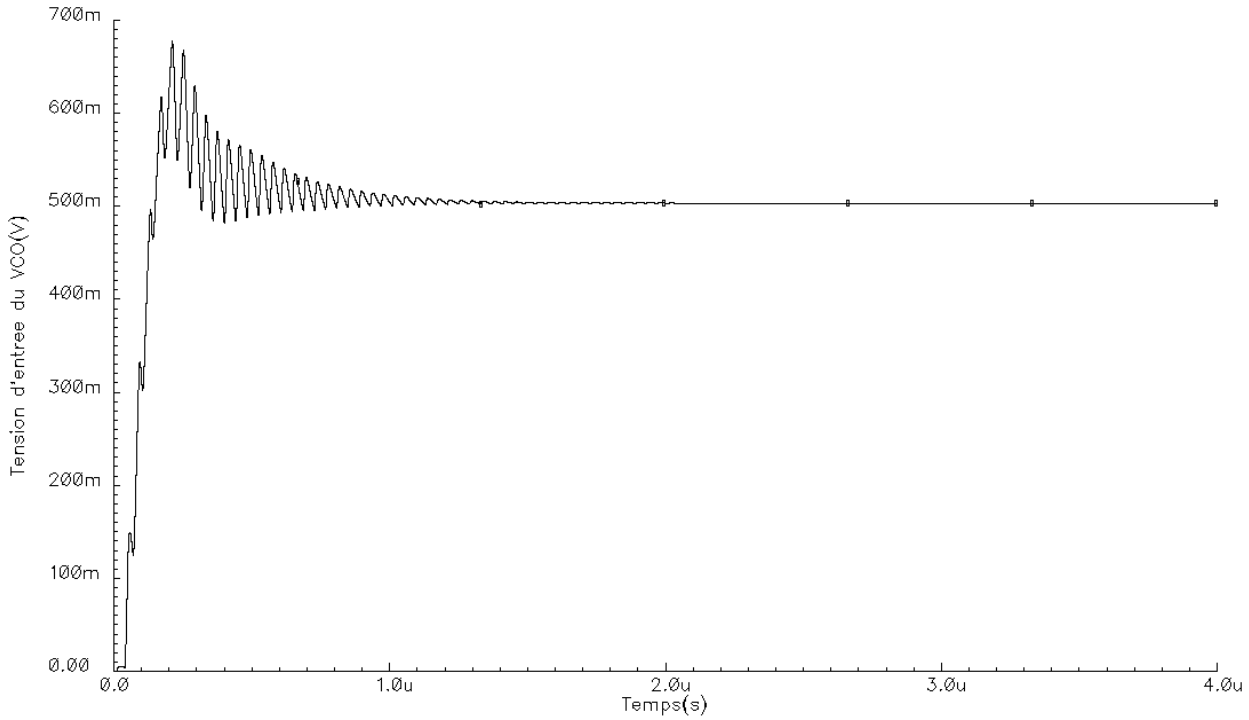


Figure 4-19: Evolution de la tension d'entrée du VCO en fonction du temps

Pour voir la qualité de la fréquence générée en sortie, on a appliqué une Transformée de Fourier Discrète (TFD) après accrochage de 4µs à 12µs avec une fenêtre rectangulaire et $1E^{+6}$ échantillons, sur le signal carré (Fout) en sortie du VCO. La Figure 4-20 donne le résultat de la TFD, après verrouillage, autour de la fréquence fondamentale de 200MHz.

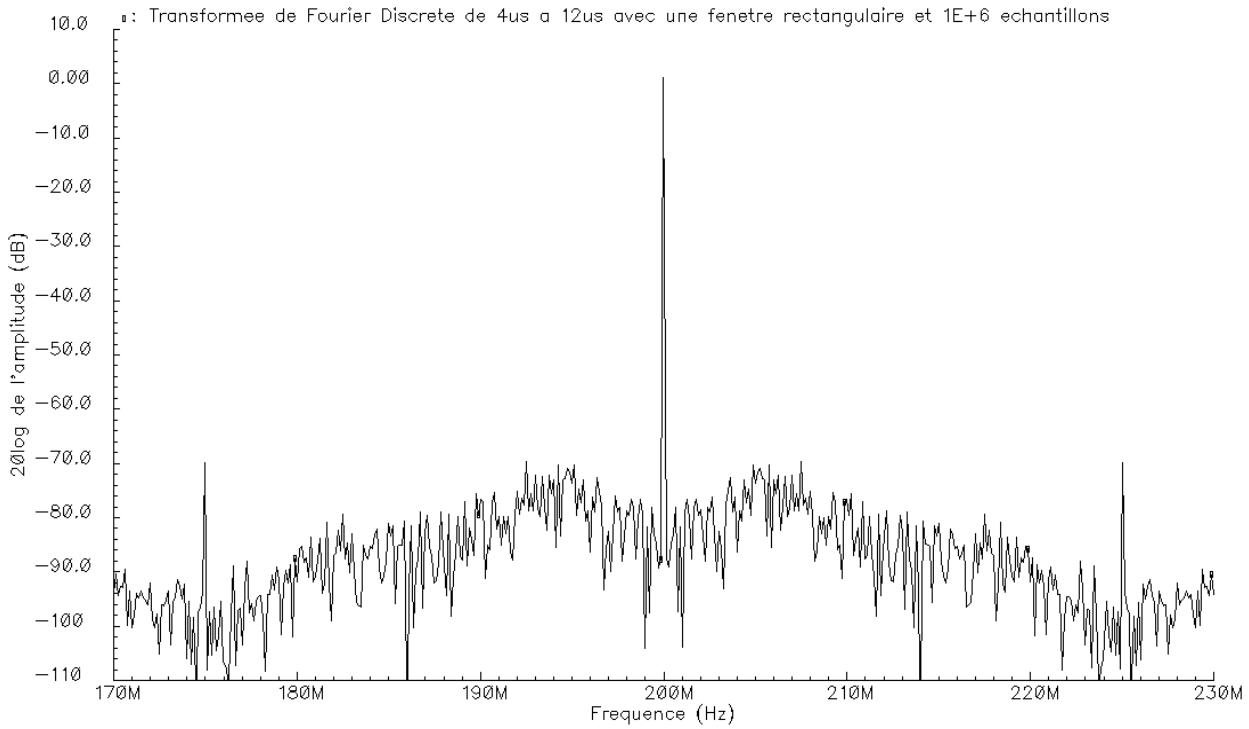


Figure 4-20: TFD autour de la fréquence fondamentale de 200MHz

Comme on peut le remarquer, le pic autour de la fréquence de 200MHz, est très étroit ce qui atteste d'une bonne qualité de l'horloge générée par la PLL. D'ailleurs la valeur du *jitter* mesurée est de 7,5ps et les conditions de mesure sont les mêmes que dans la section 4.5.3. On rappelle que le *jitter* maximum a été fixé à 60ps ce qui donne une plage de tolérance pour la période située entre 5±0,030ns. Toute valeur de période qui sort de cet intervalle n'est pas bonne.

Sur la même figure, on peut aussi voir les *reference spur* qui sont des fréquences indésirables (voir A.5) situées à 200±25MHz et qui ont une atténuation de 70dB.

4.7.3. Spécification des campagnes d'injection

L'Annexe C regroupe le détail des injections dans la PLL décrite au niveau transistors. Ces injections sont réparties suivant les blocs de la PLL. Les nœuds d'injections sont donnés à la Figure C-1, Figure C-4, Figure C-9, Figure C-11, Figure C-14 et Figure C-16 respectivement pour le filtre, le comparateur avec diviseur, le diviseur par 8, le PFD, la pompe de charge et l'oscillateur.

Les pics de courant injectés sont positifs ou négatifs pour des valeurs de Qinj de 0,2pC, 0,3pC ou 0,5pC pour les injections positives et le complément de ces valeurs pour les injections négatives. La forme du courant injecté est décrite par une double exponentielle à l'Equation 4-1. On présentera par la suite une synthèse de ces injections. Mais avant cela, on va décrire la campagne d'injection et les résultats collectés à chaque campagne :

- Les blocs concernés par les injections sont : PFD, pompe de charge, filtre, oscillateur (au nombre de 2 voir C.6), comparateur avec son diviseur et diviseur par 8. Au niveau de chaque bloc, on a fait varier Qinj ainsi que la polarité des pics injectés. Ce qui donne en tout 3x2x7=42 campagnes d'injections avec près de 235 simulations.
- Les injections sont réalisées après le verrouillage de la PLL, c'est-à-dire à partir de 3µs. En prévision de la perturbation et pour faciliter les campagnes d'injection, on laisse 2µs avant de réinjecter de nouveau. Donc on aura des temps d'injections (Tinj) à 3µs pour le nœud N1, 5µs pour N2, 7µs pour N3,… pour chaque campagne.
- La différence de tension pic-à-pic (Vpp_n_inj) est mesurée au niveau de chaque nœud d'injection.
- La durée de la perturbation (Durée_pertu) est mesurée à partir de la courbe représentant Vvcoin. En effet celle-ci change d'allure si une perturbation au niveau de la PLL apparaît. La mesure à proprement dite s'effectue en calculant le temps mis par Vvcoin pour reprendre une allure d'avant injection. Cette mesure est peu précise car elle ne fait pas appel à un outil adéquat mais repose sur une appréciation visuelle.
- La différence de tension pic-à-pic (Vpp_vcoin_inj) au niveau du signal Vvcoin est aussi mesurée. Cette mesure est à corréler avec la durée de la perturbation.

- Jitter : D'après la tolérance qu'on a fixé, le *jitter* ne doit pas dépasser 60ps et la période générée doit être égale à 5±0,030ns (section 4.5.5 pour les conditions de mesure). Cette mesure est faite sur 2µs à partir de l'instant d'injection mais en tenant compte du front montant de l'horloge généré juste avant l'injection.

4.7.4. Synthèse des résultats d'injection

Comme mentionné dans la section ci-dessus, le détail des injections est donné à l'Annexe C. Dans cette section on va présenter la synthèse de ces injections. Le Tableau 4-1 (respectivement le Tableau 4-2) présente la synthèse lors d'injections de charges négatives (respectivement positives) dans les différents blocs de la PLL. Deux paramètres sont reportés : le *jitter* et la durée de la perturbation. On se concentrera principalement sur le *jitter* en donnant des remarques sur la durée de la perturbation là où c'est nécessaire car ce dernier est juste donné à titre indicatif et sera traité plus en détail dans la section 4.7.4.3.

4.7.4.1. Injections de charges négatives

D'après le Tableau 4-1, tous les nœuds n'ont pas le même effet sur le comportement de la PLL. Ainsi il y a des nœuds qui ne font pas sortir le *jitter* de sa plage de tolérance quelle que soit la charge négative injectée (donnés en gras et soulignés sur le tableau). Comme exemple de nœuds appartenant à cette catégorie, on peut nommer le nœud numérique N1 du bloc comparateur avec diviseur ou le nœud analogique N1 du bloc pompe de charge. D'autres donnent des *jitters* plus ou moins grands en fonction de la charge injectée et peu importe la nature du nœud (analogique ou numérique). Par contre les deux nœuds les plus sensibles (donnant les plus grands *jitter*) sont des nœuds numériques et il s'agit de N5 et N7 du comparateur avec diviseur. Ceci est tout à fait prévisible dans la mesure où ces nœuds correspondent respectivement au double et à la fréquence en sortie de la PLL (signal Fout) et c'est par rapport à ce signal que le calcul du *jitter* est effectué. Les autres nœuds numériques donnant de forts *jitters* sont N9 du bloc PFD et N1 du bloc diviseur par 8, ce dernier étant le signal Fout divisé par 2.

Côté nœuds analogiques produisant de forts *jitter*, ils sont exclusivement contenus dans le bloc VCO. On peut ainsi nommer les nœuds N33, N43, N23 (pour une charge < -0,3pC) ou N31 (pour les charges -0,2pC et -0,3pC). Concernant ce nœud N31, il a un comportement qu'on ne retrouve nulle part ailleurs avec cette importance c'est-à-dire que le *jitter* diminue considérablement lorsque la charge est de -0,5pC par rapport aux autres charges. Pour tenter d'expliquer la raison de cette diminution, il faudrait tenir compte du fait que l'oscillateur est un bloc purement analogique et que de ce fait les nœuds sont interdépendants plus que pour les autres campagnes.

Pour en revenir aux nœuds ne produisant pas d'effets, on peut tenter d'expliquer cela pour les nœuds numériques du fait qu'il y ait :

- Masquage, comme dans le cas de certaines injections dans le PFD,

- Injections de pics négatifs pour un signal qui est déjà à zéro. Le nœud N4 appartient à cette catégorie, et sa sensibilité n'est visible que si la chargée injectée devient positive (en gardant exactement les mêmes niveaux pour les différents nœuds).

Tableau 4-1: Synthèse d'injection de charges négatives dans les blocs de la PLL

Bloc	Noeuds	Jitter (ps)	Durée perturbation (ns)
COMPARATEUR AVEC DIVISEUR	N1, N2, N3, N4, N6	**<15**	<=200
	N5	2695-2699	1300
	N7	4606-4792	1400-1500
PFD	N1, N3, N4, N5, N6, N7, N8, N10	**<10**	0
	N2	859-863	1200
	N9	1390-1400	1400
DIVISEUR PAR 8	N1	1110-1117	1600
	N2, N3, N4	**<9**	0
POMPE DE CHARGE	N1, N2	**<9**	0
	N3	205-495	1000-1100
	N4	10(-0,2pC) 35(-0,3pC) 92(-0,5pC)	400(-0,2pC) 500(-0,3pC) 800(-0,5pC)
	N5	64-352	800-1000
FILTRE	N1	201-498	1200
	N2	**29(-0,2pC)** **40(-0,3pC)** 65(-0,5pC)	1000
	N3	481-1113	1500
VCO	N11, N21, N41, N61, N13	100-931	400-1100
	N31	1241(-0,2pC) 1890(-0,3pC) 168(-0,5pC)	1200(-0,2pC) 1200(-0,3pC) 800(-0,5pC)
	N51	**37(-0,2pC)** 200(-0,3pC) 307(-0,5pC)	400(-0,2pC) 600(-0,3pC) 900(-0,5pC)
	N23	320(-0,2pC) 1081(-0,3pC) 1293(-0,5pC)	900(-0,2pC) 1100(-0,3pC) 1200(-0,5pC)
	N33, N43	941-1491	1000-1200
	N53	**17(-0,2pC)** 1010(-0,3pC) 1209(-0,5pC)	200(-0,2pC) 1100(-0,3pC) 1100(-0,5pC)

4.7.4.2. Injections de charges positives

Le Tableau 4-2 présente la synthèse lors de l'injection de charges positives dans les différents blocs de la PLL. Ici on retrouve les mêmes remarques que précédemment sauf que, et c'est normal, les nœuds n'ont plus la même sensibilité face aux charges injectées. Cela dit il y a quand même quelques remarques qu'on peut tirer de ces injections :

- Les valeurs de durée de perturbation pour l'ensemble des nœuds du bloc comparateur avec diviseur sont très fortes par rapport au *jitter* généré (au maximum 34). Je pencherai pour un "problème" lié à la simulation. En effet avec d'autres injections la forme de la tension à l'entrée du VCO après stabilisation (supposant que le temps après injection est suffisant pour qu'il puisse se stabiliser) diffère de celle obtenue avec ces injections.

- La durée de la perturbation lors de l'injection dans N2 du bloc filtre est très grande vis-à-vis du *jitter* (qui reste très faible). Ceci vient sans doute du fait qu'il s'agisse du nœud le plus chargé avec une capacité de 47pF.
- Quand on compare ces injections avec les précédentes, il apparaît que les charges positives produisent moins d'effets que les charges négatives. C'est une conclusion qu'on tire mais dans l'absolu ceci pourrait ne pas être le cas. En effet, en changeant de conditions d'injections on pourrait faire en sorte que certains nœuds numériques par exemple ne soient plus masqués ou bien injecter les bons pics en fonction de la valeur ('0' ou '1') du nœud concerné par l'injection. La spécification de la campagne est donc un point crucial, déterminant très fortement la confiance pouvant être accordée aux résultats.

Tableau 4-2: Synthèse d'injection de charges positives dans les blocs de la PLL

Bloc	Noeuds	Jitter min-max (ps)	Durée perturbation (ns)
COMPARATEUR +DIVISEUR	N1, N2, N3, N4, N5, N6, N7	**<34**	300-700
PFD	N2, N3, N4, N5, N6, N7, N8, N9	**<20**	0
	N1	860-865	1200
	N10	1398-1411	1400
DIVISEUR PAR 8	N1, N2, N3	**<19**	0
	N4	1383-1402	1300
POMPE DE CHARGE	N1, N2	**<20**	0
	N3	216-601	800-1100
	N4	29-51	400-500
	N5	**24(0,2pC)** **46(0,3pC)** 90(0,5pC)	300(0,2pC) 500(0,3pC) 600(0,5pC)
FILTRE	N1	214-594	1200
	N2	**26(0,2pC)** **39(0,3pC)** 66(0,5pC)	1400(0,2pC) 1400(0,3pC) 1500(0,5pC)
	N3	586-1814	1500
VCO	N11, N31, N41, N51, N23, N33	123-1075	700-900
	N21, N61, N13, N53	1130-2166	900-1100
	N43	**35(0,2pC)** **23(0,3pC)** 124(0,5pC)	200(0,2pC) 200(0,3pC) 300(0,5pC)

4.7.4.3. Evolution de la durée de la perturbation et du jitter en fonction de la tension Vvcoin pic-à-pic due à l'injection

La Figure 4-21 et la Figure 4-22 donnent l'évolution de la durée de la perturbation et du *jitter* en fonction de Vpp_vcoin_inj pour tous les nœuds d'injection et toutes les charges injectées de toutes les campagnes. L'axe des abscisses est donné en logarithme car Vpp_vcoin_inj va de quelques $10^{ème}$ de millivolts à quelques centaines de millivolts.

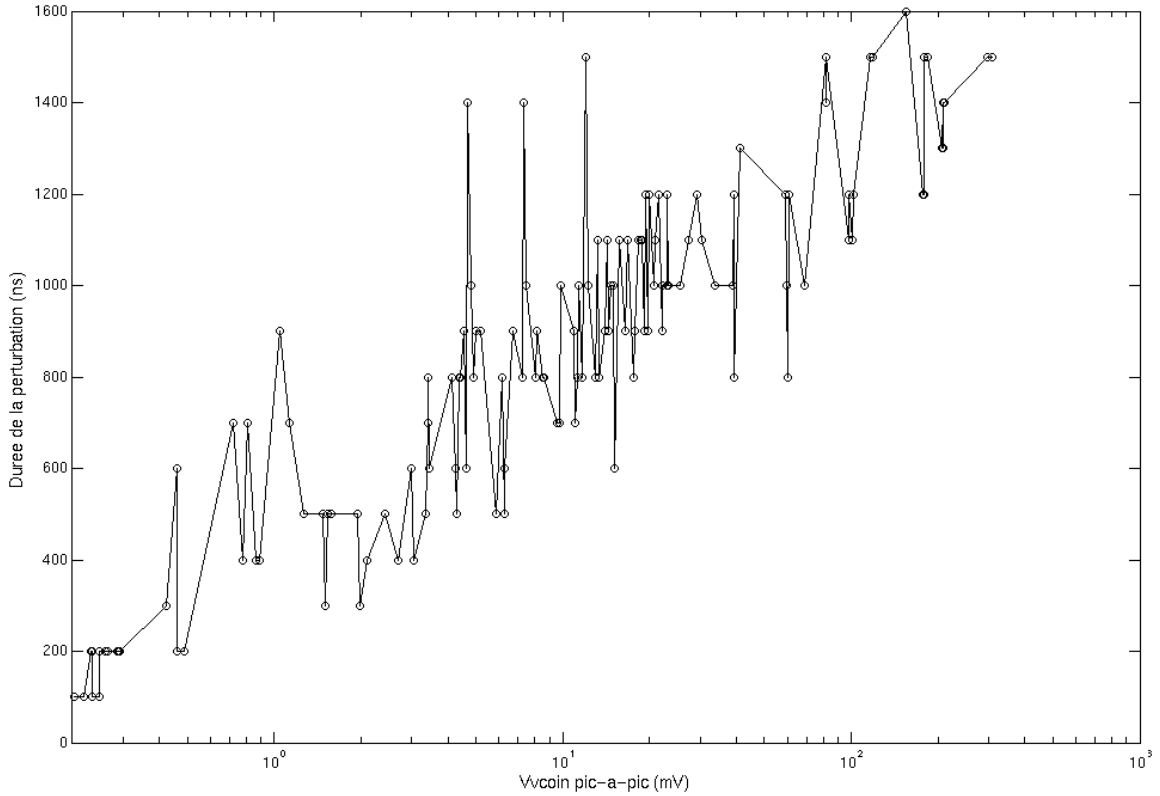


Figure 4-21: Evolution de la durée de la perturbation en fonction de Vvcoin pic-à-pic pour tous nœuds et toutes charges confondues et pour toutes les campagnes

Comme on peut le voir sur la Figure 4-21, la courbe croit presque linéairement en fonction de Vpp_vcoin_inj, même si il y a des fluctuations. Celles-ci s'expliquent par le fait que la méthode de mesure présentée à la section 4.7.3 soit peu précise. Cela dit en admettant cette imperfection, il y a quand même quelques injections comme celles se produisant à 4,68mV, 7,32mV et 12,05mV qui dépassent largement la moyenne de la courbe. En regardant de près, il s'avère qu'il s'agit de injections au niveau du nœud N2 (Figure C-1) qui se trouve être le nœud le plus chargé de toutes ces campagnes, et il est donc normal qu'il faille beaucoup plus de temps pour que la tension dans ce nœud reprenne un niveau d'avant injection.

Pour la Figure 4-22, le *jitter* croît sous forme exponentielle jusqu'à à peu près 30mV puis reprend son ascension en partant de nouveau des valeurs les plus faibles du *jitter*. Ici aussi il y a quelques points qui ne suivent pas l'allure générale de la courbe. Ainsi, à 81mV, trois injections donnent un *jitter* supérieur à 4500ns. Ces trois points correspondent aux injections dans le nœud N7 de la Figure C-4, qui se trouve être la sortie Fout de la PLL et c'est par rapport à cette sortie que les *jitters* sont calculés. De même à à peu près 41mV, le *jitter* est supérieur à 2500ps. Ces injections (au nombre de trois, même si cela n'est pas très lisible) correspondent au nœud N5 de la Figure C-4, qui encore une fois a une relation directe avec Fout puisqu'il est égal à son double. Concernant les injections dans le filtre, elles génèrent les deux plus grandes valeurs de Vpp_vcoin_inj mais avec un *jitter* beaucoup plus petit que d'autres injections.

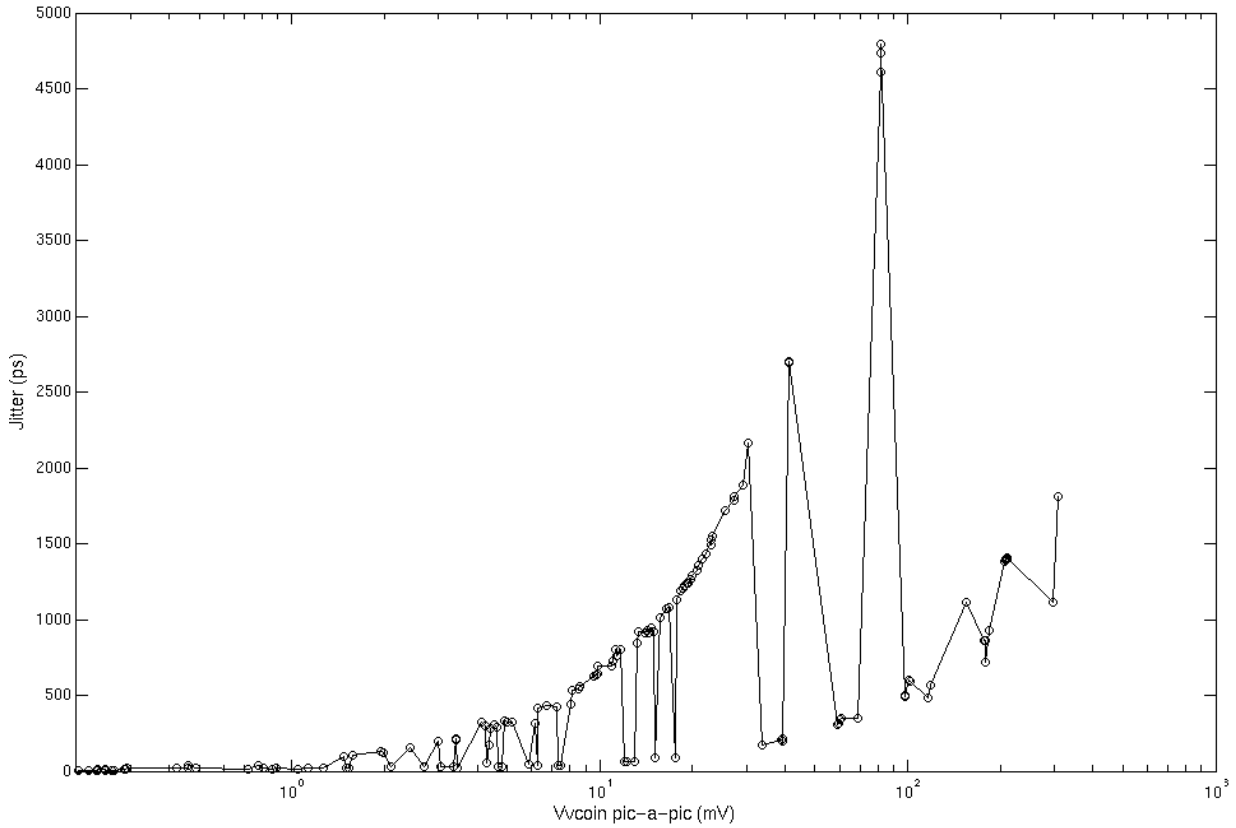


Figure 4-22: Evolution du jitter en fonction de Vvcoin pic-à-pic pour tous nœuds et toutes charges confondues et pour toutes les campagnes

4.7.4.4. Comparaison avec des injections à haut niveau

Au vu des sections 4.7.4.1 à 4.7.4.3, on peut dire qu'on retrouve la même conclusion que lors des injections à haut niveau à savoir qu'un courant de faible amplitude produit un effet sur une durée très longue par rapport à la durée de l'injection. Durant cette durée certaines fréquences générées font que le *jitter* de la PLL sort de la plage de tolérance fixée.

Pour comparer ces injections d'un point de vue quantitatif, je donne le Tableau 4-3. Il contient les résultats d'injections dans le nœud N1 du bloc filtre (Figure C-1) à haut et bas niveau. Le courant injecté à haut niveau est modélisé par une rampe dont les caractéristiques sont : [0,63mA, 60ps, 380ps, 500ps], [0,945mA, 60ps, 380ps, 500ps] et [1,57mA, 60ps, 380ps, 500ps]. Ceux-ci sont approximés à partir de la double exponentielle (utilisé à bas niveau) pour des charges respectivement de 0,2pC, 0,3pC et 0,5pC.

Tableau 4-3: Comparaison entre l'injection à haut et bas niveau dans le nœud à l'entrée du filtre

Bloc	Charge (pC)	Jitter bas niveau (ps)	Jitter haut niveau (ps)	Durée perturbation bas niveau (ns)	Durée perturbation haut niveau (ns)
FILTRE	0,2	214	188	1200	1400
	0,3	346	285	1200	1500
	0,5	594	480	1200	1600

Comme on peut le remarquer les *jitter* à haut et bas niveau ont le même ordre de grandeur même si ceux à bas niveaux sont un peu plus grands sans doute à cause du fait que le modèle du

pic injecté est beaucoup plus précis (double exponentielle). On remarque aussi que les durées de perturbation à haut niveau font apparaître une certaine progression (on passe de 1400ns à 1600ns) alors que ce n'est pas le cas à bas niveau. De plus pour la même charge, la durée de la perturbation est plus grande à haut niveau bien que le *jitter* soit plus petit.

4.8. Conclusion

Pour conclure on peut dire que les injections dans la PLL d'un courant de faible amplitude produit un effet sur une durée très longue par rapport à la durée de l'injection. Durant cette durée certaines fréquences générées font que le *jitter* de la PLL sort de la plage de tolérance fixée. Cette remarque est valable aussi bien à haut qu'à bas niveau.

Il y a une bonne corrélation entre les *jitters* à bas et haut niveau même si la PLL est beaucoup plus perturbée à haut niveau qu'à bas niveau malgré que les *jitters* soient plus grands.

La précision des résultats des injections sur la description analogique comportementale n'est bien sûr pas très élevée. Cependant, pour les nœuds où cette injection est possible, il apparaît que les principales conclusions sont tout à fait confirmées par les résultats d'injections à bas niveau. L'utilisation d'un flot d'injection mixte numérique/analogique peut donc permettre des analyses utiles très tôt dans le flot de conception d'un circuit.

Par ailleurs, les résultats obtenus sur l'étude de la PLL montrent qu'une faute de durée très brève peut avoir des conséquences sur une durée beaucoup plus importante. Cela peut remettre en cause les hypothèses classiquement utilisées dans les campagnes d'injections de fautes, notamment au niveau de la multiplicité temporelle des erreurs transitoires dans les blocs numériques. Un impact de particule unique sur un bloc PLL peut en effet entraîner une déviation de la fréquence d'horloge interne d'un circuit sur plusieurs centaines de cycles, induisant potentiellement des erreurs en salve pendant toute cette durée et donc une accumulation très rapide d'erreurs pouvant saturer les capacités des mécanismes de détection ou de tolérance implémentés.

Conclusion et perspectives

Le progrès technologique rend les circuits intégrés de plus en plus sensibles vis-à-vis de leur environnement, conduisant à une augmentation des fautes transitoires. Les techniques d'injection de fautes offrent des perspectives prometteuses pour analyser très tôt l'impact de ces fautes sur un circuit numérique. Les techniques basées sur la modification d'une description de haut niveau permettent d'obtenir une évaluation de la sûreté de fonctionnement d'un circuit dès que l'on dispose d'un modèle comportemental de celui-ci.

L'intérêt de tenir compte d'un modèle précis du système global lors d'une analyse du comportement erroné d'un circuit a été démontré. L'approche proposée permet à un concepteur d'évaluer les conséquences réelles des fautes sur l'application avec une bonne précision. Les résultats présentés sont donnés comme exemples de cas d'injections dans lesquels le comportement observé diffère sensiblement du comportement nominal, mais sans conséquence sur l'application.

Des résultats expérimentaux, obtenus à l'issue de campagnes d'injection de fautes simples sur un microcontrôleur 8051, ont montré l'intérêt d'une analyse des caractéristiques de sûreté dès les plus hauts niveaux de description. L'intérêt de combiner la classification et l'analyse de propagation d'erreur au niveau RTL a été également démontré. Les résultats d'injection ont montré que dans la plupart des cas, la combinaison de ces deux analyses permet au concepteur d'identifier des configurations erronées de signaux internes qui sont spécifiques à une classe donnée. Les résultats ont également montré que les états spécifiques dépendaient sensiblement du programme exécuté. Il vaut donc mieux tenir compte de l'application réelle au moment du durcissement d'un circuit à but général comme un microprocesseur.

Les résultats d'injections dans le microprocesseur 8051 ont également permis de montrer que l'injection d'un très faible pourcentage des fautes possibles permet déjà d'obtenir des informations utiles pour le concepteur.

L'évaluation par injection de fautes au niveau RTL d'une technique de détection d'erreurs logicielle a aussi été réalisée. Ces injections sont plus générales que celles basées sur les ISS, et c'est ce qui nous a permis de trouver des situations critiques que les mécanismes implémentés n'ont pas su détecter, et qui n'auraient pas pu être identifiés avec d'autres types d'injections de fautes.

Une dernière et importante conclusion concerne le flot d'analyse unifié pour circuits numériques, analogiques et mixtes. Une PLL décrite à haut niveau d'abstraction a été choisie comme cas d'étude. Cette étude a démontré la faisabilité de fournir un flot unifié pour des blocs numériques et analogiques ainsi que l'intérêt de l'approche en analysant tôt l'effet de fautes transitoires sur le comportement du circuit mixte. Les résultats d'injection à haut niveau ont montré que les injections dans la PLL d'un courant de faible amplitude produit un effet sur une durée très longue par rapport à la durée de l'injection. Durant cette durée certaines fréquences générées font que le jitter de la PLL sort de la plage de tolérance fixée.

Pour valider le flot, des injections de fautes ont également été réalisées au niveau transistor et comparées à celles réalisées à haut niveau. Il apparaît que, pour les nœuds où l'injection est possible, les principales conclusions à haut niveau sont tout à fait confirmées par les résultats d'injections à bas niveau.

- **Perspectives**

Concernant la modélisation du circuit dans son environnement, il serait intéressant de bâtir un nouvel outil utilisant cette approche pour étendre et affiner les possibilités d'analyse de sûreté d'un circuit. De façon complémentaire, un autre objectif serait d'arriver à durcir un circuit de façon sélective sur la base des résultats combinés de la classification et de l'analyse de propagation d'erreurs au niveau RTL. En effet, les résultats d'injection actuels ont démontré l'intérêt de cette combinaison mais ne permettent pas directement de guider le durcissement du circuit.

A terme, une interaction avec un outil de durcissement automatique permettrait de compléter le travail réalisé dès le niveau de description comportemental, en modifiant dans la description du circuit uniquement les parties sensibles identifiées lors de l'analyse. Ces parties sensibles seraient par ailleurs sélectionnées en tenant compte des réactions globales du système, constitué du circuit et de son environnement applicatif.

Il serait également intéressant d'intégrer dans le flot d'analyse un outil permettant de calculer le nombre d'injections nécessaire pour atteindre un certain niveau de confiance statistique. En effet, les résultats fournis dans cette thèse ont montré qu'un petit pourcentage d'injections permet de fournir des informations importantes sans pour autant pouvoir quantifier précisément le nombre d'injections requis.

Le flot unifié pour circuits numériques, analogiques et mixtes a été appliqué sur un convertisseur analogique-numérique modélisé à haut niveau. Cette étude de cas n'a pas été résumée dans le manuscrit car les résultats obtenus n'ont pas permis d'exploitation plus fine que de montrer la possibilité de modification de bits en sortie du convertisseur. Il serait donc intéressant de pouvoir mieux caractériser les types de circuits mixtes pour lesquels l'approche proposée donne des résultats significatifs.

Enfin, un circuit numérique durci a été développé dans le cadre du projet RNRT Duracell. Le circuit conçu a été fabriqué et des expériences d'injection de fautes par laser ont été réalisées chez Gemplus. Les résultats de ces expériences restent à dépouiller. Ils permettront d'affiner les modèles de fautes utilisés pour les expériences d'injection, au moins dans un contexte d'attaques sur des circuits sécurisés.

Références bibliographiques

[A.A. 92] A. Antola, R. Negrini, M. Sami and N. Scarabottolo, "Tolerance to transient faults in microprogrammed control units", IEEE Transactions on Reliability, vol. 27, 1992, pp. 258–264.

[A.A. 04] A. Avizienis, J. C. Laprie, B. Randell, "Dependability and its Threats: A Taxonomy", in Proceedings of the Building the Information Society: Proc. IFIP 18th World Computer Congress, Toulouse, France Jacquart, 22-27 August, 2004, R (ed) pp. 91-120.

[A.C. 99] A. Cataldo, "Intel Scans for Soft Errors in Processor Designs", EETimes, 06/16/99.

[A.D. 81] A. Dantec, "Le phénomène de latchup dans les circuits integers CMOS", Toute l'électronique, TLE, 1981, 465, p.45.

[A.M] ADVance MS User's Manual Software Version 3.0_1 Release 2003.3.

[A.M. 02] Adrian Maxim, "Low-Voltage CMOS Charge-Pump PLL Architecture for Low Jitter Operation", ESSCIRC 2002, pp. 423-426.

[B.P. 00] B. Parrotta, M. Rebaudengo, M. Sonza-Reorda, and M.Violante, "New Techniques for Accelerating Fault Injection in VHDL Descriptions", in 6th IEEE Int. On-Line Testing Workshop, Palma de Mallorca, Spain, July 3–5, 2000, pp. 61–66.

[B.N. 04] B. Nicolescu, Y. Savaria, R. Velazco, "Software Detection Mechanisms Providing Full Coverage Against Single Bit-flip Faults", IEEE Transaction on Nuclear Science, Vol. 51, No. 6, December 2004, pp 3510-3518.

[C.O. 05] Celia Lopez-Ongil, Mario Garcia-Valderas, Marta Portela-Garcia, Luis Entrena-Arrontes, "Autonomous Transient Fault Emulation on FPGAs for Accelerating Fault Grading", 11th IEEE International On-Line Testing Symposium, 2005, pp. 43-48.

[C.P. 97] Chen-Yang Pan, Kwang-Ting Cheng, "Fault Macromodeling for Analog/Mixed-Signal Circuits", Proceedings IEEE International Test Conference 1997, Washington, DC, USA, November 3-5, 1997, IEEE Computer Society Press, 1997, pp. 913-922.

[C.R. a] "Cosmic Rays", http://helios.gsfc.nasa.gov/cosmic.html.

[C.R. b] "Cosmic Rays", http://www.oulu.fi/~spaceweb/textbook/crays.html.

[D.B] Dean Banerjee, "PLL Performance, Simulation, and Design 4th edition", http://www.national.com/appinfo/wireless/files/deansbook4.pdf.

[D.C. 04] David Cimonnet, "Utilisation de LabView pour l'analyse de sûreté d'un circuit complexe", Stage de DEA INPG de mars à juin 2004.

[D.S] Dallas Semiconductor, "Clock (CLK) Jitter and Phase Noise Conversion", Application Note 3359, http://www.maxim-ic.com/appnotes.cfm/appnote_number/3359.

[D.S. 02] Debapriya Sahu, "A Completely Integrated Low Jitter CMOS PLL for Analog Front Ends in Systems on Chip Environment", 7th Asia and South Pacific Design Automation Conference/15th International Conference on VLSI Design, Bangalore, India, 2002, pp. 360-365.

[E.B. 99] E. B¨ohl, W. Harter, and M. Trunzer, "Real Time Effect Testing of Processor Faults", in 5th IEEE International On-Line Testing Workshop, Rhodes, Greece, July 5–7, 1999, pp. 39–43.

[E.J. 94] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, "Fault injection into VHDL models: the MEFISTO tool", in 24th Symposium on FTCS, June1994, pp. 66-75.

[E.N. a] Eugene Normand "Single Event Effects in Avionics", Boeing Defense & Space Group Seattle, WA 98124-2499.

[E.N. b] Eugene Normand "Single Event Upset at Ground Level", Boeing Defense & Space Group, Seattle, WA 98124-2499.

[F.C. 03] F. Corno, S. Tosato, P. Gabrielli, "System-level analysis of fault effects in an automotive environment", The 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, Boston, Massachusetts, USA, November 3-5, 2003, IEEE Computer Society Press, Los Alamitos, California, 2003, pp. 529-536.

[F.Y. 92] F. L. Yang and R. A. Saleh, "Simulation and Analysis of Transient Faults in Digital Circuits", IEEE Journal of Solid-State Circuits, vol. 27, 1992, pp. 258–264.

[G.C. 02] G. C. Cardarilli et al., "Bit-flip injection in processor-based architectures: a case study", 8th IEEE International On-Line Testing workshop, Isle of Bendor, France, July 8-10, 2002, pp. 117-127.

[G.M. 82] G. C. Messenger, "Collection of charge on junction nodes from ion tracks", IEEE Transactions on Nuclear Science, 1982, pp. 2024–2031.

[H.C. 96] H. Cha, E. M. Rudnick, J. H. Patel, R. K. Iyer and G. S. Choi, "A gate level simulation environment for alpha-particle-induced transient faults", IEEE Transactions on Computers, vol. 45, 1996, pp. 1248-1256.

[H.L. 97] Howard C. LUONG, "Monolithic Integration of a CMOS single-chip Wireless Receiver", Final Year Project Report 96/97, The Hong Kong University of Science and Technology Department of Electrical and Electronic Engineering, "http://www.ee.ust.hk/~issachsu/FYP/lh3.html".

[H.S] Les HDL simulation : principe "http://www.comelec.enst.fr/hdl/hdl_simulation.html".

[I.M. 94] I. Mouret, M. Allenspach, R.D. Schrimpf, J.R. Brews, K.F. Galloway, P. Calvel, "Temperature and angular dependence of substrate response in SEGR", IEEE Trans. on Nuclear Science, vol. 41, no. 6, pp. 2216-2221, 1994.

[I.T] http://public.itrs.net.

[J.A. 98] J. Abke, E. B¨ohl, and C. Henno, "Emulation Based Real Time Testing of Automotive Applications", in 4th IEEE International On-Line Testing Workshop, Capri, Italy, July 6–8, 1998, pp. 28–31.

[J.B. 98] J. Boué, P. Pétillon, and Y. Crouzet, "MEFISTO-L: A VHDL Based Fault Injection Tool for the Experimental Assessment of Fault Tolerance", in 28th Symposium on Fault-Tolerant Computing (FTCS), 1998, pp. 168–173.

[J.G. 01] J. Gracia, J.C. Baraza, D. Gil, and P.J. Gil, "Comparison and Application of Different VHDL-Based Fault Injection Techniques", in The IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems, San Francisco, California, USA, Oct. 24–26, 2001, IEEE Computer Society Press, 2001, pp. 233–241.

[J.K. 91] J. Karlsson, U. Gunneflo, P. Lid´en, and J. Torin, "Two Fault Injection Techniques for Test of Fault Handling Mechanisms", in International Test Conference (ITC), 1991, pp. 140–149.

[J.K. 05] Jaehong Ko, Wookwan Lee, Soo-Won Kim, "2.5GHz PLL with current matching charge-pump for 10Gbps transmitter design", ACM Great Lakes Symposium on VLSI 2005, pp. 122-125.

[J.L. 88] Jean-Claude Laprie, "Sûreté de fonctionnement et tolérance aux fautes: Concepts de base", Rapport LAAS nº 88.287, Novembre 1988.

[J.M. 96] J. G. Maneatis, "Low-jitter process-independent DLL and PLL based on self-biased techniques", IEEE JSSC, vol. 31, Nov 1996, pp. 1723-1732.

[J.S. 97] J. R. Samson,W. Moreno, and F. Falquez, "Validating Fault Tolerance Designs Using Laser Fault Injection", in The IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, Paris, France, Oct. 20–22, 1997, Los Alamitos, CA: IEEE Computer Society Press, 1997, pp. 175–183.

[J.T. 04] J. N. Tombs, F. Muñoz, V. Baena-Lecuyer, A. Torralba, L.G. Franquelo, A. Fernández-León, F. Tortosa-López, D. Gutiérrez-González, "A Hardware Approach for Seu Immunity Verification Using Xilinx Fpga's", Proc. 19th Conference on Design of Circuits and Integrated Systems, DCIS 2004. Bordeaux, France. 2004. pp. 479-484.

[J.W. 62] J. T. Wallmark, S.M. Marcus, "Minimum size and maximum packaging density of non-redundant semiconductor devices", Proc. IRE, vol. 50, pp. 286-298, March 1962.

[J.Z. 79] J.F. Ziegler and W. A. Lanford : "Effect of Cosmic Rays on Computer Memories", Science, vol. 206, p.776, 1979.

[J.Z. 04] J. Ziegler, et al., "SER – History, trends, and challenges", Cypress 2004.

[K.C. 99] K. -T. Cheng, S. -Y. Huang, and W.-J. Dai, "Fault Emulation: A New Methodology for Fault Grading", IEEE Transactions on Computer-Aided Design, vol. 18, no. 10, 1999, pp. 1487–1495.

[K.E] http://www.keil.com/

[K.H. 05] Karim Hadjiat, "Evaluation prédictive de la sûreté de fonctionnement d'un circuit intégré numérique", Thèse de l'Institut National Polytechnique de Grenoble (INPG), spécialité Microélectronique, soutenue le 10 juin 2005 à Grenoble.

[K.L. 97] K. LaBel, P. W. Marshall, C. J. Marshall M. D'Ordine, M. Carts, G. Lum, H. S. Kim, C. M. Seidlick, T. Powell, R. Abbot, J. Barth and E. Stasinopoulos, "Proton-Induced Transients in Optocouplers: In-Flight Anomalies, Ground Irradiation Test, Mitigation and Implications", IEEE Transactions on Nuclear Science, Volume 44, NO 6, December 1997, pp.1885-1892.

[L.A. 00] L. Antoni, R. Leveugle, and B. Fehér, "Using Run-Time Reconfiguration for Fault Injection in Hardware Prototypes", in The IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems, Yamanashi, Japan, Oct. 25–27, 2000, IEEE Computer Society Press, 2000, pp. 405–413.

[L.A. 02] L. Antoni, R. Leveugle, and B. Fehér, "Using Run-Time Reconfiguration for Fault Injection in Hardware Prototypes", The 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, Vancouver, Canada, Nov. 6–8, 2002, Los Alamitos, CA: IEEE Computer Society Press, 2002, pp. 242–249.

[L.B. 02] L. Berrojo, I. Gonzalez, F. Corno, M. Sonza-Reorda, G. Squillero, L. Entrena, and C. Lopez, "New Techniques for Speeding Up Fault-Injection Campaigns", in Design Automation and Test in Europe Conference (DATE), March 4–8, 2002, pp. 847–852.

[L.D. 00] L. Dai and R. Harjani, "Comparison and Analysis of Phase Noise in Ring Oscillators", Proceedings of 2000 the Intl Symp on Circuits and Systems, May 2000, pp. V-77 – V-80.

[L.M] The MathWorks – Link for ModelSim – Cosimulate and verify VHDL and Verilog using ModelSim, http://www.mathworks.com/products/modelsim/.

[L.N] "Loi normale", http://fr.wikipedia.org/wiki/Loi_normale.

[M.B. 97] M. BAZE, S. BUCHNER, "Attenuation of Single Event Induced Pulses in CMOS Combinational Logic", IEEE Trans. on Nuclear Science, Vol. 44, No 6, December 1997.

[M.H] Mei-Chen Hsueh, Timothy K. Tsai, and Ravishankar K. Iyer "Fault injection techniques and tools", http://swig.stanford.edu/~candea/teaching/cs444a-fall-2003/readings/hsueh-injection.pdf.

[M.N. 99] M. Nicolaidis, "Time Redundancy Based Soft-Error Tolerance to Rescue Nanometer Technologies", in proceedings of the 17th IEEE VLSI Test Symposium, April 26-30, 1999, p.86.

[M.P] Maxim Integrated Products, "Converting between RMS and Peak-to-Peak Jitter at a Specified BER", Application Note: HFAN-4.0.2, http://pdfserv.maxim-ic.com/arpdf/AppNotes/ 3hfan402.pdf.

[M.S. 01] M. Singh, I. Koren, "Reliability enhancement of analog-to-digital converters (ADCs)", IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems, San Francisco, California, USA, October 24-26, 2001, IEEE Computer Society Press, 2001, pp. 347-353.

[M.S. 03] M. Singh, I. Koren, "Fault Sensitivity Analysis and Reliability Enhancement of Analog-to-Digital Converters", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume 11, Issue 5 October 2003, pp. 839-852.

[M.V] Modèle VHDL synthétisable du 8051, http://www.8051.free.fr.

[N.S] Nelson Soo, "Jitter Measurement Techniques", Application Brief AB36, http://www.pericom.com/pdf/applications/AB036.pdf.

[O.A. 84] O. C. Allkofer and P. K. Grieder, Physics Data: Cosmic Rays on Earth, Fachinformationszentrum Energie, Physik, Mathematik GmbH, Karlsruhe, 1984.

[P.C. 01-a] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, and M. Violante, "Exploiting FPGA for Accelerating Fault Injection Experiments", in 7th IEEE International On-Line Testing Workshop, Taormina, Italy, July 9–11, 2001, pp. 9–13.

[P.C. 01-b] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, and A. Violante, "Exploiting FPGA-Based Techniques for Fault Injection Campaigns on VLSI Circuits", in The IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, San Francisco, California, USA, Oct. 24–26, 2001, Los Alamitos, CA: IEEE Computer Society Press, 2001, pp. 250–258.

[P.C. 01-c] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, and A. Violante, "FPGA-Based Fault Injection for Microprocessor Systems", in Asian Test Symposium,Nov. 2001, pp. 304–309.

[P.W. 02] P. R. Wilson, Y. Kilic, J. N. Ross, M. Zwolinski, A. D. Brown, "Behavioural modeling of operational amplifier faults using VHDL-AMS", Design, Automation and Test in Europe Conference (DATE), March 4-8, 2002, pp. 1133.

[R.B. 97] R. J. Betancourt, A. Hajimiri and T. H. Lee, "A 1.5mW, 200MHz CMOS VCO for Wireless Biotelemetry", IEEE Workshop on Design of Mixed-Mode Integrated Circuits and Applications, July 1997.

[R.B. 02] R. Baumann, "Soft Errors in Commercial Semiconductor Technology: Overview and Scaling Trends", IEEE 2002 Reliability Physics Symp. Tutorial Notes, Reliability Fundamentals, IEEE Press, 2002, pp. 121-01.1—121-01.14.

[R.E. 94] R. Ecoffet, S. Duzellier, P.Tastet, C. Aicardi and M. Labrunee, "Observation of Heavy Ion Induced Transients in Linear Circuits", IEEE Radiation Effects Data workshop, December 1994, pp. 72-77.

[R.L. 98] R. Leveugle, "Behavior Modeling of Faulty Complex VLSIs: Why and How?", in The Baltic Electronics Conference, Tallinn, Estonia, Oct. 7–9, 1998, pp. 191–194.

[R.L. 99] R. Leveugle, "Towards modelling for dependability of complex integrated circuits", 5th IEEE International On-Line Testing workshop, Rhodes, Greece, July 5-7, 1999, pp. 194-198.

[R.L. 00] R. Leveugle and K. Hadjiat, "Optimized generation of vhdl mutants for injection of transition errors", in 13th Symposium on Integrated Circuits and Systems Design (SBCCI2000), Manaus, Brazil, Sept. 18-24, 2000, pp. 243-248.

[R.L. 01] R. Leveugle, "A Low-Cost Hardware Approach to Dependability Validation of IPs", in The IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, San Francisco, California, USA, Oct. 24–26, 2001, Los Alamitos, CA: IEEE Computer Society Press, 2001, pp. 242–249.

[R.L. 02] R. Leveugle, K. Hadjiat, "Multi-level fault injection experiments based on VHDL descriptions: a case study", 8th IEEE Int. On-Line Testing workshop, Isle of Bendor, France, July 8-10, 2002, pp. 107-111.

[R.L. 03] R. Leveugle, K. Hadjiat, "Multi-level fault injections in VHDL descriptions: alternative approaches and experiments", Journal of Electronic Testing: Theory and Application (JETTA), Vol. 19, No. 5, October 2003, pp. 559-575.

[R.M] R. Marawar, G. Edelmon, C. Nybro, S. Duncan, C. Britton, "Combining the use of Multisim and LabVIEW for Circuit Design and Analysis in Undergraduate Electronics Curriculum", http://www.electronicsworkbench.com/pdf/msmlabvw.pdf

[R.M. 96] R. A. Mewaldt, "Cosmic Rays", http://www.srl.caltech.edu/personnel/dick/cos_encyc .html.

[R.W. 95] R. W. Wieler, Z. Zhang, and R.D. McLeod, "Emulating Static Faults Using a Xilinx Based Emulator", in IEEE Symp. FPGAs for Custom Computing Machines, Feb. 1995, pp. 110–115.

[S.B. 00] Stephen P. Buchner, Timothy J. Meehan, Arthur B. Campbell, Kenny A. Clark, and Dale McMorrow, "Characterization of Single-Event Upsets in a Flash Analog-to-Digital Converter (AD9058)", IEEE Transactions on Nuclear Science, Volume 47, NO. 6, December 2000, pp. 2358-2364.

[S.E] Single Event Effects, "http://www.eas.asu.edu/~holbert/eee460/see.html".

[S.K] Sammy Kayali, "Space Radiation Effects on Microelectronics", NASA Jet Propulsion Laboratory, http://parts.jpl.nasa.gov/docs/Radcrs_Final.pdf.

[S.K. 86] S. Kang and D. Chu, "CMOS circuit design for the prevention of single event upset", in Proceedings of IEEE International Conference on Computer Design, Oct 1986, pp. 385–388.

[S.S. 97] S. Svensson and J. Karlsson, "Dependability Evaluation of the THOR Microprocessor Using Simulation-Based Fault Injection", Technical Report No. 295, Chalmers University of Technology, Department of Computer Engineering, Sweden, November 1997.

[S.S. 02] S. Sinha, "Design of an integrated CMOS PLL frequency synthesizer" Proceedings of the IEEE MELECON'02, Cairo, Egypt, pp. 220-224.

[S.S. 05] Susi Saleh, "Méthodes de simulation des erreurs transitoires à plusieurs niveaux d'abstraction", Thèse de l'Institut National Polytechnique de Grenoble (INPG), spécialité Microélectronique, soutenue le 21 juin 2005 à Grenoble.

[T.D. 96] T.A Delong, B.W. Johnson, and J.A. Profeta III, "A Fault Injection Technique for VHDL Behavioral-Level Models", IEEE Design & Test of Computers, vol. 13, Winter 1996, pp. 24–33.

[T.M. 79] T.C. May, M.H. Woods, "Alpha-particle-induced soft errors in dynamic memories", IEEE Trans. on Electron Devices, vol. ED-26, no. 1, pp. 2-9, Jan. 1979.

[T.M. 89] T. P. Ma and P. V. Dressendorfer, Eds., "Ionising Radiation Effects in MOS devices and circuits", New York : Wiley 1989.

[T.M. 94] T. Michel, R. Leveugle, G. Saucier, R. Doucet, and P. Chapier, "Taking Advantage of ASICs to Improve Dependability with Very Low Overheads", in ED&TC, 1994, pp. 14–18.

[T.T. 96] Thomas L. Turflinger, "Single-Event Effects in Analog and Mixed-signal Integrated Circuits", IEEE Transactions on Nuclear Science, Volume 43, NO. 2, April 1996, pp. 594-602.

[U.G. 89] U. Gunneflo, J. Karlsson, and J. Torin, "Evaluation of Error Detection Schemes Using Fault Injection by Heavy-Ion Radiation", in 19th Symposium on Fault-Tolerant Computing (FTCS), 1989, pp. 340–347.

[V.S] Virtuoso Spectre Circuit Simulator, "http://www.cadence.com/products/custom_ic/spectre/index.aspx".

[W.K. 79] W. A. Kolasinski, J.B. Blake, J.K. Anthony, W.E. Price, E.C. Smith, "Simulation of cosmic-ray induced soft errors and latchup in integrated-circuit computer memories", IEEE Trans. on Nuclear Science, vol. NS-26, no. 6, pp. 5087-5091, 1979.

[W.W] Wireless Webench, "http://webench.national.com/appinfo/wireless/webench/index.cgi".

[Y.H] Yannick Hervé "VHDL-AMS : Applications et enjeux industriels" Dunod-Université - collection : Sciences-sup - préface d'Alain Vachoux.

[Z.Z. 01] Zhaofeng Zhang, Louis Tsui, Zhiheng Chen, Jack Lau, "A CMOS Self-Mixing-Free Front-End for Direct-Conversion Applications", The 2001 IEEE International Symposium on Circuits and Systems ISCAS2001, vol. 4, May 2001, Sydney, pp. 386-389.

Publications

Revues internationales

[A.A. 05] A. Ammari, K. Hadjiat, R. Leveugle, "Combined Fault Classification and Error Propagation Analysis to Refine RT-Level Dependability Evaluation", Journal of Electronic Testing: Theory and Application (JETTA), 2005, pp. 365-375.

Conférences internationales et Workshops

[A.A. 03] A. Ammari, R. Leveugle, M. Sonza-Reorda, M. Violante, "Detailed comparison of dependability analyses performed at RT and gate levels", 18th International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 03), Cambridge, MA, USA, November 3-5, 2003, pp. 336-343.

[R.L. 04-a] R. Leveugle, A. Ammari, "Early SEU Fault Injection in Digital, Analog and Mixed Signal Circuits: A Global Flow", Design Automation and Test in Europe (DATE 04), Paris, February 16-20, 2004, pp. 590-595.

[A.A. 04] A. Ammari, K. Hadjiat, R. Leveugle, "On Combining Fault Classification and Error Propagation Analysis in RT-Level Dependability Evaluation", 10th IEEE International On-Line Testing Symposium (IOLTS 04), Madeira Island, Portugal, July 12-14, 2004, pp. 227-232.

[R.L. 04-b] R. Leveugle, D. Cimonnet, A. Ammari "System Level Dependability Analysis with RT-Level Fault Injection accuracy", 19th International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 04), Cannes, October 11-13, 2004, pp. 451-458.

[K.H. 05-a] K. Hadjiat, A. Ammari, R. Leveugle "Injection of Multiple Bit-Flips for Counter Measures Validation", Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), Edinburgh, Scotland, September 2nd, 2005.

[K.H. 05-b] K. Hadjiat, A. Ammari, R. Leveugle "Early Functional Evaluation of SET Effects (Extended Abstract)", 8th European Conference on Radiation and Its Effects on Components And Systems (RADECS 05), Palais Des Congrés Du Cap D'agde, France, September 19-23 2005.

[A.A. 06] A. Ammari, R. Leveugle, B. Nicolescu, Y. Savaria, "Evaluation of a Software-Based Error Detection Technique by RT-Level Fault Injection," delta, Third IEEE International Workshop on Electronic Design, Test and Applications (DELTA'06), 2006, pp. 488-493.

Conférences nationales

[A.A. 04-c] A. Ammari, R. Leveugle, "Injection de fautes dans une PLL", Journées Nationales du Réseau Doctoral en Microélectronique (JNRDM), Marseille, France, Mai 4-6, 2004, pp. 295-297.

[K.H. 04] K. Hadjiat, A. Ammari, R. Leveugle, "Application et Combinaison de Deux Approches d'Analyse de Sûreté", Journées Nationales du Réseau Doctoral en Microélectronique (JNRDM), Marseille, France, Mai 4-6, 2004, pp. 410-412.

[K.H. 05-c] K. Hadjiat, A. Ammari, R. Leveugle, "Modélisation de fautes et génération de mutants pour analyse de robustesse de circuits sécurisés", Journées Nationales du Réseau Doctoral en Microélectronique (JNRDM), Paris, France, Mai 10-12, 2005, pp. 472-474.

Annexe A.

Description et paramétrage des blocs de la PLL décrite en langage de haut niveau

A.1. Le détecteur de phase-fréquence

Son schéma de principe est donné à la Figure A-1.

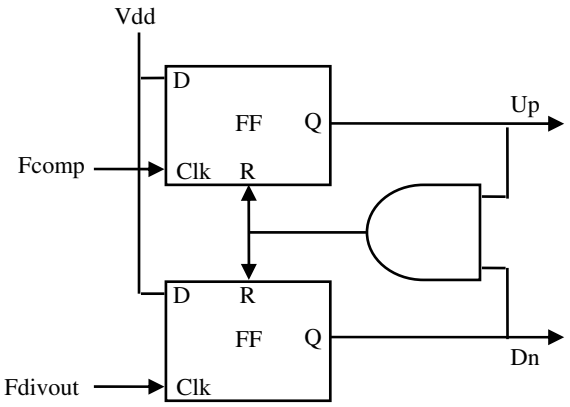


Figure A-1: Schéma de principe d'un PFD

Le PFD comporte 3 états, -1, 1 et 0. Ces états permettent respectivement à la pompe de charge de fournir du courant, d'absorber du courant ou de se placer en haute impédance.

Après avoir instancié ce modèle, dont le nom est MGC_CommLib.pfd_dz(a1), les paramètres génériques utilisés sont résumés dans le Tableau A-1.

Tableau A-1: Paramétrage du PFD

NOM	TYPE	UNITE	VALEUR	DESCRIPTION
tp	TIME	s	0,2ns	Délai de propagation
tdz	TIME	s	0,1ns	Délai de la *dead zone*[1]
edge	BIT		'1'	Type du front d'horloge : '0' descendant, '1' montant
state_init	INTEGER		0	Etat initial du PFD : -1, 0, 1

A.2. La pompe de charge

Les signaux de sortie UP et DN du PFD sont utilisés pour contrôler deux interrupteurs. Suivant les valeurs de ces signaux, et donc l'état du PFD, la pompe de charge fournit du courant (état 1 du PFD), absorbe du courant (état -1 du PFD) à travers un "terminal", ou bien se trouve dans l'état haute impédance (état 0 du PFD). Le modèle (Figure A-2) qui porte le nom de MGC_CommLib.charge_pump_ub(a1) suppose que la pompe de charge est constituée de deux transistors, un pMOS et un nMOS, dont les signaux d'entrée des grilles sont respectivement NOT(UP) et DN. Les drains des deux transistors sont reliés au même "terminal" de sortie aout.

[1] La "*dead zone*" représente une région où le PFD est insensible aux très petites erreurs de phase. La cause principale vient de la relation entre le délai de propagation du *reset* des bascules D (donc au niveau de la porte ET dans le cas de la Figure A-1) et le temps de commutation de la pompe de charge. Pour éviter cette zone, il faut faire en sorte que tdz soit supérieur au temps de commutation.

La source du transistor pMOS est connectée au niveau logique haut (1,8V), tandis que la source du transistor nMOS est connectée au niveau logique bas (0V).

Quand un transistor est OFF, il est modélisé avec une résistance rOFF. Quand il passe à l'état passant, il est modélisé comme une source de courant qui atteint linéairement, au bout d'un temps trise, la valeur Ion ou Iop. Une fois que la tension drain-source (source-drain pour le transistor pMOS) chute au dessous d'un certain niveau, la source de courant est remplacée par une résistance (rONn pour le nMOS et rONp pour le pMOS) qui correspond au passage de la région saturée vers la région ohmique.

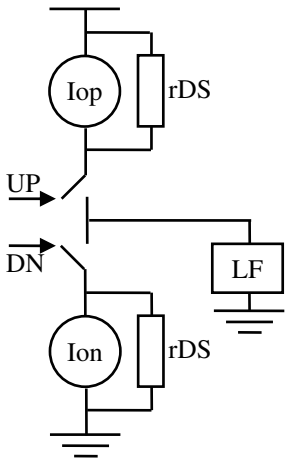


Figure A-2: Modèle de la pompe de charge avec transistors en saturation

Ce modèle prend en considération le fait que la source de courant produite par le transistor pMOS en saturation (Iop) peut ne pas être égale à celle du transistor nMOS (Ion).

Tableau A-2: Paramétrage de la pompe de charge

NOM	TYPE	UNITE	VALEUR	DESCRIPTION
trisen	REAL	s	$200,0E^{-12}$	Temps de montée de la source de courant du transistor N
trisep	REAL	s	$200,0E^{-12}$	Temps de montée de la source de courant du transistor P
Vhi	REAL	V	1,8	Tension correspondant au niveau logique haut
Vlo	REAL	V	0,0	Tension correspondant au niveau logique bas
Ion	REAL	A	$100,0E^{-6}$	Courant produit par le transistor N
Iop	REAL	A	$100,0E^{-6}$	Courant produit par le transistor P
rONn,p	REAL	Ω	100,0	Résistance modélisant le transistor N,P dans la région ohmique
rOFF	REAL	Ω	$1,0E^{9}$	Résistance modélisant les transistors dans la région bloquée
rDSn,p	REAL	Ω	$200,0E^{3}$	Résistance de saturation parallèle à la source de courant du transistor N,P

A.3. Le VCO

Le modèle qui porte le nom de MGC_CommLib.VCO_sqr_d(a1) produit à sa sortie un signal carré de fréquence variable. Cette fréquence est limitée par deux valeurs correspondant à fmin et fmax. Les signaux d'entrée et de sortie ne sont pas des signaux différentiels. La fréquence de sortie est définie par l'Equation A-1.

Equation A-1: $$Fout = \left(Kv \times (Vin - V\_f0) + Kvv \times (Vin - V\_f0)^2 + Kvvv \times (Vin - V\_f0)^3 + f0\right)$$

Dans notre cas, le VCO est considéré comme étant linéaire donc Kvv et Kvvv sont nuls. Les autres paramètres à introduire sont résumés dans le Tableau A-3.

Tableau A-3: Paramétrage du VCO

NOM	TYPE	UNITE	VALEUR	DESCRIPTION
f0	REAL	Hz	$275,0E^{6}$	Fréquence centrale
fmin	REAL	Hz	$50,0E^{6}$	Fréquence minimale
fmax	REAL	Hz	$275,0E^{6}$	Fréquence maximale
V_f0	REAL	V	0,0	Tension d'entrée correspondant à la fréquence centrale
V_fmax	REAL	V	1,0	Tension d'entrée correspondant à la fréquence min
V_fmin	REAL	V	$-1,0E^{-9}$	Tension d'entrée correspondant à la fréquence max
Kv	REAL	Hz/V	$-225,0E^{6}$	Gain linéaire du VCO
Kvv	REAL	Hz/V^2	0,0	Gain quadratique du VCO
Kvvv	REAL	Hz/V^3	0,0	Gain cubique du VCO
Rin	REAL	Ω	$1,0E^{9}$	Résistance parallèle d'entrée
Cin	REAL	F	$200,0E^{-15}$	Capacité Parallèle d'entrée

La Figure A-3 représente la courbe donnant la fréquence de sortie en fonction de la tension d'entrée.

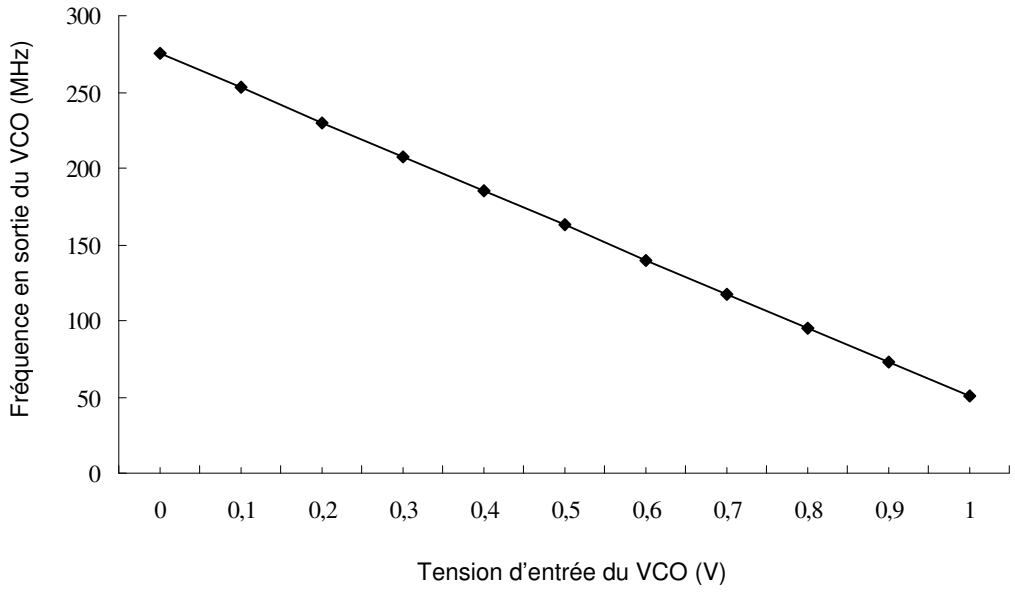


Figure A-3: Caractéristique du VCO

A.4. Le diviseur

Le diviseur est décrit par un modèle purement numérique. Le signal en sortie a une fréquence N fois plus petite que celle de l'entrée. Le rapport cyclique n'est pas de 50% (sauf pour N=2), mais le signal de sortie passe à '1' ou à '0' pendant une durée correspondant à la période du signal d'entrée. Le Tableau A-4 donne le paramétrage du diviseur.

Tableau A-4: Paramétrage du diviseur

NOM	TYPE	UNITE	VALEUR	DESCRIPTION
tPLH	TIME	s	0,1ns	Délai de propagation d'un état bas vers haut
tPHL	TIME	s	0,1ns	Délai de propagation d'un état haut vers bas
N	INTEGER		8	Facteur de division : rapport entre Fout et Fcomp
edge	BIT		'1'	Type du front d'horloge : '0' descendant, '1' montant
val_c	STD_LOGIC		'0'	Valeur du signal de sortie tout le long du comptage

A.5. Le filtre de boucle

Le filtre de boucle est sans doute l'élément qui influe le plus sur les performances de la PLL. Il est utilisé pour : (1) assurer les performances de l'asservissement (bande passante, amortissement, …) et (2) atténuer le plus possible les produits indésirables issus de l'étage précédent. Les paramètres du filtre de boucle sont les suivants :

- **Largeur de bande en boucle ouverte (Fc):** C'est la fréquence à laquelle la fonction de transfert en boucle ouverte est égale à 1 ou 0dB. Cette largeur de bande est inversement proportionnelle au temps d'accrochage.
- **Marge de phase (Φ):** C'est 180° diminué de la phase de la fonction de transfert en boucle ouverte évaluée à la fréquence Fc. Ce paramètre est typiquement choisi entre 40° et 55°. Les simulations montrent que 48° donne un temps d'accrochage optimal. Une marge de phase assez élevée diminue le pic de la réponse transitoire au détriment du temps d'accrochage. Pour une erreur de phase minimum, 50° est un bon point de départ pour la marge de phase.
- **Spur gain:** Le *spur gain* est la valeur de la fonction de transfert en boucle fermée à la fréquence de comparaison Fcomp. Le *spur gain* donne une indication relative (non absolue), du niveau de certains effets indésirables (*spurs* en anglais). Pour les PLL à diviseur entier, les *spurs* se produisent à la fréquence de sortie ± la fréquence de comparaison (ce type d'imperfection est le plus important et se nomme *reference spur*) ou bien la fréquence de sortie ± les multiples de la fréquence de comparaison. Ces *spurs* sont dus à deux phénomènes : (1) le courant de fuite lorsque la pompe de charge se trouve dans l'état haute impédance, (2) la disparité da la pompe de charge.
- **Fréquence naturelle (ωn):** Pour les systèmes du second ordre, la fréquence naturelle est la fréquence d'oscillation de la réponse transitoire. A noter que ceci s'applique pour les systèmes du second ordre, et dans notre cas, la fonction de transfert en boucle fermée de la PLL est un système au minimum du 3ème ordre (car le filtre est au minimum du 2ème ordre). Ceci veut dire que ce paramètre n'est qu'une approximation.
- **Facteur d'amortissement (ξ):** Pour les systèmes du second ordre, le facteur d'amortissement détermine la forme de l'enveloppe exponentielle qui multiplie ωn. Dans notre cas ce paramètre n'est qu'une approximation. A noter que ξ et ωn peuvent être approximés relativement à Fc et Φ.

- **Facteur d'optimisation gamma (γ):** Il est possible de concevoir deux filtres de même ordre avec les mêmes Fc et Φ, mais l'un aura un meilleur temps d'accrochage et une meilleure atténuation des *spurs* que l'autre. La différence réside dans le choix de γ. Cela dit généralement il est pris à 1.
- **Effets discrets d'échantillonnage:** Ceux-ci se produisent quand la précision est perdue en approximant les corrections du courant discret de la pompe de charge, avec sa moyenne dans le temps. Cette approximation est appelée approximation en temps continu. Pour garder une bonne précision, il faut faire en sorte que Fc soit au minimum le 1/5ème de Fcomp. Au dessus de cette valeur, la PLL peut devenir instable (c'est le cas si le rapport est supérieur à 1/3). Cela dit pour garder une bonne précision, il faut faire en sorte que ce rapport soit au moins 1/10 dans la mesure du possible.
- **L'ordre:** Différents ordres peuvent être utilisés pourvu qu'on puisse en tirer un réel bénéfice. Ainsi l'utilisation d'un ordre plus élevé permet de diminuer encore plus le niveau du *spur*, mais génère d'autres problèmes comme :
  - Augmentation du bruit des résistances (car il y a plus de résistances).
  - Augmentation du nombre d'éléments passifs et donc de la surface.
  - Les capacités d'ordre plus élevé tendent à devenir très petite à telle point que le circuit peut être affecté par la capacité d'entrée du VCO.
  - Les filtres d'ordre supérieur à 2 peuvent aboutir à des systèmes instables (on suppose que l'approximation en temps continu est respectée).

  Comme on peut le voir il y a plus d'inconvénients que d'avantages à utiliser un filtre d'ordre plus élevé. De ce fait, investir dans un filtre d'ordre plus élevé n'est bénéfique que s'il contribue à atténuer d'au moins 2dB le niveau du *spur*. Un principe de base approximatif dit que le filtre de 3ème ordre peut diminuer le niveau du *spur* si déjà Fcomp est au moins 10 fois plus grand que Fc. De même un filtre du 4ème ordre devient intéressant si Fcomp est supérieure à 20 fois Fc.
- **La stabilité du système:** D'après le critère algébrique de Routh [D.B], seul le zéro (noté T2) et les pôles (T1 pour filtre de 2ème ordre, T3 pour 3ème ordre, T4 pour 4ème ordre, ….) du filtre déterminent la stabilité du système. Ainsi pour un filtre du second ordre (Figure A-4 (a)) pour que le système soit stable il faut satisfaire l'Equation A-2 :

  Equation A-2: $T2 > T1$

  En remplaçant avec les élément du filtre on obtient l'Equation A-3.

  Equation A-3: $$R2 \times C2 > \frac{R2 \times C2 \times C1}{C1 + C2}$$

  Ceci est toujours le cas. Donc La PLL qui utilise un filtre du 2ème ordre est toujours stable (pourvu qu'on respecte l'approximation en temps continu et le fait que C1 >> Cvcoin capacité à l'entrée du VCO).

Pour les filtres du 3ème ordre (Figure A-4 (b)), la condition de stabilité est donnée par l'Equation A-4:

Equation A-4: $T2 > T1 + T3$

En remplaçant avec les éléments du filtre on obtient l'Equation A-5 dont le terme de droite n'est qu'une approximation.

Equation A-5: $R2 \times C2 > \frac{R2 \times C2 \times C1}{C1 + C2 + C3} + R3 \times C3$

La stabilité est d'autant plus importante que le gain du VCO, le gain KΦ de la pompe de charge ou la valeur N du diviseur peuvent varier énormément (en fait le calcul des éléments passifs est optimisé pour une seule fréquence).

- **Le type:** Deux types de filtre peuvent être utilisés : les filtres passifs et les filtres actifs. Le filtre passif est généralement recommandé pourvu que la pompe de charge puisse fournir la tension requise pour piloter le VCO. Un filtre passif du 2ème ordre est donné à la Figure A-4 (a). Pour obtenir un filtre du 3ème ordre, il suffit de rajouter une résistance en série et un condensateur en parallèle (Figure A-4 (b)). Les filtres actifs quant à eux sont utilisés si le VCO requiert des niveaux de tension supérieurs à ceux que peut fournir la pompe de charge. La Figure A-5 donne un exemple de filtre actif du 4ème ordre.

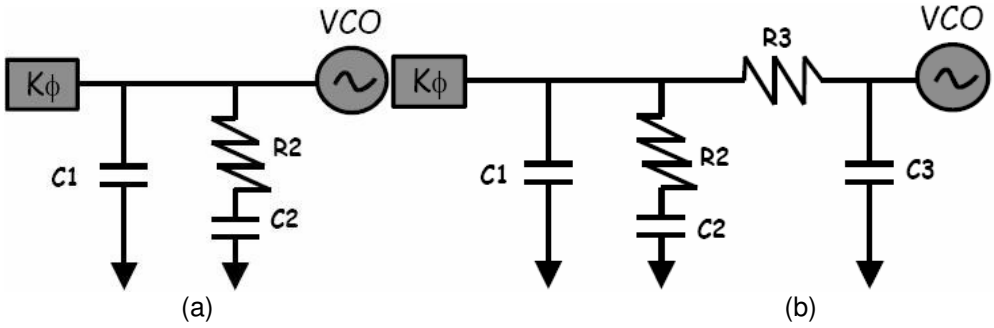


Figure A-4: (a) Un filtre passif d'ordre 2 (b) un filtre passif d'ordre 3 [D.B]

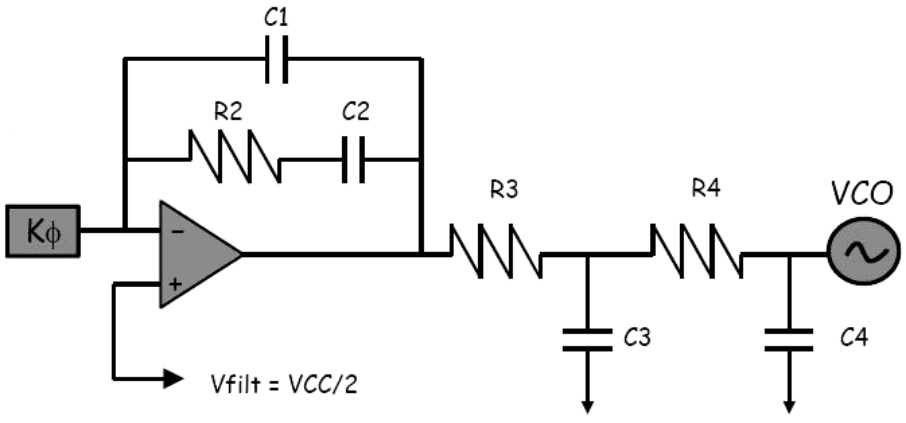


Figure A-5: Filtre actif d'ordre 4 [D.B]

Revenons à la modélisation du filtre de boucle et donc des capacités et des résistances qui le constituent. Pour cette modélisation, les résistances et les capacités sont décrites par leur

équation liant la tension au courant. Avant de calculer ces valeurs, il faut d'abord fixer les valeurs des paramètres du filtre. Les valeurs qu'on donne par la suite sont celles obtenues après plusieurs simulations et sont résumées dans le Tableau A-5 :

- Fc doit être 10 fois plus petite que Fcomp (25MHz). J'ai pris une valeur de 1600KHz.
- La marge de phase Φ a été fixée à 46°.Cette valeur fait partie de l'intervalle 40°-55°.
- Pour le type du filtre, il faut voir si la pompe de charge peut fournir les tensions requises au fonctionnement du VCO. D'après le Tableau A-2 et Tableau A-3, ceci est bien le cas car la pompe de charge peut fournir une tension entre 0 et 1,8V (en réalité elle n'atteint pas vraiment ces valeurs mais s'en rapproche), alors que le VCO fonctionne pour une plage de tension de 0 à 1V. Donc le type du filtre est passif.
- Afin de respecter les spécifications données en 4.5.2 relatives au *reference spur*, j'ai pris un filtre du 3ème ordre. Ce choix est justifié par le principe donné lors de la définition de l'ordre du filtre.

Tableau A-5: Spécification du filtre

SYMBOLE	VALEUR	UNITE	DESCRIPTION
Fc	1600	KHz	Largeur de bande en boucle ouverte
Φ	46	degrés	Marge de Phase
γ	1,5		Facteur d'optimisation gamma
KΦ	0,1	mA	Gain de la pompe de charge
Kvco	225	MHz/V	Gain du VCO voir Tableau A-3
Fout	200	MHz	Fréquence en sortie de la PLL.
Fcomp	25	MHz	Fréquence en entrée de la PLL ou fréquence de comparaison
N	8		Rapport de division entre Fout et Fcomp
T31	0,13		Rapport entre T3 et T1 avec T1 donné par l'Equation A-7

Les Equations A-6 jusqu'à A-7 [D.B], permettent de calculer les éléments du filtre.

Equation A-6: $$wc = 2\times\pi\times Fc$$

Equation A-7: $$\Phi = \tan^{-1}\left(\frac{\gamma}{wc\times T1\times(1+T31)}\right) - \tan^{-1}(wc\times T1) - \tan^{-1}(wc\times T1\times T31)$$

Equation A-8: $$T3 = T1\times T31$$

Equation A-9: $$T2 = \frac{\gamma}{wc^2\times(T1+T3)}$$

Equation A-10: $$A0 = \frac{K\Phi\times Kvco}{wc^2\times N}\times\sqrt{\frac{1+wc^2\times T2^2}{(1+wc^2\times T1^2)\times(1+wc^2\times T3^2)}}$$

Equation A-11: $$A1 = A0\times(T1+T3)$$

Equation A-12: $$A2 = A0\times T1\times T3$$

Equation A-13: $$C1 = \frac{A2}{T2^2}\times\left(1+\sqrt{1+\frac{T2}{A2}\times(T2\times A0 - A1)}\right)$$

Equation A-14: $$C3 = \frac{-T2^2 \times C1^2 + T2 \times A1 \times C1 - A2 \times A0}{T2^2 \times C1 - A2}$$

Equation A-15: $$C2 = A0 - C1 - C3$$

Equation A-16: $$R2 = \frac{T2}{C2}$$

Equation A-17: $$R3 = \frac{A2}{C1 \times C3 \times T2}$$

Cela dit, un outil disponible en ligne [W.W] permet de résoudre ces équations. Il suffit de rentrer les différents paramètres du Tableau A-5, et on obtient après arrondi les valeurs du Tableau A-6.

Tableau A-6: Valeurs des résistances et capacités du filtre

SYMBOLE	VALEUR	UNITE
C1, C3	2	pF
C2	47	pF
R2, R3	6,8	kΩ

Ces valeurs vérifient la condition énoncée en Equation A-5. La Figure A-6 représente le diagramme de bode du filtre conçu à partir des valeurs du Tableau A-6.

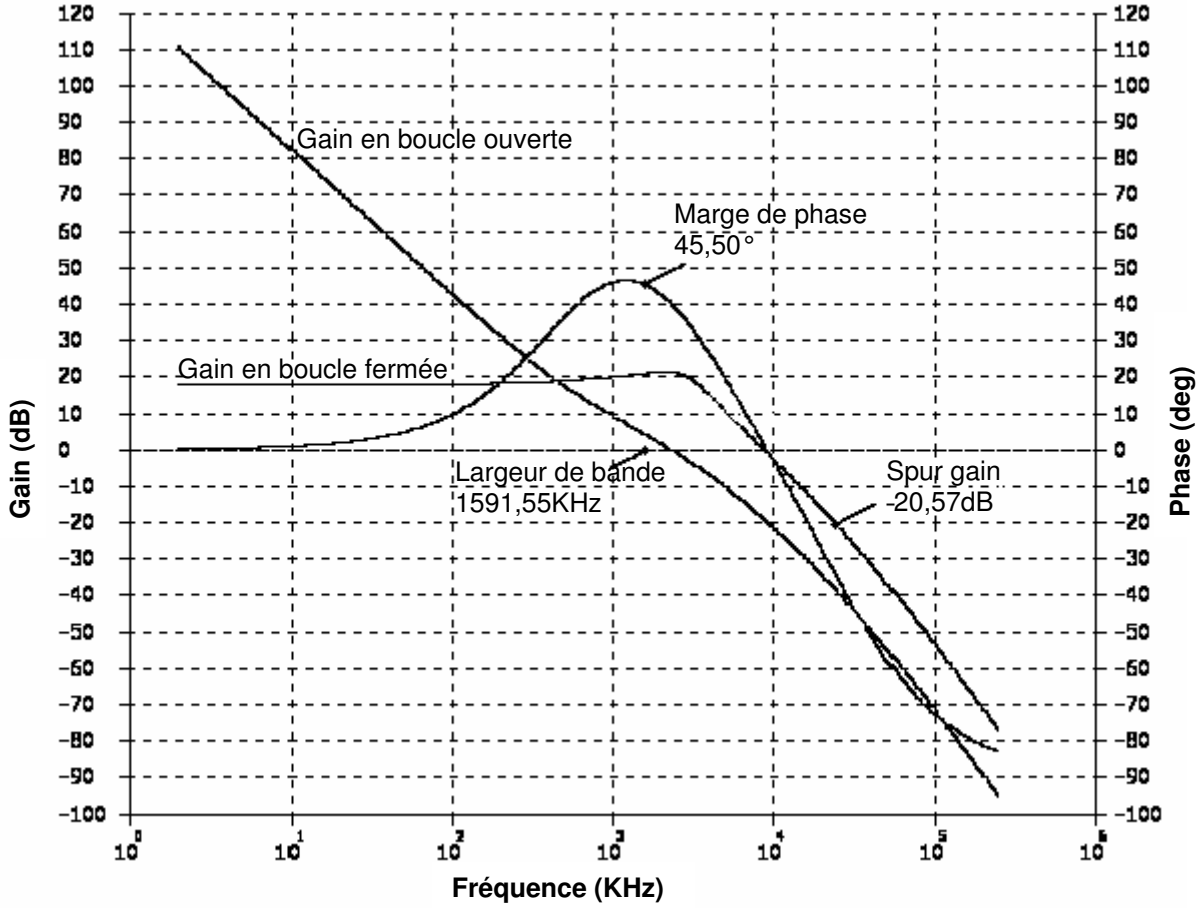


Figure A-6: Diagramme de bode du filtre utilisé

Annexe B.

Description de la PLL au niveau transistor

B.1. Description du PFD

La Figure B-1 donne le PFD au niveau portes. Pour le décrire, il suffit de faire appel aux bons éléments de la bibliothèque lesquels sont eux-mêmes conçus au niveau transistor.

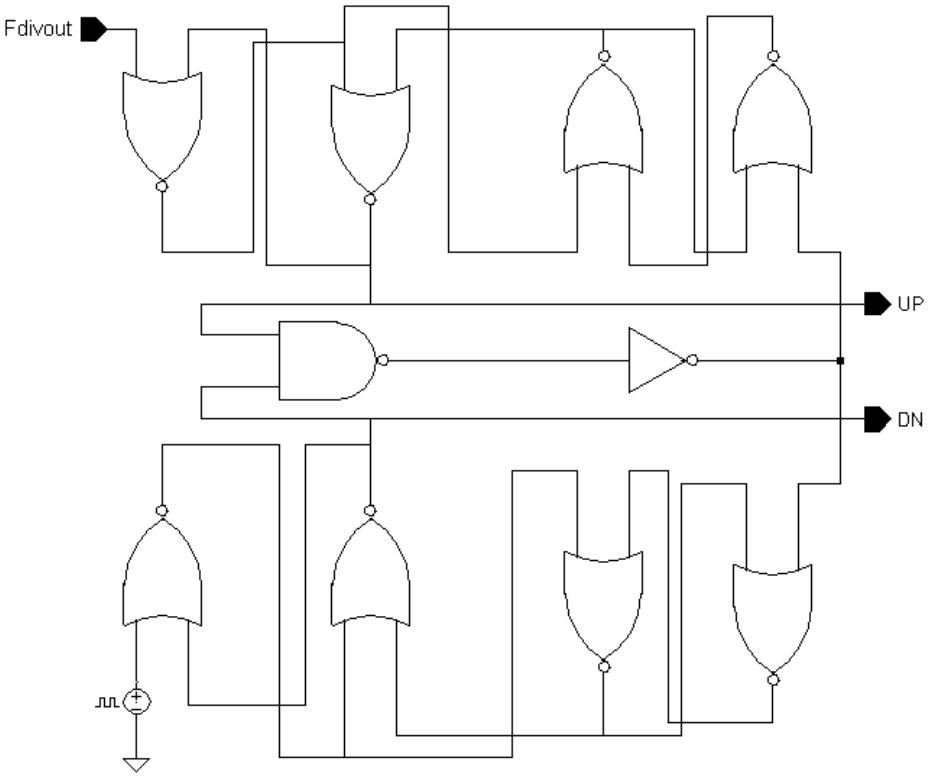


Figure B-1: PFD au niveau portes

Fdivout est l'une des deux entrées du PFD, elle provient de la sortie du diviseur par 8. L'autre entrée est connectée à un générateur de créneaux avec une fréquence de 25MHz (Fcomp : fréquence de comparaison).

B.2. Description de la pompe de charge et du filtre

L'architecture de la pompe de charge employée ici (Figure B-2) est décrite dans [S.S. 02]. D'autres architectures beaucoup plus complexes mais plus performantes existent [A.M. 02, J.K. 05]. Les entrées UP et DN sont fournies par le PFD. Vbiasp et Vbiasn sont des tensions continues de polarisation. Le signal de sortie Vvcoin sera connecté à l'entrée du VCO.

Le filtre reste le même que celui décrit en Annexe A, en dépit du fait que les caractéristiques de la PLL ont un peu changé. Ainsi le courant traversant la pompe de charge n'est plus exactement de 100µA comme décrit dans le Tableau A-2, de même la valeur absolue du gain du VCO a augmenté (voir section B.3.6).

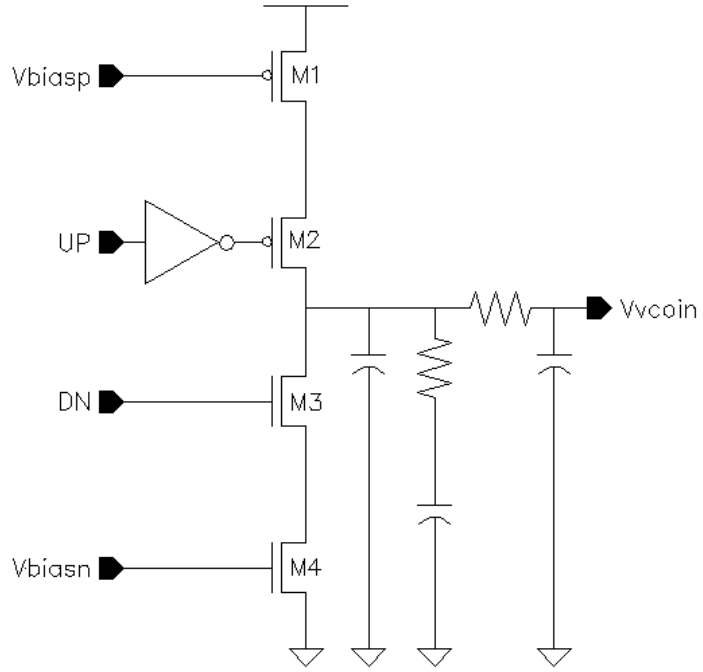


Figure B-2: Pompe de charge niveau transistor et filtre

Les transistors M1 et M4 jouent le rôle de générateur de courant et ils sont calculés de telle sorte à obtenir 100µA de courant (avec Vbiasn=700mV et Vbiasp=1V) suivant l'état des transistors M2 et M3 qui sont des interrupteurs. Ainsi si le transistor M2 est fermé et M3 est ouvert, la pompe de charge fournit du courant au filtre. Dans le cas contraire, c'est le filtre qui fournit du courant. Le Tableau B-1 donne le dimensionnement des transistors M1 à M4 de la pompe de charge.

Tableau B-1: Taille des transistors de la pompe de charge

	M1	M2	M3	M4
W/L	7µm/0,3µm	10µm/0,18µm	5µm/0,18µm	3,9µm/0,3µm

Concernant le filtre, les valeurs des éléments sont les mêmes que dans le Tableau A-6.

B.3. Description du VCO

Le VCO est composé de trois parties : l'oscillateur en anneau, le comparateur et le diviseur. L'oscillateur utilise une architecture différentielle afin d'augmenter les performances du système. Le comparateur permet de passer d'une entrée différentielle venant de l'oscillateur, vers une sortie digitale. Le diviseur permet de ramener la fréquence dans les plages désirées et d'obtenir un rapport cyclique de 50%.

B.3.1. Théorie de l'oscillation

Un oscillateur quasi-sinusoïdal comporte obligatoirement un amplificateur recevant de l'énergie d'une source extérieure et fournissant dans la boucle un gain en puissance nécessaire

pour entretenir les signaux produits. Le réseau de réaction ne permet l'apparition d'oscillations qu'à une fréquence particulière.

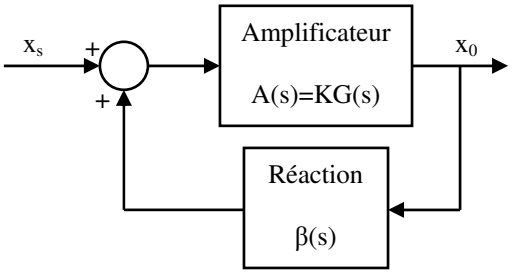


Figure B-3: Boucle de réaction négative

Pour que le système de la figure précédente puisse être le siège d'oscillations, il faut satisfaire deux conditions :

Equation B-1:
$$\begin{cases} |A(jw)\times\beta(jw)| = 1 \\ \arg(A(jw)\times\beta(jw)) = 0° \end{cases}$$

Si β(jw)=1, alors l'oscillation ne dépend que de la fonction de transfert en boucle ouverte A(jw).

B.3.2. Topologie de l'oscillateur en anneau à quatre étages

La Figure B-4 donne la topologie de l'oscillateur. Il s'agit d'une architecture différentielle permettant d'augmenter les performances globales de la PLL. Cet oscillateur est constitué de quatre cellules identiques rétro bouclées [H.L. 97].

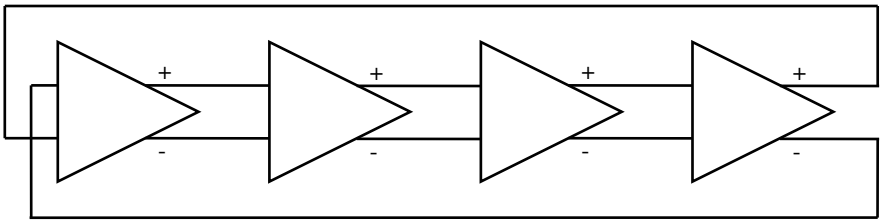


Figure B-4: Topologie de l'oscillateur

Il est à remarquer que les entrées au niveau de la première cellule sont inversées par rapport aux trois autres. Ceci est une condition nécessaire d'oscillation et fait en sorte que la fonction de transfert en boucle ouverte ait une amplification négative, car dans le cas contraire le système n'oscillera pas.

B.3.3. Description d'une cellule au niveau transistor

Plusieurs architectures existent dans la littérature [J.M. 96, R.B. 97, L.D. 00, Z.Z. 01] pour décrire ces étages qu'on appelle aussi cellules de retard. L'architecture choisie est donnée à la Figure B-5.

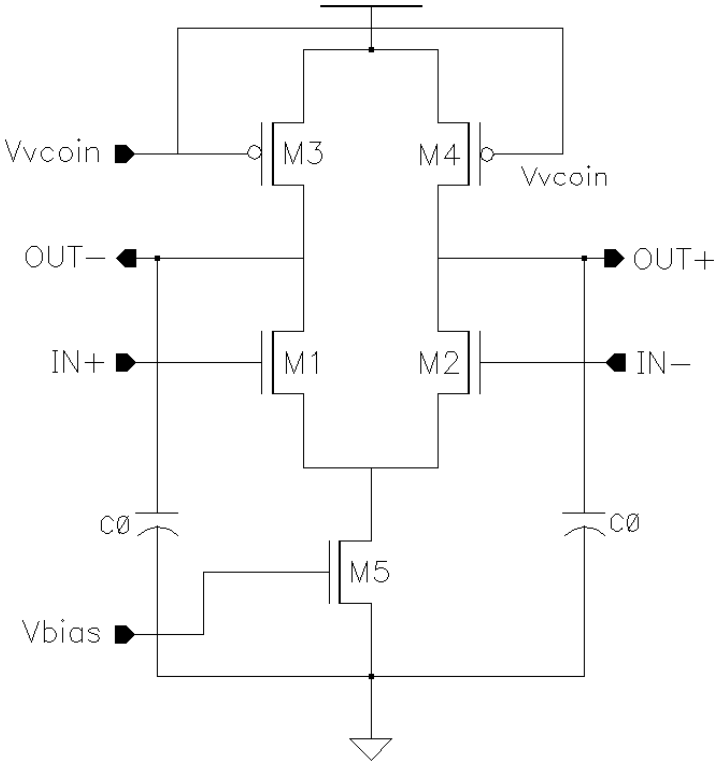


Figure B-5: Cellule de retard différentielle

Pour calculer les tailles des différents transistors, on part du fait que :

- Les transistors pmos sont équivalents à des résistances donc ceux-ci fonctionnent dans la zone ohmique pour certaines valeurs de Vvcoin. Cette tension permet de modifier la résistance de ces transistors, et par suite la fréquence d'oscillation (voir section B.3.4). Cette résistance $Rpmos_{3,4}$ est fixée par l'utilisateur et est égale à 4kΩ pour une tension Vvcoin=0V d'où :

Equation B-2: $$Rpmos_{3,4} = \left(2 \times \frac{w_{3,4}}{l_{3,4}} \times Kp \times \left(Vsg_{3,4} - |Vthp| \right) \right)^{-1}$$

- Les transistors d'entrée nmos sont calculés lorsque ceux-ci fonctionnent dans la zone active. La transconductance de ces transistors est donnée par :

Equation B-3: $$gm_{1,2} = \sqrt{4 \times Kn \times \frac{w_{1,2}}{l_{1,2}} \times Ids_{1,2}}$$

  La valeur de $Ids_{1,2}$ est égale à la moitié du courant Ids_5 qui traverse le transistor M5. Ce courant est fixé par l'utilisateur et sa valeur vaut 50µA (avec Vbias=600mV).

- Le montage possède une amplification égale à :

Equation B-4: $$Av = gm_{1,2} \times Rpmos_{1,2}$$

  Il vaut une valeur minimum pour assurer des oscillations. Cette valeur minimum est calculée à la section B.3.4 et vaut $Av = \sqrt{2}$. Dans notre cas vu les valeurs peu précises de Kn, Kp, Vthp et Vthn, j'ai pris une valeur d'amplification de 1,6.

- La capacité C0 est introduite pour choisir la fréquence d'oscillation pour une valeur de Vvcoin (par exemple Vvcoin=0V). Celle-ci est une partie de la capacité totale (Ctot) de sortie de la cellule. Sa valeur est de 58fF, et la fréquence d'oscillation (voir section B.3.4 pour son calcul) est de 550MHz pour Vvcoin=0V, C0=58fF et Rpmos=4kΩ.

Finalement en résolvant toutes ces équations on obtient après plusieurs itérations des simulations, les dimensions des transistors résumées dans le Tableau B-2.

Tableau B-2: Taille des transistors d'une cellule de retard

	M1,2	M3,4*	M5
W/L	4,8µm/0,18µm	0,76µm/0,2µm	2µm/0,18µm

* : Il y a en tout quatre cellules interconnectées comme mentionné à la Figure B-4. La dernière cellule a des tailles différentes (0,62µ/0,2µ) pour compenser l'effet introduit par l'étage suivant qui servira à passer d'une sortie différentielle à une sortie digitale avec rapport cyclique de 50% (voir section B.3.5).

B.3.4. Modèle linéaire de l'oscillateur en anneau

On suppose des petites amplitudes au niveau de chaque étage de telle sorte à ce que le modèle linéaire puisse être appliqué. Ceci conduit à la Figure B-6 :

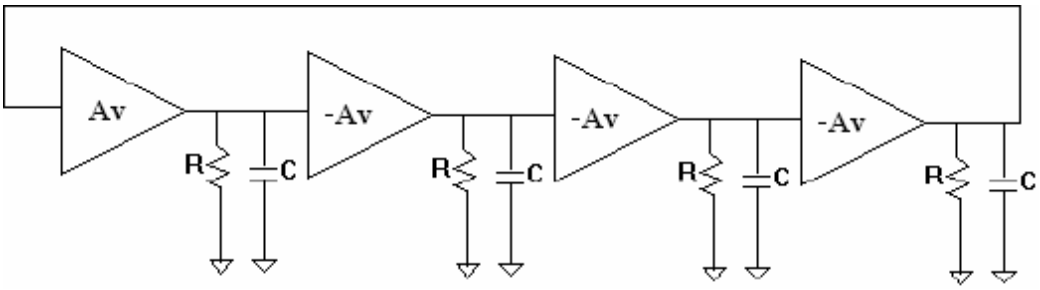


Figure B-6: Modèle linéaire de l'oscillateur en anneau à quatre étages

La résistance R et la capacité C correspondent respectivement à la résistance et à la capacité équivalente en sortie de chaque cellule.

La fonction de transfert en boucle ouverte d'un tel système est donnée par :

Equation B-5: $$H(s) = \frac{-Av^4}{\left(1+\frac{s}{w_0}\right)^4}$$

A partir de l'Equation B-1, avec β(jw)=1 et A(jw)=H(jw), on obtient ces conditions:

Equation B-6: $$\begin{cases} w_{osc} = w_0 = \dfrac{1}{R \times C} \\ Av = \sqrt{2} \end{cases}$$

Avec ces conditions, il suffit qu'un petit bruit en entrée existe (ce qui est toujours le cas) pour que ce dernier soit amplifié et petit à petit le circuit se mettra à osciller avec une forme

d'onde approximativement sinusoïdale (elle n'est pas sinusoïdale à cause de la constante RC). En théorie l'amplification du montage en entier est infinie mais en réalité, elle est limitée par les tensions d'alimentation qu'on ne peut pas dépasser. Il est à remarquer que la valeur Av donnée est le minimum pour assurer des oscillations.

B.3.5. Le comparateur avec diviseur

Comme mentionné plus haut, la sortie de l'oscillateur est à peu près sinusoïdale. Il faut transformer cette sortie en une sortie digitale qui varie entre 0V et 1,8V et ceci est le rôle du comparateur. De plus la sortie de l'oscillateur a une fréquence d'oscillation deux fois plus grande que nécessaire. C'est pour cette raison qu'un premier diviseur est employé.

Le schéma niveau transistors de ces blocs est donné à la Figure B-7.

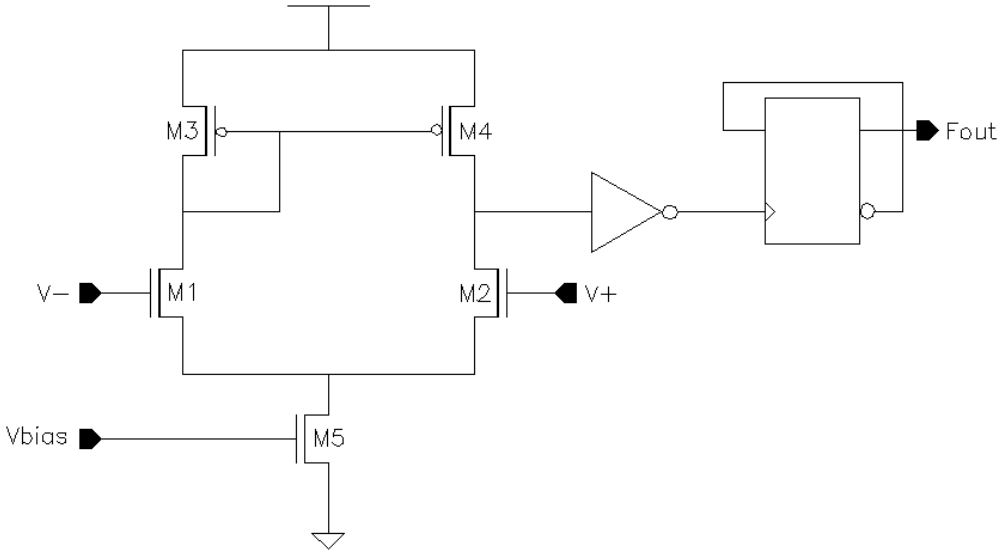


Figure B-7: Le comparateur avec le diviseur par 2

Le fonctionnement de cette architecture est largement décrit dans la littérature, on ne va pas trop s'attarder sur les équations. Cela dit pour calculer les différents transistors, on a fixé le courant de polarisation Ids_5 à 70µA (avec Vbias=740mV) et le gain de l'amplificateur différentiel à 26dB.

Tableau B-3: Taille des transistors du comparateur

	M1,2	M3,4	M5
W/L	0,5µm/0,18µm	0,6µm/0,18µm	1,5µm/0,2µm

B.3.6. Caractéristique du VCO complet

Le VCO englobe l'oscillateur à 4 cellules de retard, le comparateur et le diviseur. Ceci mène à la Figure B-8. Ce VCO fonctionne pour une plage de tension d'entrée de 0V à 0,9V comme décrit par la Figure B-9.

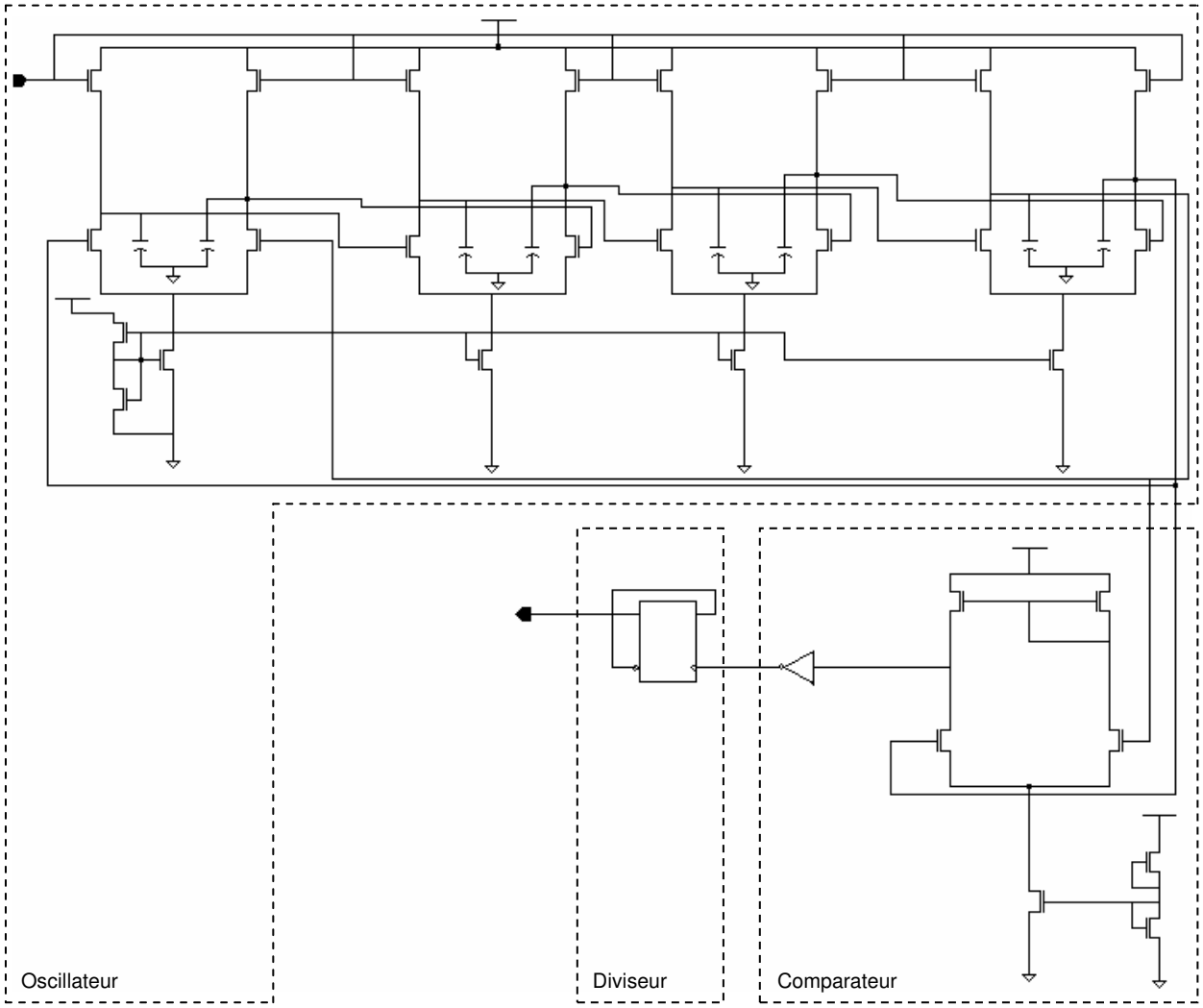


Figure B-8: Le VCO au niveau transistor

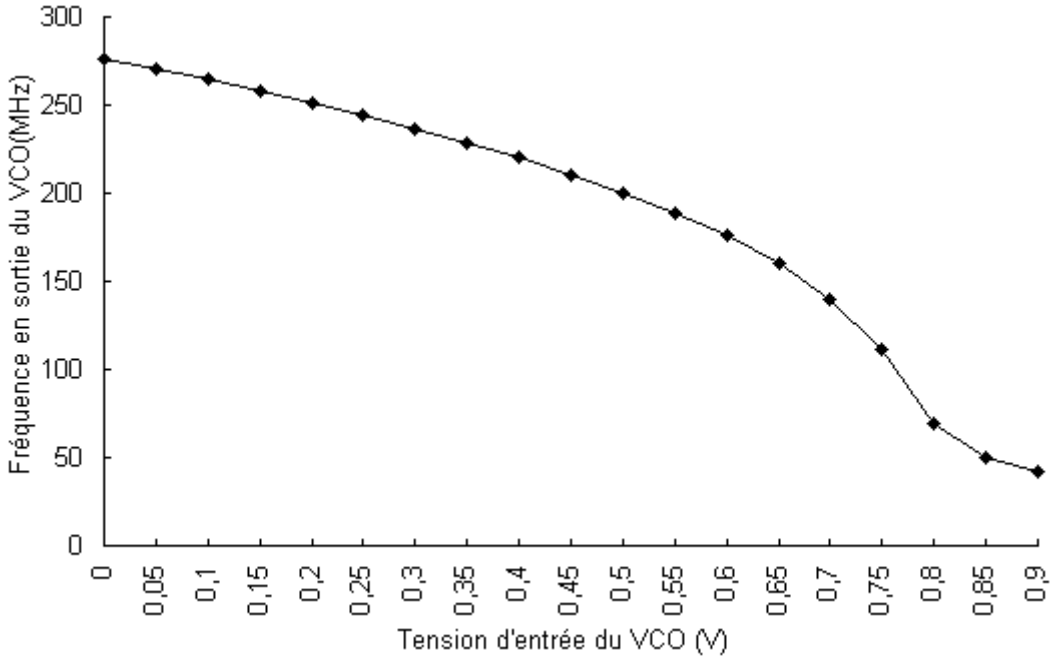


Figure B-9: Caractéristique du VCO au niveau transistor

Si on compare cette figure à la Figure A-3 qui représente la même caractéristique du modèle haut niveau, on remarque que :

- La caractéristique n'est plus linéaire. Néanmoins elle l'est autour du point qui nous intéresse, c'est-à-dire celui donnant une fréquence de sortie de 200MHz.
- Le gain linéaire, autour du point donnant une fréquence de sortie de 200MHz, est de -215MHz au niveau transistors, alors que celui au niveau VHDL-AMS est de -225MHz.
- Même si il y a différence entre les gains linéaires, le filtre n'a pas changé et ceci a une influence sur la valeur du *spur gain* dont la valeur augmente (en valeur absolue). Autrement dit le filtre atténue mieux que dans le modèle haut niveau mais cette différence reste minime. Donc on aura les mêmes courbes (sauf pour le *spur gain*) que celles données en Figure A-6.
- La tension d'entrée du VCO (Vvcoin) permettant d'obtenir 200MHz passe de 333mV pour le modèle haut niveau à environ 500mV.

B.4. Le diviseur

D'après les caractéristiques globales de la PLL, la fréquence de comparaison (Fcomp) et la fréquence de sortie (Fout) sont respectivement de 25MHz et 200MHz. Pour obtenir cette dernière il faut diviser Fout par 8. Ce dernier est obtenu en connectant trois bascules dont chacune divise par 2 (Figure B-10).

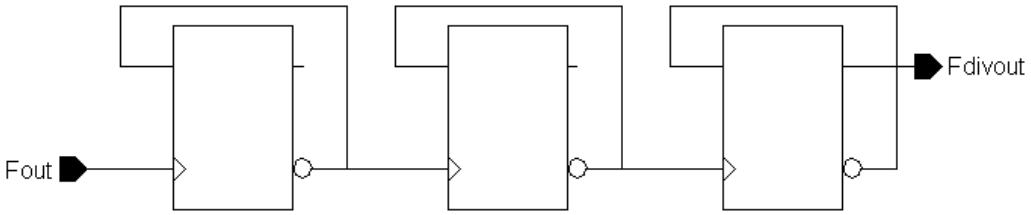


Figure B-10: Diviseur par 8

Annexe C.

Résultats d'injection de fautes dans la PLL décrite au niveau transistor

Cette section présente les résultats d'injections dans la PLL. Les résultats présentés ici sont regroupés selon les blocs fonctionnels de la PLL.

C.1. Injections au niveau du filtre

Commençons par définir les nœuds d'injection. Ceci est donné à la Figure C-1.

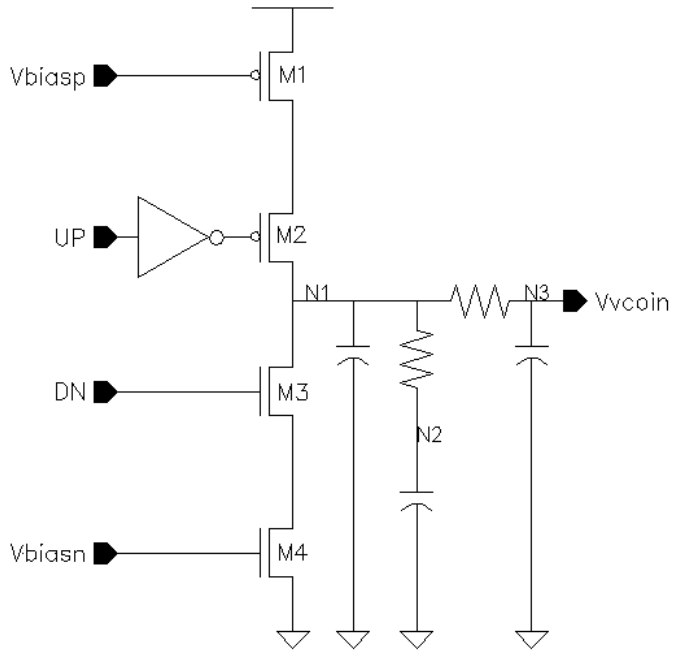


Figure C-1: Nœuds d'injection au niveau du filtre

La Figure C-2 donne l'évolution de Vpp_n_inj au niveau du nœud N1 pour des charges négatives.

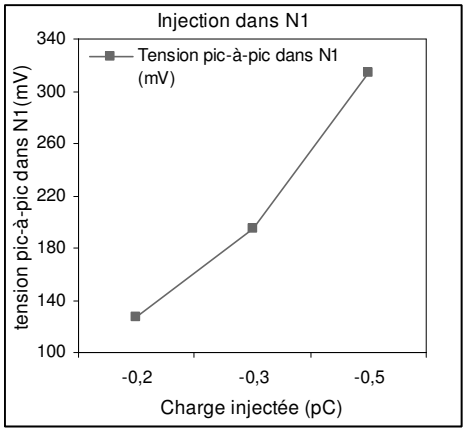
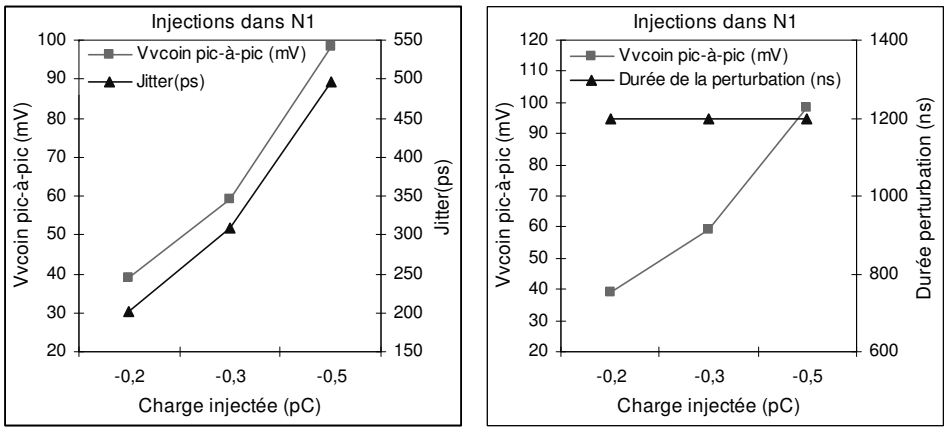


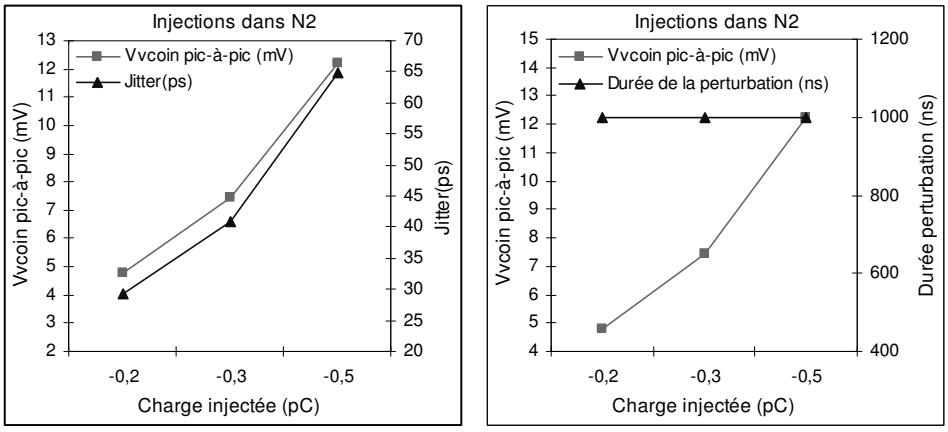
Figure C-2: Evolution de la tension pic-à-pic au niveau du nœud N1 pour différentes valeurs de Qinj

Comme on peut le voir, plus la charge augmente (en valeur absolue) plus cette différence est grande. Ceci est déjà connu d'avance; on ne fait que le confirmer encore une fois ici, et cette remarque est valable pour toutes les autres campagnes. Donc, par la suite on n'y reviendra plus.

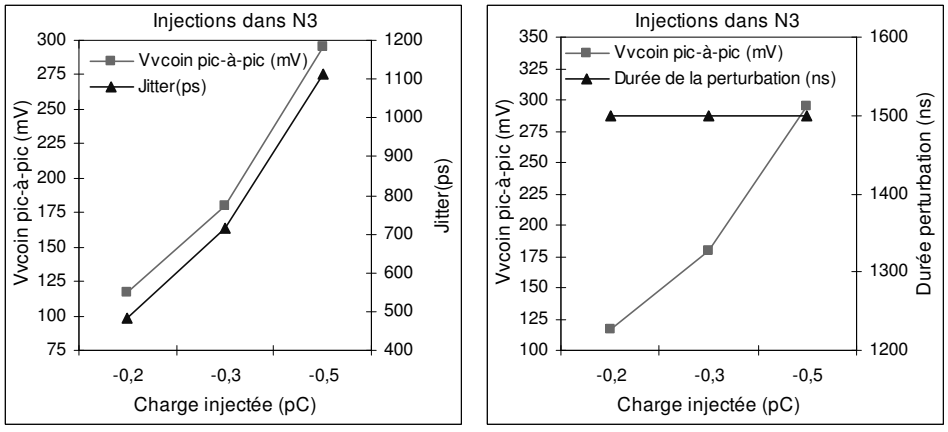
La Figure C-3 montre l'évolution de Vpp_vcoin_inj, du *jitter* et de durée_pertu en faisant varier Qinj et suivant les nœuds d'injection N1 (Figure C-3 (a)), N2 (b) et N3 (c).



(a)



(b)



(c)

Figure C-3: Evolution de la tension Vvcoin pic-à-pic, du jitter et de la durée de la perturbation pour différentes valeurs de Qinj et suivant les nœuds d'injection N1(a), N2(b) et N3(c) du filtre

En regardant de près ces courbes, on remarque que plus la tension Vpp_vcoin_inj augmente, plus le *jitter* est important. Par contre bien que la tension Vpp_vcoin_inj augmente, la durée de la perturbation (durée_pertu) ne change pas. Certes la méthode de mesure, à savoir une mesure visuelle, y est pour beaucoup. Cela dit en ne se basant que sur ces courbes, il n'y a pas

de relation directe entre la tension Vpp_vcoin_inj et durée_pertu. Ainsi pour une charge de -0,5pC dans le nœud N1 la durée de la perturbation est de 1200ns pour une tension Vpp_vcoin_inj d'à peu près 100mV, et pour une charge de -0,2pC dans le nœud N3, la tension Vpp_vcoin_inj est égale à peu près à 100mV (comme dans N1), mais la durée_pertu est cette fois de 1500ns. Donc à première vue celle-ci dépend beaucoup plus du nœud que la tension Vpp_vcoin_inj générée à cause de l'injection. On aura l'occasion d'infirmer cette hypothèse en comparant tous les durée_pertu de tous les nœuds, pour toutes les campagnes à la section 4.7.4.3.

Concernant le *jitter*, seules les injections dans N2 pour une charge injectée de -0,2pC ou -0,3pC ne dépassent pas la plage de tolérance fixée. Ceci s'explique par la présence de la grande capacité de 47pF. Pour les autres injections le *jitter* dépasse la plage de tolérance fixée.

Les remarques présentées précédemment sont aussi valables lorsque les charges injectées sont positives (en particulier pour le nœud N2).

Au vu de ces résultats, on peut dire que les injections à haut niveau et celles au niveau transistor sont concordantes.

C.2. Injections au niveau du comparateur avec diviseur

La Figure C-4 donne les nœuds d'injection.

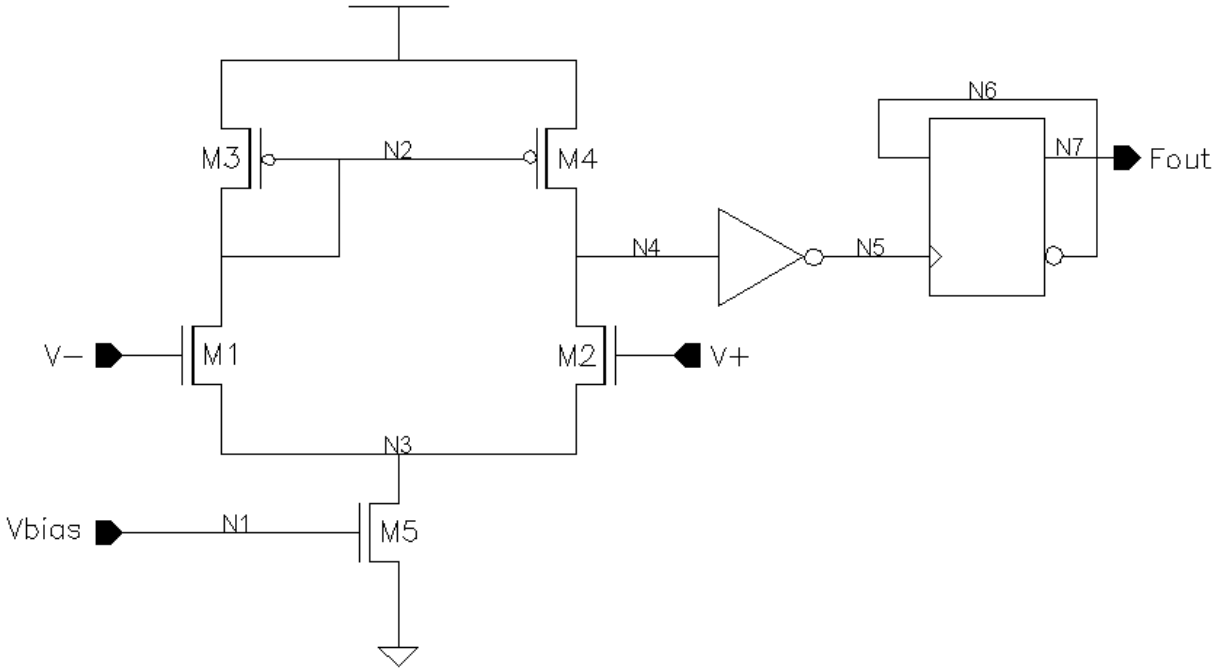


Figure C-4: Nœuds d'injection au niveau du comparateur avec diviseur

La Figure C-5 donne les résultats d'injection d'un pic négatif dans le comparateur avec diviseur. Les injections dans le nœud analogique N1 (Figure C-5 (a)) ne font pas sortir le *jitter* de sa plage de tolérance (dans le cas de N1, le *jitter* reste inférieur à 15,5ps). La durée de la perturbation ne dépasse pas les 200ns pour une tension Vpp_vcoin_inj inférieure à 0,29mV. N1

n'est pas le seul nœud où l'injection ne produit pas d'effets notables. Ainsi les nœuds N2 à N4 n'ont pratiquement pas d'effets même si les Vpp_N_inj ont des valeurs sensiblement élevées (3V dans certains cas).

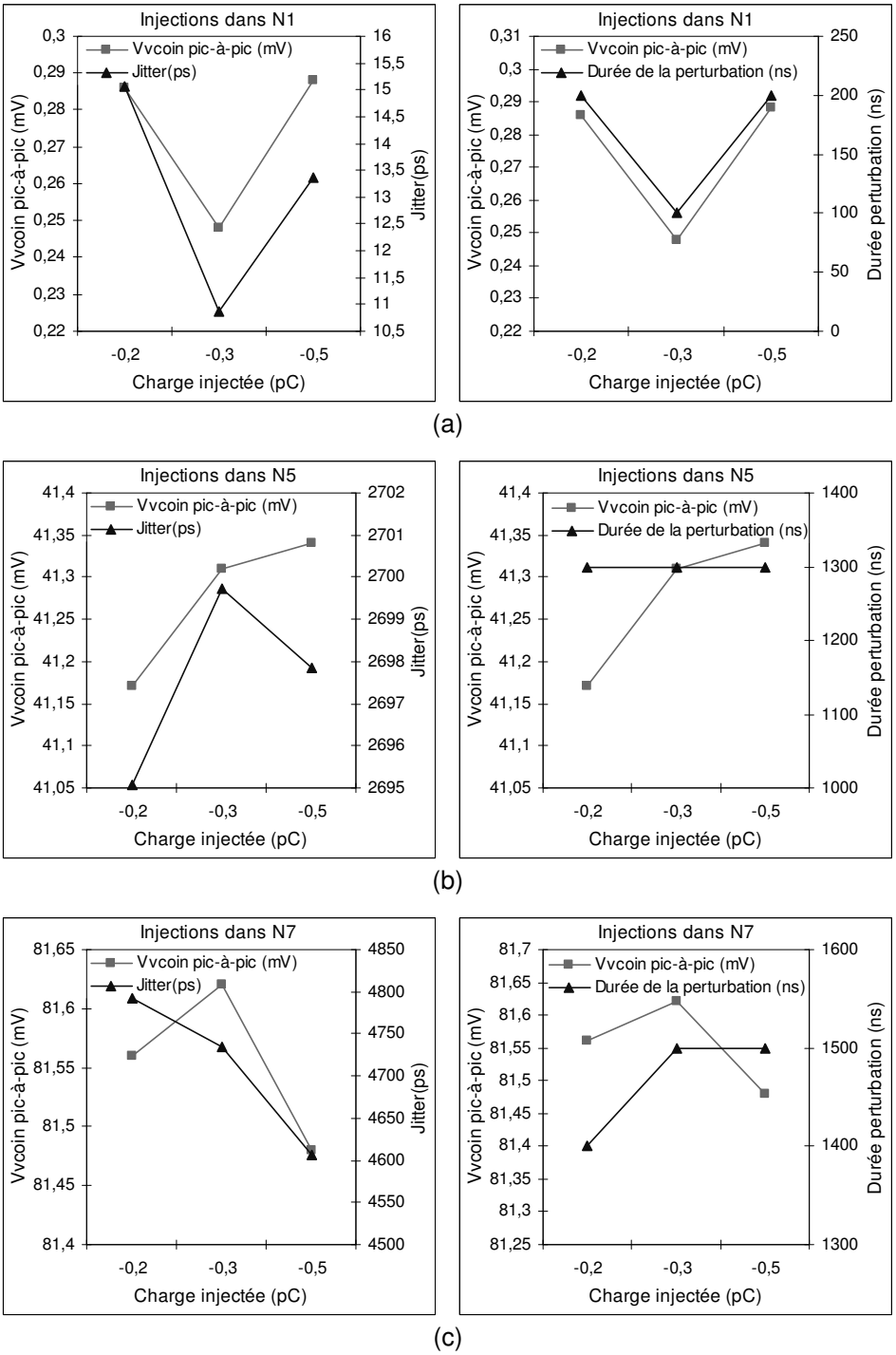


Figure C-5: Evolution de Vvcoin pic-à-pic, du jitter et de la durée de la perturbation pour différentes valeurs de Qinj et suivant les nœuds d'injection N1(a), N5(b) et N7(c) du comparateur avec diviseur

En regardant de près la Figure C-6 pour Qinj = -0,5pC, on voit que le Vvcoin n'a pas de variation notable aux moments des injections 3, 5, 7 et 9µs, qui correspondent respectivement aux nœuds N1, N2, N3 et N4. Au mieux l'injection fait varier le Vpp_vcoin_inj de 0,29mV lors de l'injection dans le nœud N1.

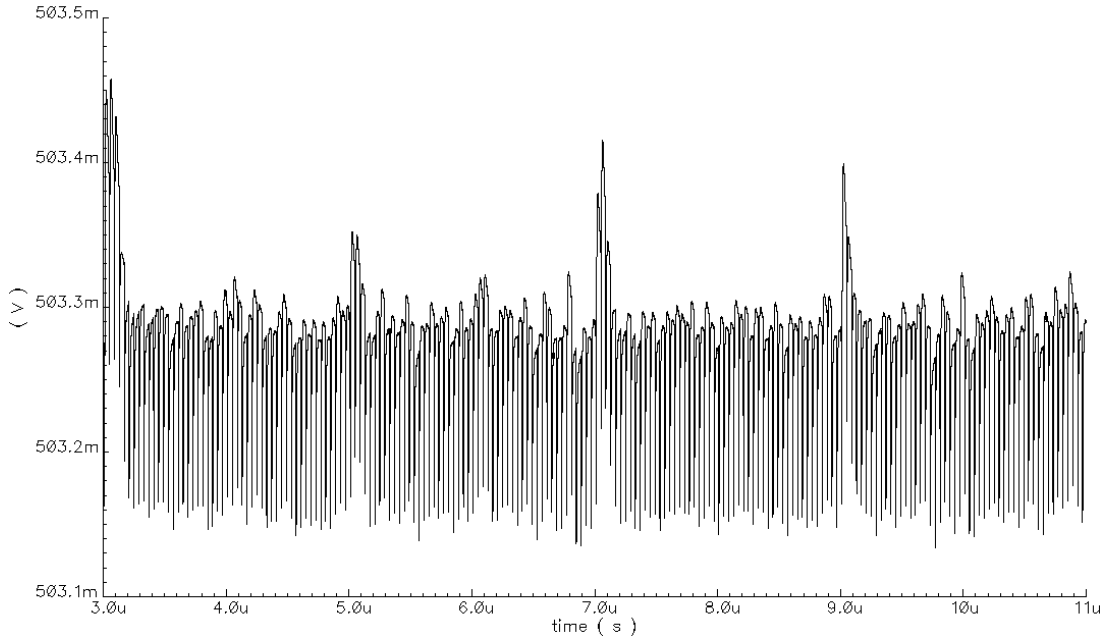


Figure C-6: Tension à l'entrée du VCO pour les temps d'injection : 3, 5, 7 et 9µs

L'injection dans le nœud N5 (Tinj 11µs) produit quant à lui un effet plus que notable. En regardant la Figure C-7, on comprend mieux pourquoi. La faute fait passer ce signal, servant d'horloge à la bascule, à '1' et vu que celle-ci est sensible aux fronts montants, la sortie est inversée alors qu'elle ne le devrait pas. La PLL par la suite corrige cette erreur (Figure C-8). Avec un pic positif, on n'aurait pas eu le même effet car les fronts montants de cette horloge ne vont pas changer. L'effet serait également réduit dans le cas d'un diviseur synchrone.

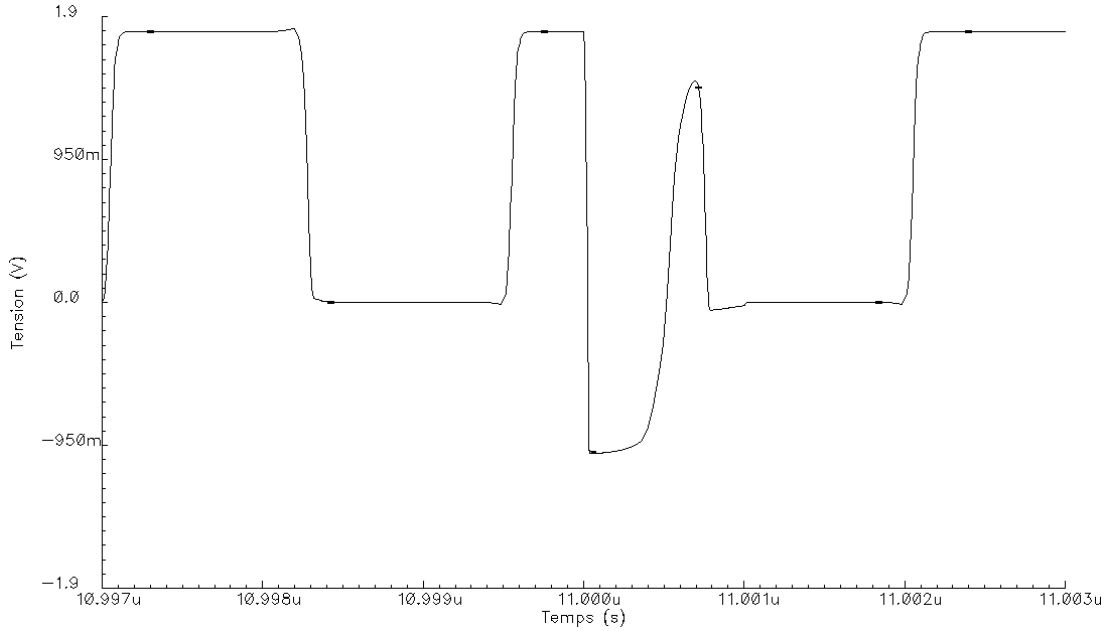


Figure C-7: Tension au niveau du nœud N5

Pour que l'injection dans le nœud N6 perturbe sensiblement la PLL, il faudrait que le signal numérique soit inversé au moment d'un front montant de l'horloge de la bascule D. Lors de ces simulations ceci n'a pas été le cas, d'où la non sensibilité de ce nœud.

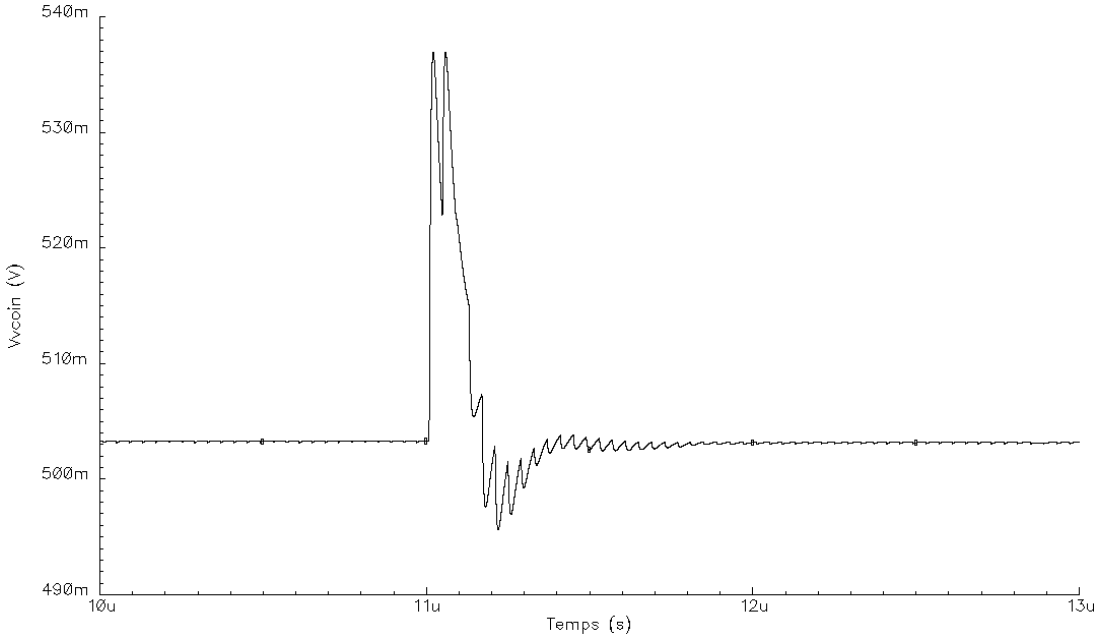


Figure C-8: Tension a l'entrée du VCO suite à l'injection dans le nœud N5

Pour N7, il s'agit de la sortie de la PLL. Le fait d'inverser ce signal va forcément perturber la PLL et plus cette perturbation se produit après un front montant "sain", plus le *jitter* sera grand comme c'est le cas ici (plus que 4800ps). En regardant la Figure C-5(c), on voit que la tension Vpp_vcoin_inj diminue lors de l'injection avec Qinj=0,5pC. Cela dit cette baisse reste relativement faible pour être négligée, et c'est ce qui explique l'allure du *jitter*. Donc grâce au pic négatif injecté, on n'a pu inverser la valeur du signal.

Avec un pic positif, et dans les mêmes conditions les nœuds N5 et N7 ne perturberont pas la PLL car ils ne vont pas créer de fronts supplémentaires. En plus les nœuds N1 à N4 et N6 ne sont pratiquement pas sensibles. D'ailleurs le *jitter* ne dépasse même pas les 30ps. Donc l'injection de pics positifs n'a pas d'effets notables sur la PLL (du moins avec cette campagne d'injection).

C.3. Injections au niveau du diviseur par 8

La Figure C-9 donne les nœuds d'injection dans le diviseur par 8.

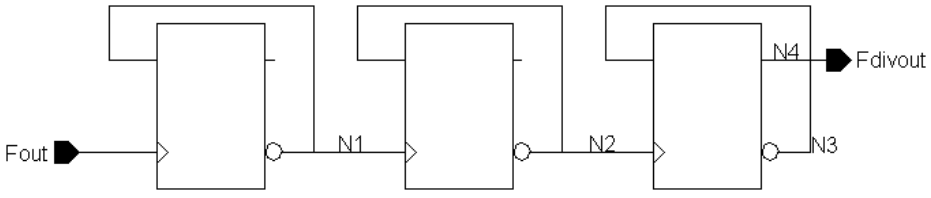


Figure C-9: Nœuds d'injection au niveau du diviseur par 8

La Figure C-10 montre l'évolution de Vpp_vcoin_inj, du jitter et de durée_pertu. Seuls les nœuds N1 et N4 sont sensibles respectivement aux pics négatifs et positifs. Ici aussi les remarques précédemment mentionnées sur l'évolution des courbes sont respectées.

Que ce soit en injectant des pics positifs ou négatifs, les nœuds N2 et N3 ne produisent pratiquement pas d'effets (*jitter* ne dépassant pas les 20ps). Pour le nœud N3, on comprend vite pourquoi. En effet il faudrait que le pic injecté se produise au moment d'un front d'horloge de la 3ème bascule (la plus à droite de la Figure C-9). Par contre pour N2, qui est une horloge, on aurait dû avoir une forte déviation du *jitter*, ceci n'a pas été le cas même si la Vpp_n_inj est encore très grande (au minimum 1,38V).

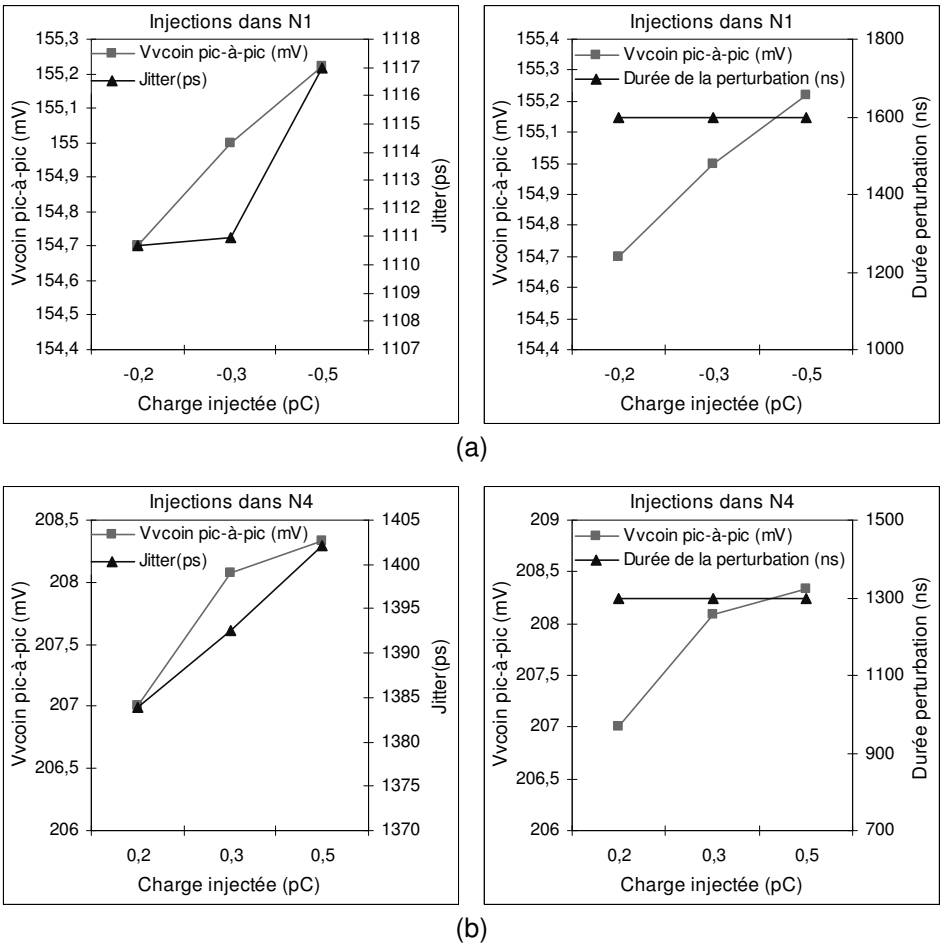


Figure C-10: Evolution de Vvcoin pic-à-pic, du jitter et de la durée de la perturbation lors de l'injection dans le diviseur par 8 pour Qinj négatif dans le nœud N1(a) et Qinj positif dans le nœud N4(b)

C.4. Injections au niveau du détecteur de phase-fréquence

La Figure C-11 donne les nœuds d'injection. Les Figures C-12 et C-13 donnent l'évolution de Vpp_vcoin_inj, du *jitter* et de durée_pertu. Les nœuds N1 et N10 sont sensibles aux pics positifs, par contre N2 et N9 le sont pour les pics négatifs. Les autres injections n'ont pas d'impact sur la PLL.

Encore une fois le *jitter* suit l'évolution du Vpp_vcoin_inj (mis à part pour N10 mais cette différence reste minime). La durée_pertu reste pour les différents nœuds.

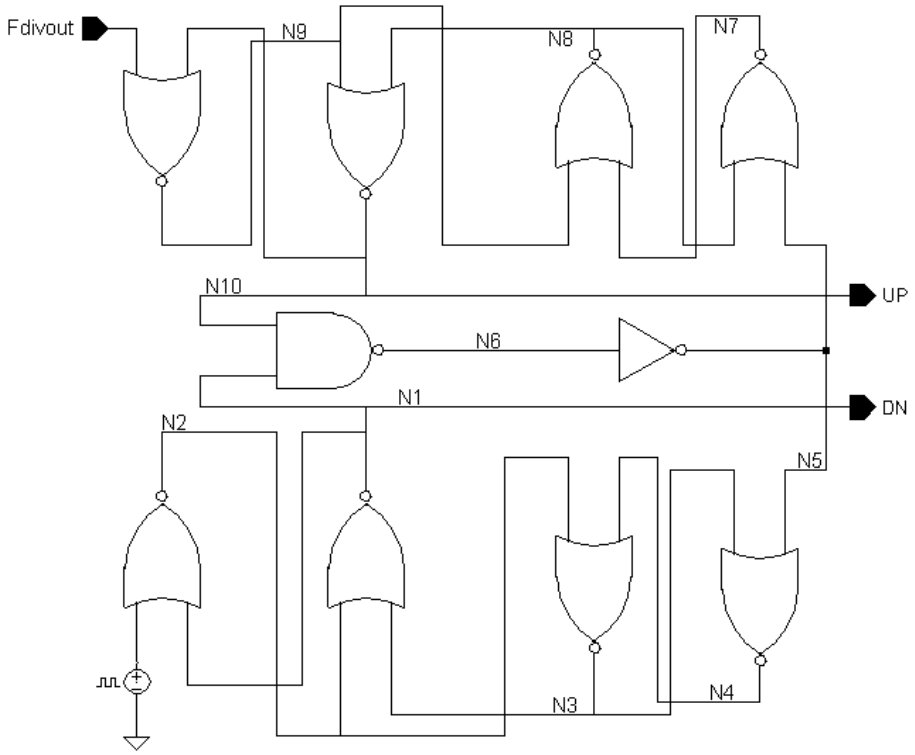
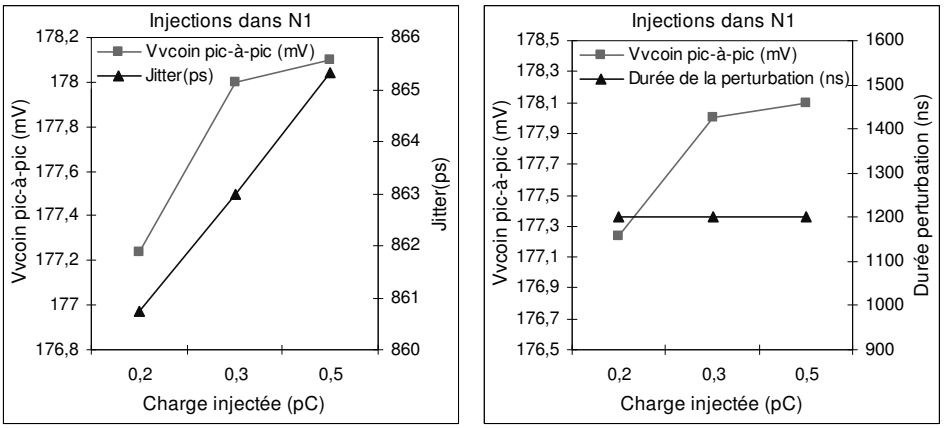
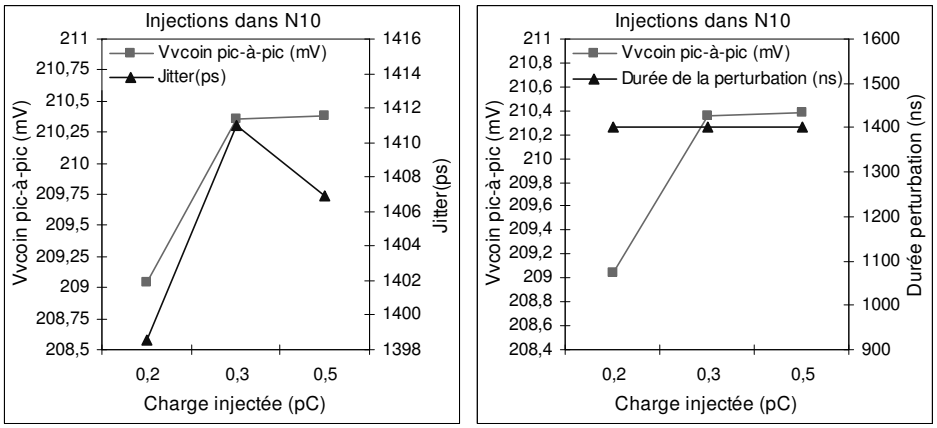


Figure C-11: Nœuds d'injection au niveau du PFD

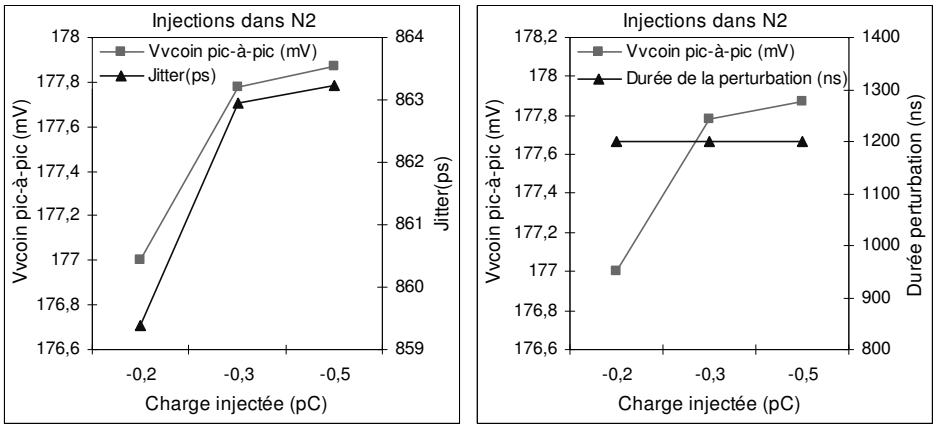


(a)

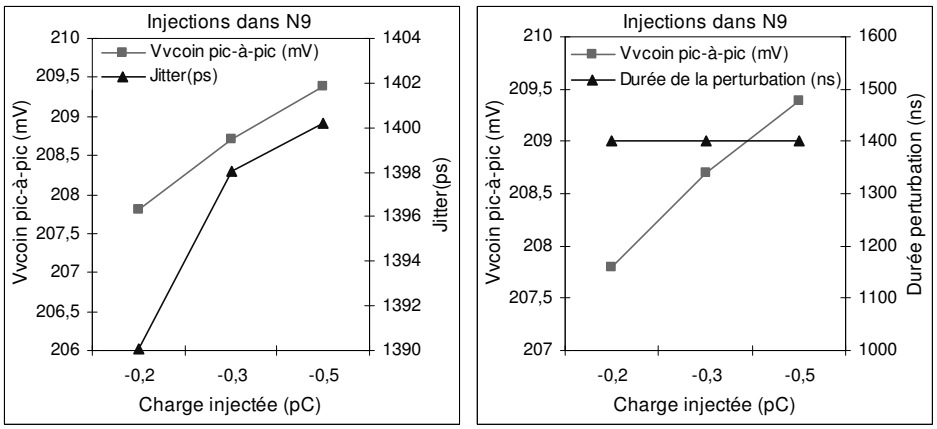


(b)

Figure C-12: Evolution de Vvcoin pic-à-pic, du jitter et de la durée de la perturbation lors de l'injection dans le PFD pour Qinj positif dans les nœuds N1(a) et N10(b)



(a)



(b)

Figure C-13: Evolution de Vvcoin pic-à-pic, du jitter et de la durée de la perturbation lors de l'injection dans le PFD pour Qinj négatif dans les nœuds N2(a) et N9(b)

C.5. Injections au niveau de la pompe de charge

La Figure C-14 donne les nœuds d'injection.

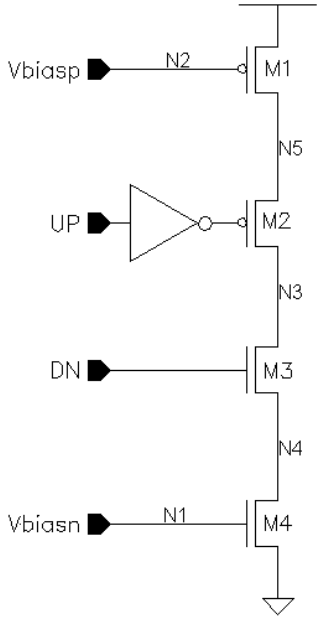
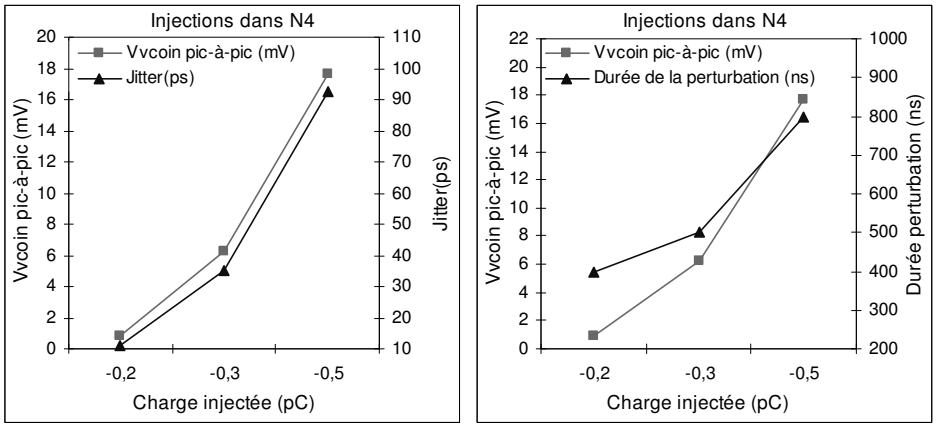


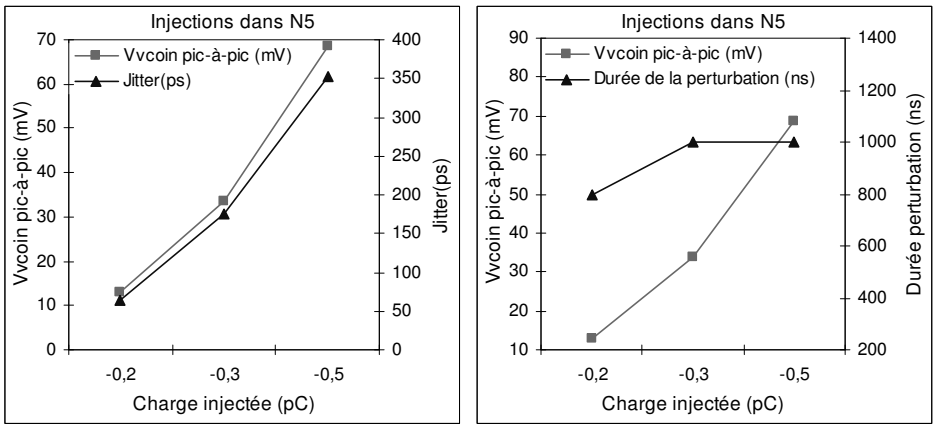
Figure C-14: Nœuds d'injection au niveau de la pompe de charge

Les injections ont montré que les nœuds N1 et N2 n'ont pas d'effets sur la PLL quelque soit le signe de la charge injectée. Le nœud N3 est le même que le nœud N1 de la Figure C-1, donc les commentaires le concernant ne seront pas repris ici.

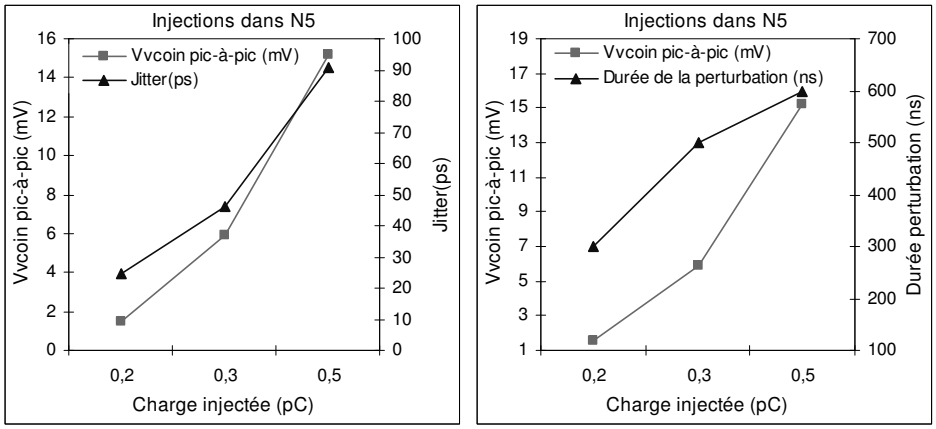
La Figure C-15 donne les résultats d'injection dans le nœud N4 (a) et N5 (b) pour des pics négatifs, et N5 (c) pour des pics positifs. Les injections de pics positifs dans le nœud N4 n'ont pas fait sortir le *jitter* de sa plage de tolérance même si dans le cas où Qinj=0,5pC, le *jitter* atteint quand même 51ps.



(a)



(b)



(c)

Figure C-15: Evolution de Vvcoin pic-à-pic, du jitter et de la durée de la perturbation lors des injections dans la pompe de charge pour Qinj négatif dans les nœuds N4(a), N5(b) et Qinj positif dans N5(c)

Les mêmes remarques que lors des autres injections sont valables ici à savoir l'augmentation du *jitter* avec la tension Vpp_vcoin_inj. Par contre durée_pertu n'est plus constante, comme c'était le cas auparavant.

Dans le cas des injections dans N5, avec Qinj=0,3pC, on retrouve un *jitter* de 46ps. Même si cette valeur reste inférieure à 60ps (valeur maximum du *jitter* fixée), ceci ne veut pas pour autant dire qu'on puisse considérer qu'il n'y a pas d'impact notable. En effet les périodes extrêmes servant au calcul de ce *jitter* sont : 5,0325ns et 4,9864ns. La première ne fait pas partie de la plage de tolérance à savoir 5±0,030ns.

C.6. Injections au niveau de l'oscillateur

Dans la section B.3.3, on a présenté l'architecture de l'oscillateur utilisé. Il exploite 4 cellules de retard. Vu le nombre important de nœuds d'injection, on s'est limité aux injections dans la première et la troisième cellule (on compte de gauche à droite par rapport à la Figure B-8). Chacune de ces injections représente une campagne à part.

Les nœuds d'injection qui en résultent sont donnés à la Figure C-16. Les nœuds Nx1 concernent la première cellule et ceux dénommés Nx3 concernent la troisième cellule.

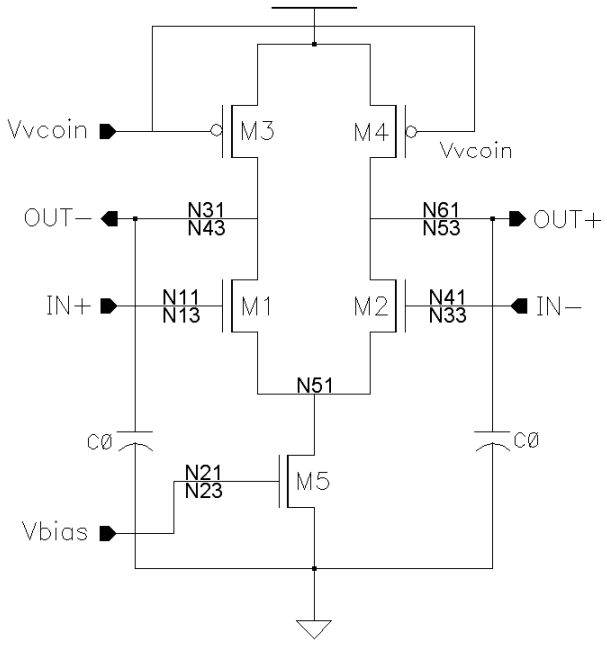
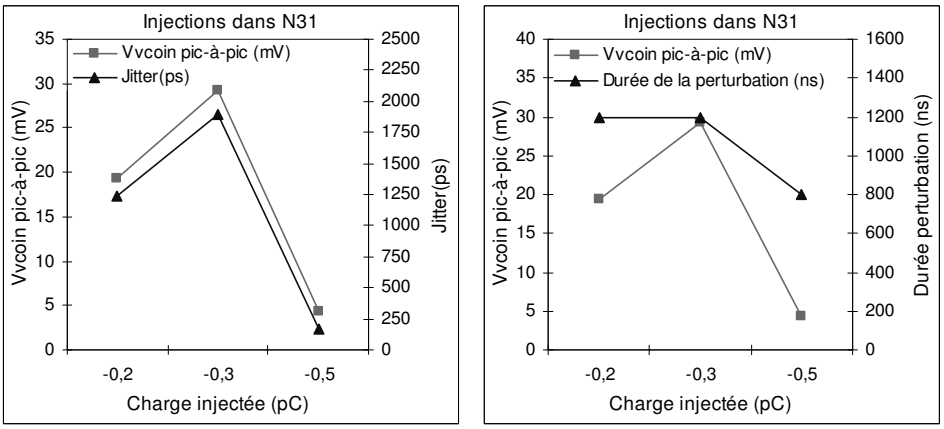


Figure C-16: Nœuds d'injection au niveau de l'oscillateur

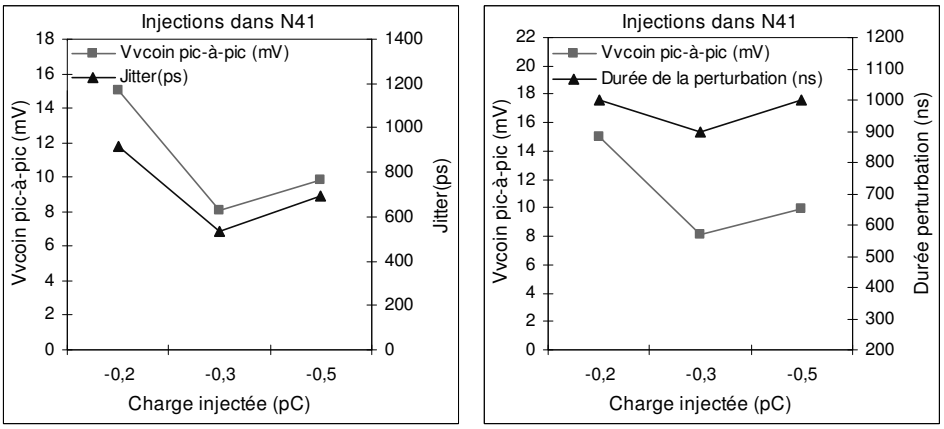
Concernant les résultats d'injections, seuls les nœuds N53 pour Qinj=0,2pC et N43 pour Qinj=0,2pC ou 0,3pC ne font pas sortir le *jitter* de sa plage de tolérance. Tous les autres cas donnent des *jitters* supérieurs à 60ps (le maximum relevé est de 2166ps). Les mêmes remarques que lors des autres campagnes sont valables ici, hormis les injections dans les nœuds N31 et N41 pour des charges négatives et le nœud N11 pour des charges positives. Le résultat de ces injections est donné à la Figure C-17. On s'était habitué à ce que la tension Vpp_vcoin_inj croisse lorsque Qinj augmente (en valeur absolue), or ici Vpp_vcoin_inj diminue pour certaines

charges même si Vpp_n_inj au niveau de chaque nœud augmente (en valeur absolue). Par contre le *jitter* ainsi que durée_pertu suivent l'évolution de Vpp_vcoin_inj.

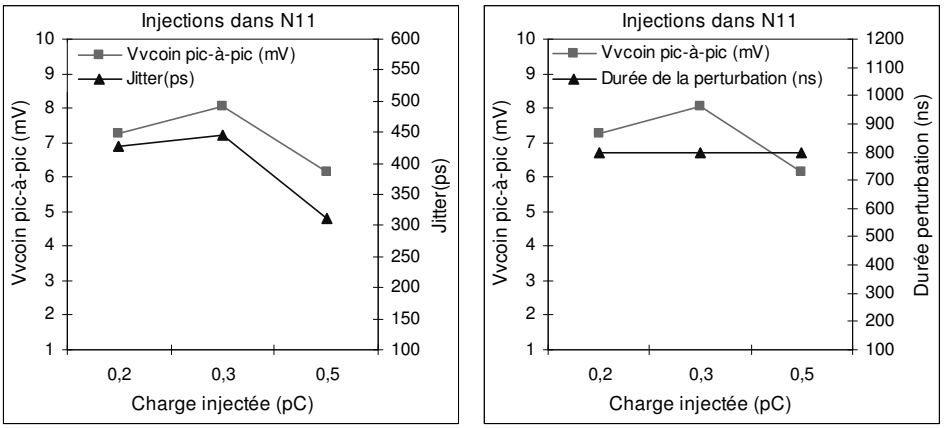
Pour tenter d'expliquer la raison de cette évolution de Vpp_vcoin_inj, il faudrait tenir compte du fait que l'oscillateur est un bloc purement analogique et que de ce fait les nœuds sont interdépendants plus que pour les autres campagnes.



(a)



(b)



(c)

Figure C-17: Evolution de Vvcoin pic-à-pic, du jitter et de la durée de la perturbation lors des injections dans l'oscillateur pour Qinj négatif dans les nœuds N31(a), N41(b) et Qinj positif dans le nœud N11(c)

RESUME

La probabilité des fautes transitoires augmente avec l'évolution des technologies. Plusieurs approches ont été proposées pour analyser très tôt l'impact de ces fautes sur un circuit numérique. Il est notamment possible d'utiliser une approche fondée sur l'injection de fautes dans une description VHDL au niveau RTL. Dans cette thèse, nous apportons plusieurs contributions à ce type d'analyse.

Un premier aspect considéré est la prise en compte de l'environnement du circuit numérique lors des campagnes d'injection. Ainsi, une approche basée sur une analyse de sûreté de fonctionnement multi-niveaux a été développée et appliquée sur un exemple. Les injections sont réalisées dans le circuit numérique décrit au niveau RTL alors que le reste du système est décrit à un niveau d'abstraction plus élevé. L'analyse des résultats montre que certaines défaillances apparaissant au niveau du circuit n'ont en fait aucun impact sur le système.

Nous présentons ensuite les avantages de la combinaison de deux types d'analyses : la classification des fautes en fonction de leurs effets, et l'analyse plus détaillée des configurations d'erreurs activées dans le circuit. Une campagne d'injection de fautes de type SEU a été réalisée sur un microcontrôleur 8051 décrit au niveau RTL. Les résultats montrent que la combinaison des analyses permet au concepteur de localiser les points critiques, facilitant l'étape de durcissement. Ils montrent également que, dans le cas d'un processeur à usage général, les configurations d'erreurs peuvent être dépendantes du programme exécuté. Cette étude a également permis de montrer que l'injection d'un très faible pourcentage des fautes possibles permet déjà d'obtenir des informations utiles pour le concepteur. La même méthodologie a été utilisée pour valider la robustesse obtenue avec un durcissement au niveau logiciel. Les résultats montrent que certaines fautes ne sont pas détectées par les mécanismes implémentés bien que ceux-ci aient été préalablement validés par des injections de fautes basées sur un simulateur de jeu d'instructions.

Le dernier aspect de cette thèse concerne l'injection de fautes dans des blocs analogiques. En fait très peu de travaux traitent du sujet. Nous proposons donc un flot global d'analyse pour circuits numériques, analogiques ou mixtes, décrits au niveau comportemental. La possibilité d'injecter des fautes dans des blocs analogiques est discutée. Les résultats obtenus sur une PLL, choisie comme cas d'étude, sont analysés et montrent la faisabilité de l'injection de fautes dans des blocs analogiques. Pour valider le flot, des injections de fautes sont également réalisées au niveau transistor et comparées à celles réalisées à haut niveau. Il apparaît une bonne corrélation entre les résultats obtenus aux deux niveaux.

MOTS-CLES

VLSI, descriptions comportementales, circuits numériques, circuits analogiques, analyse de sûreté, injection de fautes, SEU, SET

TITLE

DEPENDABILIY ANALYSIS OF COMPLEX CIRCUITS DESCRIBED IN A HIGH LEVEL LANGAGE

ABSTRACT

The probability of transient faults increases with the evolution of the technologies. Several approaches have been proposed to early analyze the impact of these faults in a digital circuit. It is in particular possible to use an approach based on the injection of faults in a RT-Level VHDL description. In this thesis, we make several contributions to this type of analysis.

A first considered aspect is to take into account the digital circuit's environment during the injection campaigns. So, an approach based on multi-level dependability analysis has been developed and applied to an example. The injections are performed in the digital circuit described at the RT-Level while the rest of the system is described at a higher level of abstraction. The results' analysis shows that failures appearing at circuit's level have in fact no impact on the system.

We then present the advantages of the combination of two types of analyses : classification of faults with respect to their effects, and a more detailed analysis of error configurations activated in the circuit. An injection campaign of SEU-like faults was performed on a 8051 microcontroller described at RT-Level. The results show that the combination of the two type analyses allows a designer to localize the critical points, facilitating the hardening stage. They also show that, in the case of a general processor, the error configurations can be dependent on the executed program. This study also demonstrates that injecting a very small percentage of the possible faults gives useful information to the designer. The same methodology has been used to validate the robustness obtained with a software hardening. The results show that some faults are not detected by the implemented mechanisms although those were previously validated by fault injections based on an instruction set simulator.

The last aspect of this thesis concerns the fault injection in analog blocks. In fact very few works cover this subject. We thus propose a global analysis flow for digital, analog or mixed circuits, described at behavioral level. The possibility to inject faults in analog blocks is discussed. The results obtained on a PLL, chosen as case study, have been analysed and show the feasibility of fault injections in analog blocks. To validate this flow, fault injections were also performed at transistor level and compared to those performed at high level. It appears a good correlation between the results obtained at the two levels.

KEYWORDS

VLSI, behavioral descriptions, digital circuits, analog circuits, dependability analysis, fault injections, SEU, SET

Laboratoire TIMA, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France.

ISBN : 2-84813-092-X **ISBNE : 2-84813-092-X**