Estimation à haut-niveau des dégradations temporelles dans les processeurs : méthodologie et mise en oeuvre logicielle

Clément Bertolini

N° d'ordre : 4968

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ BORDEAUX 1

ÉCOLE DOCTORALE DES SCIENCES PHYSIQUES ET DE L'INGÉNIEUR

Par Clément BERTOLINI

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : ÉLECTRONIQUE

ESTIMATION À HAUT-NIVEAU DES DÉGRADATIONS TEMPORELLES DANS LES PROCESSEURS : MÉTHODOLOGIE ET MISE EN ŒUVRE LOGICIELLE

Directeur de recherche : François MARC

Equipe d'accueil : CEA LIST / DACLE / LCE

Soutenue le 13 Décembre 2013,

devant la commission d'examen formée de :

Mme MANEUX Cristell	Professeur à l'Université Bordeaux 1	Présidente du jury
Mme. ANGHEL, Lorena	Professeur à l'Institut Grenoble INP	Rapporteur
M. GRANADO, Bertrand	Professeur à l'Université Pierre et Marie Curie	Rapporteur
M. MARC, François	Maitre de Conférences, HDR à l'Univ. Bordeaux 1	Directeur de thèse
M. HERON, Olivier	Ingénieur Chercheur au CEA LIST	Encadrant de thèse
M. VENTROUX, Nicolas	Ingénieur Chercheur au CEA LIST	Encadrant de thèse

À Marthou, ma grand-mère.

REMERCIEMENTS

Je tiens à remercier tout d'abord l'ensemble des laboratoires du CEA LIST DACLE. Je pense aux ingénieurs, aux chercheurs et aux techniciens avec qui j'ai partagé mon travail, mes activités sportives et mes sorties pendant ces dernières années. Ensuite, mes remerciements vont à mes encadrants Olivier HERON et Nicolas VENTROUX qui m'ont ouvert les portes au monde de la recherche industrielle. Je remercie mon directeur de thèse pour ses avis aiguisés sur la signification scientifique de mon travail. J'ai commencé mon aventure de thèse avec Tushar, dont je suis reconnaissant pour son accueil et son ambition résistante à toute épreuve. Je salue aussi Patrick et Lukas, qui m'ont aidé dans les meilleurs moments comme dans les pires. Enfin, ma famille a beaucoup compté et je salue Maman, Papa, Tatie Manique, Géraldine, Espérence, Janine, Béatrice, Pierrot, Vincent, Tatie Huguette, Tonton Fady, Tatie Chantal, Tonton Jean-Claude, Sacha, Sébastien, Lionel. Mamie Marthou, tu es là pour nous tous et je te dédie ces travaux.

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GÉNÉRALE

De nos jours, les Circuits Intégrés (CI) numériques sont utilisés de plus en plus intensivement dans les systèmes embarqués. Les demandes de fonctionnalités et de performances des processeurs croissent et imposent un cahier des charges de plus en plus exigeant. Pour cela, il est courant de concevoir des Systèmes Multi Processeurs sur Puce ou MultiProcessor System-on-a-Chip (MPSoC) en anglais. Ils intègrent de multiples processeurs et mémoires sur une seule puce électronique, ce qui permet de bonnes performances et répond à la demande de fonctionnalité.

Nous définissons le vieillissement comme la variation des caractéristiques temporelles d'un CI pendant sa durée de vie. Actuellement, les méthodes de conception des MPSoC intègrent le vieillissement en ajoutant des marges sur la fréquence maximale de fonctionnement du produit final notamment. À cause de l'augmentation de l'occurrence des défaillances matérielles, les marges sur les performances des produits actuels ont été augmentées et deviennent trop importantes. Pour réduire ces marges et continuer à proposer des produits compétitifs, il devient nécessaire de prendre en compte le vieillissement pendant la conception, dès les premières phases. Les techniques actuelles de conception des MPSoC se basent sur des simulations à haut niveau d'abstraction. Les effets de la conception sur le vieillissement est un sujet encore au stade de recherche et nécessite une validation des modèles de dégradation à haut niveau.

Nous défendons l'idée dans cette thèse que l'estimation à haut niveau de la dégradation des temps de propagation des processeurs permet d'estimer le vieillissement des MPSoC. Notre méthodologie s'intègre dans les outils de conception des MPSoC et elle est validée pour un ensemble d'applications. Les phénomènes de dégradation étudiés dans cette thèse sont l'Instabilité de Température de Grille ou Negative Bias Temperature Instability (NBTI) en anglais et l'Injection de Porteurs Chauds ou Hot Carrier Injection (HCI) en anglais ; ils contribuent tous les deux au vieillissement du circuit. Ces phénomènes sont basés sur les transistors et les MPSoC en contiennent jusqu'à plus d'un milliard. Nous étudions les méthodes à haut niveau d'abstraction pour la rapidité de simulation et ainsi permettre l'exploration architecturale des MPSoC.

Objectifs

Notre objectif est de proposer une méthodologie basée sur la propagation du modèle de dégradation du HCI établi au niveau porte logique, qui puisse simuler le vieillissement des processeurs au niveau fonctionnel. Pour cela, nous définissons une nouvelle méthodologie d'estimation à

haut niveau des dégradations temporelles des CI. Nous validons notre méthode et donnons les résultats de dégradation des marges temporelles pour un processeur embarqué.

Organisation du manuscrit

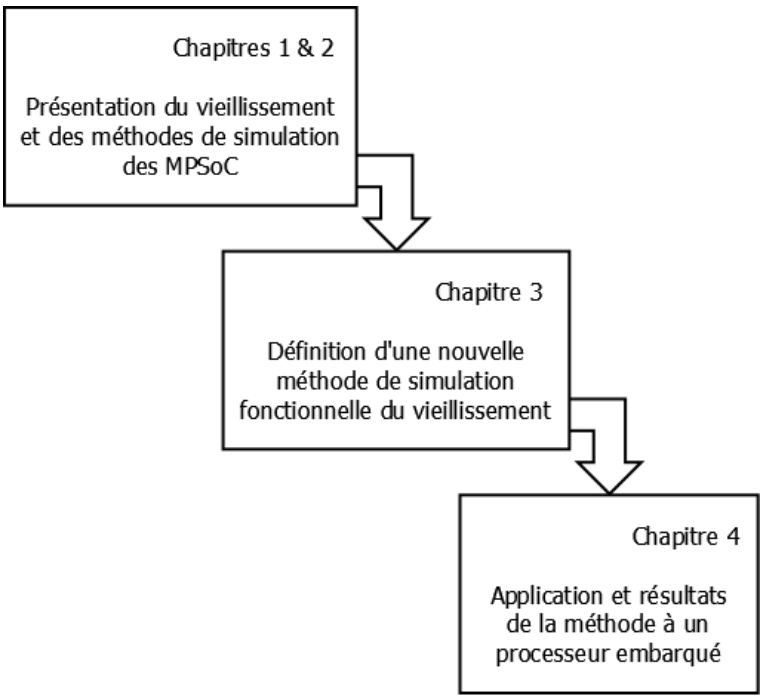
Le manuscrit comporte quatre chapitres et son déroulement est représenté sur la figure ci-contre. Le premier chapitre présente le contexte du vieillissement dans les CI. Les notions de vieillissement des transistors sont expliquées par les mécanismes NBTI et HCI de la technologie Complementary Metal-Oxide-Semiconductor (CMOS). Ensuite, nous présentons les effets du HCI sur une porte logique puis sur un processeur. L'extension au NBTI est possible avec un paramètre différent des modèles de dégradation et ne modifie pas la suite de l'étude. Nous démontrons que le vieillissement est fonction de l'évolution des temps de propagation des signaux internes du circuit. Le paramètre clé est la proximité des temps de propagation à un seuil critique, défini par le concepteur pour une application donnée. Le respect du seuil critique est nécessaire au bon fonctionnement du circuit pendant sa durée de vie. L'enjeu de l'estimation des dégradations temporelles des MPSoC est la simulation des temps de propagation.

Le deuxième chapitre dresse l'état de l'art des méthodes de simulation des dégradations. On évalue en particulier les performances de simulation du vieillissement pendant la conception des MPSoC. Pour cela, nous comparons la rapidité de simulation par rapport à la précision des résultats. Nous verrons qu'il n'existe pas de modèle pour simuler les dégradations au niveau fonctionnel, qui soit validé.

Nous proposons alors une nouvelle méthode décrite dans le troisième chapitre, permettant de simuler les dégradations d'un CI à haut niveau. En particulier, une technique de simulation fonctionnelle est mise au point. Elle se base sur un modèle au niveau porte logique des chemins d'un circuit. Le jeu d'instructions sert de base pour construire le modèle de dégradation d'un circuit.

Le quatrième chapitre met en application la technique précédente sur un cœur de processeur embarqué. Nous présentons ces caractéristiques et la plateforme expérimentale de simulation utilisée pour obtenir les résultats de dégradations. Nous présentons aussi la validation de la méthode avec différents benchmarks. La précision et la performance de notre méthode de simulation sont chiffrées. Enfin, nous donnons les résultats d'analyse temporelle du processeur. Nous identifions les instructions les plus agressives pour le vieillissement et les parties les plus sensibles du circuit.

Enfin, nous concluons nos travaux et donnons les perspectives d'évolution à court terme et à long terme.



Composition du manuscrit.

CHAPITRE 1

DÉGRADATIONS TEMPORELLES DANS LES SYSTÈMES SUR PUCE

La dégradation temporelle des System-on-Chip (SoC) est reliée à la dégradation de chaque transistor. Par exemple, pour les MPSoC, l'estimation des dégradations temporelles nécessite de traiter un grand nombre de transistors. L'objectif de ce chapitre est de présenter l'effet des dégradations au niveau des transistors et au niveau du système.

Plus précisément, nous présentons les principaux mécanismes de dégradation de la technologie Metal-Oxide-Semiconductor (MOS) et leur effets sur les transistors jusqu'au système. On présente les origines et les conséquences des dégradations. Ceci permet de comprendre les techniques de simulation des dégradations qui seront présentées au chapitre 2. Certaines se basent sur le mécanisme de dégradation du transistor [Bor+03] alors que d'autres sont à des niveaux d'abstraction plus élevés [Xia+10 ; Gup11].

Nous présentons tout d'abord le transistor Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) et l'impact de sa miniaturisation sur les paramètres électriques. Ensuite, nous décrivons les différents mécanismes de dégradation dans le transistor. Enfin, nous expliquons les effets de ces mécanismes sur les paramètres temporels d'une cellule logique et du circuit.

1.1 Miniaturisation des transistors

La réduction du facteur d'échelle a deux principales conséquences sur les transistors MOSFET. Les performances électriques et temporelles augmentent ; d'un autre côté, les mécanismes de défaillance sont favorisés. Nous commençons par décrire les paramètres électriques les plus sensibles à la réduction du facteur d'échelle.

1.1.1 Constitution et fonctionnement des transistors MOSFET

Le transistor MOSFET est l'élément de base des circuits intégrés MOS. Il existe deux types de transistors MOSFET : les Transistors à effet de champ MOS à canal N (nMOS) et les Transistors à effet de champ MOS à canal P (pMOS). La figure 1.1 présente une vue en coupe d'un transistor MOSFET à canal N. Un courant peut circuler du Drain jusqu'à la Source en passant par le Canal, sous certaines conditions que nous décrivons ci-dessous. Pour un transistor à canal N, les régions

de la Source et du Drain contiennent du silicium dopé N. La grille est en métal séparée de la région dopée P par la couche d'oxyde, de volume $W \times L \times t_{ox}$.

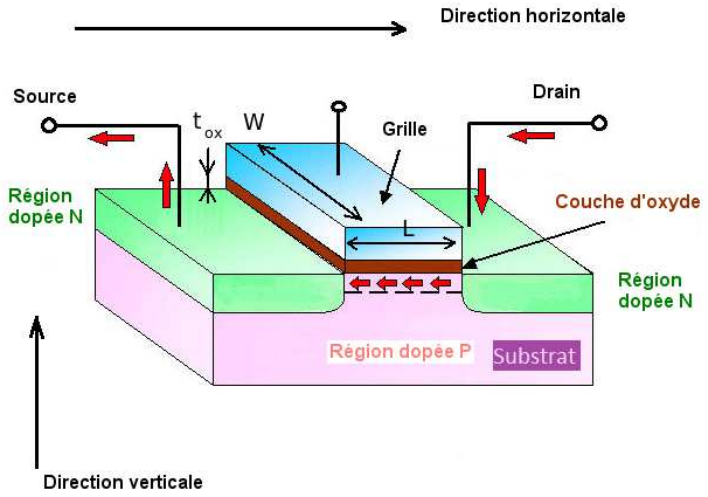


Figure 1.1 – *La Source, le Drain et la Grille sont les trois composants semi-conducteurs du transistor MOS. Un courant peut circuler selon la direction des flèches rouges, en fonction de la polarité de la Grille. [SH04]*

La tension de seuil du transistor est appelée V_t ; elle impose un seuil à la tension V_{gs} prise entre la source et la grille du transistor. Lorsque $V_{gs} > V_t$, un courant peut traverser le canal (fermé) entre le Drain et la Source. Lorsque ce courant n'est limité que par la tension de grille, on dit que le transistor est saturé. À l'inverse, lorsque $V_{gs} < V_t$, le courant Drain Source est nul ; le canal est ouvert.

La tension de seuil V_t dépend des matériaux constituant le transistor, du dopage des régions du Substrat, du Drain et de la Source, de leur géométrie, des défauts dans l'isolant [SH04] et de la température.

1.1.2 Tendances de la miniaturisation

La miniaturisation permet l'accroissement des performances des CI. Un plus grand nombre de transistors sont intégrés pour une même surface (augmentation de la densité d'intégration). Cette tendance permet également d'accroître la fréquence de commutation des transistors.

La vitesse de miniaturisation suit la loi de Moore, qui prévoit une augmentation par un facteur deux du nombre de transistor d'une puce tous les 18 mois [HP06]. La miniaturisation ne se fait pas dans un contexte idéal : les tensions ne diminuent pas à la même vitesse que l'épaisseur d'oxyde (figure 1.2). Sur ce graphique trois données sont représentées : la tension d'alimentation V_{dd}, la tension de seuil V_t et l'épaisseur d'oxyde t_{ox}. L'axe des abscisses montre la longueur de canal.

Sur le graphique, le facteur de réduction est représenté par la pente unité, de valeur $L = 0,7$. On remarque que l'épaisseur d'oxyde diminue à une vitesse proche de la loi de Moore. Par contre les tensions V_{dd} et V_t ont des vitesses inférieures car les pentes sont moindres.

Ceci a pour effet d'augmenter les champs électriques verticaux et latéraux. C'est une conséquence de la relation $E = V_{dd}/t_{ox}$ où E est le champ électrique, V_{dd} la tension d'alimentation

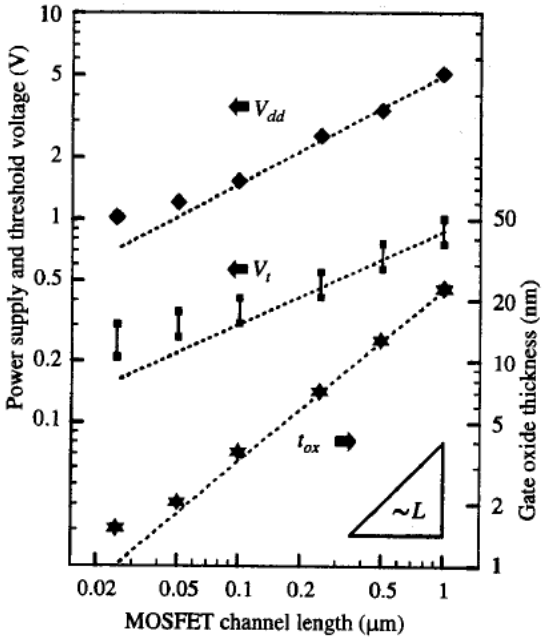


Figure 1.2 – *Évolution des tensions et de l'épaisseur d'oxyde d'un transistor en fonction de la longueur du canal. Les pentes d'évolution en pointillé sont obtenues avec les points entre* $0,1\,\mu m$ *et* $1\,\mu m$*. Le triangle en bas à droite montre la pente unité, correspondant à la proportionnalité avec la longueur du canal. [Ste08]*

et t_{ox} l'épaisseur d'oxyde. Les phénomènes de dégradation des transistors tels que l'injection de porteurs chauds (HCI) apparaissent.

1.2 Vieillissement dans les systèmes sur puce

Dans cette section, nous définissons le vieillissement des SoC. Les principaux mécanismes de défaillance (le Time-Dependent Dielectric Breakdown (TDDB), le NBTI, le HCI et l'électromigration EM) sont introduits ; ils contribuent au vieillissement des SoC. Nous étudions le détail des mécanismes NBTI et HCI car ce sont les phénomènes de dégradations qui contribuent le plus au vieillissement. Enfin, nous résumons les conséquences des dégradations sur les paramètres électriques des transistors.

1.2.1 Définition du vieillissement

Le vieillissement d'un système est défini comme l'augmentation de son taux de défaillance [KK07]. Le taux de défaillance λ, exprimé dans l'unité Failure in Time (FIT), a pour valeur la probabilité que le système fonctionne à la date $t + dt$ sachant qu'il fonctionne à la date t. Il a été constaté que le taux de défaillance d'un SoC suit une loi d'évolution en fonction du temps comme décrit figure 1.3. Pendant les premières heures d'utilisation, le taux de défaillance est élevé car il se peut que des produits contenant des défauts de fabrication n'aient pas été mis de côté. Puis le système ayant passé cette période, il est probable qu'il n'a pas de défaut de fabrication et le taux de défaillance est constant. Seul l'environnement extérieur peut entraîner des défaillances du système [Gup+09a]. Enfin, après une durée d'utilisation, le vieillissement apparaît et entraîne des défaillances du système. Le taux de défaillance augmente alors rapidement.

Les mécanismes de dégradation expliqués ci-après contribuent au taux de défaillance lors de la période de fin de vie du système.

1.2.2 Principaux mécanismes de défaillance des transistors MOS

Les mécanismes de défaillance de la technologie MOS connus à ce jour sont le claquage d'oxyde dans le temps ou TDDB, les phénomènes liés aux porteurs chauds (Hot Carriers) et les instabilités de température (Bias Temperature Instability). Les lignes d'interconnexions peuvent souffrir de l'électromigration (EM) ou de migration sous contrainte (Stress migration). Pour l'industrie, les modèles issus de [JED10] sont souvent utilisés car ils permettent de relier directement l'effet de la tension d'alimentation et de la température sur ces phénomènes. Seul le TDDB ne dispose pas de modèles de ce type, car son apparition est décrite par des grandeurs statistiques. Nous ne l'étudions pas dans la suite car dans le cas particulier d'un transistor d'un circuit donné, les modèles actuels ne permettent pas de calculer sa dégradation. Enfin, l'électromigration concerne les connexions électriques entre les transistors, qui ne sont pas l'objet de notre étude. Nous décrivons donc le détail des mécanismes d'apparition du HCI et du NBTI ci-dessous.

1.2.2.1 Instabilité de Température de Grille (NBTI)

Le NBTI est un phénomène d'usure des transistors MOSFET à canal P. Des états d'interface ΔN_{it} sont créés à la frontière entre le silicium et l'oxyde de silicium d'une part. D'autre part, des défauts apparaissent dans le substrat sous la forme de charges positives. Ces défauts sont générés lorsque le champ électrique vertical est faible ($E_{ox} < 6\,\text{MV/cm}$ à $10\,\text{MV/cm}$), le transistor est en inversion ($V_g < 0\,\text{V}$) et que la température est élevée (entre $30\,^{\circ}\text{C}$ et $200\,^{\circ}\text{C}$).

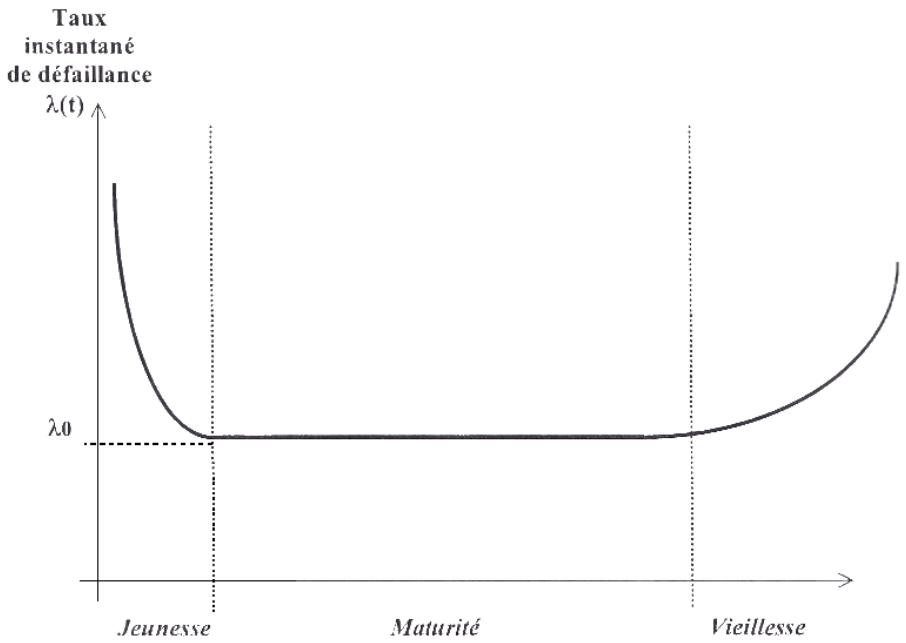


Figure 1.3 – *Évolution du taux de défaillance en fonction du temps d'utilisation d'un SoC. La mortalité infantile apparaît tôt dans la phase de vie, suivie par une période utile et finalement les effets du vieillissement apparaissent en fin de vie. [DoD98]*

Dans les années 2000, le modèle de dégradation utilisé par les industriels (JEDEC [JED10]) pour l'extrapolation des durées de vie était une loi puissance de coefficient $0,25$. Ensuite, le modèle a été amélioré en prenant en compte la saturation et la relaxation de la dégradation. Les durées de vie ainsi extraites deviennent alors plus proches de la réalité. Nous détaillons comment ces estimations peuvent être calculées.

Les nouveaux procédés de fabrication rendent les effets du NBTI dominants pour les technologies à largeur de grille inférieure à 130 nm. Le canal P des transistors est disposé d'une manière qui implique une augmentation des champs électriques dans ces transistors et le traitement de la surface Si–SiO_2 rend la dégradation plus importante. Aussi, la réduction non linéaire des tensions d'alimentation augmente les champs électriques, qui eux-mêmes contribuent ensuite à l'élévation en température. On retrouve les conditions privilégiées d'apparition du NBTI décrite précédemment. Enfin, la diminution de la consommation électrique augmente l'effet de la dégradation.

Deux régions sont impliquées dans la dégradation, le canal en inversion et le débordement des régions drain-source sur la région de la grille, qui sont décrites figure 1.4.

Proche du canal en inversion, le champ électrique dépend des propriétés semi-conductrices des matériaux alors que dans les régions de débordement, ce sont les effets de bord du champ électrique qui opèrent. Il y a donc une forte dépendance du phénomène NBTI avec la géométrie du transistor. Deux phénomènes générant les états d'interface ont été distingués selon les liaisons chimiques qui entrent en jeu. Lorsque les tensions de stress V_g sont élevées, plusieurs phénomènes physiques sont activés et créent des dégâts permanents de l'interface. Il a aussi été démontré que le NBTI est un phénomène impliquant des trous en équilibre thermique et non accélérés à travers le canal [Str+09].

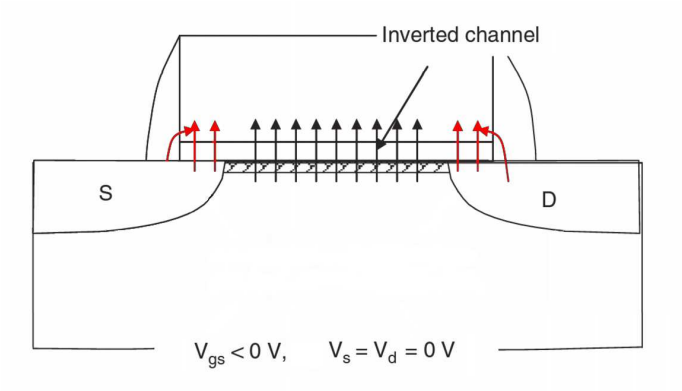


Figure 1.4 – *Régions d'apparition des dégradations dues au NBTI dans un transistor MOSFET à canal P en inversion : débordement des jonctions Source et Drain sur la Grille (flèches rouges) et Canal en inversion (flèches noires). [Str+09]*

Les mécanismes physiques et chimiques impliqués dans la dégradation NBTI concernent les états d'interface d'une part et les défauts d'autre part. Ces deux phénomènes sont représentés figure 1.5 a). La phase de relaxation du NBTI est décrite figure 1.5 b).

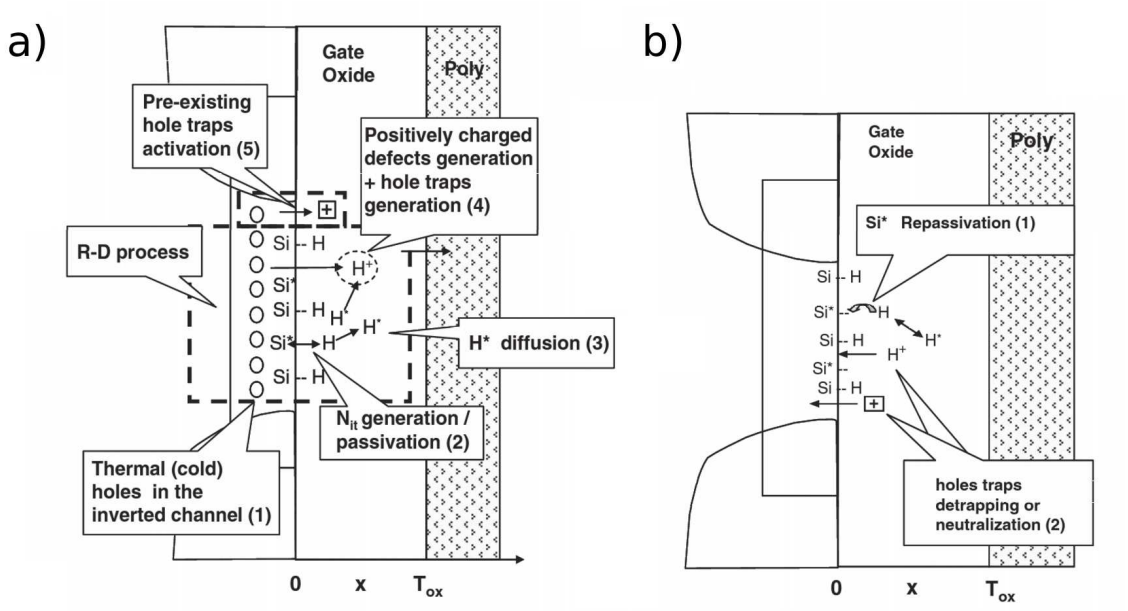


Figure 1.5 – *a) Génération des états d'interface dû aux trous dans le canal par réaction Si–H et diffusion d'ions H^+. Les défauts chargés positivement sont générés et activés le premier processus. b) Relaxation du NBTI soit par repassivation des ions silicium soit par élimination ou neutralisation des pièges à trous. [Str+09]*

La nature du phénomène NBTI est complexe et différents types de stress doivent être considérés. Ici, nous considérons des alternances transistor bloqué - transistor saturé. Dans ce cas, il a

été montré que deux phénomènes participent à la création de charges positives et impliquent une phase de relaxation du phénomène NBTI [Hua+03]. D'une part des états d'interface ΔN_{it} sont créés et d'autre part, des défauts sous la forme de charges positives sont activés ou générés. Ces derniers possèdent une composante permanente et une composante réversible.

Les états d'interface influent sur la mobilité des porteurs en surface du canal et la tension de seuil augmente. Cela entraîne des déviations du courant de saturation ou de la transconductance en fonction de l'application que l'on fait du transistor.

Pour résumer, la génération d'état d'interface est une fonction puissance du temps de stress. La température a un effet sur le vieillissement qui suit la loi d'Arrhenius. Les énergies d'activation sont de l'ordre de $0,1$ eV à $0,3$ eV pour les états d'interface et de $0,03$ eV pour les pièges à trous. Enfin, le rapport cyclique est un paramètre majeur qui influe sur les états d'interface. Un résumé des effets combinés des états d'interface et des pièges à trous est présenté pour chaque paramètre p sur le tableau 1.1.

Paramètre	t	E_{ox}	Rapport Cyclique	I_d	T
Influence sur Δp	↗	↗	↗	↘	↗

Tableau 1.1 – *Sens de variation des déviations des paramètres électriques Δp d'un transistor soumis au NBTI en fonction de ses conditions au repos et du temps de stress t. [Str+09]*

Le NBTI est constamment actif dans les circuits [Hof+10] et dépend de la température, du rapport cyclique en entrée des transistors et des tensions.

1.2.2.2 Injection de Porteurs Chauds (HCI)

Les porteurs, qu'ils soient des électrons ou des trous peuvent circuler avec une grande énergie cinétique lorsqu'ils traversent de forts champs électriques. Plusieurs mécanismes contribuent à augmenter leur énergie. Quand l'énergie moyenne des porteurs devient beaucoup plus grande que l'énergie associée au réseau de silicium, on les qualifie de « chauds ». Les porteurs chauds peuvent gagner assez d'énergie pour être injectés dans l'oxyde de grille ou causer des dégâts à l'interface et rendre instable les caractéristiques électriques du transistor. Les porteurs chauds peuvent être générés de différentes manières et nous présentons ici le cas du processus d'avalanche dans l'oxyde de grille. Pour les applications CMOS, les transistors fonctionnent dans le mode saturé ou bloqué. Les porteurs chauds sont alors générés en grande partie dans le canal, lorsque le transistor est en phase de conduction. Le courant circulant dans le canal est maximal lors des commutations, c'est donc à ce moment que la dégradation des porteurs chauds est maximale.

La tension V_{ds} est prise entre la source et le drain du transistor. Sa valeur de saturation est notée V_{dsat}. On définit la phase de conduction saturée d'un transistor l'état pour lequel $V_{gs} > V_t$ et $V_{ds} > V_{dsat}$. Nous considérons maintenant le cas d'un transistor dopé N dans sa phase de conduction saturée.

Le phénomène de dégradation des porteurs chauds du canal est un processus en trois étapes. Premièrement, il y a gain d'énergie dans le canal, puis impact ou injection des porteurs énergétiques. Ensuite, il y a génération de dégâts physiques ou de pièges dans l'oxyde de grille, ou à son interface avec le canal. Dans la majorité des cas, les porteurs chauds finissent leur parcours dans le drain mais une faible portion peut être déplacée vers l'interface avec suffisamment d'énergie

pour être injectée dans l'oxyde ou créer des états d'interface. Les dégâts modifient les propriétés électriques du transistor (par exemple la tension de seuil V_t) et peuvent affecter la fonctionnalité d'un circuit intégré.

Les principes physiques en jeu dans l'injection des porteurs chauds parcourant le canal sont illustrés figure 1.6. Cinq processus participent à la dégradation.

1. Les dégâts des porteurs chauds sont générés le long du canal dans un MOSFET en inversion ($V_{gs} > V_t$). Après le point de pincement du canal, certains porteurs ont reçu beaucoup d'énergie. Le réchauffement des porteurs est donc directement relié à l'arrivée de la condition de saturation ($V_{ds} > V_{dsat}$). La tension $V_{dsat} - V_{ds}$ est appliquée sur une faible distance proche de la fin du drain. Le champ électrique créé est très important et les porteurs situés dans la région non pincée ont leur distribution d'énergie déséquilibrée.
2. Les porteurs énergétiques perdent leur énergie par ionisation et contribuent au courant de substrat.
3. Les porteurs chauds ont suffisamment d'énergie pour casser les liaisons Si–H à l'interface. Des liaisons incomplètes apparaissent et piègent les porteurs issus de la bande d'énergie interdite. On les nomme *états d'interface*.
4. Les porteurs chauds peuvent aussi dépasser la barrière d'énergie de l'interface Si–SiO_2 et une fois injectés dans l'oxyde, peuvent y être piégés.
5. Les dégâts induits par l'injection des porteurs chauds du canal impliquent une modification des paramètres du transistor MOSFET. L'instabilité des paramètres du transistor peut induire des pannes à long terme pour des circuits utilisant de tels transistors.

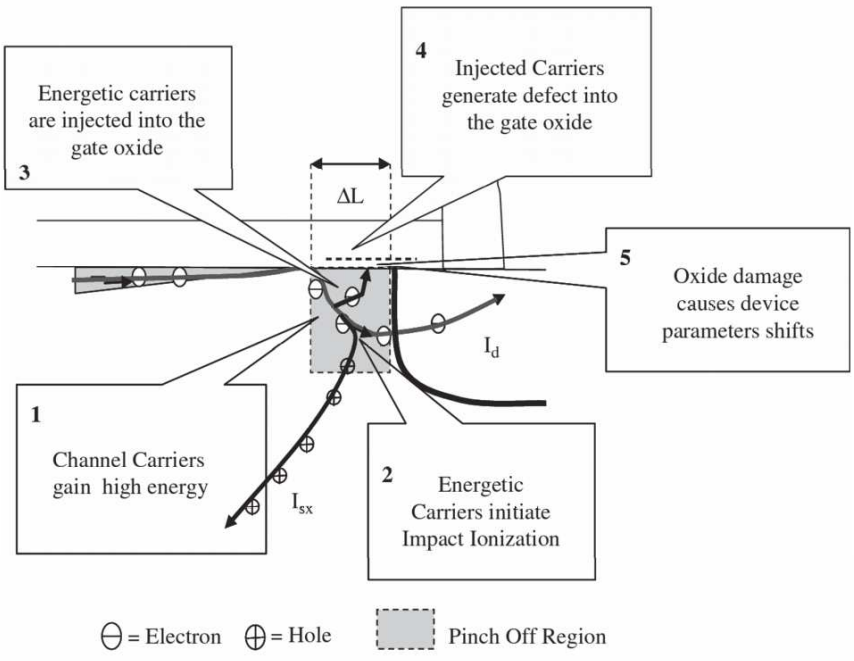


Figure 1.6 – *Étapes du processus d'injection de porteurs chauds (HCI) dans un transistor MOSFET à canal N en saturation. La région de pincement a une longueur Ls. [Str+09]*

Le champ électrique est le principal acteur dans la génération de porteurs chauds. Il régule aussi leur injection dans l'oxyde de grille. Selon la polarisation dans la région de saturation, le champ électrique latéral dans la zone de pincement réchauffe les porteurs du canal, alors que le champ vertical favorise ou restreint l'injection de trous ou d'électrons dans l'oxyde en modulant la barrière de potentiel de l'interface Si–SiO_2.

Dans la zone non pincée d'un transistor MOSFET en saturation, le champ électrique s'étend sur deux dimensions. Le calcul précis du champ latéral près du drain est très complexe car il s'agit de résoudre simultanément une équation de transport de courant et d'une équation de Poisson. Aussi, la réduction du facteur d'échelle a imposé d'utiliser différents agencements internes des transistors pour respecter un compromis entre performance et fiabilité. Ainsi, l'évaluation du champ électrique dans la zone de pincement dépend fortement de la configuration des jonctions source/drain.

En résumé, le HCI apparaît lors des commutations du transistor. À la différence du NBTI, le HCI apparaît dans la zone de déplétion, et dépend de l'activité du signal d'entrée du transistor. De plus, il a été observé que le phénomène HCI apparaît seulement pour des commutations fréquentes [Hof+10]. Seulement à partir de 50 % d'activité, on observe les effets dus au HCI figure 1.7 pour un temps de stress de 10^6 s. Sur cette figure, on peut voir la variation du fonctionnement d'un circuit, selon trois types de stimuli en comparant la dégradation de fréquence après un temps de stress de 10^6 s, qui est identique pour les activités faibles (0,01 % et 6,25 %) mais plus marquée pour une activité de 50 %.

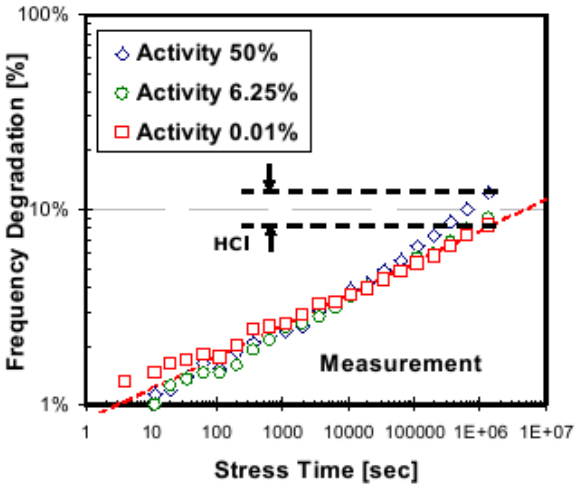


Figure 1.7 – *Dégradation en fréquence d'un oscillateur en anneau pour une technologie de* 40 *nm. Trois courbes en fonction du temps de stress sont obtenues avec une activité des signaux de* 0,01 %*,* 6,25 % *et* 50 %*. L'effet du HCI apparaît pour une activité de* 50 % *et au-delà. [Hof+10].*

Nous comparons maintenant les effets du HCI et du NBTI sur les transistors MOSFET.

1.2.2.3 Comparaison entre HCI et NBTI

L'étude présentée dans [Str+09] a déterminé que le HCI est dominant à température ambiante (30 °C) pour les transistors nMOS alors que le NBTI et le HCI sont majeurs à haute température

(125 °C). Sur la figure 1.8, on observe les effets du NBTI et du HCI sur chaque type de transistor par les trois courbes empilées, sauf pour le cas du NBTI sur transistor nMOS, pour lequel le NBTI n'apparaît pas.

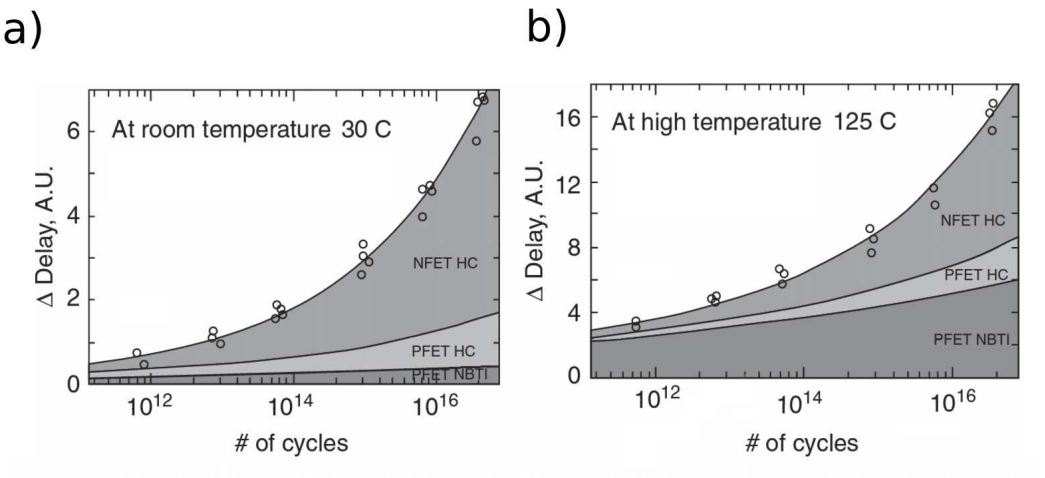


Figure 1.8 – *Variations de délais d'oscillateurs en anneaux en fonction du nombre d'oscillation (cycles) pour deux températures de stress. Les effets du HCI et du NBTI sur les transistors pMOS et nMOS sont distingués. [Str+09]*

On remarque sur la figure 1.8 que la dégradation du temps de propagation due au NBTI sur les transistors pMOS est importante pour une température de 125 °C ainsi que sur les transistors nMOS pour le HCI. Aussi, il y a très peu de dégradation des transistors pMOS à température ambiante (graphe a) alors qu'à haute température (graphe b), l'amplitude de la dégradation est comparable à celle du HCI. Ainsi, le NBTI et le HCI peuvent avoir des effets du même ordre de grandeur sur des circuits CMOS.

Conscients des effets combinés du HCI et du NBTI sur les circuits, leurs modélisations les plus récentes imposent des contraintes sur la simulation. Le NBTI comprend une phase de relaxation à chaque changement d'état et son modèle nécessite un ensemble de paramètres à plusieurs instants. Le HCI ne dépend que des chengements d'état sans distinction de leur moment d'apparition. Par rapport à celui du HCI, le modèle du NBTI est donc beaucoup plus volumineux. Pour garder des méthodes de simulation rapides et remplir l'objectif de la thèse, il est nécessaire que le modèle soit le plus simple possible, et c'est pourquoi nous choisissons de décrire uniquement les modèles du HCI pour un transistor dans la suite du document. Les études de modélisation du NBTI sont encore au stade de recherche et d'autres études doivent être menées pour définir simplement le modèle du NBTI d'un circuit.

1.3 Modèle de dégradation du HCI dans le transistor

Le paragraphe précédent a permis de comprendre les effets du HCI et son impact dans la dégradation des paramètres du transistor. Nous présentons dans cette section un modèle issu de la littérature qui permet de quantifier les effets de la dégradation HCI sur le transistor.

Les modèles utilisés actuellement pour estimer les effets de la dégradation due au HCI sont basés sur d'anciennes technologies de fabrication des transistors [Pag03]. Le HCI est un mécanisme de défaillance paramétrique dont l'effet sur le circuit peut être exprimé de façon déterministe [SH04]. Historiquement, un des premiers modèles pour la défaillance HCI représente l'amplitude de la dégradation d'un paramètre physique p en fonction du temps de stress t par deux coefficients empiriques A et n [JED10].

$$\Delta p = A \cdot t^n \tag{1.1}$$

Ce modèle suggère que la loi de dégradation de la tension de seuil d'un transistor en fonction du temps est une fonction puissance. Le temps de stress t dépend de l'utilisation du transistor, donc de son utilisation.

Des modèles plus précis ont été reportés plus récemment notamment le LEM (Lucky Electron Model) [Hu+85]. Le LEM est plus couramment utilisé dans l'industrie et suppose une relation entre les états d'interface, la déviation d'un paramètre p du transistor et la largeur de grille : $\Delta p \propto L_{eff}^{\alpha} \cdot \Delta N_{it}(t)$. Plus un transistor a une grille large et plus les effets dus au HCI sont importants.

De plus dans [Hu+85], la valeur des états d'interface est exprimée en fonction des courants de drain I_d, de substrat I_{sub}, de la largeur W de la grille et de la température T, comme on le voit dans l'équation 1.2.

$$\Delta p \propto L_{eff}^{\alpha} \left[t \cdot \frac{I_d}{W} \cdot \left(\frac{I_{sub}}{I_d} \right)^m \right]^n exp(-\Delta H/(kT)) \tag{1.2}$$

k est la constante de Boltzmann, m et n sont des constantes technologiques, t est le temps de stress et ΔH un paramètre empirique modélisant l'effet de la température, de valeur négative.

Ainsi, on peut estimer l'impact des paramètres du transistor sur sa sensibilité au HCI (tableau 1.2). On remarque que seul le courant I_d est directement relié à l'utilisation du transistor. Pour des circuits numériques, les états du transistor sont saturé et bloqué et dépendent de l'activité du circuit. Il faut donc connaître le signal logique en entrée du transistor pendant le temps d'utilisation pour déterminer la dégradation due au HCI.

Paramètre	W	L	I_d	I_{sub}	T
Influence sur Δp	↘	↗	↘	↗	↘

Tableau 1.2 – *Sens de variation des déviations Δp des paramètres électriques d'un transistor soumis au HCI en fonction de ses conditions au repos. [Str+09]*

En conclusion, nous avons vu deux des principaux modèles permettant de quantifier la dégradation d'un paramètre électrique du transistor. Il est montré que le HCI est lié à l'activité du circuit [Hof+10]. C'est pourquoi nous explorons le lien entre activité du circuit et HCI dans la suite du document. Des améliorations ont été apportées pour tenir compte de la température et de la géométrie des transistors et sont utilisées dans l'industrie [Hu+85].

1.3.1 Effet sur les temps de propagation d'une porte logique CMOS

Nous montrons ici l'impact du HCI sur une porte logique simple. Nous prendrons l'exemple d'une porte inverseuse CMOS.

Il a été montré que la mesure des temps de propagation peut être reliée au vieillissement des transistors [Aga+07]. Nous adoptons une approche similaire, en calculant l'impact de la dégradation HCI sur les temps de propagation des portes logiques CMOS. Pour cela, nous étendons les principales conséquences des dégradations des transistors à une porte logique CMOS. D'une part, le temps de propagation d'une porte est exprimé en fonction des paramètres technologiques. D'un autre côté, l'influence des paramètres environnementaux sur ce temps de propagation permet de comprendre les effets des conditions d'utilisation.

Dans cette section, nous calculons le temps de propagation d'un circuit en fonction de la dégradation HCI. D'abord, nous présentons le modèle d'une porte logique. Ensuite, nous analysons les effets du HCI sur les conditions de fonctionnement.

1.3.1.1 Temps de propagation

On modélise la grille des transistors par une capacité et les interconnexions par des circuits RC. En regroupant les capacités du réseau et de la grille, le modèle d'un inverseur est déterminé sur le schéma électrique figure 1.9.

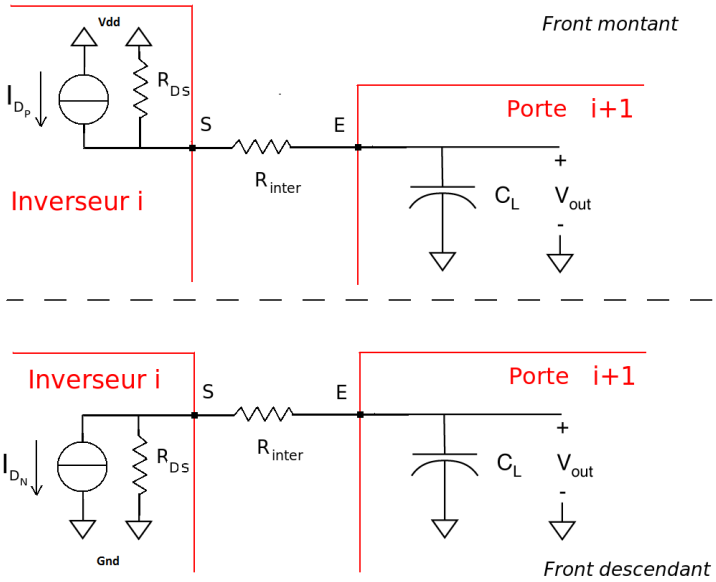


Figure 1.9 – *Modélisation d'une porte inverseuse chargée par une autre porte lors d'une transition en sortie. La source de courant et la résistance R_{DS} modélisent le transistor MOS lors d'une transition. L'interconnexion est représentée par sa résistance R_{inter} et la porte i+1 par sa capacité d'entrée C_L.*

Définition : On appelle temps de propagation d'une porte logique le temps mis par la sortie pour atteindre 50 % de sa valeur finale, mesuré à partir du moment où l'entrée a atteint 50 % de sa valeur finale [Sap04].

Lors d'une transition de la sortie de l'inverseur numéroté i, le modèle représenté figure 1.9 définit une charge/décharge d'un condensateur à courant constant selon le sens de la transition. La porte inverseuse numérotée i est modélisée par une source de courant I_D, R_{DS} chargée par une capacité C_L représentant l'impédance d'entrée de la porte $i+1$. La résistance R_{inter} du fil de

connexion entre les portes est négligée par rapport à la résistance R_{DS}. On peut calculer le temps τ_r de montée du signal en sortie pour atteindre sa valeur finale lors d'une transition montante en sortie et τ_f dans le cas d'une transition descendante.

$$\begin{aligned} \tau_r &= \frac{1}{2} \cdot V_{dd} \cdot C_L \cdot \frac{1}{I_{D_p}} \\ \tau_f &= \frac{1}{2} \cdot V_{dd} \cdot C_L \cdot \frac{1}{I_{D_n}} \end{aligned} \tag{1.3}$$

avec I_{D_p} et I_{D_n} les courants de saturation respectivement des transistors pMOS et nMOS.

Pour des portes contenant plusieurs entrées, nous distinguons le changement de polarité en sortie pour chacune des entrées par les deux chemins de « pull » représentés figure 1.10. Par rapport à l'entrée A, les transitions *HL* (High-Low pour état haut-état bas) apparaissent lors d'un front descendant par décharge de la capacité C_{ox} à travers le réseau de pull down. On note alors le temps de propagation à la descente $tp_k^{HL,A}$. De manière opposée, les transitions *LH* (Low-High pour état bas-état haut) apparaissent lors d'un front montant dues à la charge de la capacité C_{ox} à travers le réseau de « pull up ». On note alors le temps de propagation à la montée $tp_k^{LH,A}$.

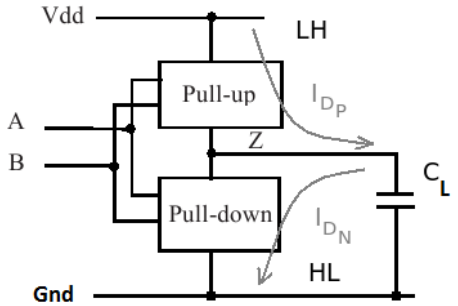


Figure 1.10 – *Modélisation de charge-décharge d'une porte logique par un réseau de « pull-up » dans le cas d'un front descendant (HL) et par un réseau « pull-down » dans le cas d'un front descendant (LH). Les courants I_Dn et I_Dp associés sont aussi représentés. [BC09]*

Plus précisément, les temps de propagation dépendent de l'environnement d'utilisation des portes inverseuses. Pour modéliser ces effets nous nous basons sur le modèle de Shockley [Sho50]. On décrit le paramètre I_D en fonction des tensions de seuil V_t, de grille, d'alimentation V_{dd} et des dimensions du transistor W et L (largeur et longueur). La capacité équivalente de l'oxyde est notée C_{ox} et sa permittivité μ. L'équation du temps de propagation d'une porte inverseuse 1.3 devient alors tp^{LH} pour les fronts montants et tp^{HL} pour les fronts descendants du signal de sortie.

$$tp^{LH} = \frac{L}{\mu \cdot W \cdot C_{ox} \left(V_{dd} - V_{tp}\right)^2} \cdot C_{ox} \cdot V_{dd} \tag{1.4}$$

$$tp^{HL} = \frac{L}{\mu \cdot W \cdot C_{ox} \left(V_{dd} + V_{tn}\right)^2} \cdot C_{ox} \cdot V_{dd} \tag{1.5}$$

avec V_{tp} la tension de seuil (positive) du transistor pMOS et V_{tn} la tension de seuil (négative) du transistor nMOS. À partir de ces résultats, l'influence des paramètres physiques sur les temps de propagation peut être étudiée.

1.3.1.2 Influence du vieillissement

Dans [Mar+09], les circuits tests sont des inverseurs CMOS de géométrie $Wn = 3\,\mu m$, $Wp = 6\,\mu m$ et $L = 130\,nm$ avec un oxyde d'épaisseur $2,2\,nm$. L'alimentation est portée à $V_{dd} = 3,2\,V$ et deux types de stress sont appliqués : une tension nulle ($Vin = 0\,V$) / tension élevée ($Vin = V_{dd}$) et un signal carré unipolaire d'amplitude V_{dd} de fréquence 1 kHz ou 500 kHz.

L'augmentation des courants de saturation des transistors N et P montrent que les temps de propagation de la porte logique augmentent avec la fréquence du signal. Aussi, il apparaît figure 1.11, que le type de stress influe fortement sur l'amplitude de la dégradation avec notamment une forte dégradation du transistor nMOS sous un stress haute fréquence.

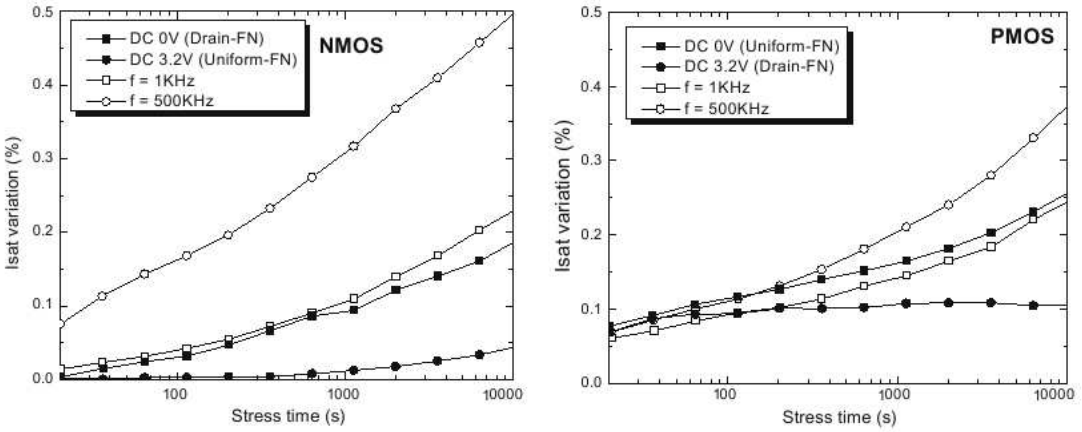


Figure 1.11 – *Évolution du courant de saturation pour les transistors nMOS et pMOS pour différent profil de stress HCI : stress continu (DC) à tension nulle et à tension maximale (3,2 V) ; stress alternatif à fréquence basse (1 kHz) et fréquence élevée (500 kHz). [Mar+09]*

Dans cette section, nous avons présenté la modélisation du temps de propagation d'une porte logique et son évolution sous l'effet du vieillissement. Nous allons maintenant expliquer le lien entre le modèle de dégradation et une cellule logique dans le but de l'appliquer à tous types de circuits, notamment lors de leur conception.

1.4 Modèle de dégradation d'une porte logique

Nous présentons les hypothèses pour modéliser la dégradation des délais d'une porte logique. L'équation de délai est explicitée et comparée avec les approches de la littérature.

Les délais d'une porte logique sont caractérisés par un ensemble d'arcs temporels entre une entrée et une sortie. Un exemple est donné figure 1.12. Chaque arc temporel représente le délai de propagation possible d'une seule transition à travers la porte logique, par exemple $t_{A->Z}$. Le temps de propagation d'un arc temporel dépend de la taille du transistor, de la *sortance* de la porte, de la direction de la transition en entrée et des conditions de température et de tension.

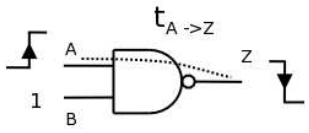


Figure 1.12 – *Exemple d'arc temporel dans une porte NON ET à 2 entrées. Une des deux entrées est supposée être constamment à la valeur logique '1'.*

La dégradation du temps de propagation d_{aged} d'une porte logique en fonction de son temps de propagation initial d_0 est représentée par l'incrément Δd dû au vieillissement.

$$d_{aged} = d_0 + \Delta d \tag{1.6}$$

Ce dernier terme traduit les effets du HCI sur les transistors internes de la porte logique parcourue par un courant ou lorsque sa sortie commute.

Nous adoptons le modèle exprimant le terme Δd similaire à celui proposé dans [Hua+11]. Nous nous basons sur les hypothèses établies pour ce modèle, rappelées ci-dessous.

- Les gradients de température ne sont pas pris en compte ;
- et le *slew rate* sur les entrées n'est pas considéré.

Le *slew rate* représente la vitesse de variation maximale que peut reproduire la sortie d'une porte logique. On considère donc ici que les signaux sont des échelons purs partout dans le circuit. En reprenant l'équation 1.1 de dégradation d'un paramètre du transistor dans le cas où $p = d$, on peut exprimer la dégradation de temps de propagation directement en fonction du temps de stress t.

$$\Delta d = B \cdot t^n \tag{1.7}$$

avec B et n des constantes issues de la technologie du fondeur et t le temps de stress du transistor. B dépend de la tension de drain seulement dans le cas du LEM [Str+09].

Un transistor interne est sous l'influence du HCI uniquement lorsqu'un courant le traverse. Cette condition dépend de l'activité des entrées et de la topologie des transistors internes. Ainsi la condition de stress dépend de l'utilisation du circuit et peut s'écrire comme suit.

$$t = \theta \cdot TC \tag{1.8}$$

θ est le temps pendant lequel le courant traverse le transistor ou bien le temps de commutation de la porte logique. TC est le nombre de fois où la condition de stress est vérifiée. Pour une période donnée, la valeur de TC est incrémentée dès qu'il y a un front LH ou HL en entrée.

En collationnant les équations 1.8, 1.7 et 1.6, le temps de propagation d'une porte logique peut s'exprimer comme suit.

$$d_{aged} = d_0 + C \cdot (TC)^n \tag{1.9}$$

Le terme C est une constante valant $B \cdot \theta^n$ et TC représente le nombre de commutations sur l'entrée de la grille des transistors.

Les autres paramètres électriques du transistor varient en fonction du temps d'utilisation, de la température, et de la polarisation. Les modèles de dégradation au niveau transistor permettent de

calculer précisément la variation de la tension de seuil, de la transconductance, etc. en fonction du temps de fonctionnement [Tu+93]. Cependant, les cellules standards contiennent des associations complexes de transistors. Il faut relier la variation d'un paramètre des transistors avec la variation d'un paramètre de la cellule. Les approches proposées par Lorenz et. al. [LBS10] ont explicité ce lien en décomposant chaque cellule en porte logique simple, puis en composant les données statistiques de commutation. Chaque signal d'entrée est supposé indépendant des autres signaux.

D'un autre côté, Huard et. al. [Hua+11] ont étudié expérimentalement l'application au niveau porte logique d'un modèle niveau transistor. Il est alors possible d'estimer les dégradations relatives de délai. La dépendance avec les différentes cellules standards est calculée par un coefficient de sensibilité.

1.5 Effet du HCI sur les caractéristiques temporelles d'un circuit

Un circuit est un ensemble de portes logiques et de registres interconnectés. Une horloge permet de synchroniser les traitements de données. C'est un signal périodique permettant de contrôler l'instant de mémorisation des données dans les registres. Notre objectif ici est de calculer les dégradations temporelles d'un circuit dues au HCI. Il faut donc séparer les portes logiques et les registres. Le modèle de dégradation précédemment défini est applicable à toutes les portes logiques d'un CI et nous n'étudions pas l'effet du HCI sur les registres. Nous présentons la méthode de calcul des dégradations de délai d'un circuit en fonction des registres et des portes logiques. Nous en donnons l'expression mathématique de la dégradation de délai d'un chemin. Ensuite, nous montrons les conditions de fonctionnement d'un CI en fonction des délais. Avec l'expression de la dégradation de délai, nous établissons alors la condition de fonctionnement d'un CI en fonction du vieillissement. Enfin, nous décrivons le problème du calcul des dégradations temporelles d'un circuit en fonction des chemins.

1.5.1 Caractéristiques temporelles d'un chemin combinatoire

Dans cette section, nous définissons les termes employés pour étudier la dégradation temporelle d'un circuit. La notion de chemin temporel, chemin physique et chemin fonctionnel sont introduites. On définit le délai d'un chemin et un exemple sur un circuit simple est fourni.

Un chemin combinatoire est un ensemble de portes combinatoires interconnectées qui relient une sortie d'un registre A à une entrée de donnée d'un registre B. Le délai d'un chemin est le temps de propagation total de la logique combinatoire, calculé le long d'un chemin combinatoire. Il correspond à la somme des temps de propagation des portes et des interconnexions se trouvant sur le chemin en question (chemin temporel).

Il peut exister deux types de chemins temporels selon qu'on considère la polarité et les conditions des transitions ou non. Dans le cas où on ne considère ni le sens de transitions des signaux ni les conditions de transition, on parle de chemin physique. Dans l'autre cas, on parle de chemin fonctionnel. Il peut exister plusieurs chemins fonctionnels correspondant à un seul chemin physique. Par exemple, un chemin physique p constitué de m portes logiques $g_1, ..., g_i, ..., g_m$, a son délai d dépendant des temps de propagation t_{pi} des portes logiques g_i.

$$d = \sum_{i=1}^{m} d_i^{\max} \text{ avec } d_i^{\max} = max\left(t_{pi}^{HL,e}; t_{pi}^{LH,e}\right) \tag{1.10}$$

Ici, on suppose que d_i est le temps de propagation maximal de la porte i par rapport à son entrée e (équation 1.10).

Un chemin fonctionnel est défini par la valeur logique des entrées des portes traversées par le chemin. Le délai d d'un chemin fonctionnel est calculé par l'équation 1.11 en fonction de l'entrée e et de la polarité de la transition tr ; on a $d_i(e,tr) = t_{pi}^{HL,e}$ pour un front descendant en entrée et $d_i(e,tr) = t_{pi}^{LH,e}$ pour un front montant.

$$d = \sum_{i}^{m} d_i(e,tr) \tag{1.11}$$

Le délai maximal et le délai minimal d'un circuit correspondent au chemin le plus long et au chemin le plus court respectivement. Ces dénominations ne correspondent pas au nombre de portes constituant les chemins mais bien à la somme des temps de propagation de chaque porte constituant les chemins.

La figure 1.13 montre un exemple de circuit contenant un chemin entre des éléments séquentiels (*flip-flops*). Dans cet exemple, le chemin le plus long et le chemin le plus court sont tracés en considérant que le délai de chaque porte logique est identique.

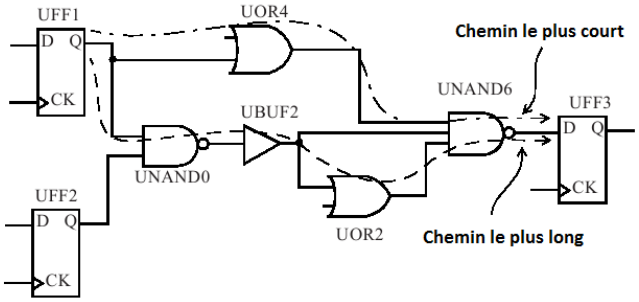


Figure 1.13 – *Exemple de chemin le plus long et le plus court dans un circuit combinatoire simple. Les délais de toutes les portes logiques sont considérés identiques. [BC09]*

1.5.2 Conditions temporelles de fonctionnement des CI

Dans un circuit numérique, la période d'horloge détermine le bon fonctionnement du système. Il faut que les registres puissent mémoriser les données correctement pour permettre le traitement séquentiel des données. Deux contraintes majeures concernent les éléments séquentiels : le temps de validité (*setup* en anglais) et le temps de maintien (*hold* en anglais) des données.

Le temps de validité T_{setup} est le temps minimal entre l'arrivée du signal d'horloge et l'arrivée d'une donnée qui permet à un registre d'enregistrer correctement la valeur logique sur son entrée D.

Le temps de maintien T_{hold} est le temps minimal entre le départ d'une donnée et l'arrivée du signal d'horloge qui permet à un registre d'enregistrer correctement la valeur logique sur son entrée D.

Dans un circuit, il existe en général une bascule *flip-flop* qui envoie une donnée et une autre bascule qui la reçoit. Cette dernière doit assurer la contrainte de temps de validité. Le contrôle de cette contrainte est faite sur le chemin le plus long depuis la bascule qui envoie la donnée jusqu'à celle qui la reçoit. Les horloges de ces bascules peuvent être identiques ou décalées dans le temps.

Le contrôle du temps de validité s'effectue après qu'une donnée soit envoyée, au moment où elle est reçue, à savoir au front d'horloge suivant.

La quantité T_{ck2q} représente le temps de propagation entre l'entrée d'horloge et la sortie d'une bascule *flip-flop*. L'architecture des bascules *flip-flop* montre que ce chemin emprunte au maximum deux portes logiques NAND. Les quantités T_{launch} et $T_{capture}$ représentent les délais entre les différentes connexions du circuit à l'horloge. Ils dépendent de l'arbre d'horloge du circuit et peuvent comprendre plusieurs portes logiques notamment pour réduire la consommation électrique [She+10] ou les pics de courant [LT10].

Pour qu'un circuit fonctionne correctement, il faut d'assurer que le temps d'arrivée des données respecte le temps de validité des bascules. Le temps d'arrivée d'un signal vaut $T_{launch} + T_{ck2q} + d$. Le temps de validité d'une bascule vaut T_{setup} et doit respecter la condition de la formule 1.12.

$$T_{launch} + T_{ck2q} + d < T_{capture} + T - T_{setup} \tag{1.12}$$

Par exemple sur la figure 1.14 le temps d'arrivée du signal d'horloge sur la bascule UFF1 doit respecter la condition de validité pour assurer que le circuit entier fonctionne.

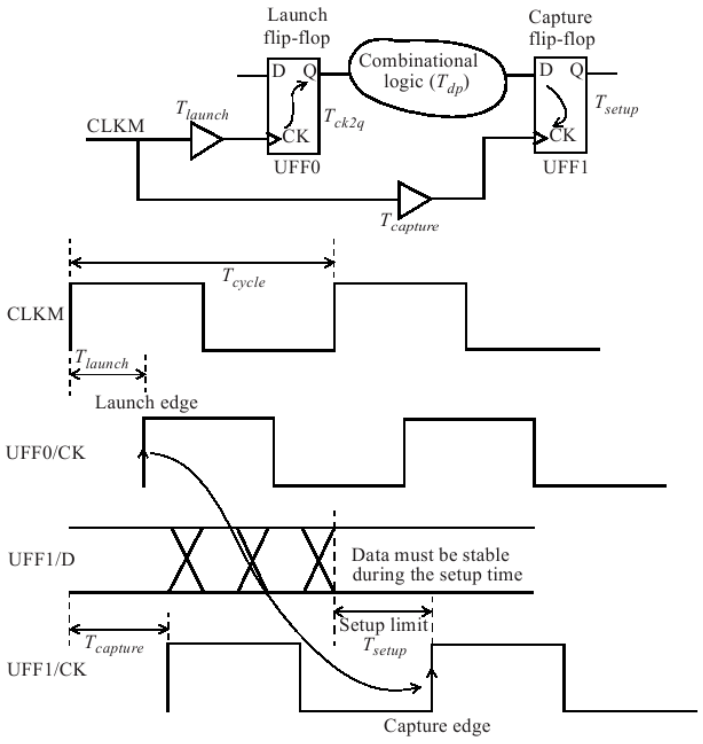


Figure 1.14 – *Illustration de la contrainte de temps de validité des signaux d'un circuit synchronisé par une horloge CLKM. La donnée est envoyée par la bascule UFF0 et reçue par la bascule UFF1 après avoir emprunté le chemin menant à la bascule UFF0 en un temps T_{launch}, traversé la bascule UFF0 depuis l'entrée d'horloge CK jusqu'à la sortie Q en un temps T_{ck2q} et parcouru la logique combinatoire pendant une durée d. [BC09]*

De manière opposée, lorsqu'une bascule *flip-flop* reçoit une donnée, il faut s'assurer que la donnée envoyée pendant le cycle d'horloge suivant ne vienne pas modifier sa valeur avant qu'elle soit prise en compte. Le contrôle de cette contrainte est faite sur le chemin le plus court, depuis la bascule qui envoie la donnée jusqu'à celle qui la reçoit. Le contrôle du temps de maintien s'effectue après qu'une donnée soit reçue, au moment où une nouvelle donnée est envoyée, cette fois au même cycle d'horloge.

Cette seconde condition est satisfaite si le temps d'arrivée des données respecte le temps de maintien des bascules. Le temps d'arrivée d'un signal vaut $T_{launch} + T_{ck2q} + d$. Le temps de maintien d'une bascule vaut T_{hold} et doit respecter la condition de la formule 1.12.

$$T_{launch} + T_{ck2q} + d > T_{capture} + T_{hold} \tag{1.13}$$

Par exemple sur la figure 1.15, le temps d'arrivée des données sur l'entrée de la bascule UFF1 doit respecter la condition de maintien pour assurer que le circuit entier fonctionne. Le signal logique le plus rapide est celui parcourant le chemin minimal ayant pour délai d_{min}. Il faut s'assurer donc que $T_{capture} < T_{launch} + T_{ck2q} + d_{min} - T_{hold}$.

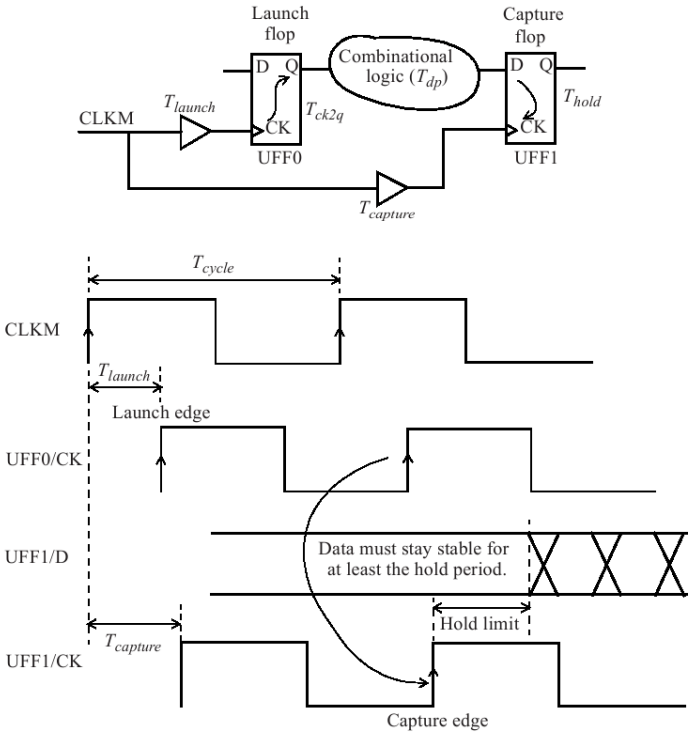


Figure 1.15 – *Illustration de la contrainte de temps de maintien des signaux d'un circuit synchronisé par une horloge CLKM. La deuxième donnée est émise par l'entrée CK de la bascule UFF0 après avoir emprunté le chemin menant à la bascule UFF0 en un temps T_{launch}, traversé la bascule UFF0 depuis l'entrée d'horloge CK jusqu'à la sortie Q en un temps T_{ck2q} et parcouru la logique combinatoire pendant une durée d. [BC09]*

En résumé, la condition de fonctionnement d'un circuit est fixée après que la période d'horloge T soit définie par l'équation 1.14. C'est le temps de validité qui entre en jeu car elle impose une

limite haute pour les valeurs de d. La condition de maintien impose seulement une limite basse pour le temps d.

$$d < T - T_{launch} - T_{ck2q} + T_{capture} - T_{setup} \tag{1.14}$$

La différence entre la période d'horloge et le délai de propagation du signal d'entrée, représente la marge temporelle de la bascule, notée ST. On a donc formule 1.15 l'expression de la marge temporelle ST.

$$ST = T - d \tag{1.15}$$

Nous supposons ici que l'expression $T - T_{launch} - T_{ck2q} + T_{capture} - T_{setup}$ a une valeur constante pendant la durée de vie du circuit. Avec le vieillissement, seul la valeur de d augmente. Lorsque la dégradation est trop importante, les données attendues ne sont plus reçues à temps par la bascule et il apparaît une faute dans le circuit. En d'autres termes, la relation 1.14 n'est plus vérifiée. Cette faute se transforme en erreur si la donnée reçue est différente de celle initialement attendue [Arl+06]. Cette condition est vérifiée si par exemple, le chemin le plus long contient un grand nombre de portes logiques soumises à la dégradation HCI.

La condition de fonctionnement d'un circuit a lieu lorsque l'inégalité 1.14 est respectée. À partir de l'équation 1.14, on peut écrire une autre expression de la condition de fonctionnement du circuit : $ST > T_{launch} + T_{ck2q} - T_{capture} + T_{setup}$. En supposant que $T_{launch} + T_{ck2q} - T_{capture} + T_{setup} << ST$, la condition devient $ST > 0$.

Plus précisément, le délai vieilli d_{aged} d'un chemin de la logique combinatoire composé de m portes logiques $g_0, ..., g_i, ..., g_{m-1}$ est exprimé équation 1.16.

$$d_{aged} = \sum_{g_i} d_{aged}(i) \tag{1.16}$$

Avec $d_{aged}(i)$ le temps de propagation de la porte logique g_i vieillie. En combinant les équations 1.9 et 1.16, le temps de propagation peut s'écrire en fonction des commutations $TC(i)$ des entrées e_i.

$$d_{aged} = d_0 + C \cdot \sum_{g_i} (TC(i))^n \tag{1.17}$$

d_0 est le délai initial du chemin temporel. Le terme $\sum_{g_i} (TC(i))^n$ est appelé « commutations d'un chemin » dans la suite du document. Il représente l'évolution de la dégradation temporelle du chemin. On nomme *chemin critique initial* le chemin temporel qui a le plus long délai de propagation, sans condition de stress HCI.

Il est aussi intéressant de noter qu'une porte logique soumise à une dégradation de son temps de propagation peut bouleverser le calcul des chemins temporels. Par exemple, si on se place dans le cas de la figure 1.16, on remarque que le chemin c-z est initialement le plus long car la somme des temps de propagation des portes logiques A et B pour ce chemin vaut 3, qui est plus grande que celle du chemin a-z et b-z (respectivement 1 et 2). Mais en prenant en compte le vieillissement, si le délai de la porte A par rapport à l'entrée b augmente (ici à la valeur de 3) alors le délai vieilli du chemin b-z devient 5 et il devient le chemin le plus long ; il y a donc un échange entre les chemins c-z et b-z. Dans le cas où c-z est un chemin critique initial du circuit, les contraintes de temps de validité seront respectées initialement mais peuvent être dépassées au cours de la vie du circuit.

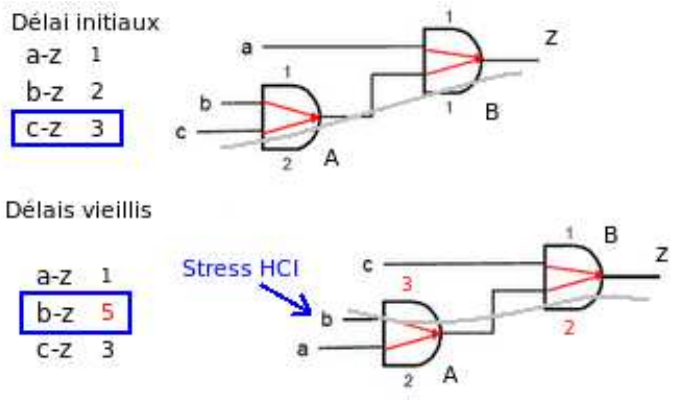


Figure 1.16 – *Le circuit en haut est décrit au repos par les délais des chemins a-z, b-z et c-z. Le circuit en bas est décrit après un stress HCI sur l'entrée b seulement. Les délais des chemins sont changés ainsi que le chemin maximal.*

Dans cette section, nous avons présenté l'effet du HCI sur la dégradation des temps de propagation d'un circuit CMOS. Pour que le circuit fonctionne, on doit s'assurer que les délais sont inférieurs à une limite définie par les contraintes temporelles des bascules et de l'horloge.

1.6 Conclusion

Dans le contexte de réduction du facteur d'échelle, les phénomènes de dégradation réduisent la durée de vie des SoC à cause de la réduction non idéale de leur tension d'alimentation. Nous avons expliqué les principaux mécanismes de dégradation des transistors MOSFET : le HCI et le NBTI. Nous avons montré que le vieillissement implique des variations de temps de propagation des portes logiques. Ainsi les chemins peuvent avoir un délai qui varie au cours du temps. Cette variation peut se traduire par une violation des conditions temporelles de fonctionnement d'un CI et causer une erreur.

Finalement, les solutions actuelles consistent à ajouter des marges de sécurité par exemple sur la période d'horloge lors de la phase de conception. Les marges temporelles couramment ajoutées pour parer aux variations de procédé, tension et température sont présentées figure 1.17. L'intervalle de confiance est représenté pour la valeur de variation due au procédé.

Pour tenir compte des dispersions des paramètres technologiques, de tension et de température, il a été montré que les marges temporelles combinées peuvent atteindre 35 % pour une technologie 65 nm [Gup+09b] en considérant le pire cas de variation de procédé. Concevoir un SoC en fonction du vieillissement permet de compenser la baisse de la fiabilité des transistors ; cette baisse apparaît avec l'évolution de la technologie [ITR11].

Nous avons vu dans les sections précédentes que le vieillissement dépend du circuit, de l'activité, des paramètres technologiques et des conditions de fonctionnement. Pour comprendre les effets du vieillissement, des simulations sont nécessaires. Dans le prochain chapitre nous présentons les techniques actuelles de simulation du vieillissement des SoC.

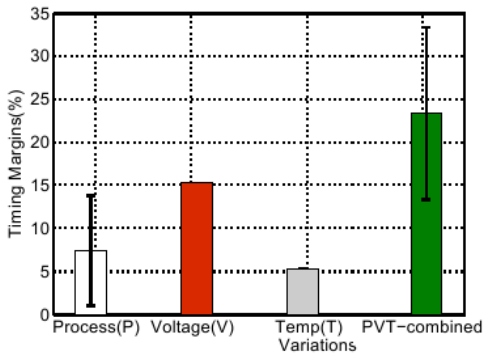


Figure 1.17 – *Impact des variations du procédé de fabrication, de la tension d'alimentation et de la température (PVT) sur les marges de délais pour une technologie* 65 *nm. [Gup+09b]*

CHAPITRE 2

MÉTHODES EXISTANTES DE SIMULATION DU VIEILLISSEMENT D'UN CIRCUIT INTÉGRÉ

Le précédent chapitre a montré le vieillissement des transistors et son effet sur les caractéristiques temporelles des CI. Avec les prochaines générations de procédé de fabrication, la phase de conception doit être adaptée pour tenir compte du vieillissement dès les premières étapes de développement des MPSoC.

Les premières étapes de conception consistent à simuler rapidement le circuit. Pour cela, on utilise les langages de description de haut niveau. Plusieurs modèles existent pour décrire simplement un circuit complexe. On trouve les langages de type Architecture Description Language (ADL), comme par exemple ArchC [Rig+04]. Il y a de plus en plus d'efforts pour parvenir à proposer de tels langages pour modéliser diverses propriétés des architectures [Bel+08 ; Sch+07 ; KN08 ; Mor+11]. Il est à noter qu'aucune d'entre elles ne permet de modéliser la fiabilité du circuit, en particulier la dégradation des délais.

Il existe deux principaux verrous pour calculer les dégradations au niveau fonctionnel. D'un côté, il n'y a pas de méthode éprouvée dans la littérature ou bien d'outils commerciaux permettant de décrire le comportement de la dégradation due au HCI et au NBTI au niveau fonctionnel. Seules les études [Hua+11 ; LGS09] permettent d'évaluer les effets au niveau porte logique. Elles impliquent une surestimation des défaillances, comme expliqué dans ce chapitre. D'un autre côté, les simulations fonctionnelles rapides d'exécution, ne sont pas assez précises pour évaluer les délais du circuit. Or nous avons vu au chapitre 1 que le calcul des délais est nécessaire pour évaluer les dégradations.

Objectifs

Notre objectif est de définir une méthode de conception rapide permettant de fabriquer des SoC fiables. Or, les effets du vieillissement apparaissent en fin de vie d'un produit. Ils doivent donc être simulés et accélérés. Aussi, pour estimer les dégradations pendant la conception d'un CI, il faut des temps de simulation courts. Ainsi, les principaux besoins d'une méthode de simulation du vieillissement pendant la conception sont :

– une simulation assez rapide pour pouvoir être exécutée itérativement pendant les premières étapes de conception ;
– et une prise en compte des conditions d'utilisation et de la topologie du circuit.

Nous présentons et comparons dans ce chapitre, les méthodes les plus récentes permettant de simuler le vieillissement des circuits numériques, lors de la phase de conception. De nos jours, les systèmes embarqués comportent de plus en plus de transistors et de possibilités d'arrangement de ces transistors, que nous appelons configurations. Il est nécessaire que les simulations soient assez rapides pour estimer les dégradations dans toutes les configurations données.

Enfin, un circuit peut être modélisé au niveau le plus élevé par une fonction reliant les interactions entrées/sorties. Il est clair qu'il y a une perte d'information sur la fiabilité du circuit au fur et à mesure que l'on s'élève dans les niveaux d'abstraction, mais que l'on gagne en temps de simulation. Un compromis entre niveau de modélisation, précision des modèles et temps de simulation doit donc être fait comme suggéré dans [BES08].

Définitions

Dans ce chapitre, nous appelons durée de vie le temps spécifié dans le cahier des charges d'un produit, pendant lequel le système remplit toutes ses fonctions. Dans le cas de SoC, c'est le temps pendant lequel les données en sortie sont conformes aux entrées, après avoir été traitées par un algorithme. Du point de vue interne au circuit, cette condition est vérifiée si chaque temps de propagation des chemins n'excède pas une limite fixée par le point de fonctionnement du circuit.

Pour permettre des simulations rapides de la durée de vie, il est possible de définir quatre niveaux d'abstraction. Une version assez répandue pour un processeur, établie par Gajski et al. [Gaj+92] est résumée dans le tableau 2.1.

1. Le niveau système/Electronic System Level (ESL) est le moins détaillé et décrit des processeurs ou des mémoires globalement. Le niveau fonctionnel est restreint au déroulement d'instruction ou à un flot de données.
2. Pour la microarchitecture, les registres ou les unités de calcul (ALU) sont décrits par les transferts de données qu'ils impliquent. Par exemple, une addition nécessite un accès aux ressources du processeur, l'utilisation de l'ALU, le stockage du résultat et la gestion des débordements.
3. Quant au niveau logique, c'est une description du parcours des signaux logiques dans un circuit. Chaque bit est décrit ainsi que les portes logiques permettent de réaliser la fonction du système. La connectivité des portes logiques entre elles est appelée *netlist* ; elle contient des nœuds, qui sont des fils de connexion entre les entrées/sorties des portes logiques.
4. Enfin, les tensions et les courants aux bornes de chaque transistor, en fonction du temps définissent le niveau transistor.

Plan du chapitre

Ce chapitre débute par la description des méthodes de conception et les contraintes qu'elles imposent pour intégrer la fiabilité. Les quatre sections suivantes correspondent aux quatre niveaux de description du circuit, allant du plus détaillé (niveau transistor) au moins détaillé (niveau système). On décrit et compare leurs avantages et inconvénients pour simuler des systèmes complexes comme les MPSoC. Enfin, nous positionnons la thèse dans ce contexte.

Niveau d'abstraction	Description	Modélisations
Système/ESL	Synoptiques algorithmes	Paramètres fonctionnels à chaque état/phase
Fonctionnel	Instructions, Flots de données	Paramètres fonctionnels à chaque cycle/instruction
Microarchitecture	Register Transfer Level (RTL)	Paramètres fonctionnels à chaque cycle/transfert
Porte logique	Équations booléennes Diagramme d'état	Paramètres tempo- rels instantanés
Transistor	Fonction de transfert Diagrammes temporels	Paramètres élec- triques instantanés

Tableau 2.1 – *Principaux niveaux d'abstraction d'un circuit numérique. Chaque niveau définit une description du système utilisée pour calculer les dégradations (tension, courant, temps, transfert logique, état du système). [Gaj+92]*

2.1 Niveau transistor

Au niveau transistor, la plupart des études utilisent des modèles empiriques obtenus par caractérisation des paramètres physiques de la défaillance sous conditions accélérées ou non.

Les outils basés sur un simulateur de type SPICE [1] et des modules de dégradations physiques prédisent les dérives des caractéristiques électriques d'un circuit. Cette analyse de la fiabilité par simulations électriques a pour but de détecter les dispositifs sensibles de l'architecture du circuit intégré.

Nous présentons ici deux des principaux simulateurs commerciaux existants. Enfin, nous présentons une méthodologie de simulation d'un processeur entier, basée sur des modèles mathématiques.

2.1.1 Principe de simulation

Actuellement, le consensus pour estimer les dégradations est l'approche « Bottom-up ». Elle consiste à propager des informations précises de la physique de la dégradation vers les premières étapes de conception. Un modèle de dégradation est associé à chaque élément du circuit, lors de la conception. Les simulateurs du vieillissement sont imbriqués dans les simulateurs électriques (de type SPICE) et estiment la dégradation de performance des circuits [NP73]. Chaque transistor subit un stress qui est fonction de sa polarisation et des conditions environnementales (comme la température par exemple). L'effet du stress est calculé individuellement par des modules de dégradations physiques. Par exemple, pour le HCI, les modules de dégradations physiques sont basés sur l'équation $Degradation \propto A \cdot t^n$ [TS83]. A et n sont déterminés initialement par une ou plusieurs expériences sur un transistor ; t représente le temps. Il est courant d'utiliser la tension de

1. Simulation Program with Integrated Circuit Emphasis

seuil V_t pour estimer le NBTI des transistors PMOS [Hua+03]. Le courant drain source sert pour le HCI dans les transistors NMOS [Pag03]. Ensuite, les contraintes courant/tension de chaque transistor sont mises à jour et la simulation SPICE est lancée. Il est alors possible d'estimer les variations de la tension de seuil, de la mobilité, etc. des transistors dégradés. À partir de ces étapes, on peut simuler le fonctionnement du circuit après dégradation.

En résumé, la simulation de la fiabilité d'un système implique l'utilisation de deux modules assurant l'interface entre le simulateur électrique SPICE et les modules de dégradations physiques [BES08]. L'environnement de simulation de tels outils est montré figure 2.1.

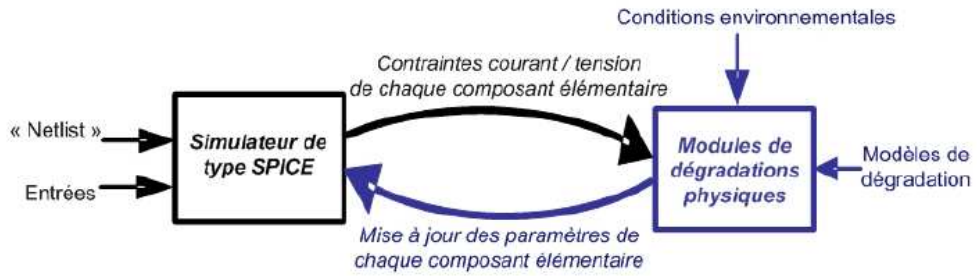


Figure 2.1 – *Schéma synoptique d'un simulateur de fiabilité des circuits intégrés au niveau transistor. [BES08]*

2.1.2 Principaux simulateurs

Le premier simulateur utilisant cette méthode est *BERT* [Tu+93], développé par l'université de Berkeley en Californie. Il comprend différents modules pour simuler les effets de l'électromigration, du HCI et du TDDB. Il est basé sur le simulateur *CAS*[2] [Lee+88] et opère sur les tensions en sortie de tous les transistors du circuit.

BERT a été proposé dans le but de simuler les dégradations d'un circuit en un temps proche du temps de simulation fonctionnelle d'un circuit matériel, décrit par ses transistors. Cependant la complexité des circuits actuels fait que le calcul de tous les profils de courant et tension d'un circuit nécessite d'importantes ressources. Par exemple, un circuit comportant 1 million de transistors comporte au moins 2 millions de variables courants-tensions à calculer si on se limite à la tension et au courant de sortie. Une de ses extensions commerciales est connue sous le nom de *RelXpert* [Cada].

Le deuxième simulateur au niveau transistor présenté dans cette section est *UltraSim*, développé par Cadence pour des applications SoC en technologie inférieure à 130 nm [Cadb]. Des modèles pour les dégradations causées par les porteurs chauds et pour le NBTI sont disponibles et permet une diminution du temps de simulation par un facteur 10 à 1 000 grâce à un simulateur *FastSPICE* qui est moins précise que SPICE (l'erreur est inférieure à 1 % par rapport au niveau SPICE).

BERT et *UltraSim* sont les plus connus pour simuler le temps de vie des circuits. Ces simulateurs sont utilisés après l'étape de synthèse logique.

2. Circuit Aging Simulator

La principale limitation des simulations au niveau transistor est le temps d'exécution, qui peut être très long pour les MPSoC, notamment si de nombreux cas applicatifs doivent être évalués.

2.2 Niveau porte logique

Au niveau porte logique, les travaux [Hua+11 ; LGS09] ont permis d'évaluer le NBTI et le HCI par des simulateurs au niveau porte logique. D'un côté, les approches empiriques nécessitent une instrumentation des cellules logiques par des simulateurs de fiabilité dont les temps de simulation sont élevés. D'un autre côté, les méthodes probabilistes utilisent une activité identique sur toutes les portes logiques et surestiment la dégradation. Les estimations sont donc pessimistes. Nous décrivons ces deux méthodes dans cette section.

2.2.1 Méthode empirique

L'approche présentée dans [Hua+11] permet d'analyser des effets du HCI et du NBTI sur les délais du circuit au niveau porte logique (qui est représenté par la *netlist* ici). Il est possible d'analyser les délais des portes après vieillissement avec un simulateur de fiabilité couplé à des analyses temporelles statiques ou STA [3]. Le modèle estime la dégradation des délais de chaque cellule standard de la *netlist* du circuit. Ainsi, l'analyse temporelle permet d'estimer la dégradation des chemins. D'intéressantes contributions ont été réalisées dans ce domaine comme [Hua+11]. Une nouvelle approche « Bottom-up », appelée méthodologie « Design-in », permet d'analyser le HCI et le NBTI au niveau porte logique. Une validation pour la technologie 45 nm a été réalisée sur silicium [Rui+11]. Un outil de simulation de performance est couplé à un outil propriétaire pour évaluer le taux de dégradation de chaque cellule logique de la *netlist*. Le délai de propagation initial d'une cellule standard est remplacé par le délai obtenu avec le modèle vieilli. L'analyse temporelle fournit le délai des chemins après dégradation. L'approche est facilement intégrable à de nouvelles technologies et elle est compatible avec les outils de conception au niveau porte logique.

Plus précisément, la dégradation du temps de propagation d'une porte logique est estimée par trois paramètres (équation 2.1).

$$\Delta d = S \cdot (POT)^n \tag{2.1}$$

Le premier paramètre est la sensibilité S qui dépend de la pente d'évolution de la tension en entrée et/ou de la capacité. POT est le temps de maintien au niveau logique haut du signal d'entrée. Il correspond au temps de stress correspondant au profil de mission sous la tension nominale. Le troisième paramètre est l'exposant n, qui quantifie la loi puissance entre le temps et la dégradation du temps de propagation.

Nous présentons les approches empiriques pour construire ces modèles. Les librairies de cellules standards sont caractérisées afin de générer des modèles pour le NBTI et le HCI. Un modèle niveau porte logique est intégré dans une nouvelle librairie qui contient des informations de dégradation de chaque porte logique. Ainsi, le concepteur peut évaluer la fiabilité d'un circuit dès la phase de conception au niveau porte logique, simplement en utilisant cette librairie et en spécifiant l'évolution des signaux logiques. La possibilité d'utiliser un simulateur logique, un modèle probabiliste ou une condition de pire cas pour décrire les signaux rend la méthode très flexible.

3. Static Timing Analysis

2.2.2 Méthode probabiliste

Dans [LGS09], une méthode similaire est utilisée pour calculer les dégradations de délai dues au NBTI et au HCI. Trois étapes permettent d'obtenir une estimation des dégradations de délai des cellules logiques et des chemins (figure 2.2).

1. Profil d'utilisation : les conditions de fonctionnement pendant la durée de vie t_{life} sont la tension V_{eff} et la température T_{eff}.
2. Détermination des signaux : l'activité des signaux d'entrée des portes logiques permet de calculer la déviation des paramètres électriques des transistors. Elle est obtenue par deux métriques : SP (Signal Probability) est la probabilité que le signal s a la valeur logique '1' pendant la durée t_{life} ; TD (Transition Density) est la probabilité que le signal s change d'état pendant la durée t_{life}. SP est relié à la dégradation NBTI alors que TD conditionne le HCI. Trois modes de calcul sont possibles : une simulation logique du circuit ou une propagation des valeurs depuis les entrées primaires par des méthodes probabilistes ou un pire cas. Le pire cas apparaît lorsque $TD = 100\,\%$, cas où le signal change d'état à chaque cycle d'horloge. Ceci représente un stress HCI permanent.
3. Analyse temporelle : les paramètres des cellules logiques sont calculés et analysés pendant le temps t_{life}, comme par exemple la tension de seuil et le courant de saturation (équations 2.2). Ceci permettra de calculer la déviation des paramètres temporels de la porte logique au cours du temps de vie.

$$\begin{aligned} \Delta V_{th} &= f1(V_{eff}, T_{eff}, P_{\text{stress}} \cdot t_{\text{life}}, L) \\ \Delta I_{on} &= f2(V_{eff}, T_{eff}, P_{\text{stress}} \cdot t_{\text{life}}, W) \end{aligned} \tag{2.2}$$

Les déviations dépendent de la tension V_{eff}, la température T_{eff}, le temps de stress total t_{life} et les tailles des transistors (W et L).

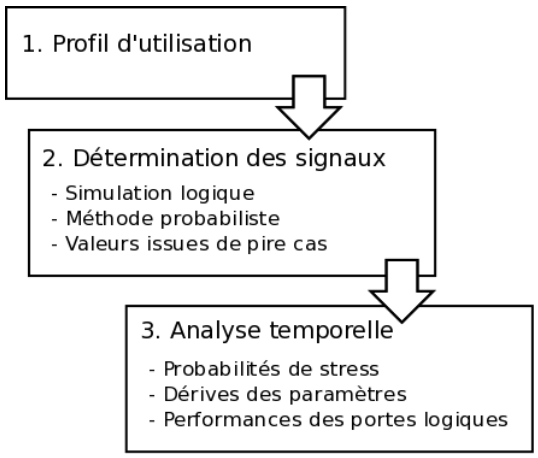


Figure 2.2 – *Méthode de simulation des délais dynamiques à partir du profil utilisateur au niveau cellule standard. Les signaux des portes logiques peuvent être décrits au cycle près, statistiquement ou par un pire cas mathématique. Enfin l'analyse des délais est réalisée par le calcul du stress, des déviations et des performances des portes logiques. [Lor12]*

Les erreurs entre le niveau transistor et le modèle proposé ont été mesurées dans un pire cas. Pour cela, un simulateur SPICE donne les valeurs de temps de propagation en fonction d'un profil de mission pire cas (le circuit doit fonctionner 20 années sous une température de 125 °C). Puis la méthode est appliquée pour calculer les temps de propagation en fin de mission. L'erreur est calculée entre les résultats du simulateur SPICE et les résultats issus de chaque cellule. Les erreurs commises sur la valeur des temps de propagation sont de l'ordre de 1 % de la dégradation totale. Ceci prouve l'intérêt d'utiliser le niveau porte logique pour calculer les dégradations d'un circuit quelle que soit son utilisation. Les temps de simulation ne sont pas reportés au niveau porte logique mais le temps de calcul préliminaire de la fiabilité au niveau transistor est 2 fois à 5 fois plus important, sans augmentation significative de mémoire utilisée par l'outil [Par+06]. Ainsi les temps de simulation au niveau porte logique sont inférieurs à ces valeurs, mais peuvent nécessiter plus de mémoire pour stocker les modèles notamment.

En modélisant toutes les cellules standards, il est donc possible de déterminer les dégradations de délais des chemins logiques pour tout type de signaux logique. Cependant les résultats ne sont valides que pour des librairies de cellules standards qui ont été préalablement caractérisées. Cette méthode laisse donc un travail important au fournisseur de cellules standards. Par contre, les résultats sont assez précis pour permettre de sélectionner une porte logique dont la dégradation serait trop importante par exemple. Cette tâche est laissée au concepteur sans automatisation disponible pour l'instant.

Les simulations des portes logiques sont donc assez avantageuses . Elles sont rapides en temps de simulation et peu coûteuses en mémoire ; dans le cas d'un MPSoC entier par exemple elles ne considèrent que certaines de ses portes logiques. Par contre, elles restent moyennement intéressantes à utiliser dans les méthodes de conception car elles nécessitent de disposer de la description porte logique du circuit. Or cette description n'est généralement pas disponible dès la toute première phase de conception d'une architecture. Aussi, elles nécessitent de disposer des informations technologiques du fabricant. Dans le cas des cellules standards, ces informations sont facilement utilisables directement dans les outils de conception. Pour d'autres techniques de conception, il peut y avoir un effort important d'intégration aux outils utilisés.

2.3 Niveau microarchitecture

Dans cette section, nous décrivons la méthode de simulation de la fiabilité au niveau microarchitecture. Une décomposition hiérarchique du système permet d'utiliser des simulateurs architecturaux existants. Les informations de fiabilité des transistors sont propagées à plus haut niveau.

Des travaux académiques [Sri+04] ont permis d'estimer la fiabilité d'un système tel un MPSoC au niveau fonctionnel, décrit au cycle près. RAMP [4] permet l'analyse de la fiabilité des microprocesseurs. C'est un outil d'analyse des fautes permanentes développé par l'université de l'Illinois et IBM [Sri+03]. Différents mécanismes de défaillance sont décrits au niveau transistor et contribuent au vieillissement du circuit. Les paramètres d'entrée sont :

- le profil applicatif qui doit être décrit au cycle près ;
- la puissance ;
- la température ;
- et le *floorplan* de la puce.

4. Reliability-Aware Microprocessor

RAMP utilise un simulateur propriétaire de consommation électrique et un simulateur académique de température, pour alimenter le modèle de dégradation du circuit. Ce modèle est construit par combinaison hiérarchique des modèles de défaillance d'éléments uniques et par la physique du phénomène de dégradation. L'idée est de construire un modèle de défaillance générique du système, à partir des équations du temps de défaillance des transistors [JED10]. L'approche se veut simple et utilise la somme des taux de défaillances au niveau système. Des informations peu précises sont donc suffisantes (pas besoin de synthèse logique) et permettent de comparer la métrique Cumulative Failure Rate (CFR) pour différentes configurations d'une architecture. Nous détaillons le CFR dans la suite.

La métrique utilisée au niveau transistor est le taux de défaillance λ. Le taux de défaillance est la probabilité que le système fonctionne à la date $t + dt$ sachant qu'il fonctionne à la date t. Son unité est le FIT. À partir du niveau supérieur, on peut aussi exprimer le CFR, défini comme la somme cumulée des taux de défaillance, qui sont calculés en des instants discrets i du temps et multipliés par le temps T_i. Les mécanismes de défaillance sont supposés indépendants les uns des autres et constants, ce qui implique que la fiabilité du système est la somme des taux de défaillance de ces blocs (modèle *sum-of-failure-rates* ou SOFR).

$$CFR(n) = \sum_{i=0}^{n} \lambda_i \cdot T_i \tag{2.3}$$

On note aussi le Mean Time-To-Failure (MTTF) du système, qui correspond au *Median Time-To-Failure* dans le domaine des probabilités. Il est défini comme le temps pour lequel la probabilité que le système soit fonctionnel est de 50 %.

À partir des modèles de fiabilité série, les taux de défaillance sont composés pour chaque bloc hiérarchique pour obtenir une grandeur proportionnelle au CFR du circuit, en fonction du temps. Les défaillances sont considérées indépendantes et chaque bloc contribue identiquement aux pannes du système. Aussi, chaque transistor et chaque via ont les mêmes caractéristiques. La fiabilité du circuit est alors calculée uniquement en fonction des librairies de cellules standards. Les librairies technologiques TSMC 40 nm contiennent par exemple des modèles non-linéaires pour calculer délai et consommation. Cependant, aucun modèle de fiabilité pour le HCI et le NBTI n'est disponible à ce jour.

RAMP permet de simuler les blocs architecturaux d'un système et de calculer rapidement sa fiabilité. Les hypothèses sont assez fortes et le résultat peut seulement être comparé pour plusieurs configurations d'une architecture par exemple. Enfin, son application directe à un MPSoC reste coûteuse en temps de calcul car un MPSoC peut contenir jusqu'à 1 million de portes logiques.

2.4 Niveau fonctionnel et supérieur

D'autres travaux considèrent le niveau fonctionnel [Gup+09a ; Gup11 ; Sri+04] toujours en utilisant la méthodologie « Bottom-up ». Mais cette fois, l'objectif est de simuler une grandeur proportionnelle aux dégradations et de comparer différentes configurations d'une architecture. Les traces de consommation électrique et de température permettent de simuler le MTTF au niveau fonctionnel.

2.4.1 Niveau fonctionnel

Récemment, RAAPS [Gup+09a] a été introduit pour simuler la fiabilité des MPSoC au niveau fonctionnel avec des modèles précis. Un simulateur du jeu d'instructions est augmenté d'une

donnée de consommation électrique (Power-ArchC). Un circuit peut donc être directement étudié de cette façon par les langages tels SystemC. La fiabilité d'un processeur de type Microprocessor without Interlocked Pipeline Stages (MIPS) a été calculée en quelques minutes.

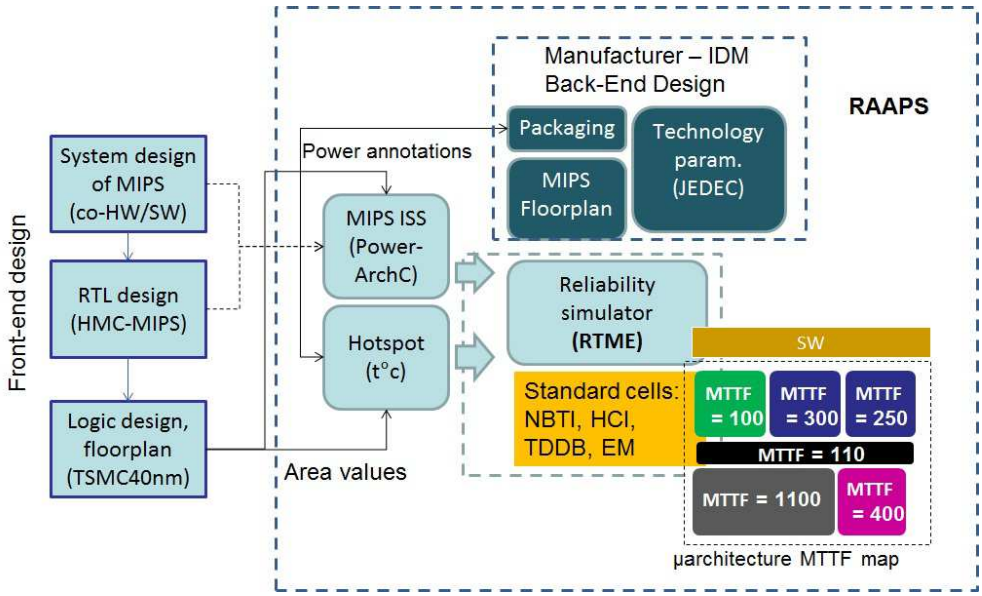


Figure 2.3 – *Méthode de simulation de la fiabilité avec le simulateur RAAPS. L'annotation du jeu d'instructions permet de calculer consommation et température, puis les informations technologiques de floorplan et de boîtier thermique sont utilisées. [Gup11]*

Une première étape consiste à décrire le système depuis sa conception à haut niveau (co-HW/SW) jusqu'au niveau porte logique, en passant par la description Register Tranfer Level (RTL). Les deux premières étapes permettent d'annoter la consommation électrique dans le simulateur Power-ArchC. Ensuite, le simulateur « Hotspot » [Ska+03] fournit les résultats de température grâce aux informations de surface silicium, aux données du fondeur, au *floorplan* et aux paramètres de packaging. Enfin, les informations de température et de consommation sont fournies au simulateur *RTME*. La fiabilité est estimée grâce aux librairies de cellules standards par une cartographie des valeurs MTTF de la microarchitecture. La méthodologie globale est présentée figure 2.3.

L'utilisation d'un Instruction Set Simulator (ISS) réduit fortement les temps de simulations. Le calcul de la consommation électrique est par exemple, de 10 à 15 fois plus rapide qu'avec « Wattch » [BTM00]. De plus, les résultats sont très précis pour une architecture proche de celle utilisée pour générer les modèles de consommation (MIPS). Par contre, pour de nouvelles technologies et architectures, il se peut que les modèles doivent être générés à nouveau, laissant un grand travail au concepteur.

2.4.2 Niveau applications

Dans [Cos+09] une simulation architecturale basée sur une caractérisation de la performance et de la puissance des MPSoC est utilisée. Trois mécanismes sont pris en compte : l'électromigration, le TDDB et le cyclage thermique. Les dégradations générées par les trois phénomènes physiques sont quantifiés relativement entre elles pour s'affranchir des constantes technologiques.

Les défaillances sont supposées indépendantes et le MTTF moyen est calculé pour une simulation complète. Les équations technologiques de dégradations issues de [JED10] sont évaluées. Une des principales différences avec l'approche RAMP est que les variations de température sont caractérisées.

La consommation électrique et l'IPC [5] sont déterminés par le simulateur « Wattch » [BTM00], précis au cycle d'horloge près. De plus, le logiciel HotSpot [Ska+03 ; Ska04] est utilisé avec une période plus grande que le cycle d'horloge. Ceci permet d'accélérer les temps de simulation par rapport aux précédentes méthodes. Par exemple, le simulateur « SimPoint » [Per+05] permet de capturer les différentes phases dans l'exécution complète d'une application. Cela permet de réduire la simulation des cycles d'exécution à quelques points représentatifs du comportement à plus haut niveau de l'application. En résumé, un ensemble de simulations préliminaires est réalisé pour différents paramètres de tension et de fréquence. Une base de données est alors construite et permet de renseigner sur l'IPC et la puissance consommée pour une longueur de programme quelconque. Enfin le *Schedule Manager* prend des décisions de migration ou d'arrêt des tâches, ou encore de changements de tension-fréquence. La méthodologie globale, présentée figure 2.4 rend possible les simulations de l'ordre de la dizaine de secondes, ce qui permet de capturer les effets de la température et de la fiabilité.

Les résultats expérimentaux sont calibrés par des données issues de la littérature pour un procédé de fabrication de 65 nm.

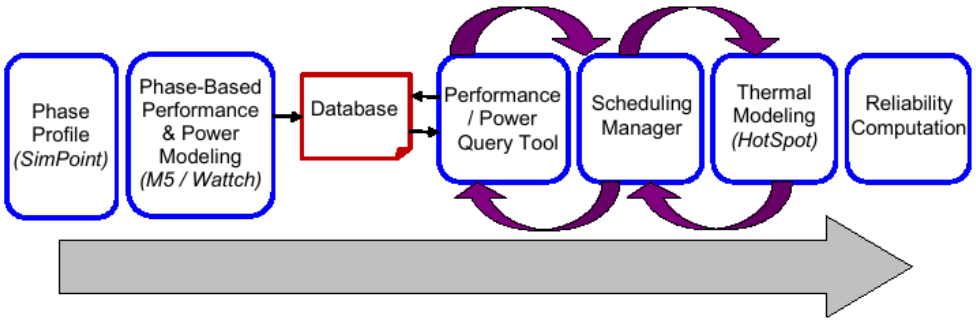


Figure 2.4 – *Méthodologie de simulation de la fiabilité en fonction des applications exécutées sur un SoC. La base de données permet d'accélérer la simulation d'un MPSoC complet et nécessite deux étapes préliminaires de repérage des phases des applications ainsi que le calcul détaillé de performance et de consommation électrique. [Cos+09]*

D'autres études du même auteur [Cos+06] utilisent des simulations plus rapides qu'au niveau architectural. Toutefois le modèle de fiabilité dépend seulement des valeurs instantanées de tension et température. Les instants passés ne sont donc pas pris en compte. En revanche, les résultats doivent être calibrés par des mesures ou des formules mathématiques disponibles pour chaque modèle de processeur.

5. Instruction Per Cycle

2.4.3 Niveau système/ESL

Enfin, notons les approches de simulation [RMD07] basées sur des états du système. Par exemple, différentes tensions d'alimentation, ou différents types d'exécution peuvent se résumer à une consommation électrique et/ou une température constantes. Les dégradations sont ensuite calculées à partir de mesures réalisées dans le cadre d'un partenariat avec un fabricant.

Le point majeur est la possibilité de spécifier certains mécanismes de défaillance comme actifs ou non selon l'état du système. Il est supposé ne pas avoir d'électromigration pendant la phase de veille du processeur, étant donné que les courants ne circulent pas dans les interconnexions. Sur le tableau 2.2, différents états de fonctionnement sont résumés dans la première colonne. Pour certains d'entre eux, il existe une distribution et un paramètre qui représente au mieux l'apparition des défaillances. À l'état de repos, les défaillances apparaissent uniformément entre le temps d'entrée en veille t_{min} et de sortie t_{max}. Enfin l'expression du taux de défaillance est donnée pour chaque état. On note que l'électromigration est active lors de l'activité (état actif) ou du repos du système et le TDDB apparaît tout le temps, sauf en phase de veille. Enfin, les variations thermiques ne sont considérées que lors des instants de veille.

État	Distribution	Paramètres	Taux de défaillance
Actif	Exponentiel	$\phi_{core,f-V}$	$\lambda_a^{EM} + \lambda_a^{TDDB}$
Repos	Toute	Données mesurées	$\lambda_i^{EM} + \lambda_i^{TDDB}$
Transition	Uniforme	t_{min}, t_{max}	
Veille	Toute	Données mesurées	$\lambda_s^{TDDB} + \lambda_s^{TC}$

Tableau 2.2 – *Modélisation des états d'un système (veille, activité, transition ou repos) pour la simulation de la fiabilité. Dans chaque cas, le type de distribution des défaillances est spécifié ainsi que les paramètres servant à calculer leur effets (taux de défaillance, tension/fréquence, mesures). [RMD07]*

Un simulateur a été proposé pour calculer la fiabilité en fonction de l'état du système. Chaque cœur est représenté comme une machine d'état représentant sa fiabilité et sa consommation dans chacun de ses états. Aussi le taux de défaillance est constant pendant un état. La température est estimée par des modèles mathématiques simples simulant le boîtier et l'interface avec l'extérieur.

Les évolutions entre les états du système sont simulées ou des méthodes statistiques sont utilisées [Gui11]. Les besoins en ressources de simulation sont faibles ; dans l'exemple proposé ci-avant, seuls quatre états sont utilisés pour un processeur complet. Par contre l'adaptation à des simulateurs existants, notamment HotSpot n'est pas encore réalisée et les résultats de température ne sont pas validés.

2.5 Synthèse des méthodes de simulation du vieillissement et positionnement de l'étude

Nous résumons sur les tableaux 2.3 et 2.4 les méthodes de simulations du vieillissement (type de mécanismes, de simulateur et niveau d'abstraction).

Auteurs	Mécanismes	Simulateur	Niveau d'abstraction
[AVG07]	NBTI	SPICE	Transistor
[Rui+11] [LGS09]	HCI, NBTI	PrimeTime	Porte logique
[Sri+03]	TDDB, NBTI	Wattch Hotspot	Micro- architecture
[Gup+11a]	EM, HCI TDDB, NBTI	Wattch Hotspot	Fonctionnel
[Cos+09]	EM, TDDB, TC	SimPoint	Système/ESL
[RMD07]	EM, TDDB, TC	Machine d'état	Système/ESL

Tableau 2.3 – *Présentation des méthodes existantes de simulations de la fiabilité. Les mécanismes de défaillances et les types de simulateurs sont reportés ainsi que leur niveau d'abstraction.*

Notre objectif était de définir une méthode de conception rapide permettant de simuler le vieillissement des processeurs. Il est nécessaire de disposer de paramètres à haut niveau d'abstraction pour que :

- Les simulations soient rapides et que la précision soit bonne, bien que réduite ;
- on puisse lancer les simulations dès les premiers cycles de conception.

Les avantages et inconvénients des différentes approches pour simuler des CI pendant la conception sont les suivants (cf. tableau 2.4).

- Les simulations au niveau transistor offrent une très grande précision pour le calcul de la fiabilité du système. Cependant, l'exploration de différentes configurations d'un circuit nécessite un temps de simulation très important. Elles nécessitent une description physique du circuit, très volumineuse et complexe dans le cas de processeur.
- Les approches de calcul de fiabilité niveau porte logique présentent un temps de simulation assez grand pour une bonne précision. Bien qu'elles s'intègrent simplement dans les méthodologies de conception, elles incluent le détail des portes logiques, qui ne sont pas faciles à analyser dans le cadre de CI avec de nombreuses portes logiques par exemple.
- Les approches basées sur la microarchitecture permettent d'accélérer notablement le temps de simulation des processeurs mais ne disposent pas de modèles de dégradation validés.
- Enfin, les approches par annotation d'un ISS ou par simulation architecturale permettent une exploration aisée car le temps de simulation est faible. Cependant, elles considèrent

que le chemin le plus sensible est le chemin critique pendant la durée de vie du circuit. Or, il a été démontré que d'autres chemins peuvent enfreindre les contraintes de temps [LGS09]. Aussi, elles utilisent des hypothèses fortes sur la dégradation et surestiment sa valeur.

Auteurs	Méthode de simulation	Avantages	Inconvénients
[AVG07] [Tu+93] [Cadb]	SPICE FastSPICE	Analyse des transistors Grande précision	Très long temps de simulation Description nécessaire (*netlist*)
[Rui+11] [LGS09]	RTL	Analyse des portes et chemins logiques Intégrable dans un flot de conception Bonne précision	Long temps de simulation Description des portes logiques nécessaire
[Sri+03]	Traces cycle Monte Carlo	Intégrable dans un flot de conception Prise en compte des applications	Forte hypothèse (densité uniforme de probabilité de défaillance) Pas de lien entre technologie, température et fiabilité
[Gup+11a]	ISS annoté	Prise en compte des données fondeur Intégration simple dans les simulateurs du jeu d'instructions	Perte en précision des chemins logiques Pas de modèle de dégradation disponible
[Cos+09]	SimPoint Base de donnée	Simulation très rapide Intégration simple dans les simulateurs du jeu d'instructions	Calibration nécessaire Pas de modèle de dégradation disponible

Tableau 2.4 – *Comparaison des avantages et inconvénients des méthodes de simulation de la fiabilité pendant la phase de conception d'un circuit.*

À notre connaissance, aucune étude n'a proposé de modèle de dégradation au niveau fonctionnel pour simuler les MPSoC qui prenne en compte les chemins du circuit. C'est pourquoi nous proposons une nouvelle méthode de simulation au niveau fonctionnel dans le prochain chapitre. Elle est basée sur des modèles empiriques et elle est validée au chapitre 4 pour un processeur embarqué.

CHAPITRE 3

SIMULATION À HAUT NIVEAU DE L'EFFET DU HCI

Les effets des dégradations sur les circuits sont de plus en plus importants comme nous l'avons vu au chapitre 1. En particulier le HCI et le NBTI entraînent des effets négatifs sur les caractéristiques temporelles des circuits. La simulation des effets du vieillissement devient nécessaire lors de la conception, pour être appliquée aux MPSoC par exemple.

Le chapitre 2 a décrit les techniques actuelles de simulation des dégradations. En particulier, les modèles de dégradation disponibles dans la littérature sont validés pour les niveaux transistor et porte logique. Or, à ces niveaux, les simulations sont lentes et donc mal adaptées à la simulation itérative, utilisée pendant la conception. Les niveaux supérieurs doivent donc être utilisés pour simuler la dégradation des circuits. Cependant, l'utilisation de niveaux supérieurs ne permet pas de décrire les portes logiques. Et nous avons vu que les modèles de dégradation les plus récents qui soient validés pour une technologie donnée, sont au niveau porte logique.

Notre objectif est d'estimer la dégradation temporelle des CI. Les principaux besoins sont :

- une simulation assez rapide pour pouvoir être exécutée itérativement pendant les premières étapes de conception ;
- et une prise en compte des conditions d'utilisation et de la topologie du circuit.

Nous proposons une méthodologie d'estimation des effets du HCI avec l'aide d'une simulation fonctionnelle. Nous l'appelons **simulation fonctionnelle du HCI**. Pour le NBTI, une technique similaire peut être employée avec un modèle de dégradation du NBTI ramené au cycle, qui soit validé. Pour éviter les répétitions, nous n'étudions que le cas du HCI dans ce chapitre.

Le simulateur des dégradations dues au HCI est construit à partir d'un ISS, de la description comportementale du système et du modèle de dégradation d'une porte logique pour le HCI. Les ISS simulent les instructions sans le détail des temps de propagation. Le niveau fonctionnel utilisé dans les ISS permet de grandes vitesses de simulations qui sont donc adaptées aux MPSoC. Nous proposons d'inclure le modèle de dégradation du HCI au niveau porte logique, obtenu par une simulation préliminaire, dans un ISS. Ainsi, il est possible de calculer la dégradation par une simulation fonctionnelle en un temps court. Cette technique prend en compte la tension

d'alimentation, la fréquence, les applications et des informations structurelles du circuit pour étudier le HCI. En modifiant le fonctionnement de l'ISS, il est possible de calculer les effets dus au HCI.

La description comportementale du système permet de définir les signaux d'entrée/sortie du circuit. Le problème est de pouvoir estimer la valeur des signaux internes au niveau fonctionnel. Nous nous plaçons dans le cas d'un processeur, exécutant des instructions, et nous proposons un modèle pour calculer les dégradations en fonction des instructions. Cette technique permet de ne pas ralentir l'ISS du système tout en fournissant les données de dégradation. Le modèle de dégradation au niveau porte logique permet de définir le délai initial et les constantes technologiques du circuit pour quantifier la dégradation. Nous utilisons un modèle empirique que nous évaluons par simulation au niveau fonctionnel de la dégradation HCI d'un processeur. La description de ces techniques est développée selon le plan suivant.

L'organisation de ce chapitre se compose en cinq parties.

1. Nous présentons la méthodologie employée pour construire le simulateur fonctionnel du HCI.
2. Nous définissons les termes techniques utilisés pour estimer les dégradations.
3. Dans l'objectif de validation et de construction des modèles d'instructions, nous mettons en place le simulateur des effets du HCI au niveau porte logique. Les outils standards de conception sont utilisés et augmentés d'une donnée de dégradation.
4. Sur la base des travaux au niveau porte logique, nous proposons un nouveau modèle de dégradation d'un circuit, appliqué au niveau fonctionnel.
5. Enfin, nous proposons l'intégration de la simulation fonctionnelle du HCI dans un ISS, pour un processeur.

Les deux derniers points représentent les contributions majeures du travail réalisé. Nous concluons finalement sur les caractéristiques de la méthode de simulation fonctionnelle proposée et sur sa validation. Nous commençons par décrire les détails de notre méthodologie et les définitions des grandeurs mathématiques utilisées.

3.1 Méthodologie

La méthodologie se développe comme suit. Dans le cas d'un processeur au niveau fonctionnel, seul le jeu d'instructions est disponible. Nous allons alors intégrer un modèle de dégradation pour calculer les délais en fonction des instructions. Une micro-instruction est un mot sur plusieurs octets stocké dans la mémoire d'instructions et contenant une opération avec les opérandes associés. Le processeur est ensuite configuré pour exécuter l'opération. Ce processus, depuis la lecture de la mémoire d'instructions jusqu'au calcul du résultat est exécuté séquentiellement, en plusieurs cycles. Dans notre cas, nous supposons qu'une seule micro-instruction est exécutée à chaque cycle. On considère la micro-instruction lue de la mémoire d'instructions. De ce fait, les commutations générées pendant un cycle d'horloge au niveau porte logique sont représentées par une valeur correspondante pour le niveau fonctionnel. On parle de modèle d'instructions pour relier les commutations des chemins et le jeu d'instruction. Nous considérons deux modèles selon que l'on considère les commutations pendant l'exécution de chaque micro-instruction au cycle i (instruction courante) ou bien entre le passage entre les micro-instructions des cycles i et $i+1$ (paires).

Le choix d'étudier les paires de micro-instructions est motivé par le modèle de dégradation du HCI présenté au chapitre 1, section 4. Les commutations, puis par extension, la dégradation due au HCI, dépend du changement d'état du circuit et donc des changements de micro-instructions exécutées.

Dans la suite du chapitre, nous présenterons la méthode d'estimation des dégradations dues au HCI par ses 3 étapes.

- Le point de départ est une description au niveau porte logique du circuit à étudier.
- La deuxième étape est la définition d'un modèle de dégradation du circuit appliqué au niveau fonctionnel. Ce modèle est capable d'estimer les retards en l'absence de description détaillée des parties internes du circuit. Nous utilisons le simulateur au niveau porte logique pour construire les modèles d'instructions.
- La troisième étape consiste à augmenter un simulateur fonctionnel par le modèle proposé. On peut alors simuler le comportement de la dégradation due au HCI en fonction des instructions.

3.2 Définitions

Les commutations dans les circuits logiques

Une commutation ou un basculement d'un signal est un changement de son état logique de la valeur '0' à la valeur '1' ou de la valeur '1' à la valeur '0'.

Le temps simulé T_{sim} est un multiple de la période d'horloge T_{clk} : $T_{sim} = N_{sim} \cdot T_{clk}$, où N_{sim} est le nombre de cycle de simulation.

Le nombre de commutations par période d'horloge α_s d'un signal s est le nombre de commutations du signal pendant une période d'horloge. Nous donnons un exemple de son calcul figure 3.1.

Le nombre de commutations TC_s d'un signal s est le nombre total de commutations pendant une simulation de durée T_{sim}.

$$TC_s = \sum_{i=1}^{N_{sim}} \alpha_{s,i} \tag{3.1}$$

Le nombre de commutations TC_p d'un ensemble de nœuds est égal à la somme des nombres de commutations des nœuds, qui sont en nombre N, pendant la simulation durant N_{sim} cycles.

$$TC_p = \sum_{j=1}^{N} TC_s(j) = \sum_{j=1}^{N} \sum_{i=1}^{N_{sim}} \alpha_{s,i}(j) \tag{3.2}$$

Le nombre de commutations moyen par nœud $\overline{TC}$ d'un circuit est égal au rapport entre le nombre de commutations et le nombre de nœuds N_c du circuit.

$$\overline{TC} = \frac{TC_c}{N_c} \tag{3.3}$$

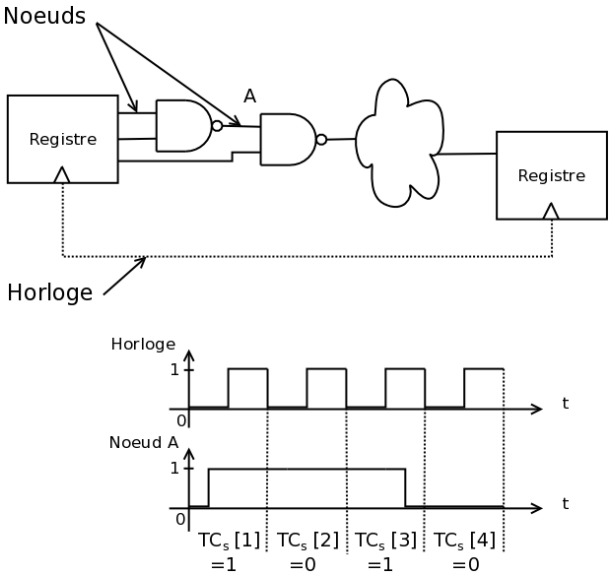


Figure 3.1 – *Illustration d'un circuit combinatoire et de registres séquences par une horloge. Les valeurs de commutations par période d'horloge $TC_s[i]$ du signal sur le nœud A sont comptées à chaque période d'horloge i.*

Un nœud ou un fil est un signal logique dont la valeur est soit '0' soit '1' et qui relie des portes logiques entre elles.

Une porte logique permet de réaliser une fonction logique simple ou complexe. On peut citer par exemple les fonctions simples ET, OU, et d'inversion et les fonctions complexes AOI. Cette appellation signifie ET-OU-Inversion et comporte une combinaison de plusieurs portes de fonction logique ET, OU et d'inversion.

Les cellules standards sont des portes logiques associées à une bibliothèque technologique, définie par le fabricant pour un certain type de procédé de fabrication et des conditions d'utilisation (température, polarisation).

Les bibliothèques de cellules standards contiennent des éléments combinatoires et séquentiels. On donne figure 3.2 une représentation schématique d'une cellule standard simple pour la fonction logique NON-ET.

Les commutations d'un chemin sont la somme des commutations des cellules logiques du chemin. Sur la figure 3.3, on représente un ensemble $g_i, i \in [1;N]$ de portes logiques dont on veut connaître les commutations. Au niveau d'un nœud i, on a TC_{si} commutations. La formule 3.4 permet de calculer les valeurs de commutations TC_p pour un chemin p.

$$TC_p = \sum_{si \in [s1;sN]} TC_{si} \tag{3.4}$$

3.3 Modèle de dégradation au niveau porte logique

L'objet de cette section est de présenter les hypothèses de modélisation des dégradations au niveau porte logique. Les modèles disponibles dans la littérature se basent sur la description des

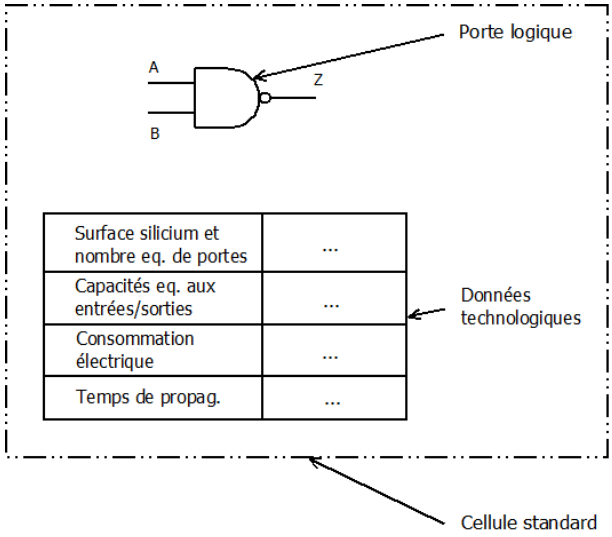


Figure 3.2 – *Composition d'une cellule standard simple (fonction logique NON-ET). Une porte logique relie les entrées/sorties primaires et les données technologiques sont fournies par le fabricant.*

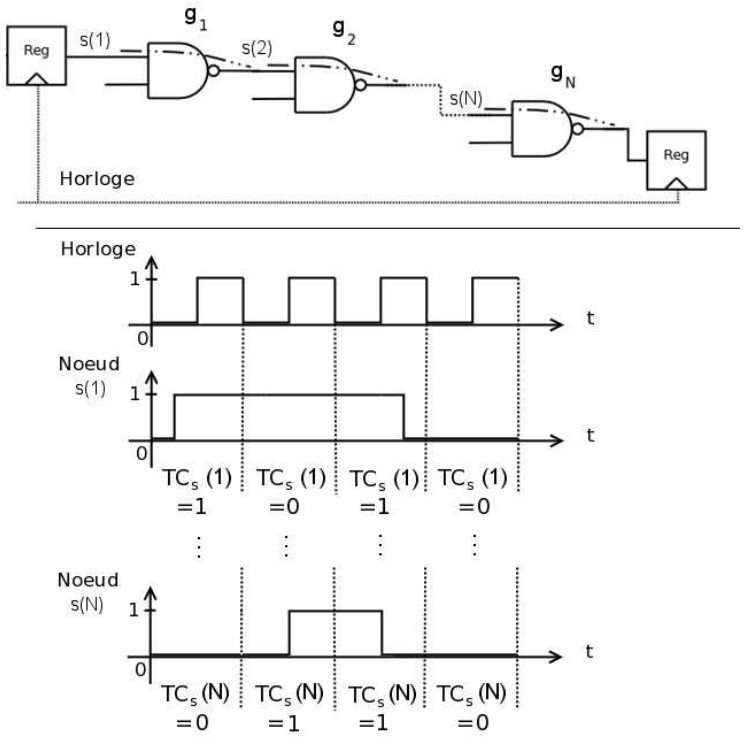


Figure 3.3 – *Illustration du calcul des commutations d'un chemin logique défini par un point de départ, un ensemble de cellules standards et un point d'arrivée.*

transistors du circuit. Après un rappel des notions de la dégradation HCI d'une porte logique, nous détaillons la méthode utilisée dans ces approches pour abstraire les modèles niveau transistor. Nous utilisons le modèle de dégradation du HCI pour d'une porte logique présenté au chapitre 1, section 4. Les détails de cette méthode sont présentés dans cette section.

3.3.1 La dégradation des délais

Nous comparons dans ce paragraphe le modèle de délai que nous utilisons avec les deux techniques les plus récentes de calcul de la dégradation des délais [Hua+11 ; Lor12].
Les temps de propagation d'une porte logique dépendent d'un ensemble P de paramètres et de chaque transistor (dont l'ensemble est G). Alors le temps de propagation final d_{aged} peut s'écrire en fonction du temps de propagation initial d_{fresh} et de la sensibilité χ aux paramètres Δp_m de chaque transistor.

$$d_{\text{aged}} = d_{\text{fresh}} + \Delta d = d_{\text{fresh}} + \sum_{p \in P} \sum_{m \in G} \chi_{m,p} \cdot \Delta p_m \tag{3.5}$$

Dans notre cas, nous considérons que le délai de propagation d'une porte se réfère à un seul arc temporel. Ce résultat peut être extrapolé à tous les arcs temporels d'une porte de la même manière que dans [Hua+11]. La relation mathématique avec les paramètres technologiques est présentée ci-après.

Lorenz a montré qu'on peut utiliser un modèle linéaire pour représenter les variations de délais d'une porte avec une très bonne précision [Lor12]. Pour les transistors MOS affectés par le NBTI et le HCI, il est possible d'exprimer la dégradation en fonction de la tension de seuil et du courant de saturation. Comme expliqué au chapitre 1, le courant de saturation joue un rôle majeur dans le phénomène du HCI et nous nous limiterons à son étude. Toutefois, sa valeur dépend de la température, la tension d'alimentation et de la technologie de fabrication du transistor. Dans cette étude nous ne prenons pas en compte ces effets. Par contre, nous présentons ci-après la modélisation du temps de stress que nous avons adopté.

Si on suppose les transistors de mêmes dimensions et alimentés par la même tension, on peut écrire le temps de propagation final d_{aged} d'une porte logique par la formule 3.6.

$$d_{\text{aged}} = d_{\text{fresh}} + \sum_{m \in G} \chi_{m,I_{on}} \cdot \Delta I_{on}(t) \tag{3.6}$$

Or ΔI_{on} est le courant de conduction du transistor et varie selon une loi puissance d'après le modèle de Takeda et Suzuki [TS83] lorsque le transistor est sous stress pendant un temps t : $\Delta I_{on}(t) = A \cdot t^n$ avec A constante expérimentale. On peut alors relier le temps de stress et la variation de I_{on} par la formule 3.7 présentée au chapitre 1, avec les constantes technologiques C et n. On suppose de plus que le temps de stress est proportionnel au nombre de commutation du signal d'entrée de la porte logique. Aussi, le temps de stress et la sensibilité de chaque transistor sont identiques, ce qui permet d'écrire $\sum_{m \in G} \Delta I_{on}(t) = \sum_{m \in G} A \cdot t^n \approx B \cdot t^n$. En définissant θ comme l'inverse du nombre moyen de commutation ($\theta = t/\overline{TC}$) et $C = B \cdot \theta^n$ on retrouve bien l'expression de l'équation 3.7 dans le cas où n a pour valeur 1. D'autres valeurs de n nécessitent plus de ressources de simulation car il faut calculer les temps de stress de chaque chemin et de chaque nœud du circuit. Dans la suite, nous nous limiterons au cas $n = 1$.

$$d_{\text{aged}} = d_{\text{fresh}} + C \cdot \left(\overline{TC}\right)^n \tag{3.7}$$

Ici, la température n'est pas considérée car les temps de variation de la température sont beaucoup plus grands que les variations des signaux logiques [MAR10]. Ainsi, la prise en compte de la température ne se fait qu'au niveau d'abstraction plus élevé, discuté dans la suite du document.

Dans les formules de dégradation des délais présentées ici, les commutations doivent être calculées. Nous présentons ci-après la méthode de calcul des commutations à haut niveau.

3.3.2 Le calcul des commutations au niveau fonctionnel

Notre objectif est de calculer les dégradations au niveau fonctionnel. Pour cela, nous détaillons les formules de dégradation d'un circuit. En particulier, nous calculons les commutations des chemins au niveau fonctionnel.

Les commutations d'un chemin j sont calculées à partir pour l'ensemble des $m(j)$ portes logiques interconnectées qui forment le chemin. On calcule le nombre de commutations $sumTC(j)$ du chemin j, par la formule 3.8 où $TC_{s_e(j)}$ est le nombre de commutations du nœud entrant $s_e(j)$ du chemin j.

$$sumTC(j) = m(j) \cdot \left(TC_{s_e(j)}\right)^n \tag{3.8}$$

Au niveau porte logique, nous avons établi dans la section 5, au paragraphe 2, que les dégradations de délais sont calculées par l'équation 3.5. En l'appliquant au calcul du délai final $d_{aged}(j)$ du chemin j on obtient l'équation 3.9.

$$d_{aged}(j) = d_0 + C \cdot \sum_i TC(i,j)^n \tag{3.9}$$

Au niveau fonctionnel, les signaux en entrée des chemins permettent d'estimer les dégradations de délais. Dans ce cas, on considère que les commutations $TC(i)$ des portes logiques des chemins sont identiques pour toutes les portes i du chemin. Dans le cas où $n = 1$, le délai de propagation final $d_{aged}(j)$ d'un chemin du bloc j est majoré par la formule de l'équation 3.10.

$$d_{aged}(j) = d_0 + C \cdot \sum_{g_i} TC(i,j)^n < d_0 + C \cdot sumTC(j) \tag{3.10}$$

$sumTC(j)$ est le nombre de commutations en comptant les commutations sur les signaux du chemin j. d_0 est le délai de propagation initial du chemin. C est une constante reliée aux données technologiques. Nous faisons ici l'hypothèse que $\forall i \in [1;m], (TC(i,j))^n < sumTC(j)$.

Nous avons présenté les hypothèses de calcul des dégradations au niveau porte logique. En particulier, la dégradation d'un transistor et des délais servent à simuler les dégradations d'un circuit au niveau porte logique. Enfin, la méthode de calcul des commutations au niveau RTL nous permet de proposer un modèle au niveau fonctionnel dans les sections suivantes.

3.4 Simulation du HCI au niveau porte logique

Nous reprenons ici une méthode standard de simulation des dégradations d'un circuit au niveau porte logique. Elle utilise les travaux antérieurs de simulation du HCI au niveau porte logique [Hua+11 ; Lor12]. Les conditions de dégradations sont accélérées pour permettre l'analyse de l'effet de la dégradation du circuit par un temps de simulation court. Enfin, les chemins les plus affectés par le vieillissement sont identifiés. Les deux étapes sont décrites ci-dessous.

1. L'environnement est mis en place pour calculer l'activité des nœuds et il est mis à jour à chaque cycle de simulation.
2. Les résultats de simulation en fin d'exécution sont relevés.

La simulation des dégradations est implémentée dans des outils de simulation de circuits numériques. Nous avons ajouté une fonctionnalité pour réaliser la simulation des dégradations.

Le modèle de dégradation des délais d'une porte logique, défini dans la section précédente est utilisé pour calculer les temps de propagation du circuit vieilli, que nous appelons temps de propagation final du circuit. Nous nous inspirons des approches de vérification lors de la conception de circuits. Pour cela, des vecteurs sont appliqués sur les entrées du circuit et une simulation est lancée. Les résultats affichés par le simulateur sont alors comparés avec un seuil défini par le cahier des charges du circuit. Dans notre cas, une estimation des temps de propagation des signaux est effectuée. Nous avons choisi de calculer l'état final du circuit en un cycle car l'utilisation des données initiales permet de calculer les dégradations du circuit vieilli en une étape unique, avec très peu d'erreur par rapport à une mise à jour à chaque cycle d'horloge [Lor12].

Nous détaillons ci-après la méthode de calcul des dégradations au niveau porte logique, figure 3.4. Dans le cas d'un processeur, les stimuli sont fournis par une application, compilée pour l'architecture du processeur. La bibliothèque technologique sert à effectuer la simulation des cellules logiques générées lors de la synthèse logique. Ensuite, les trois résultats suivants sont reportés en fin de simulation :

– les marges temporelles initiales (ou en début de simulation) ;
– les chemins critiques ;
– et les commutations des signaux.

Un outil a été développé pour évaluer les paramètres du modèle de dégradation des portes logiques (*Cell delay updater*). Avec trois paramètres, la simulation des délais du circuit vieilli est rendue possible en construisant une nouvelle bibliothèque technologique. D'abord, les commutations des signaux sont récupérées depuis la simulation logique. Ensuite, l'équation 3.7 de dégradation de délai est utilisée. Enfin, la bibliothèque technologique de cellules standards est mise à jour. Pour cela, les chemins sont identifiés, puis les instances des cellules qui appartiennent à ces chemins sont mises à jour. Nous donnons un aperçu figure 3.5 du résultat de la mise à jour de la bibliothèque et nous décrivons la procédure mise en place dans la fonction *Cell delay updater*.

Pour mettre à jour les délais de la bibliothèque technologique, trois étapes sont nécessaires.

1. L'ensemble des cellules du circuit sont parcourues et celles qui appartiennent aux chemins critiques sont sélectionnées.
2. Les commutations et leurs polarités à chaque entrée des cellules concernées sont calculées en fin de simulation.
3. Les temps de propagation en fonction des fronts montants et descendants sont incrémentés, selon le modèle de dégradation.

Les valeurs initiales de temps de propagation sont obtenues à partir de la bibliothèque technologique et pour le point de fonctionnement initial, fourni par le concepteur. Un point de fonctionnement est défini par des renseignements sur le procédé de fabrication, la température et la tension d'alimentation. L'évolution de ces valeurs au cours de la simulation n'est pas prise en compte durant cette étape.

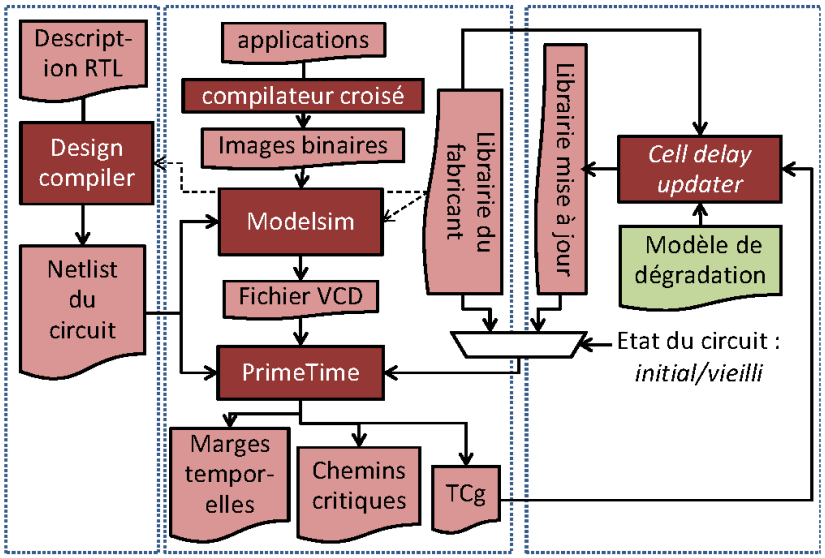


Figure 3.4 – *Méthode de simulation des dégradations au niveau porte logique. Les premières étapes de conception (RTL et netlist) sont utilisées pour effectuer une simulation dynamique des délais. Enfin, la bibliothèque technologique est mise à jour en fonction du modèle de dégradation pour calculer les délais du circuit vieilli.*

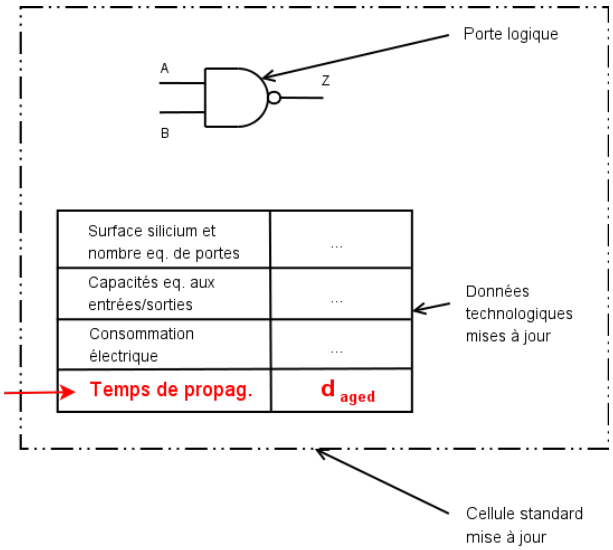


Figure 3.5 – *Illustration du résultat de mise à jour d'une cellule standard après l'étape de calcul des temps de propagation finaux* d_{aged}.

Finalement, une dernière simulation des temps de propagation est effectuée en utilisant la bibliothèque mise à jour. L'outil d'analyse *PrimeTime* est réutilisé de façon automatique en sélectionnant la nouvelle bibliothèque. Les marges temporelles en fin de simulation sont alors reportées.

À partir de cette simulation, il est possible d'étudier le circuit au niveau porte logique. Cependant, nous cherchons une méthode de simulation utilisable pendant la conception. Il est donc nécessaire de simplifier les données disponibles au niveau porte logique au maximum pour limiter la mémoire occupée et accélérer les calculs. Pour cela, nous proposons deux techniques présentées ci-dessous. La sélection des chemins les plus critiques permet de réduire la mémoire occupée par les données de dégradation et la simulation accélérée diminue le temps de simulation.

3.5 Modèle de dégradation au niveau fonctionnel pour un processeur

Le but ici est de présenter le contexte d'étude du processeur. Ensuite, nous caractérisons les valeurs d'activité de commutation en fonction des instructions. Enfin, nous construisons les modèles de dégradation des chemins au niveau fonctionnel.

Dans le cas d'un processeur au niveau fonctionnel, seul le jeu d'instructions est disponible. Nous allons alors intégrer un modèle de dégradation pour calculer les délais en fonction des micro-instructions. Pour cela, nous supposons que chaque cycle est représenté par une micro-instruction exécutée. On considère la micro-instruction lue de la mémoire d'instructions dans notre étude. De ce fait, les commutations générées pendant un cycle d'horloge au niveau porte logique sont représentées par une valeur correspondante pour le niveau fonctionnel. On parle de modèle d'instructions pour relier les commutations des chemins et le jeu d'instructions. Nous considérons deux modèles décrits ci-après.

1. Le modèle « U » considère les commutations pendant l'exécution de chaque micro-instruction au cycle i.
2. Le modèle « P » considère les commutations lors du passage de chaque paire de micro-instructions entre les cycles i et $i+1$.

Dans la suite, nous présentons la méthode de construction des modèles d'instructions. Pour cela, le déroulement de l'exécution d'une micro-instruction dans la microarchitecture d'un processeur est présenté. Puis, la caractérisation des commutations au niveau porte logique est détaillée. Enfin, la construction des modèles d'instructions est définie.

3.5.1 Microarchitecture de processeur étudiée

Nous présentons ici la micro architecture d'un processeur de type Reduced Instruction-Set Computer (RISC). La distinction entre parties combinatoires et séquentielles est décrite. Cette description permet par la suite de construire le modèle de dégradation au niveau fonctionnel.

Lors de la conception matérielle d'un processeur, la ou le micro architecte définit plusieurs registres qu'il insère entre des blocs combinatoires. Ces blocs correspondent aux différentes fonctionnalités du circuit. Par exemple, on trouve en première étape la lecture des mots d'instruction depuis la mémoire d'instructions. L'adresse est calculée et le transfert avec la mémoire d'instructions est géré. Ces fonctionnalités sont regroupées sous le nom « Fetch ». Enfin la valeur

effectivement lue et l'adresse sont stockées dans un registre synchronisé par l'horloge.
La seconde étape concerne le décodage des mots d'instruction lus, dénommée « Decode ». Le mot d'instruction est scindé en différents champs. Par exemple, sur une architecture de type RISC [HP06], les instructions d'appel de procédure possèdent un champ identifiant codé sur 4 bits et un champ d'adresse codé sur 28 bits. L'identifiant caractérise le nom de l'instruction, alors que l'adresse identifie l'endroit mémoire où est stockée la procédure à appeler.
Ensuite, vient la configuration des unités de l'architecture que va utiliser l'instruction (« Execute ») . Selon les ressources matérielles disponibles, une instruction va nécessiter un accès à certaines ressources matérielles globales du processeur et/ou aux autres étages. Puis, si l'instruction nécessite un accès à la mémoire de données, un étage « Memory » sert à y accéder. Enfin, la mise à jour éventuelle de la file de registres est réalisée par la partie « Writeback ».
Ces fonctionnalités appelées étages sont connectées les unes à la suite des autres, par des registres qui viennent accueillir les résultats de chaque cycle. Les différents étages du pipeline ainsi que la file de registres internes du processeur sont décrits figure 3.6.

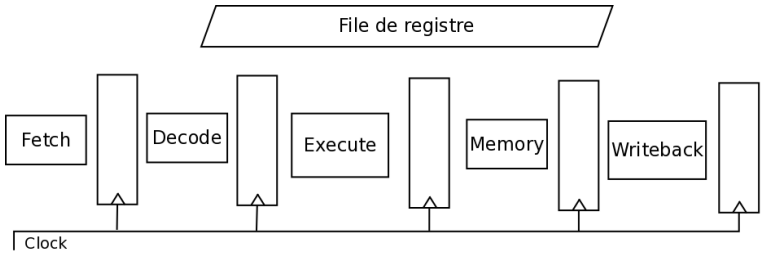


Figure 3.6 – *Définition d'une architecture pipelinée de 5 étages. Les différents blocs architecturaux représentent le parcours des instructions séquencé par une horloge.*

Nous considérons ici une architecture avec un pipeline, une partie de contrôle et une partie de calcul des données ; elles sont représentées figure 3.7. Il existe plusieurs étages et un additionneur. Nous étudions une architecture de type *load/store*, pour laquelle les micro-instructions d'accès à la mémoire sont séparées des autres micro-instructions.

Ce type d'architecture ne restreint pas la méthodologie proposée. Pour d'autres types, il suffira de repérer les différentes micro-instructions et de les classer selon les ressources qu'elles activent. Par exemple, les unités de multiplication peuvent être étudiées en prenant en compte la micro-instruction associée. Le jeu d'instructions peut être plus étendu dans ce nouveau cas, mais il contient un nombre fini de micro-instructions. Cette condition est suffisante pour pouvoir appliquer notre méthodologie. Nous présentons ci-dessous la définition et la classification des micro-instructions que nous avons choisies.

Une micro-instruction est un mot logique d'une taille standard comprenant un identifiant unique et un champ de données. L'identifiant permet de définir une opération logique alors que le champ de données permet de définir les registres internes ou les emplacements de la mémoire participant à l'opération. Nous considérons qu'une micro-instruction est définie par son identifiant unique.

Chaque micro-instruction fait varier différents signaux d'un circuit, selon sa fonctionnalité et les données qu'elle contient. Par exemple, pour un pipeline contenant 5 étages, le parcours de trois micro-instructions est représenté figure 3.8 en commençant par le chargement de la première micro-instruction (*sw*) dans le premier étage jusqu'à son arrivée dans le dernier étage. Entre temps les micro-instructions suivantes (*lw* et *nop*) sont exécutées parallèlement. Ainsi, pour un

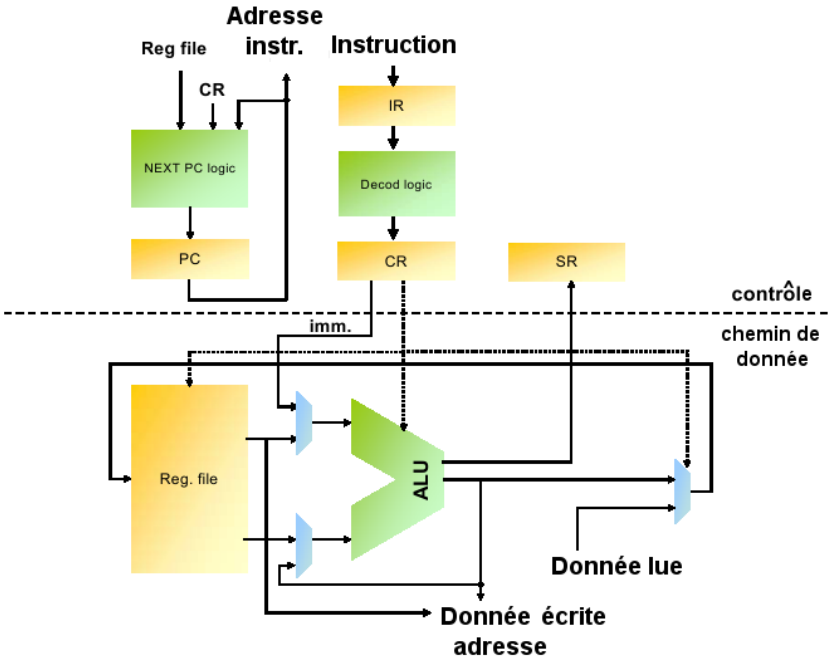


Figure 3.7 – *Unités d'exécution dans une architecture RISC avec un pipeline à 5 étages.*

étage particulier, une micro-instruction est présente pendant un cycle d'horloge. Il existe aussi des cas pour lequel la micro-instruction ne parcourt pas un étage. Dans notre exemple, lors de branchements, les deux derniers étages ne sont pas utilisés.

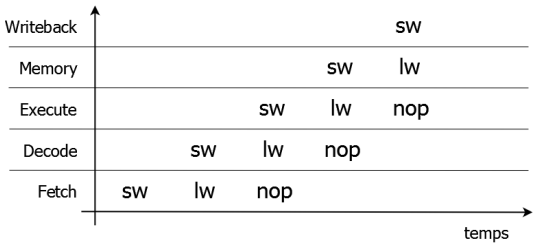


Figure 3.8 – *Étapes d'exécution de la séquence constituée des micro-instructions sw, lw et nop dans un pipeline à 5 étages.*

3.5.2 Caractérisation des commutations en fonction des micro-instructions

Nous présentons ici la méthode choisie pour modéliser les commutations, après avoir introduit les techniques similaires de la littérature. Notre méthode comprend l'écriture de programmes spécifiques et la simulation logique permettant de relever les commutations. Les commutations sont d'abord calculées par exécution de programmes unitaires avec une description au niveau porte logique du circuit. Ensuite, les modèles d'instructions sont introduits ; ils permettent de relier le nombre de commutations et les micro-instructions.

Comparaison à l'existant

Les commutations des signaux dépendent des changements de micro-instructions exécutées, selon le schéma figure 3.8. Nous faisons l'hypothèse que l'exécution d'une instruction est séquentielle dans le pipeline ; en d'autres termes, nous ne considérons pas la partie de contrôle de la figure 3.7. Sous cette hypothèse, la consommation électrique du CI à chaque cycle dépend seulement des changements ayant lieu dans le pipeline [TMW94]. Notre étude s'appuie sur la méthode de modélisation de puissance au niveau RTL. Selon le classement effectué dans [MPS98], nous utilisons des macro-modèles (ou des régressions) pour modéliser la puissance au niveau d'abstraction supérieur. Aussi, nous nous inspirons de l'approche [Gup+10a], qui permet de caractériser la consommation électrique d'un processeur MIPS en fonction des instructions (*ILPC*).

Nous effectuons tout d'abord une synthèse logique du système (cf. figure 3.4). Puis, nous mettons en place deux approches de modélisation qui considèrent respectivement l'instruction courante (qui est lue de la mémoire programme lors d'un cycle d'exécution) avec le modèle « U » et le changement d'instructions avec le modèle « P ». Les modèles sont ceux mentionnés au début de la section 5. Nous caractérisons le nombre moyen de commutations par chemin pour les deux modèles.

Les techniques de modélisation de la consommation ont montré que les paires d'instructions permettent d'obtenir des modèles avec peu d'erreur [Kla98], mais que le parcours de tous les cas possibles peut être trop long pour une utilisation lors de la conception de circuit. Dans notre cas, le modèle « P » est établi pour les paires et peut être appliqué à des processeurs suffisamment simples, à titre d'étude seulement. De plus, comme la dégradation due au HCI dépend des changements d'état des sigaux logiques, le modèle « P » est le plus adapté pour étudier les effets du HCI, dans le cas du processeur que nous présentons au chapitre suivant. Le modèle « U » est une approche plus réaliste car le modèle obtenu est suffisament simple pour les méthodes de conception.

Contrairement à la technique dans [Gup+10a], où un modèle est créé pour le processeur entier, nous considérons chaque chemin séparément. Aussi, au lieu de simuler des applications typiques de l'embarqué afin de générer les modèles, nous parcourons des séquences de caractérisation qui couvrent toutes les instructions du processeur. Une technique appliquée aux DSP [1] est présentée dans [Lau+04], où la puissance du circuit est mesurée sur circuit, en fonction de paramètres à haut niveau. Ces paramètres sont le taux de parallélisme et le taux de *cache miss* ou encore la fréquence du processeur et le mode mémoire. La phase de caractérisation nécessite une étude précise du parcours des instructions dans la microarchitecture du processeur. Dans notre cas, nous n'imposons pas une telle étude et nous parcourons systématiquement l'ensemble du jeu d'instructions du processeur.

Description de la méthode utilisée

Au total, notre méthode de génération des modèles de dégradation au niveau fonctionnel comporte 6 étapes que nous décrivons ci-dessous.

1. Chaque micro-instruction est séparée en un identifiant unique et un champ de données. Nous avons choisi d'utiliser l'identifiant unique car il conditionne le nombre d'étages parcourus. On peut ainsi discriminer les micro-instructions qui parcourent moins d'étages que d'autres, ce qui est le cas des branchements par exemple. Ensuite, le champ de données

1. Digital Signal Processor

peut prendre différentes valeurs pour un même identifiant. C'est pourquoi nous utilisons une seconde étape pour parcourir les valeurs possibles d'opérandes.

2. Les opérandes sont générés aléatoirement, pour parcourir le maximum de possibilités de valeur de données dans une micro-instruction. Un champ de données se décompose différemment selon le type d'instruction.
   Nous avons choisi de générer aléatoirement les valeurs de la file de registres et du champ de données des micro-instructions, sauf pour les micro-instructions de branchement. Pour celles-ci, les adresses sont prédéterminées.
3. Des programmes unitaires sont générés pour chaque instruction. Ils sont construits pour exécuter des micro-instructions avec les différents opérandes définis précédemment. Ces programmes diffèrent selon les deux modèles d'instructions et selon le type d'instruction. Chaque programme unitaire permet de générer les valeurs de commutation pour une micro-instruction (modèle « U ») ou une paire de micro-instructions (modèle « P ») et parcours les différents cas d'opérandes. Par exemple, pour la micro-instruction *(add.l)*, nous présentons l'algorithme du programme unitaire correspondant sur la figure 3.9. Les quatre premières micro-instructions permettent de vider le pipeline. Ensuite, un ensemble d'opérandes issu de la liste d'opérandes aléatoires est utilisé pour remplir les champs de la micro-instruction. Cette étape est appelée *condition*. Enfin quatre autres micro-instructions *nop* permettent de mesurer seulement l'effet dans chaque étage de la micro-instruction étudiée.

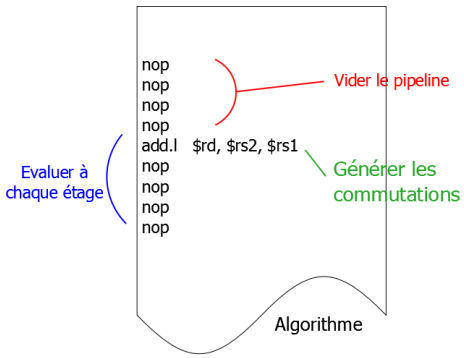


Figure 3.9 – *Algorithme utilisé pour évaluer les commutations générées par la micro-instruction (add.l).*

Dans le cas des paires de micro-instructions (modèle « P »), les micro-instructions sont exécutées les unes à la suite des autres.

4. Les stimuli sont générés par l'exécution des programmes unitaires. La simulation logique est réalisée grâce aux outils au niveau porte logique, décrits dans la section précédente. Par construction, les programmes unitaires permettent de relever les commutations pour chaque micro-instruction séparément. Ceci est compatible avec le niveau fonctionnel, pour lequel une instruction unique est exécutée à chaque cycle d'exécution.
5. Les commutations sont relevées sur les signaux appartenant à chaque chemin, et pour un état particulier du circuit. Nous décrivons ces deux caractéristiques ci-après.
   * Un chemin est constitué d'un ensemble de portes logiques, selon la définition adoptée dans ce chapitre. Nous observons les commutations en entrée de chaque porte logique constituant un chemin.

∗ Dans le cas du modèle « U », nous souhaitons mesurer l'effet d'une micro-instruction particulière. Pour cela, nous fixons le pipeline dans un état neutre en exécutant les micro-instructions *nop*. Aussi, les prologues et épilogues nécessaires au démarrage des programmes et la phase d'initialisation des registres sont ignorées.

Ensuite, pour chaque identifiant d'instruction, il existe plusieurs possibilités de valeur du champ de données et de la file de registres. Nous choisissons de parcourir un ensemble de valeurs aléatoires, fixé initialement pour tous les programmes unitaires. Par exemple, la micro-instruction *add.l* utilise des mots sur l'entrée et la sortie de l'ALU. Dans un processeur, l'Unité Arithmétique et Logique ou ALU en anglais, effectue les opérations mathématiques et logiques sur les données des registres. Dans ce cas, nous parcourons différents opérandes parmi toutes les valeurs possibles pour les mots en entrée de l'ALU.

Nous utilisons deux approches pour parcourir des valeurs possibles du champ de données. Elles peuvent être générées automatiquement et se basent sur l'exécution des programmes unitaires. 20 valeurs sont générées selon une loi uniforme parmi l'ensemble des valeurs d'un nombre entier non-signé codé sur 32 bits (soit un intervalle $[0;4\,294\,967\,296]$).

∗ On parcourt l'ensemble des conditions pour 20 instructions, selon la génération aléatoire des opérandes.

∗ On parcourt l'ensemble des conditions pour 20 paires d'instructions, selon une double génération aléatoire des opérandes.

6. Deux modèles sont générés à partir des mesures effectuées à l'étape précédente. Chaque micro-instruction ins_i est caractérisée en calculant la donnée $sumTC(ins_i)$ au moment où la micro-instruction est exécutée (modèle « U »). Chaque paire de micro-instructions (ins_i, ins_j) est caractérisée en calculant la donnée $sumTC(ins_i, ins_j)$ (modèle « P »). Enfin, la moyenne des commutations est effectuée et sa valeur est stockée dans le modèle. On donne figure 3.10 un exemple pour la paire de micro-instructions *(addc.l, jmp.l)*.

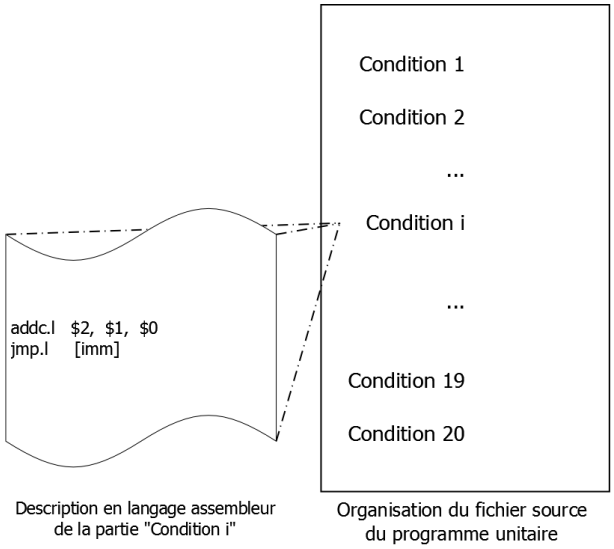


Figure 3.10 – *Description de l'algorithme et du code assembleur pour générer les modèles d'instructions. Les commutations sont mesurées pour chacune des* 20 *conditions, lors de l'exécution.*

Finalement, pour chaque micro-instruction du jeu d'instructions, l'analyse des dégradations est lancée et le résultat est stocké dans le modèle. Cette méthode est résumée figure 3.11 et l'estimation des dégradations par micro-instruction se fait en deux étapes présentées ci-après.

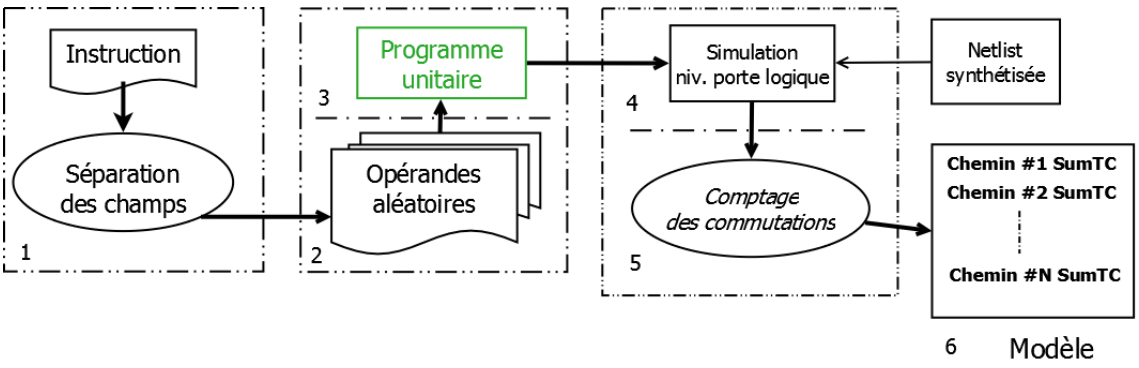


Figure 3.11 – *Méthode générale de caractérisation des dégradations du niveau porte logique au niveau fonctionnel. Le parcours du jeu d'instructions et des opérandes permet de caractériser l'exécution.*

Pour parcourir des opérandes différents, il est nécessaire de réaliser plusieurs exécutions. Ainsi, le modèle obtenu est représentatif du cas où l'exécution est avec un opérande quelconque.

Récapitulatif de la caractérisation au niveau instruction

Les programmes unitaires permettent de parcourir le jeu d'instructions et différentes possibilités de valeur d'opérande pour chacune des instructions. La simulation au niveau porte logique permet de calculer les commutations. Enfin, un modèle est créé pour chaque micro-instruction en calculant les commutations de chaque chemin. Pour le modèle « U », on stocke dans un tableau à deux dimensions les valeurs $sumTC(ins_i)$ en fonction des n micro-instructions ins_i et des N chemins P_k. Pour le modèle « P », on stocke dans un tableau à deux dimensions les valeurs $sumTC(ins_i, ins_j)$ en fonction des $\frac{n \cdot (n+1)}{2}$ paires de micro-instructions (ins_i, ins_j) et des N chemins P_k. Le parcours du jeu d'instructions permet d'obtenir un modèle au niveau fonctionnel pour chaque micro-instruction comme présenté figure 3.12. Les commutations sont alors définies pour toute exécution qui se ramène à une suite de micro-instructions exécutées les unes à la suite des autres.

Pour le modèle « U », le résultat de la caractérisation est donné dans le tableau 3.1 a) et le modèle « P » est dans la partie b).

Si le jeu d'instructions est très grand, la génération des modèles peut être coûteuse en ressources de calcul. Nous évaluons la complexité de la génération des deux modèles définis ci-dessus.

Pour le modèle « U », l'effort de simulation est proportionnel au nombre de micro-instructions du jeu d'instructions. Pour le modèle « P », on considère que le passage d'une micro-instruction 'A' à une micro-instruction 'B' est identique au passage de 'B' à 'A' en termes de commutation. Cette propriété découle du fait que l'activité est indépendante du sens de variation des signaux logiques. Ainsi, le parcours de chaque paire du jeu d'instructions représente un effort proportion-

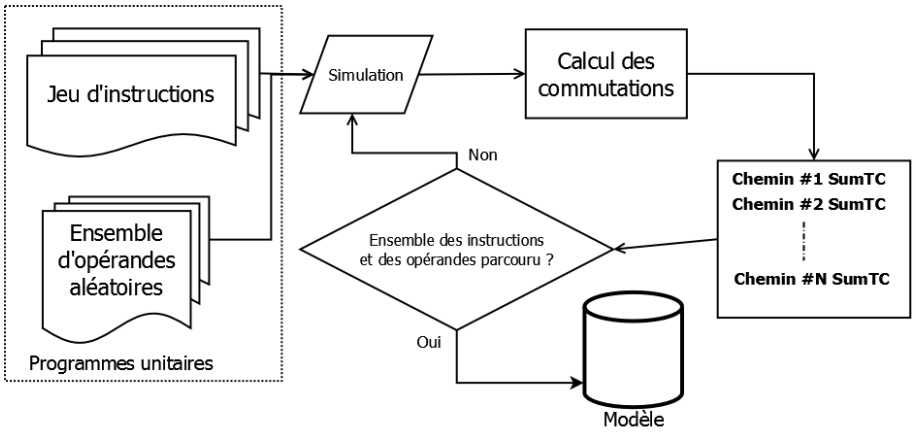


Figure 3.12 – *Méthode de caractérisation des commutations en fonction des micro-instructions, à partir du jeu d'instructions.*

a) Modèle « U »	Chemin 1	Chemin 2	...	Chemin N
ins_1	sumTC(1)	sumTC(2)	...	sumTC(N)
ins_2	sumTC(1)	sumTC(2)	...	sumTC(N)
...	...	...	...	...
ins_n	sumTC(1)	sumTC(2)	...	sumTC(N)

b) Modèle « P »	Chemin 1	Chemin 2	...	Chemin N
(ins_1, ins_2)	sumTC(1)	sumTC(2)	...	sumTC(N)
(ins_1, ins_3)	sumTC(1)	sumTC(2)	...	sumTC(N)
...	...	...	...	...
(ins_1, ins_n)	sumTC(1)	sumTC(2)	...	sumTC(N)
(ins_2, ins_3)	sumTC(1)	sumTC(2)	...	sumTC(N)
...	...	...	...	...
(ins_2, ins_n)	sumTC(1)	sumTC(2)	...	sumTC(N)
...	...	...	...	...
(ins_{n-1}, ins_n)	sumTC(1)	sumTC(2)	...	sumTC(N)

Tableau 3.1 – *Contenu des modèles d'instructions « U » et « P ». Pour chaque chemin le terme sumTC est calculé lors de la simulation au niveau porte logique.*

nel à $\frac{n^2}{2}$ exécutions et analyses, où n est le nombre total d'instructions du jeu. De même, on peut montrer que l'effort de simulation a pour complexité le nombre d'instructions considérées dans le modèle. Ainsi, avec une suite de plus de 2 instructions, le besoin en temps de simulation et en mémoire RAM augmente selon une loi puissance avec la taille du jeu d'instructions. Pour garder des temps de simulation courts dans cette première étude, le modèle est établi avec des paires d'instructions.

Dans la suite, nous détaillons l'utilisation du modèle dans le cas où un programme quelconque est exécuté sur le processeur, pour calculer les dégradations temporelles.

3.5.3 Modèle de dégradation du processeur

Nous exprimons ici la dégradation de délai du processeur, à partir des résultats de délai des portes logiques et des chemins.

D'après la formule 3.9, la dégradation de délai a pour expression $d_{aged}(j) = d_0 + C \cdot \sum_i TC(i,j)^n$. On peut écrire la formule en fonction de l'activité de commutation d'après la formule 3.1 : $d_{aged}(j) = d_0 + C \cdot \sum_i \sum_{k=1}^{N_{sim}} \alpha_{i,k}$. En remarquant que $\sum_i \sum_{k=1}^{N_{sim}} \alpha_{i,k} = \sum_{k=1}^{N_{sim}} \sum_i \alpha_{i,k}$ (inversion des opérateurs de sommation) et en reprenant la définition du terme $sumTC$ de l'équation 3.8, on obtient la formule 3.11.

$$d_{\text{aged}} = d_0 + C \cdot \sum_{k=1}^{N_{sim}} sumTC(k) \tag{3.11}$$

Ensuite, il existe deux cas selon le modèle d'instructions « U » ou « U ».

– Pour le modèle « U », on considère que $\forall\, k \in [1; N_{sim}]\,, sumTC(k) = D_1 \cdot sumTC(ins_k)$ avec ins_k la micro-instruction lue depuis la mémoire d'instructions au cycle d'exécution k ; D_1 est une constante.

  On considère que l'exécution d'un programme au niveau fonctionnel se résume à une liste de micro-instructions L issues de la mémoire d'instructions et exécutées sur l'architecture. Pour une suite L, les dégradations de délais sont estimées par la formule 3.12.

$$d_{\text{aged}} = d_0 + C_1 \cdot sumTC(L) \tag{3.12}$$

  $sumTC(L)$ est le nombre de commutation issu du premier modèle, et cumulé pour toutes les micro-instructions de la suite de micro-instructions L. Plus précisément, si la suite L vaut $[ins_1; ins_2; ...]$, on calcule $sumTC(L)$ par la formule $sumTC(L) = \sum_{k=1}^{N_{sim}} sumTC(ins_k)$ où $sumTC(ins_k)$ est repris du modèle « U » pour la micro-instruction ins_k. d_0 est le délai de propagation initial d'un chemin. Le terme C_1 est une constante reliée aux données technologiques valant $C \cdot D_1$.

– Pour le modèle « P », on définit Q comme la suite $[(ins_1, ins_2), (ins_2, ins_3), ...]$ de paires de micro-instructions, établie avec la liste L. On a $sumTC(Q) = \sum_{k=1}^{N_{sim}} sumTC(ins_k, ins_{k+1})$ où $sumTC(ins_k, ins_{k+1})$ est repris du modèle « P » pour la paire (ins_k, ins_{k+1}). On considère que $\forall\, k \in [1; N_{sim} - 1]\,, sumTC(Q) = D_2 \cdot sumTC(ins_k, ins_{k+1})$ avec respectivement ins_k et ins_{k+1} les micro-instructions lues depuis la mémoire d'instructions aux cycles d'exécution k et $k+1$; D_2 est une constante.

Dans ce cas, on peut aussi exprimer les dégradations de délais par la formule 3.13 similaire à la formule 3.12.

$$d_{\text{aged}} = d_0 + C_2 \cdot sumTC(Q) \tag{3.13}$$

d_0 est le délai de propagation initial d'un chemin. Le terme C_2 est une constante reliée aux données technologiques valant $C \cdot D_2$.

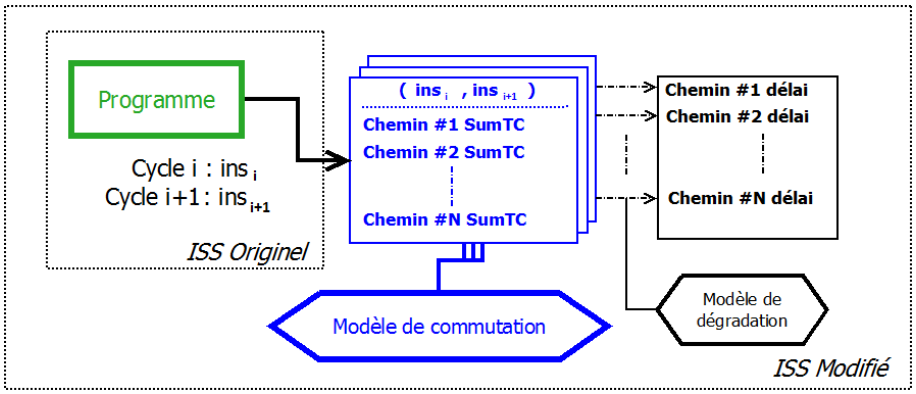


Figure 3.13 – *Différence entre un simulateur fonctionnel existant et le simulateur augmenté d'une donnée de dégradation, grâce au calcul des commutations et des dégradations de délais.*

Dans cette section, nous avons présenté les modèles de dégradation au niveau fonctionnel. La caractérisation des commutations a été présentée pour les deux modèles d'instructions « P » et « U ». Enfin, les modèles de délai ont été présentés pour permettre d'être intégrés à un simulateur au niveau fonctionnel. Nous présentons cette intégration dans la section suivante.

3.6 Simulation des dégradations au niveau fonctionnel

Dans cette section, nous présentons la simulation des dégradations du circuit au niveau fonctionnel et ses caractéristiques. Elle est basée sur l'expression de la dégradation de délai formulée à la section précédente et permet de simuler les dégradations temporelles d'un circuit. Nous présentons aussi les simulateurs ISS

Notre méthodologie est résumée figure 3.14. La première étape consiste à décrire l'architecture au niveau fonctionnel, puis la *netlist* du circuit est générée pour une technologie donnée (I). Les simulations fonctionnelle et porte logique sont lancées (II) et le modèle de commutations est repris des simulations au niveau porte logique pour le modèle « P » (III). Enfin, les marges temporelles au niveau fonctionnel sont reportées pour le circuit dégradé (IV). Ce dernier calcul est réalisé grâce à l'équation 3.13.

Les marges temporelles sont estimées au niveau fonctionnel. Nous comparons maintenant l'intégration d'une telle méthode pendant la conception d'un circuit avec son simulateur ISS.

3.6.1 Simulation des dégradations avec les ISS

Un simulateur de jeu d'instructions ou ISS joue un rôle central dans la conception d'une architecture. Il doit être rapide à exécuter, précis, paramétrable et intégrable simplement dans une

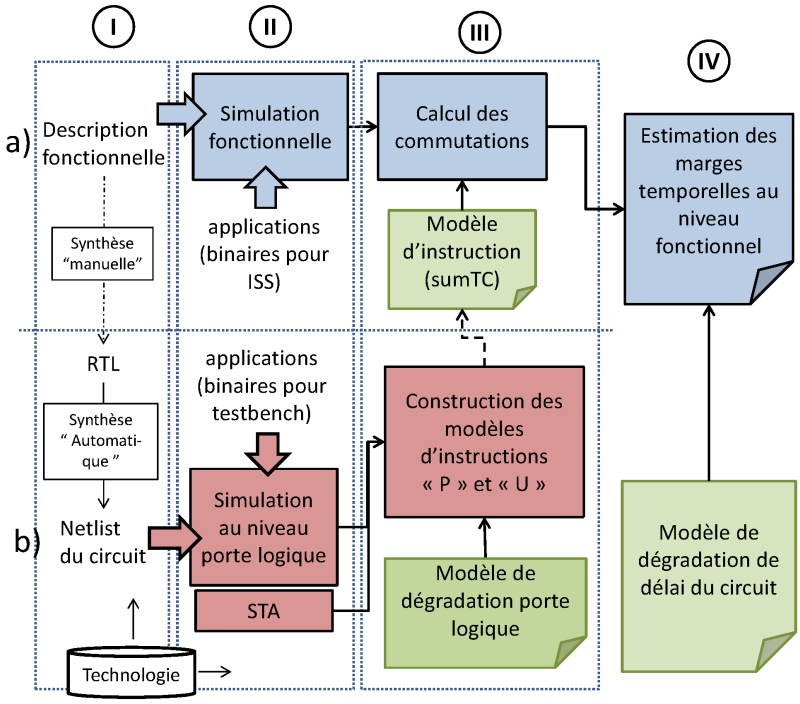


Figure 3.14 – *Détails de la méthodologie de simulation des marges temporelles. Les étapes au niveau fonctionnel sont représentées dans la partie du dessus a) et les étapes préliminaires, réalisées au niveau porte logique sont dans la partie du dessous b).*

méthode de conception des MPSoC. Le simulateur ISS émule le comportement d'un processeur en exécutant les micro-instructions dudit processeur sur une machine hôte. Il est possible d'obtenir des descriptions fonctionnelles ou précises au cycle près. D'un côté, le modèle fonctionnel élimine les détails internes de la partie matérielle, comprenant à la fois les registres et les parties combinatoires. Il est ainsi possible de générer un tel simulateur dès les premières étapes de conception d'un MPSoC, pour pouvoir débuter les étapes de développement logiciel. Durant cette phase, les temps de simulation et de construction du modèle sont déterminants pour pouvoir explorer rapidement les espaces de conception. D'un autre côté, les modèles précis au cycle près proposent un modèle entre le niveau RTL et le niveau fonctionnel. La plupart des éléments architecturaux sont représentés, notamment les registres internes. On peut ainsi déterminer plus précisément les performances d'une architecture. Par contre, les simulations sont plus longues que pour le niveau fonctionnel et le temps de définition du modèle est plus important.

Les simulateurs ISS peuvent être obtenus à partir de simulateurs existants ou bien générés automatiquement, par un langage de description d'architecture ADL. Les langages ADL mixtes représentent un bon compromis entre le modèle fonctionnel et précis au cycle près, pour l'exploration architecturale. Le langage « ArchC » [Rig+04] permet par exemple de générer un simulateur du jeu d'instructions en langage SystemC, selon une architecture du jeu d'instructions ou ISA [2]. Différents niveaux de description sont possibles grâce à la méthode TLM [3] et l'intégration avec les méthodologies de conception des MPSoC est aisée. Nous choisissons de ne pas modifier directement un ISS existant afin que notre méthode soit la plus générique possible. Tous types de simulateur peuvent être utilisé, en y intégrant les données de dégradations. Nous verrons dans le chapitre un exemple d'application de notre méthodologie avec un ISS simple.

3.7 Conclusion

Notre méthode d'estimation des dégradations comporte trois parties. D'abord, l'activité puis la dégradation sont simulées au niveau porte logique. Ensuite, le modèle de dégradation au niveau fonctionnel est construit, à partir du modèle au niveau porte logique. Enfin, la simulation fonctionnelle est lancée et donne les résultats de dégradation des chemins au niveau fonctionnel. Le simulateur fonctionnel proposé dans la section 6 permet d'estimer les marges temporelles après dégradation.

Dans le prochain chapitre, nous évaluons la précision et la performance de notre simulateur fonctionnel sur un cas d'étude. Aussi, nous comparons les deux modèles d'instructions utilisés pour le modèle de dégradation au niveau fonctionnel.

2. Instruction Set Architecture
3. Transaction Level Modeling

CHAPITRE 4

VALIDATION ET EXEMPLE D'APPLICATION DE LA SIMULATION FONCTIONNELLE DES DÉGRADATIONS

Le but de ce chapitre est de valider la méthode de simulation fonctionnelle proposée au chapitre précédent. Le modèle d'instructions a été construit seulement pour certaines valeurs des opérandes de chaque instruction. Il faut évaluer la précision du modèle proposé au niveau fonctionnel en fonction des valeurs au niveau porte logique. Pour cela, on applique dans un premier temps la méthode à un cœur de processeur embarqué, appelé AntX et développé au CEA LIST. La précision des estimations au niveau fonctionnel est évaluée pour différents benchmarks. Les benchmarks sont des programmes qui servent de référence pour évaluer des architectures de systèmes embarqués.

Parmi les résultats du chapitre, nous comparons d'abord les deux modèles d'instructions, selon que l'on considère les instructions uniques ou les paires d'instructions.

Ensuite, la performance et la précision de la simulation fonctionnelle est évaluée. Ceci montre l'intérêt de notre méthode de simulation fonctionnelle pour explorer rapidement différentes applications.

Enfin, on présente un exemple d'analyse temporelle du cœur de processeur AntX au niveau fonctionnel et au niveau porte logique.

Le chapitre débute avec la description de l'environnement logiciel et matériel utilisé pour étudier les dégradations du processeur AntX. Puis, nous donnons les résultats de précision et de performance de la simulation fonctionnelle.

Enfin, nous donnons les résultats de l'analyse temporelle du processeur AntX.

4.1 Description du cas d'étude logiciel

Notre environnement logiciel contient un ensemble de benchmarks, de compilateurs, de fichiers de configurations et de simulateurs de circuits numériques. Nous commençons par présenter l'architecture du processeur AntX. Puis, nous décrivons les benchmarks qui sont exécutés sur ce processeur. Nous terminerons par le protocole de simulation fonctionnelle. Dans la suite, nous appelons micro-instruction un mot logique contenant une instruction particulière et des opérandes particuliers, issus de la mémoire d'instructions.

4.1.1 Description de l'architecture du processeur AntX

Le processeur AntX [Bec+11] est utilisé dans les systèmes embarqués et il est compatible avec le multithreading. Son jeu d'instructions est du type RISC et il est composé d'un pipeline de 5 étages. Ci-dessous, nous présentons une description de chacun des étages du pipeline au niveau RTL. Enfin, nous donnons les résultats de sa synthèse logique.

La microarchitecture du circuit comprend les fonctions « Fetch », « Decode », « Execute », « Memory » et « Writeback » décrites ci-dessous. La figure 4.1 illustre ces étages et les barrières de registres qui les séparent. Toutes les fonctions qui participent au contrôle du parcours des instructions dans le pipeline sont regroupées sous la dénomination « Control Pipe ». La file de registres internes est décrite par le bloc « Register Files ».

- L'étage « Fetch » est codé par une machine d'état ; elle lit les micro-instructions sur 16 ou 32 bits. Le compteur programme qui définit l'adresse de lecture des micro-instructions depuis la mémoire d'instructions, est géré par cette machine d'état.
- L'unité de décodage « Decode » permet de sélectionner les ressources utiles à l'exécution de l'instruction. Par exemple, une addition nécessite d'utiliser l'Unité Arithmétique et Logique (ALU) alors qu'un branchement a seulement besoin de contrôler le compteur programme.
- Le traitement des opérandes est ensuite effectué par l'unité « Execute ». Cette unité contient un additionneur et des registres à décalages sur 32 bits. Les opérandes sont codées sur 2 mots de 16 ou 32 bits selon le type de micro-instruction. Ils sont issus soit de la micro-instruction (valeur immédiate, mot de 32 bits), soit de la file de registres (Register files, mot de 16 ou 32 bits), soit de la sortie de l'ALU (mot de 16 bits).
- La sortie de l'ALU entre dans l'étage « Memory ». Deux cas d'accès à la mémoire sont possibles. Soit une donnée doit être lue depuis la mémoire de donnée et la réception d'un mot depuis mémoire est démarrée. Soit une donnée est écrite et l'écriture dans la mémoire est lancée.
- Enfin, le bloc « Writeback » sélectionne l'entrée de la file de registres et y inscrit le résultat de l'étage « Execute » ou « Memory » selon le type d'opération.

Dans le cas d'une conception d'un circuit avec des cellules standards, il est courant de procéder à une synthèse logique automatique. Cette étape permet de représenter la fonction du circuit par des portes logiques spécifiques. Le fabricant fournit alors toutes les informations technologiques du circuit synthétisé pour ces portes logiques spécifiques. Il est donc possible d'extraire certains paramètres physiques avant même que le circuit soit fabriqué. Nous utilisons en particulier les informations de temps de propagation et les marges temporelles dans la suite de l'étude.

La description RTL du processeur a été développée dans des travaux antérieurs à cette thèse. Ici, nous avons effectué une synthèse logique du cœur de processeur sur la technologie TSMC

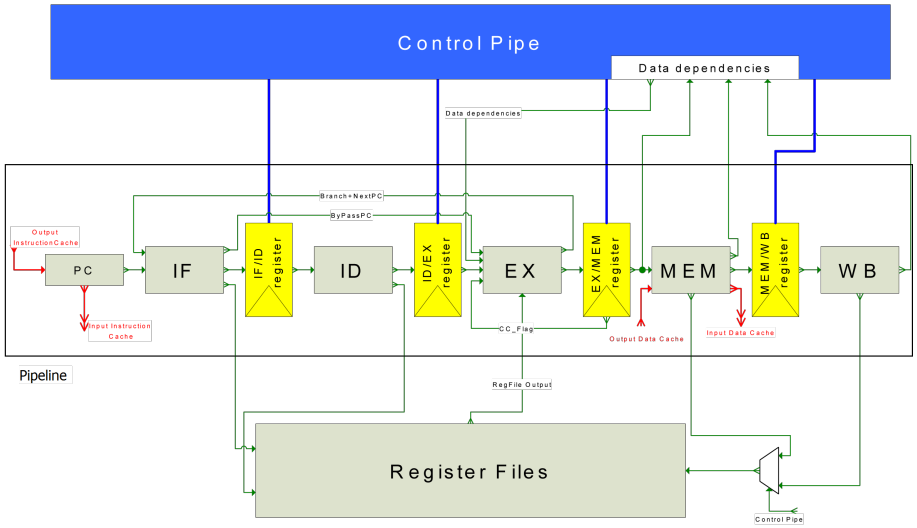


Figure 4.1 – *Microarchitecture du processeur AntX : définition du pipeline, de la file de registres internes et des registres de contrôle de l'exécution. L'interface avec la mémoire de données est réalisée par le bloc « Memory » (Input/Output Data Cache) et par le bloc « Fetch » pour la mémoire d'instructions (Input/Output Instruction Cache).*

40 nm grâce à l'outil *Design Compiler* de *Synopsys*. Chaque étage est alors traduit en cellules standards. La répartition des nœuds au niveau porte logique pour chacun des étages est reportée sur le tableau 4.1 à titre d'illustration.

Les conditions de synthèse sont une température ambiante de 25 °C et une tension d'alimentation $V_{dd} = 1,1$ V. Les fichiers de configuration écrits en langage TCL [1] comportent quatre étapes pour générer la description des portes logiques, selon la méthodologie préconisée par l'éditeur du logiciel de synthèse (Synopsys). D'abord, les fichiers sources en langage VHDL sont lus et compilés pour vérifier la syntaxe et les connexions entre registres. Enfin la synthèse logique est réalisée et le résultat est enregistré dans un fichier de sortie en langage VHDL.

La fréquence maximale de fonctionnement est fixée à 200 MHz. La surface silicium est estimée à 11 892 μm^2, ce qui représente environ 5 000 portes logiques NON-ET.

Structure	Fetch	Decode	Execute	Memory	Writeback	Reg. File & Ctrl. Pipe	Total
Nombre de nœuds	229	1373	1199	369	74	1841	5088
Proportion par rapport au cœur	5%	27%	24%	7%	1%	36%	100%

Tableau 4.1 – *Exemple de répartition des nœuds dans les étages du pipeline du processeur AntX pour une synthèse effectuée avec la librairie TSMC 40 nm (Low-power, High-k).*

Nous remarquons que les blocs « Decode », « Execute » et « Reg. file » sont les plus gros blocs

1. Tool-Kit Language

de l'architecture. La possibilité de lire des mots sur 16 ou 32 bits rend l'étage de décodage complexe. Les unités de traitement des données dans le bloc « Execute » consomment également une partie importante de surface. Enfin, la file de registres internes est grande car nous avons généré ses éléments à l'aide de cellules standards, qui ne sont pas optimisées pour des structures de mémoires. Dans la suite de l'étude, nous ne considérons pas la structure interne de la file de registres.

4.1.2 Propriétés de l'analyse temporelle du processeur

Une liste de chemins temporels est construite avec l'outil *PrimeTime* de *Synopsys*. L'analyse temporelle utilise les hypothèses suivantes :

1. Les faux chemins c.-à-d. les chemins pour lesquels il est impossible qu'une transition se propage du fait des conditions combinatoires, sont ignorés.
2. Pour chaque bascule, seul le chemin le plus long arrivant sur l'entrée est reporté et les autres chemins sont ignorés.

Pour rappel, on note chemin critique initial le chemin temporel qui a le plus long délai de propagation, sans condition de stress HCI. Le chemin critique initial a un délai de propagation de $4,54\,ns$ dans notre cas. Le nombre total de chemins est égal à *238*.

Aussi, la marge temporelle de la bascule est égale à la différence entre la période d'horloge et le délai de propagation du signal d'entrée de la bascule. Dans notre cas, les marges temporelles sont calculées pour les 238 chemins, qui sont les plus longs pour chaque bascule (d'après l'hypothèse 2) et qui ne contiennent pas de faux chemins (d'après l'hypothèse 1). On donne ainsi les temps de propagation des signaux dans les parties les plus critiques du circuit. Ces parties sont identifiées pour un état initial du circuit.

4.1.3 Outils de simulation matérielle

Le circuit décrit par les cellules standards est appelé *netlist*. La *netlist* est simulée en utilisant des outils de simulation présentés sur la figure 4.2. D'un côté, les portes logiques du cœur du processeur sont simulées. D'autre part, les mémoires sont simulées du point de vue de leur comportement. Les interfaces cœur/mémoire sont réalisées par les ports d'entrée/sortie d'instructions et de données. Le compteur programme est connecté au port de sortie d'instructions du cœur. Le port d'entrée est relié à la micro-instruction lue. Les connexions sortantes et entrantes de données sont reliées aux ports d'entrée/sortie de données du cœur.

Les outils de simulation génèrent aussi l'horloge et le signal de réinitialisation du cœur de processeur et des mémoires ainsi que l'arbitrage des bus des mémoires.

4.1.4 Outils de simulation logicielle (*benchmarks*)

Les benchmarks sont écrits en langage *C* ou *Assembleur*. La chaîne de compilation standard *GNU* est utilisée pour générer le fichier binaire, à partir du code source et après avoir fait l'édition des liens. On précise l'adresse de chargement de l'application en mémoire et éventuellement les valeurs initiales des variables d'entrée. Enfin, un utilitaire de conversion en langage machine permet d'obtenir le code binaire des micro-instructions, au format hexadécimal. Ce code est chargé en mémoire d'instructions par les outils de simulation matérielle. En fin de simulation,

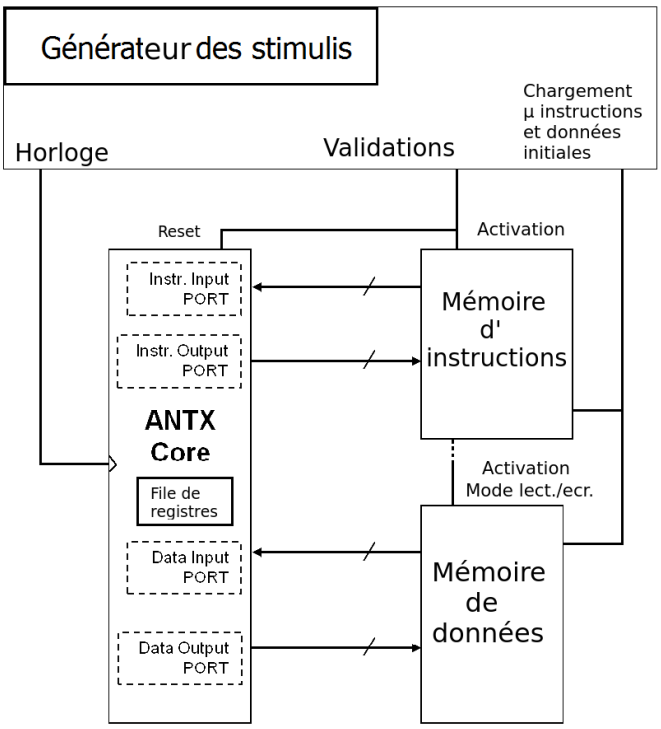


Figure 4.2 – *Outils de simulation contrôlant le fonctionnement du cœur de processeur AntX et des mémoires.*

on obtient un fichier au format VCD [2]. Ce fichier enregistre les valeurs des signaux logiques du cœur de processeur au cours de la simulation. Nous avons choisi un ensemble représentatif d'application s'exécutant sur un système embarqué, que nous appelons « benchmarks ».

Le nom et la fonction réalisée par chaque benchmark sont présentés respectivement dans la première et la deuxième colonne du tableau 4.2. Nous étudions différentes données d'entrée des benchmarks, générées aléatoirement. Le type de donnée est décrit dans la troisième colonne. Enfin, le nombre et les valeurs maximales et minimales des données ou l'ensemble des valeurs possibles est reporté dans la quatrième colonne. Les niveaux d'optimisation sont reportés dans la cinquième colonne. Les combinaisons des paramètres « fonction réalisée » et « valeurs en entrée » permettent de parcourir plusieurs cas d'exécution. Les benchmarks « fir », « fft » et « motion » réalisent des traitements de signal et d'image. Les programmes « basicmath » et « crc » exécutent des opérations mathématiques complexes. « bitcount » et « sort » effectuent des comparaisons de données. Ainsi, notre étude prend en compte différents types d'algorithmes utilisés dans les systèmes embarqués.

Benchmark	Fonction réalisée	Type d'entrée	Valeurs en entrée	Optimisations
sort	Tri de mots	10 chaînes de caractères 16 bits	10 mots du dictionnaire français	O0
bitcount	Comptage binaire	255 Nombres sur 32 bits	10 entiers non-signés entre 0 et 4 294 967 296	O0
crc	Calcul de hachage	32 Caractères Unicode	32 caractères de l'alphabet	O2
basicmath	Opérations de calcul	10 nombres décimaux	10 entiers non-signés entre 0 et 4 294 967 296	O3
motion	Estimation mouvement	2 imagettes 16x16 pixels	2 matrices de 16x16 entiers non-signés entre 0 et 4 294 967 296	O3
fft	Calcul de FFT	2 séquences num. sur 16 bits de longueur 32	64 entiers signés entre $-2\,147\,483\,648$ et $2\,147\,483\,647$	Os
fir	Filtrage discret			Os

Tableau 4.2 – *Types d'exécutions considérés pour les benchmarks avec leurs fonctionnalités, les données traitées et les niveaux d'optimisations disponibles.*

À partir du tableau 4.2, nous construisons un ensemble de données, par tirage aléatoire selon une loi uniforme entre les valeurs maximales et minimales. Dans le cas de « sort » et « bitcount »,

2. Value Change Dump

les tirages sont uniformes sur l'ensemble des mots ou lettres possibles. Les fichiers binaires sont compilés avec les niveaux d'optimisation reportés sur la cinquième colonne du tableau 4.2.

Le dernier cas d'exécution que nous considérons (non montré ici) est le « pire cas d'activité ». Il apparaît quand chaque signal de la *netlist* commute à chaque cycle d'horloge. Il correspond aux méthodes actuelles pour estimer la borne supérieure des dégradations temporelles d'un circuit.

Pour chaque benchmark, il y a une différence entre le binaire associé au niveau fonctionnel et celui associé au niveau porte logique. Le prologue et l'épilogue sont différents car ils indiquent l'adresse de départ et de retour du programme dans l'espace mémoire. Pour des raisons techniques, ces adresses sont différentes entre le niveau fonctionnel et le niveau porte logique. Cependant, ces sections sont très courtes (2 instructions) et sont exécutées une seule fois. Elles n'influent donc pas sur les résultats de dégradation car des milliers d'instructions sont exécutées. Aussi, la différence d'adresse influe peu sur la dégradation.

Dans la suite nous décrivons la procédure de simulation fonctionnelle. La plateforme de simulation est décrite ainsi que son implémentation, à partir des résultats du modèle de dégradation du chapitre 3, section 5.

4.2 Évaluation de la précision et des performances du simulateur fonctionnel

Dans cette section, le premier objectif est d'évaluer la précision de nos estimations au niveau fonctionnel. Cette évaluation prendra en compte les deux modèles d'instructions introduits au chapitre 3, section 5, paragraphe 2. Le second objectif est d'évaluer le gain en performance de la simulation fonctionnelle. Nous rappelons et discutons des deux modèles d'instructions « P » et « U » au niveau fonctionnel qui ont été générés dans le cas du processeur AntX. À chaque cycle, le modèle « P » considère les paires d'instructions et le modèle « U » considère une instruction unique. Nous présentons tout d'abord la procédure d'évaluation. Les performances des simulations sont ensuite rapportées. Enfin, la précision des estimations est donnée pour les deux modèles d'instructions.

4.2.1 Procédure d'évaluation

Le vieillissement est simulé par notre chaîne d'outils présentée dans la section précédente. Nous suivons huit étapes pour évaluer la précision de la simulation fonctionnelle pour un benchmark avec une option de compilation, un ensemble de données et pour un modèle de dégradation fonctionnel. Les quatre premières étapes opèrent au niveau fonctionnel et les quatre dernières au niveau porte logique. Un schéma est donné figure 4.3 avec les deux simulations exécutées pour calculer la précision, la robustesse et les performances.

1. Un benchmark est sélectionné avec ses paramètres et compilé par le compilateur croisé.
2. L'ISS est utilisé pour simuler le comportement du processeur avec un des deux modèles d'instructions.
3. Le temps de simulation et les instructions exécutées sont relevés.
4. La dégradation $D_{func}(j)$ de chaque chemin annoté j est calculée au niveau fonctionnel.
5. La *netlist* du processeur est simulée avec le même benchmark.

6. Le temps de simulation et les instructions exécutées sont relevés.
7. La dégradation $D_{gate}(j)$ de chaque chemin j est calculée au niveau porte logique.
8. L'erreur entre le niveau fonctionnel et le niveau porte logique est calculée.

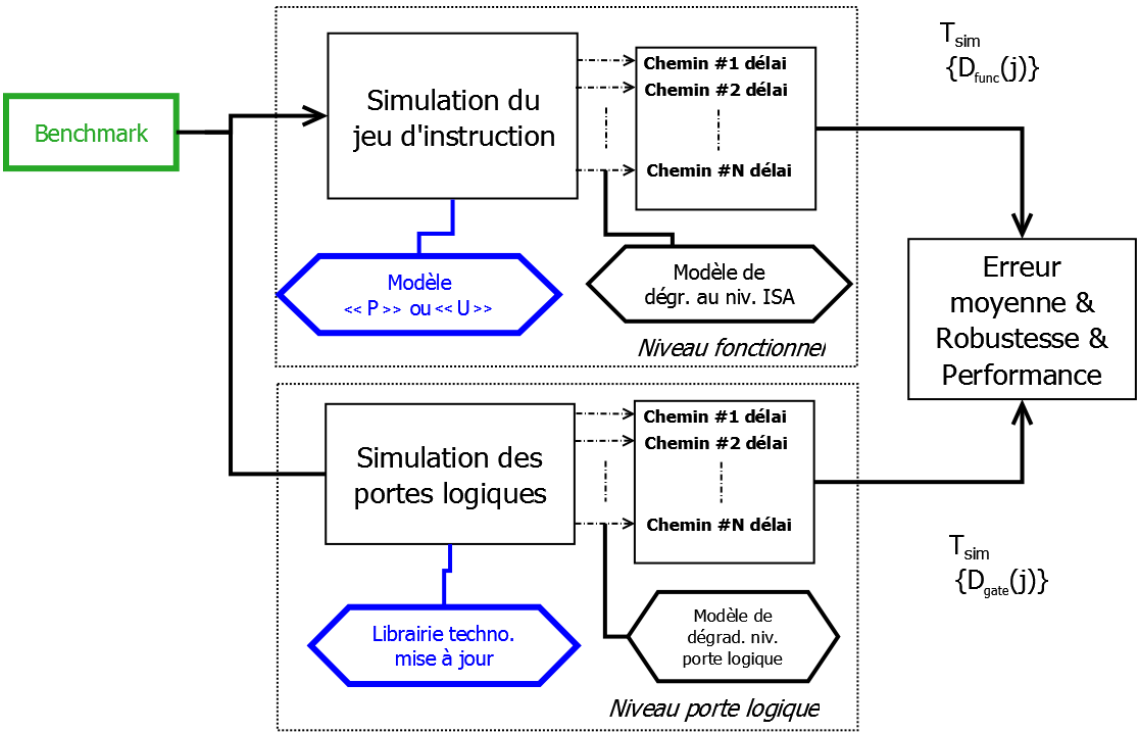


Figure 4.3 – *Procédure de comparaison entre le niveau fonctionnel et le niveau porte logique pour l'exécution d'un benchmark.*

Nous exécutons les 7 benchmarks présentés dans la section précédente en parcourant à chaque fois les huit étapes. Les deux modèles d'instructions « P » et « U » sont utilisés séparément, puis leurs résultats seront comparés entre eux. Sauf mention contraire, les valeurs en entrée des benchmarks sont gardées constantes dans ce chapitre ; nous parlons alors de données de référence. Dans le cas contraire, nous évaluons différentes exécution en changeant les valeurs en entrée.

4.2.2 Plateforme de simulation fonctionnelle

Pour calculer les dégradations au niveau fonctionnel, nous décrivons les outils utilisés dans l'étape 1 : un ISS et un modèle de dégradation. Nous présentons les résultats du profil d'exécution, nécessaires à la suite de notre étude.

Un ISS permet de générer la trace d'exécution des différents benchmarks. Cette trace comporte le nom des instructions exécutées issues de la mémoire d'instructions. On suppose que chaque instruction est exécutée en un seul cycle d'horloge. La chaîne de compilation croisée permet de générer le binaire du benchmark au niveau fonctionnel. Puis le simulateur du jeu d'instructions fournit la trace d'exécution fonctionnelle du benchmark. En résumé, la simulation fonctionnelle est couplée aux modèles de dégradations selon la méthode présentée figure 4.4.

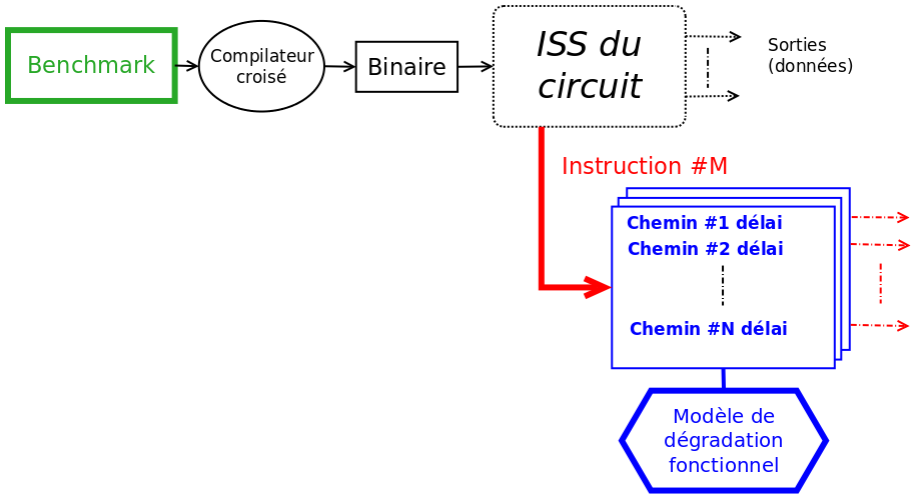


Figure 4.4 – *Méthode d'évaluation des commutations à partir des modèles d'instructions.*

À chaque instruction simulée, repérée par son nom *#M*, notre modèle fournit l'estimation des commutations pour tous les chemins numérotés de 1 à N avec $N = 238$ en accord avec le résultat de l'analyse temporelle initiale, présentée à la section 1, paragraphe 2.

Le profil d'exécution des benchmarks est donné figure 4.5. Sur ce graphe, nous reportons les instructions lues de la mémoire programme lors de l'exécution sur la plateforme de simulation fonctionnelle. Les paramètres des benchmarks sont ceux du tableau 4.2 ; ils sont exécutés entièrement. Nous classons les instructions selon leur fonctionnalité. Le groupe « arith » représente les instructions qui effectuent des calculs arithmétiques dans l'ALU du processeur. Les instructions qui lisent ou écrivent dans la mémoire de données sont identifiées par le groupe « load/store ». Celles qui gèrent le compteur programme s'appellent « branch » ; les instructions n'effectuant aucune opération sont dénommées « nop ».

D'un côté, nous calculons les dégradations au niveau porte logique avec la plateforme au niveau porte logique et le modèle de dégradation de la littérature. D'un autre côté, nous appliquons la simulation des dégradations au niveau fonctionnel présentée au chapitre 3, section 6 au processeur AntX. Tout au long de cette section, nous comparons les deux modèles d'instructions « P » et « U » au niveau fonctionnel. Le modèle de dégradation des délais utilisé ensuite est celui du chapitre 3, section 5, troisième paragraphe. La constante C de ce modèle a pour valeur $2,32.10^{-6}$ ns à la fois au niveau porte logique et au niveau fonctionnel. Elle conditionne l'incrément de délai d'une porte logique dû au vieillissement, à chaque cycle de simulation.

Dans la suite, nous commençons par présenter les performances de la simulation fonctionnelle. Ensuite, la précision est définie et évaluée pour les deux modèles d'instructions.

4.2.3 Performance de la simulation fonctionnelle

4.2.3.1 Comparaison entre les deux modèles d'instructions

On s'intéresse ici au temps de simulation des dégradations au niveau fonctionnel pour les deux modèles d'instructions. Nous reportons les temps de simulation et le nombre d'instructions exécutées pour les modèles « P » et « U » sur la figure 4.6. On représente sur l'axe des ordonnées

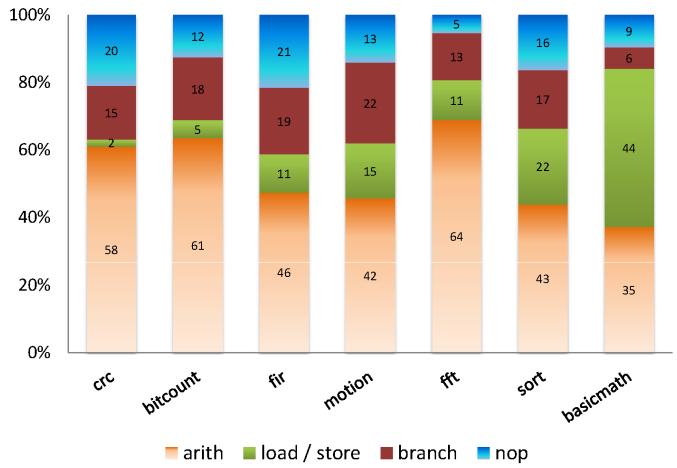


Figure 4.5 – *Profil d'instructions exécutées lors de l'exécution fonctionnelle complète des benchmarks.*

le temps de simulation en secondes. Le nombre d'instructions exécutées est représenté sur l'axe des abscisses. Nous traçons 7 points sur chaque graphe correspondant à la simulation entière d'un benchmark. Les données et les niveaux d'optimisation des benchmarks sont ceux du tableau 4.2 dans toute cette section, sauf indication contraire.

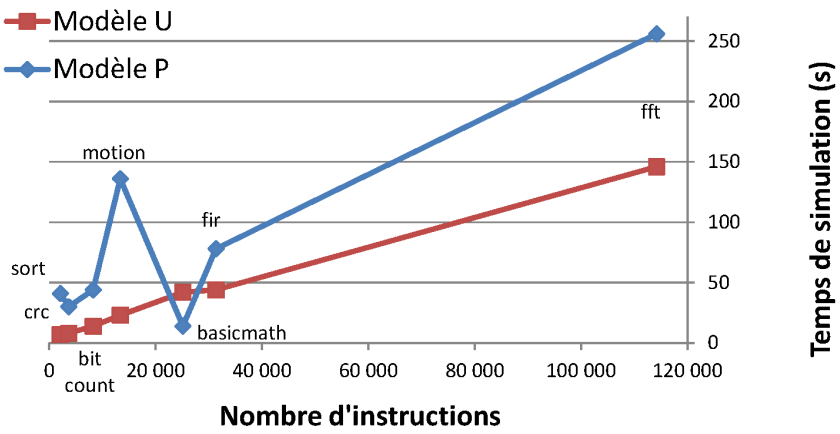


Figure 4.6 – *Temps de simulation au niveau fonctionnel avec les modèles d'instructions « P » et « U » pour les différents benchmarks.*

Nous avons différents nombres d'instructions exécutées allant de 10 000 à 100 000, ce qui

permet de comparer l'analyse des dégradations en fonction de la longueur des benchmarks. Nous remarquons que les benchmarks

Pour le modèle « P », on remarque que la courbe du temps de simulation en fonction du nombre d'instructions est croissante sauf dans deux cas. Pour le benchmark « basicmath », le temps d'exécution est plus petit que pour tous les autres benchmarks. À l'inverse, pour « motion », le temps d'exécution est plus grand que pour « fir » alors que ce dernier contient plus d'instructions simulées. Si on se réfère au types d'instructions reportées sur le tableau 4.5, « motion » contient le plus de branchement et « basicmath » a le plus grand nombre de chargement/déchargement mémoire. Or, dans le cas de la simulation fonctionnelle avec le modèle « P », les branchements ne sont pas modélisés et remplacés par des instructions *nop*, ce qui allonge le temps de simulation de « motion ». Aussi, les accès mémoire sont plus rapides à simuler car seul le comportement des mémoires est pris en compte, ce qui implique un temps court de simulation de « basicmath ».

Pour le modèle « U », on remarque que la relation entre le temps de simulation et le nombre d'instructions simulées est une fonction croissante. Cette fonction est presque linéaire.

En moyenne, le nombre d'instructions simulées par seconde s'élève à 38.10^6 Instr/s pour le modèle « P » et 49.10^6 Instr/s pour le modèle « U ». Nous avons utilisé un PC sous Linux 2.6, avec un processeur *x86* cadencé à 3 GHz et 4 Go de RAM. Le temps de simulation pour le modèle « U » est donc $1,4$ fois inférieur à celui obtenu pour le modèle « P ».

On évalue maintenant la mémoire nécessaire à la simulation. Le processeur AntX dispose d'un jeu de 60 instructions. Dans le cas du modèle « P », il faut 390 Mo en mémoire vive pour stocker toutes les paires d'instructions. Dans le cas du modèle « U », il faut seulement 14 Mo. Ces deux valeurs particulières permettent d'explorer le jeu d'instructions sans dépasser la quantité maximale de mémoire vive requise, avec les moyens de simulation habituels. Par contre, pour des jeux d'instructions plus importants comme dans le cas d'architecture CISC [3] par exemple, il faut tenir compte de cette différence. Une des raisons est que la taille du modèle « P » augmente de manière quadratique avec le nombre d'instructions. Pour le modèle « U », l'augmentation est linéaire.

4.2.3.2 Comparaison entre le niveau porte logique et le niveau fonctionnel

La simulation au niveau porte logique permet d'obtenir des résultats de dégradations plus précis, au détriment du temps de simulation. Nous comparons le temps d'analyse au niveau porte logique avec celui au niveau fonctionnel. Au niveau porte logique on simule un benchmark avec la plateforme et on réalise une analyse temporelle des dégradations à chaque cycle d'horloge. Au niveau fonctionnel, on exécute un benchmark avec le simulateur ISS et on calcule les dégradations à chaque cycle. Le temps de simulation ajouté au temps de calcul des dégradations est appelé temps d'analyse.

Sur la figure 4.7, on représente en ordonnée les temps d'analyse en secondes, en échelle logarithmique. L'abscisse montre le nombre d'instructions. Les temps d'analyse au niveau porte logique et au niveau fonctionnel pour le modèle « U » sont reportés. Les niveaux d'optimisation sont ceux du tableau 4.2 et les données sont identiques pour les deux simulations.

On remarque que la durée d'analyse est une fonction croissante du nombre d'instructions exécutées pour les deux niveaux d'abstraction. Seul le benchmark « crc » au niveau porte logique fait

3. Complex Instruction-Set Computer

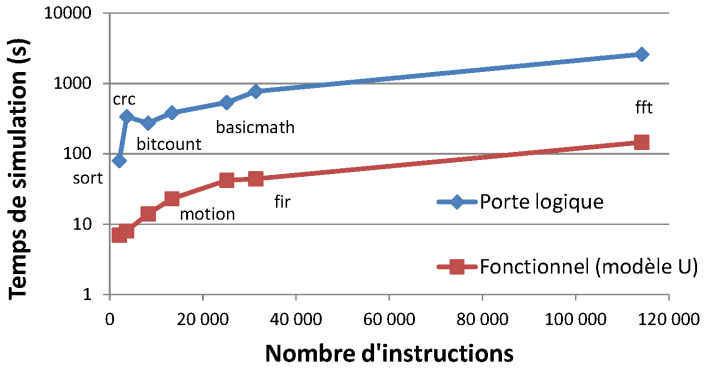


Figure 4.7 – *Temps d'analyse et de simulation des dégradations en fonction du nombre d'instructions exécutées pour les niveaux porte logique et fonctionnel.*

exception à la règle. On observe que l'analyse au niveau fonctionnel est en moyenne 10 fois plus rapide qu'au niveau porte logique.

En résumé, nous venons de montrer les performances de la simulation fonctionnelle pour les deux cas de modèles d'instructions « P » et « U ». Le nombre d'instructions simulées par seconde est de l'ordre de 10^7. Le gain de temps par rapport au niveau porte logique est d'un facteur 10. Bien que le modèle « U » soit plus avantageux en termes de temps de simulation et de mémoire, nous allons voir dans la suite les avantages du modèle « P » en termes de précision.

4.2.4 Précision

Pour calculer la précision des estimations des dégradations au niveau fonctionnel, nous définissons une fonction d'erreur pour un chemin j (équation 4.1). Le terme $ST_{func}(j)$ est la marge temporelle du chemin j au niveau fonctionnel et $ST_{gate}(j)$ celle au niveau porte logique.

$$\text{Erreur(j)} = \frac{ST_{func}(j) - ST_{gate}(j)}{ST_{gate}(j)} \tag{4.1}$$

On définit la dégradation relative par la formule 1.15 avec ST la marge temporelle, que l'on décline au niveau porte logique et au niveau fonctionnel.

$$D(j) = \frac{ST_{\text{aged}} - ST_{\text{fresh}}}{ST_{\text{fresh}}} \tag{4.2}$$

On calcule également l'activité pour évaluer la précision entre le niveau porte logique et le niveau fonctionnel. Pour rappel, l'activité de commutation d'un circuit est définie par les commutations de l'état logique de tous ses nœuds internes. Nous utilisons aussi l'activité α_c d'un circuit, définie comme le rapport entre le nombre moyen de commutations du circuit $\overline{TC_c}$ durant le temps simulé T_{sim}. Cette grandeur formulée par l'équation 4.3 sert à comparer les valeurs calculées dans chacun des niveaux d'abstraction.

$$\alpha_c = \frac{\overline{TC_c}}{T_{sim}} \tag{4.3}$$

4.2.4.1 Effet des benchmarks

Dans ce paragraphe, nous calculons les valeurs d'activité, de dégradation relative et d'erreur telles que définies au paragraphe précédent. Nous comparons les données au niveau fonctionnel et celles au niveau porte logique comme expliqué figure 4.3 dans le but de trouver une relation entre elles.

Dans un premier temps, les valeurs d'activité sont reportées sur la figure 4.8. Les données au niveau fonctionnel sont reportées sur l'axe des abscisses. L'axe des ordonnées correspond aux valeurs relevées au niveau porte logique. On exécute différents benchmarks en entier afin de générer différents profils d'activité sur le circuit. Au niveau fonctionnel, on utilise le modèle « U ». Chaque point de donnée est tracé pour les 7 benchmarks et pour 10 valeurs d'entrée différentes, soit un total de 70 points. Les valeurs d'entrée sont générées par 10 tirages aléatoires selon une loi uniforme, ayant pour intervalle les valeurs du tableau 4.2.

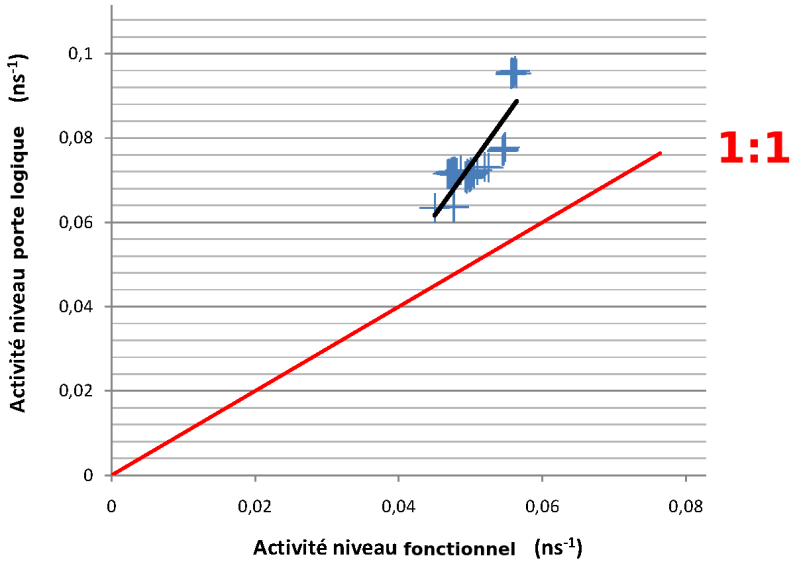


Figure 4.8 – *Relation entre l'activité niveau porte logique et celle au niveau fonctionnel du processeur AntX pour différents benchmarks et différentes valeurs en entrée.*

Les estimations au niveau fonctionnel sont toujours inférieures aux valeurs du niveau porte logique, quel que soit le benchmark exécuté. On remarque aussi que l'activité au niveau fonctionnel est moins dispersée qu'au niveau porte logique.

Dans un deuxième temps, nous comparons les valeurs de dégradation relative. Sur le graphique 4.9, l'axe des ordonnées reporte les valeurs en % de la dégradation relative au niveau fonctionnel et l'axe des abscisses représente les valeurs en % au niveau porte logique. Chaque chemin, analysé séparément, est représenté par un point de donnée sur le graphique. Les 7 benchmarks sont exécutés en entier et leurs résultats sont regroupés par couleur ou marqueur de donnée. Les niveaux d'optimisation sont ceux du tableau 4.2 et les données celles de référence.

On remarque que toutes les valeurs de différences sont positives, même lorsque la dégradation

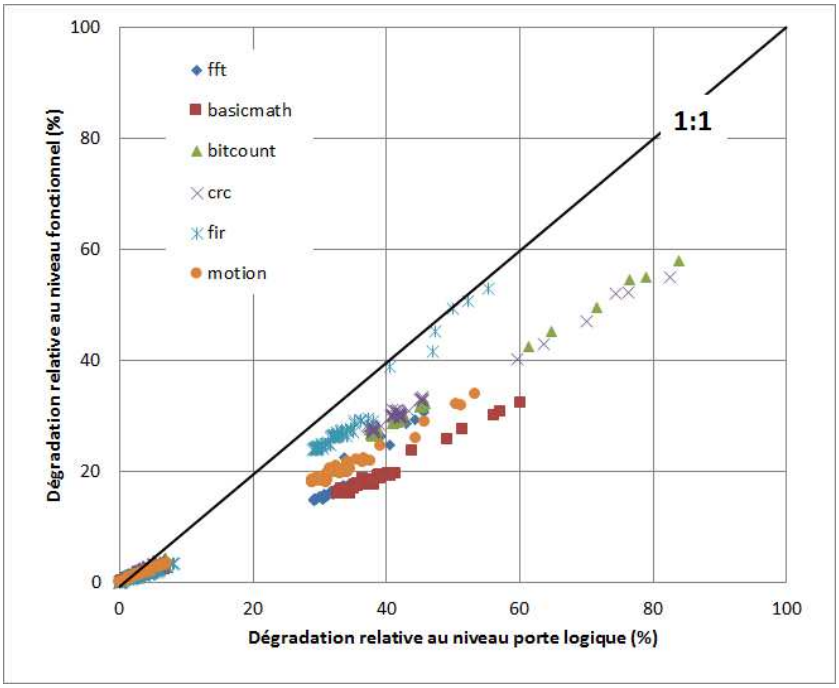


Figure 4.9 – *Relation entre les dégradations relatives estimées au niveau fonctionnel et au niveau porte logique. Les différents benchmarks sont simulés avec le modèle « P » et tous les chemins sont représentés.*

relative est inférieure à 10 % comme on le voit sur la figure 4.10. La dégradation relative au niveau fonctionnel est moitié moindre par rapport à celle au niveau porte logique. Les points de données sont alignés pour un benchmark donné et séparés en 2 groupes, selon que $D_{gate} < 10\%$ ou que $D_{gate} > 20\%$. Ces 2 groupes représentent deux types de chemins du processeur. Pour le premier groupe, représenté sur la figure 4.10, on remarque que la droite reliant les points pour un benchmark donné est d'autant plus proche de la droite 1 :1 que la durée d'exécution est plus grande (en se référant aux nombre d'instructions au niveau porte logique, présentés sur la figure 4.7). Pour le deuxième groupe, on note deux observations. D'une part le classement valide pour le premier groupe n'est plus respecté. D'autre part, la dispersion des résultats sont est plus importante, ce qui est une conséquence de l'hypothèse du modèle de dégradation au niveau fonctionnel, présentant une limitation pour de grandes valeurs de dégradation. On conclut que pour des valeurs faibles de dégradation, le modèle au niveau fonctionnel respecte les résultats au niveau porte logique. Pour des valeurs importantes de dégradation, les résultats sont dispersés et il faut étudier les chemins plus précisément.

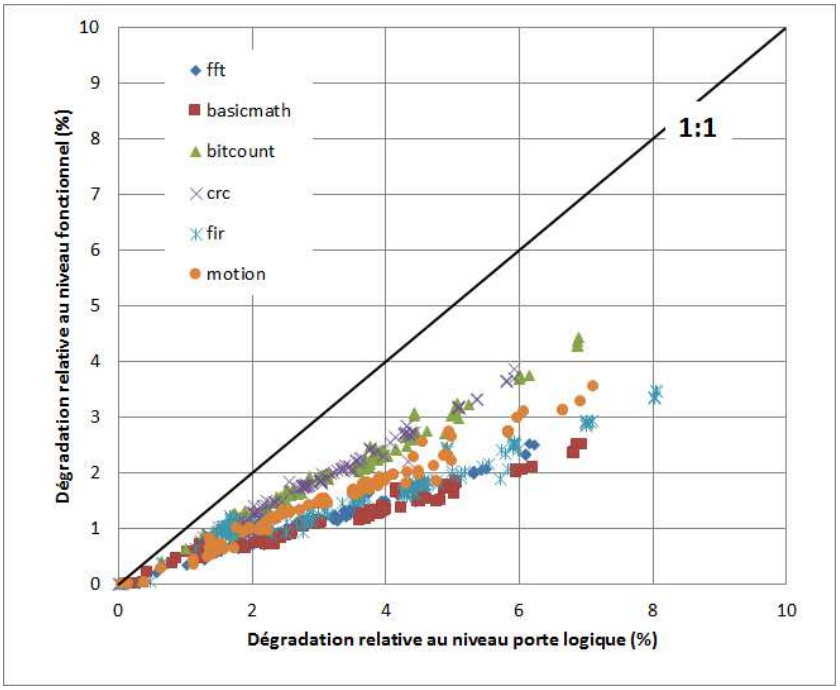


Figure 4.10 – *Zoom de la figure 4.9 pour les chemins dont la dégradation relative au niveau fonctionnel est inférieure à* 10 %.

Troisièmement, nous détaillons les résultats d'erreur par chemin. Sur le graphique figure 4.11, on reporte l'erreur entre les estimations au niveau fonctionnel et celles au niveau porte logique pour chaque benchmark, ainsi que pour le pire cas. L'axe des ordonnées reporte l'erreur en % et l'axe des abscisses représente les 238 chemins de l'analyse temporelle initiale. Les chemins les plus longs ont les indices les plus petits. Chaque exécution entière des 6 benchmarks est représentée par une courbe de même couleur (le benchmark « sort » n'est pas étudié ici). Nous utilisons le modèle « P »,les niveaux optimisations sont ceux du tableau 4.2 et les données celles de référence.

Nous observons différents groupes de chemins avec des erreurs similaires. Le groupe G1

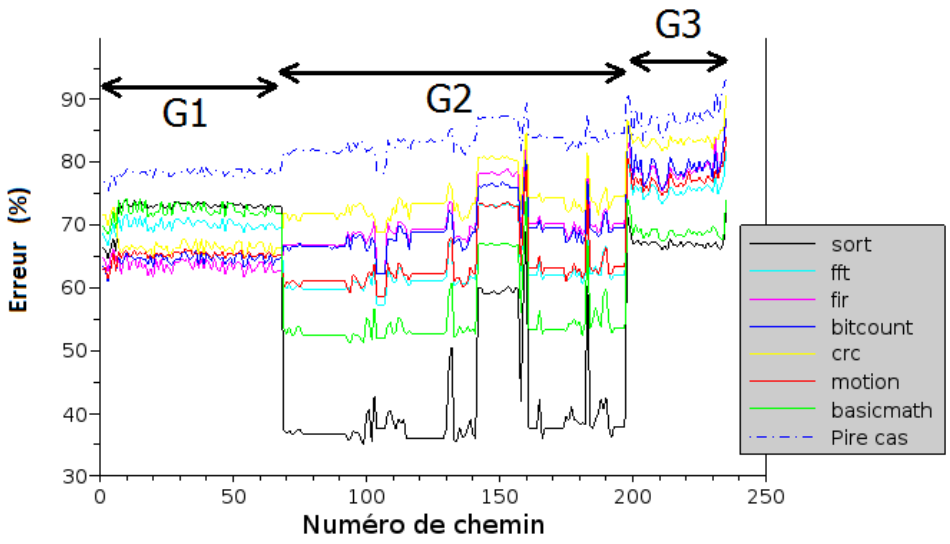


Figure 4.11 – *Erreur entre les estimations des dégradations au niveau fonctionnel et au niveau porte logique. Le modèle « P » est utilisé ici pour différents benchmarks ; on donne aussi le pire cas d'exécution.*

contient les chemins les plus longs qui ont une erreur d'environ 70%. Le groupe G2 contient les chemins de longueurs intermédiaires dont l'erreur est plus dispersée entre 40% et 80%. Enfin le groupe G3 contient les chemins les plus courts pour lesquels l'erreur varie 60% et 80%. Pour le groupe G1, nous remarquons que le benchmark « sort » est très proche du pire cas. Pour les groupes G2 et G3, c'est le benchmark « crc » qui est le plus proche du pire cas. Pour le modèle « U », nous réitérons les mêmes simulations et obtenons le graphique figure 4.12.

En conclusion, notre méthode permet de connaître plus précisément la dégradation que dans le pire cas. Cette amélioration dépend des chemins considérés et des benchmarks est la moins favorable pour « sort » et « crc ».

4.2.5 Conclusion sur la performance et la précision des modèles

Les performances de notre méthode de simulation fonctionnelle ont été évaluées pour un cas d'étude. En particulier les vitesses de simulation des deux modèles ont été évaluées à une dizaine de millions d'instructions par seconde. **Le temps d'analyse des dégradations au niveau fonctionnel est** 10 **fois plus rapide qu'au niveau porte logique.**

Le niveau fonctionnel, représenté par les modèles « P » et « U », introduit une erreur par rapport au niveau porte logique. Nous avons calculé la dégradation relative au niveau porte logique et au niveau fonctionnel et calculé l'erreur, pour différents cas d'exécution. Pour les chemins dont la dégradation relative est faible, l'erreur est constante, ce qui permet d'établir une proportionnalité entre le niveau porte logique et le niveau fonctionnel. Les différences observées proviennent des points suivants.

1. Les modèles d'instructions contiennent des valeurs pour des cas d'exécution particuliers ; ces valeurs sont obtenues avec les programmes unitaires et la méthode de caractérisation

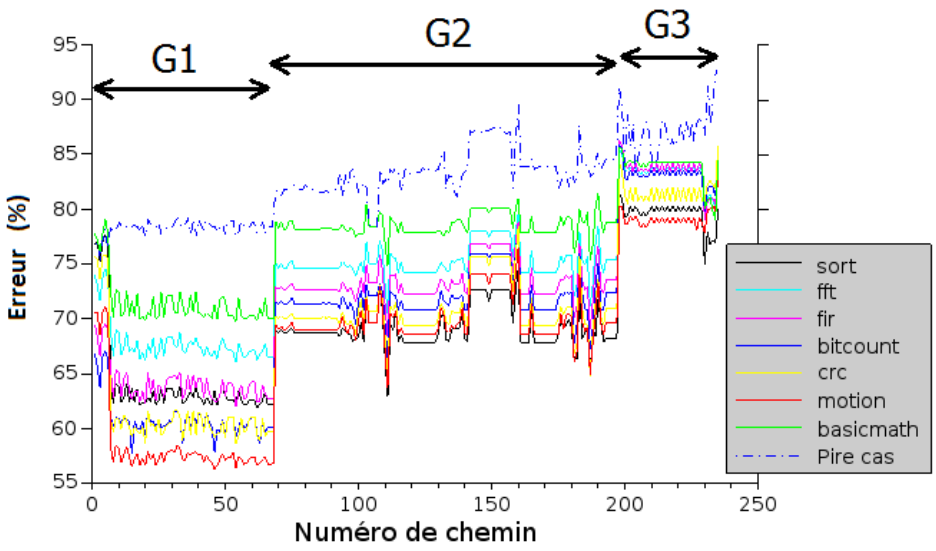


Figure 4.12 – *Erreur entre les estimations des dégradations au niveau fonctionnel et au niveau porte logique. Le modèle « U » est utilisé ici pour chaque benchmark ; on donne aussi le pire cas d'exécution.*

décrits au chapitre 3, section 5, paragraphe 2. D'autre part, les valeurs au niveau porte logique sont calculées à chaque cycle pendant l'exécution effective du benchmark. L'état du pipeline n'est pas identique dans les deux cas. En particulier, les micro-instructions exécutées avant et après les instructions du modèle sont différentes de celles exécutées par des benchmarks quelconques.

2. À chaque cycle d'exécution, la différence entre le niveau porte logique et le niveau fonctionnel se cumule dans le temps. Les simulations des benchmarks comportent des dizaines de milliers de cycles, c'est pourquoi l'erreur est importante en fin de simulation.
3. Le modèle de dégradation décrit à la section 3, paragraphe 2 utilise une majoration de la dégradation de délai, grâce à la grandeur $sumTC(j)$, représentant les commutations du chemin j. Une évaluation plus précise consisterait à tenir compte de tous les signaux internes du circuit pour calculer la dégradation de délai au niveau fonctionnel.

Maintenant que nous avons présenté les caractéristiques de notre simulation fonctionnelle des dégradations, nous l'appliquons pour estimer les dégradations temporelles du processeur AntX, dues au HCI. Nous utilisons la plateforme de simulation fonctionnelle du HCI décrite à la section 2, paragraphe 2. Il reste à définir quel modèle de dégradation du processeur nous allons utiliser entre le modèle « P » et le modèle « U ».

Le modèle « U » est construit pour chaque micro-instruction une à une, et représente le meilleur compromis pour évaluer les dégradations de processeurs complexes, contenant un grand nombre d'instructions. Le modèle « P » est établi pour les paires et peut être appliqué à des processeurs suffisamment simples. Toutefois, comme la dégradation due au HCI dépend des changements d'état des sigaux logiques, le modèle « P » est le plus adapté pour étudier les effets du HCI. C'est pourquoi il est utilisé ici à titre d'étude, dans le cas du processeur AntX.

Nous utilisons uniquement le modèle « P » dans la suite du document.

4.3 Analyse temporelle du processeur AntX sous l'effet du HCI

L'analyse temporelle consiste à estimer la dégradation des temps de propagation des signaux dans la logique combinatoire et à identifier les chemins les plus sensibles au HCI. Nous présentons la méthode d'analyse et les résultats pour le processeur AntX, au niveau fonctionnel et au niveau porte logique.

4.3.1 Méthode d'analyse

L'analyse temporelle du processeur AntX est menée en sept étapes comme décrites ci-dessous.

1. La description RTL est effectuée pour une configuration choisie de la microarchitecture.
2. La synthèse logique est lancée pour obtenir la *netlist* du circuit.
3. L'analyse temporelle initiale est effectuée sans dégradation et le modèle fonctionnel de dégradation est généré.
4. Le simulateur fonctionnel est annoté avec le précédent modèle.
5. Les applications sont compilées.
6. Chaque application est simulée au niveau fonctionnel.
7. Une analyse temporelle au niveau porte logique permet d'obtenir des valeurs précises des temps de propagation.

Nous décrivons le détail des étapes ci-après.

1. La première étape nécessite de définir les connexions matérielles du processeur. Le jeu d'instructions est de type réduit et définit les unités de lecture, décodage et d'exécution dans le pipeline du processeur. La file de registres contient 16 éléments de 32 bits chacun et conditionne le bus de connexion mémoire. Il n'y a pas de contrôleur d'interruption ni de coprocesseur.
2. L'étape 2 consiste à paramétrer le logiciel *Design Compiler* (*Synopsys*) et à générer la *netlist* du processeur et les informations technologiques. Nous avons repris la méthode standard proposée par Synopsys pour exécuter la synthèse. Elle comprend 5 étapes décrites ci-dessous.
   - L'importation des fichiers VHDL et la génération du *design* qui représente le processeur.
   - La mise à plat de la hiérarchie du design et la configuration des signaux d'entrée sortie et de l'horloge.
   - L'importation de la librairie technologique TSMC 40 nm.
   - La compilation du design avec l'outil Ultra [Syn] et les options « -timing_high_effort_script » et « -no_boundary_optimization ».
   - La génération du fichier contenant la *netlist* au format VHDL et des paramètres technologiques du design au format SDC [4].

4. Synopsys Design Constraints

3. L'étape 3 consiste d'une part à calculer les délais de propagation initiaux, sans condition de stress. Les résultats des marges temporelles sont donnés figure 4.13. Sur ce graphe, on représente la distribution des marges temporelles initiales du processeur AntX par un histogramme, pour les 238 chemins. L'axe des abscisses est gradué par les valeurs de marges temporelles en *ns* ; il est divisé en cent intervalles de longueur égale. L'axe des ordonnées représente le nombre de chemins qui appartiennent à un intervalle. Les hypothèses pour le calcul des chemins sont présentées dans la section 1, paragraphe 2.

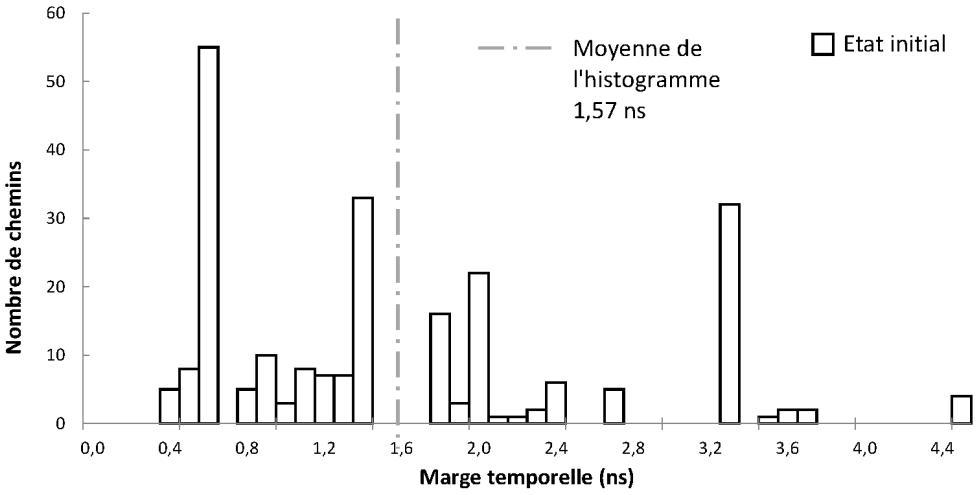


Figure 4.13 – *Distribution des marges temporelles initiales pour le processeur AntX pour l'ensemble des chemins considérés. La valeur moyenne parmi les chemins est représentée en pointillés.*

Sur ce graphique, on remarque que toutes les marges temporelles sont positives. Ceci confirme le respect des contraintes temporelles comme présenté au chapitre 1, section 5, paragraphe 2. En d'autres termes, le circuit est fonctionnel pour la fréquence d'horloge choisie à 200 MHz. La moyenne des marges temporelles vaut $1,57$ ns. D'autre part, le modèle de dégradation fonctionnel est généré comme décrit au chapitre 3, figure 3.14.

4. Les étapes 4 et 5 sont celles présentées figure 4.4.

4.3.2 Exploration des applications au niveau fonctionnel (étape 6)

L'analyse temporelle au niveau fonctionnel est utilisée pour l'exploration des applications sélectionnées. Pour rappel, le modèle d'instructions est un tableau à double entrée reliant instructions et commutations. Les 7 applications et le scénario du pire cas sont exécutés sur AntX avec notre ISS. Pour chaque application, on calcule en fin de simulation la dégradation des chemins, avec le modèle d'instructions.

Nous calculons la marge temporelle moyenne, notée $\overline{ST}$ égale à la moyenne des marges temporelles des chemins annotés. Aussi, nous reportons l'écart type, noté $\overline{Stdev}$ calculé par la formule $\frac{\sigma_{ST}}{\overline{ST}}$. Il représente la dispersion des valeurs de marges temporelles sur les 238 chemins. Enfin,

l'analyse permet aussi d'identifier le chemin le plus sensible au HCI « a priori ». Ce chemin est le chemin le plus long après dégradation.

Les résultats d'analyse issus de l'étape 7 sont donnés sur le tableau 4.3. La première colonne représente les benchmarks. La deuxième colonne représente la marge temporelle moyenne. La troisième colonne indique la marge temporelle minimale obtenue après l'exécution du benchmark. La quatrième colonne montre la marge temporelle maximale et la cinquième l'écart type. Chaque benchmark est exécuté entièrement sur la plateforme de simulation fonctionnelle. Les niveaux d'optimisation et les données sont ceux du tableau 4.2.

benchmark	$\overline{ST}$	STmin	STmax	$\overline{Stdev}$
basicmath	0,820	-1,04	4,34	1,30
bitcount	1,346	-0,25	4,68	1,17
crc	1,483	0,09	4,69	1,09
fir	0,784	-1,68	4,59	1,52
motion	1,210	-0,53	4,63	1,23
sort	1,498	0,17	4,65	1,06
pire cas	-1,179	-6,50	4,69	3,17

Tableau 4.3 – *Marge temporelle moyenne, minimale, maximale et écart type après simulation fonctionnelle des dégradations (étape 7) pour différents benchmarks et dans le pire cas.*

Sur le tableau 4.3, on remarque que le benchmark « sort » est le benchmark le moins sévère car il a la plus petite marge temporelle moyenne. À l'opposé, « fir » a le plus fort impact sur les marges temporelles moyennes. D'autre part, « fir » conduit également à la plus petite marge temporelle minimale $STmin$ et au plus grand écart-type $\overline{Stdev}$. Par contre, « basicmath » conduit à la plus petite marge temporelle maximale ST_{max}. On conclut que « fir » est le benchmark le plus impacté par le HCI.

D'autre part, les benchmarks « sort » et « crc » ont toutes leurs marges temporelles positives, ce qui signifie que les conditions temporelles sont respectées. À l'inverse, les benchmarks « fir », « motion », « bitcount » et « basicmath » ont des chemins avec des marges temporelles négatives. Dans le scénario pire cas, la dispersion des marges temporelles est très grande.

Enfin, nous identifions les chemins les plus sensibles du processeur AntX. Sur le graphique 4.14, on représente en ordonnée la marge temporelle en *ns*. L'axe des abscisses est indexé par le numéro de chemin dans l'ordre croissant des marges temporelles initiales. Pour chaque benchmark, on représente la marge temporelle des 70 chemins initiaux et en fin de simulation fonctionnelle.

On remarque que les chemins les plus sensibles sont « a priori » les chemins indexés de 1 à 7 car ils ont les marges temporelles finales les plus petites, dans les 7 cas d'application.

Dans la suite, nous allons analyser les dégradations temporelles au niveau porte logique. Nous avons analysé ici des benchmarks avec des temps d'exécution différents, ce qui affecte les résultats de dégradation en fin de simulation (figure 4.6). Nous allons analyser les dégradations avec

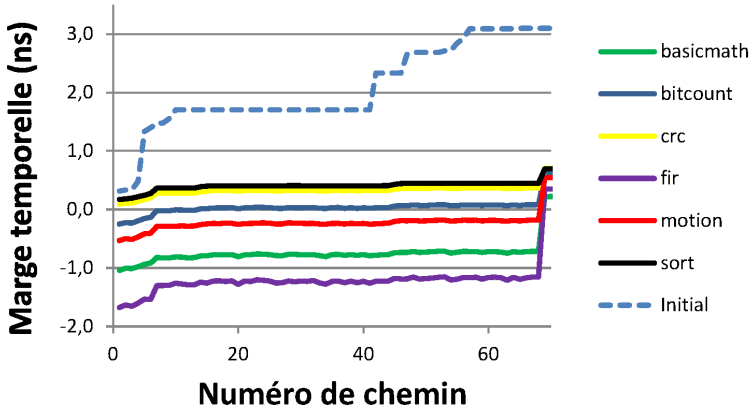


Figure 4.14 – *Résultats des marges temporelles par simulation fonctionnelle pour différentes exécutions.*

des temps simulés identiques pour éliminer cette disparité.

4.3.3 Analyse temporelle au niveau porte logique

La simulation au niveau porte logique permet de calculer de manière plus précise la dégradation des marges temporelles en fonction des benchmarks exécutés. On pourra se référer au chapitre 3, section 4 pour plus de détails. Dans ce paragraphe, nous calculons les dégradations au niveau porte logique, avec des temps simulés identiques pour chaque benchmark. On montre l'identification du chemin sensible, des instructions agressives et de la fréquence maximale de fonctionnement. Le but est de donner les résultats de dégradation pour le processeur AntX, indépendamment des applications.

Au niveau porte logique, les délais des chemins sont calculés à chaque cycle d'horloge. La librairie technologique des cellules logiques est également mise à jour à chaque cycle d'horloge avec les temps de propagation après dégradation. Les marges temporelles initiales et après dégradation sont représentés sur le graphe XY de la figure 4.15. Sur ce graphique, on reporte les marges temporelles initiales et après simulation pour les 238 chemins. L'axe des ordonnées représente les valeurs de marges temporelles initiales en *ns* et l'axe des abscisses les valeurs après avoir simulé le benchmark « bitcount » pendant 17 133 ns (temps simulé). Ici, nous analysons les benchmarks pendant une durée de 17 133 ns qui est la même pour tous les benchmarks. Cette durée est la durée d'exécution du benchmark le plus court. Nous obtenons alors un nombre d'instructions exécutées quasiment identique pour tous les benchmarks. Les niveaux d'optimisation et les données sont ceux du tableau 4.2.

On remarque que la valeur moyenne des marges temporelles après simulation est de $-0,28$ ns

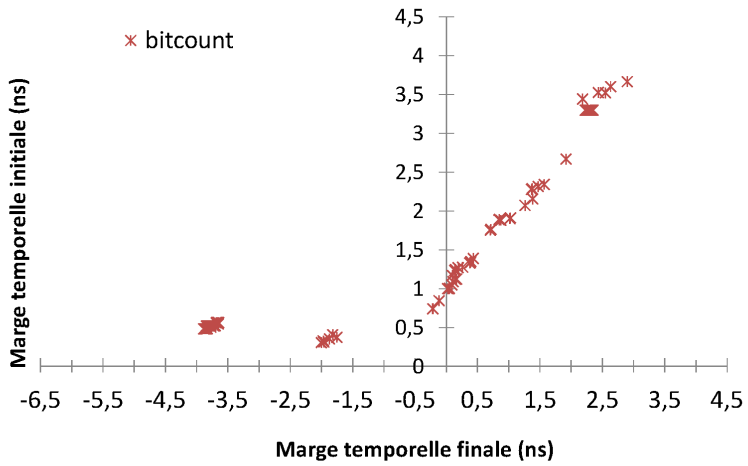


Figure 4.15 – *Simulation au niveau porte logique pour le benchmark « bitcount ». Les marges temporelles après un temps simulé de* 17 133 *ns sont représentées en fonction des marges initiales.*

soit une différence de $-117\,\%$ par rapport à la valeur moyenne initiale ($1,57$ ns). La valeur négative indique qu'il y a violation des contraintes de temps du circuit.

Dans un second temps, les résultats de dégradation pour le benchmark « motion » sont donnés figure 4.16 pour les 238 chemins et les mêmes conditions de simulation.

Dans le cas de « motion », la valeur moyenne des marges temporelles après simulation est de $0,08$ ns soit une différence de $-90\,\%$ par rapport à la valeur moyenne initiale.

Pour les benchmarks « motion » et « bitcount », on compte 83 violations du temps de pré chargement des bascules (le délai de ces chemins, ajouté au temps de pré chargement dépasse la période d'horloge). La marge temporelle de ces 83 chemins est négative.

Dans le pire cas, on obtient les résultats figure 4.17, toujours dans les mêmes conditions de simulation.

Il y a 157 violations du temps de pré chargement. On remarque que la valeur moyenne des marges temporelles après simulation est de $-1,1$ ns soit une différence de $-170\,\%$ par rapport à la valeur moyenne initiale. La valeur négative indique que les contraintes temporelles ne sont pas respectées.

Après l'analyse de toutes les applications, nous avons identifié « basicmath » comme le plus sévère pour la dégradation car il conduit à la plus petite valeur de marge temporelle moyenne.

Nous étudions maintenant les valeurs moyennes, maximales et minimales et les écarts types des marges temporelles.

Sur le tableau 4.4, on remarque que le benchmark « motion » est le moins agressif. À l'opposé, « basicmath » a le plus fort impact sur la valeur moyenne. D'autre part, « bitcount » conduit à la plus petite marge temporelle minimale ST_{min} ; « basicmath » conduit à la plus petite marge temporelle maximale ST_{max} ; et « bitcount » conduit au plus grand écart-type $\overline{Stdev}$. On conclut

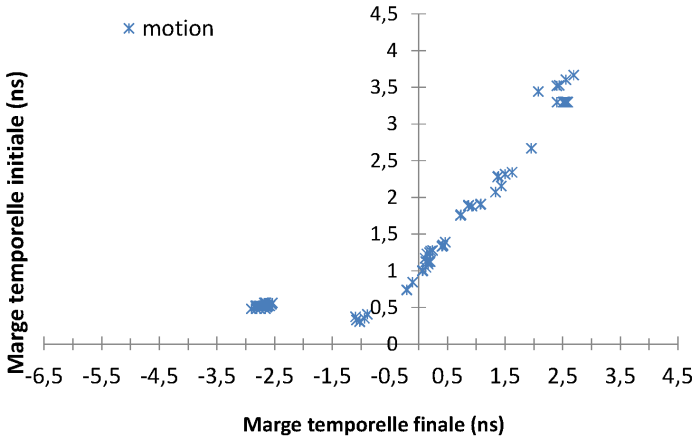


Figure 4.16 – *Simulation au niveau porte logique pour le benchmark « motion ». Les marges temporelles après un temps simulé de* 17 133 *ns sont représentées en fonction des marges initiales.*

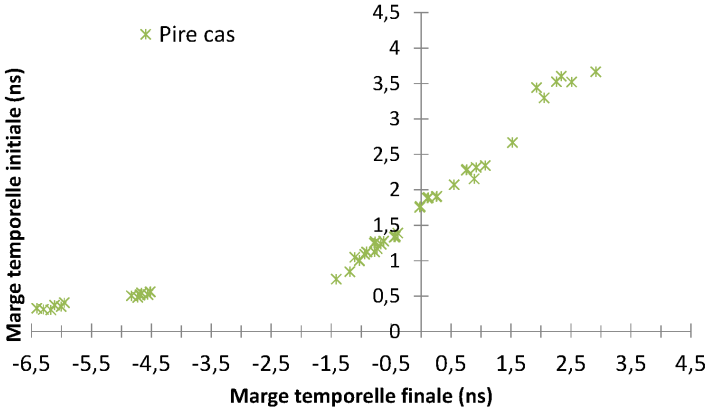


Figure 4.17 – *Simulation au niveau porte logique pour le scénario du pire cas d'exécution. Les marges temporelles après un temps simulé de* 17 133 *ns sont représentées en fonction des marges initiales.*

benchmark	$\overline{ST}$	STmin	STmax	$\overline{Stdev}$
basicmath	-0,321	-3,47	4,59	2,08
bitcount	-0,279	-3,89	4,66	2,28
crc	-0,209	-3,78	4,68	2,25
fft	-0,098	-3,62	4,67	2,22
fir	-0,150	-3,11	4,67	1,91
motion	0,081	-2,90	4,68	1,88
sort	-0,181	-3,75	4,62	2,21
pire cas	-1,104	-6,41	4,43	2,57

Tableau 4.4 – *Marge temporelle moyenne, minimale, maximale et écart type après simulation des dégradations au niveau porte logique, pour différents benchmarks et dans le pire cas.*

que « bitcount » est le benchmark qui produit la dégradation moyenne la plus élevée.

4.3.3.1 Identification des chemins sensibles

Sur la figure 4.18 les marges temporelles du processeur sont représentées pour les 70 premiers chemins, dans l'état initial et après l'exécution de chaque benchmark, ainsi que dans le pire cas. Les chemins sont indexés de A à H, selon le classement initial par marge temporelle décroissante. On utilise la plateforme de simulation au niveau porte logique et on restreint le temps simulé à 17 133 ns. Les niveaux d'optimisation et les données sont ceux du tableau 4.2.

Parmi l'ensemble d'applications, nous observons un groupe bien distinct de chemin dont les numéros vont de 7 à 70 dont les délais se dégradent le plus en fin de simulation. Ils sont situés dans l'étage « Execute » du pipeline du processeur AntX. Les groupes A à F contiennent chacun un seul chemin situé dans la partie contrôlant les opérations de l'ALU (débordement ou retenue par exemple). Les chemins les plus longs en fin de simulation sont regroupés sous les noms G et H. Ces deux groupes sont constitués d'un ensemble de chemins traversant l'ALU et en particulier ceux participant à l'additionneur du processeur. Le groupe G est le bus de résultat de l'ALU et le groupe H est situé dans le module calculant l'adresse mémoire.

On remarque que les chemins des groupes A à F ont les marges temporelles les plus petites. Les groupes A à F contiennent un nombre important de cellules logiques. En effet, les chemins avec les marges temporelles initiales les plus petites contiennent le plus grand nombre de portes logiques. Cette propriété implique que pour le pire cas, le nombre de commutations sur ces chemins est le plus grand. A fortiori, les dégradations de délai sur ces chemins sont les plus grandes.

Par contre, les groupes G et H ont les plus petites marges temporelles après dégradation, lorsqu'on considère des exécutions typiques. C'est pourquoi une approche pire cas ne peut pas identifier les parties sensibles du processeur et notamment les modules les plus sensibles à la dégradation HCI. Dans les deux scénarios, les modules sensibles sont différents et nécessiteront des techniques différentes de masquage ou d'équilibrage des effets du HCI.

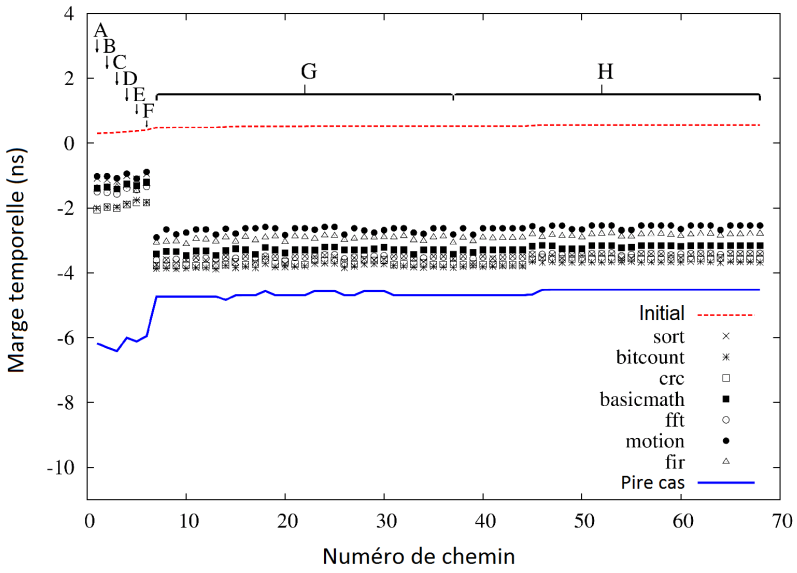


Figure 4.18 – *Marges temporelles des chemins les plus longs pour différentes exécutions. On représente aussi les marges initiales ainsi que le scénario du pire cas.*

Nous venons d'analyser les unités sensibles du processeur. Dans la suite, nous identifions l'impact de la partie logicielle sur la dégradation.

4.3.3.2 Identification des instructions agressives

Dans ce paragraphe, nous identifions les instructions qui génèrent la plus grande dégradation sur les 238 chemins. Pour cela, nous analysons le profil des benchmarks exécutés. Un profil est obtenu à partir de la plateforme de simulation au niveau porte logique. Les niveaux d'optimisations et les données sont ceux du tableau 4.2. Chaque instruction lue de la mémoire d'instructions est mémorisée, au fur et à mesure du déroulement du benchmark. Ensuite, les instructions sont regroupées selon leurs identifiants. Nous avons utilisé des regroupements selon le type d'opération effectué dans l'unité « Execute ». Par exemple, le groupe « add » comprend les additions sur 16 ou 32 bits. Le groupe « sub » correspond aux soustractions, « logic » se réfère aux manipulations de bits (masquage ou forçage), « shift » représente les décalages logiques et « non ALU » regroupe les opérations qui n'utilisent pas les ressources de l'ALU, comme les branchements et les accès mémoires.

Comme on le remarque figure 4.19, « bitcount » utilise le plus d'opérations de décalage (« shift ») comparé aux autres benchmarks. Ensuite, « bitcount » est le benchmark contenant le plus d'opérations de décalages, alors que « motion » en exécute le moins. Comme l'étage « Execute » du pipeline d'AntX contient les chemins les plus longs et participe aux décalages, on peut conclure que le benchmark qui contient le plus d'opérations de décalages génère le plus de dégradation sur ces chemins.

Nous présentons maintenant une application de l'estimation de la fréquence maximale de fonc-

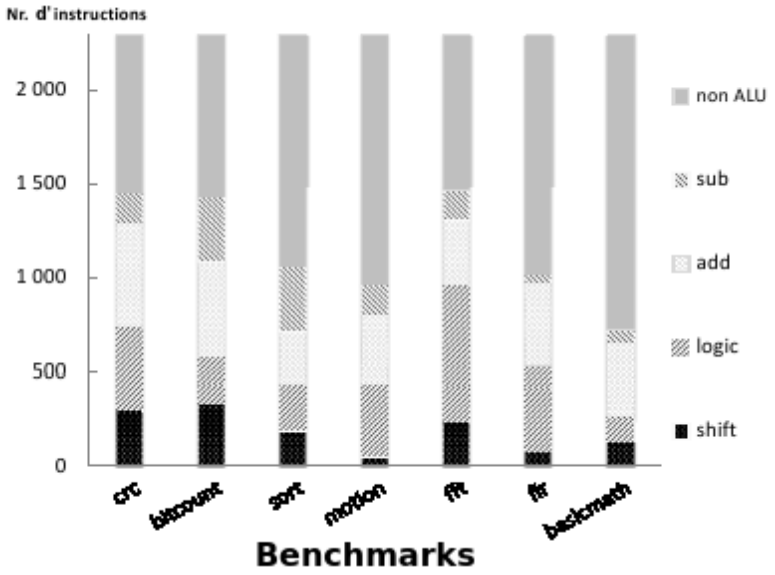


Figure 4.19 – *Profil d'instructions exécutées dans les benchmarks pour des temps d'exécutions égaux.*

tionnement.

4.3.3.3 Fréquence maximale de fonctionnement

Cette partie discute d'une méthode de calcul par extrapolation de la dégradation de la fréquence maximale sur la durée de mission.

Nous identifions la paire d'instructions la plus agressive, en utilisant la plateforme de simulation au niveau fonctionnel. Puis, nous utilisons la formule 4.4 pour calculer la fréquence maximale de fonctionnement sur une période d'utilisation par exemple de $T_{\text{life}} = 2ans$. Le temps de simulation est égal à $T_{sim} = 17\,133$ ns. Le paramètre du modèle de dégradation C_2 dans l'équation 3.13 a pour valeur arbitraire $2,32.10^{-6}$ ns et nous considérons seulement le chemin critique en fin de simulation.

$$\Delta Fmax_{\text{life}} \propto \Delta Fmax_{\text{sim}} \cdot \left(\frac{T_{\text{life}}}{T_{\text{sim}}}\right)^{n} \cdot \frac{A_{\text{life}}}{A_{\text{sim}}} \tag{4.4}$$

Dans cette équation, basée sur les travaux de Takeda et Suzuki [TS83], le terme $\frac{A_{\text{life}}}{A_{\text{sim}}}$ représente le rapport entre la tension nominale (pendant la période d'utilisation d'AntX) et la tension de stress utilisée pendant la simulation. $Fmax_{\text{sim}}$ est la fréquence maximale qui satisfait aux conditions temporelles du chemin critique initial $\left(\frac{1}{4,54ns}\right) \approx 220\,\text{MHz}$. Pour $n = 0,5$ et un ratio de $\frac{A_{\text{life}}}{A_{\text{sim}}} = 10^{-7}$, la déviation en fréquence vaut $\Delta Fmax \approx 8,55\,\%$ après un temps $T = 2\,\text{ans}$.

Synthèse

Cette section a permis d'évaluer la précision et la performance de la méthode de simulation fonctionnelle des dégradations, et de présenter des résultats d'analyse temporelle du cœur de processeur AntX.

Pour comparer avec le niveau porte logique, une simulation au niveau porte logique a permis de repérer les chemins qui sont le plus impactés par le vieillissement. Le circuit décrit par son code RTL est d'abord traduit en langage plus détaillé (*netlist*), avec les informations de la bibliothèque technologique. A partir de cette description, une première simulation est effectuée pour calculer les chemins. Ensuite, les stimuli sont fournis, pour calculer les commutations de chaque signal. Enfin, un outil vient calculer les délais en fin de simulation, selon le modèle de dégradation au niveau porte logique et l'extrapolation choisie. A partir de simulations de courtes durées, on obtient les dégradations du circuit en fin de vie.

Ensuite, nous avons calculé les variations de fréquence maximale de fonctionnement pour la paire d'instructions la plus agressive. Ceci permet de définir une marge de sécurité d'un processeur dès les premières étapes de sa conception. Il suffit de fixer la fréquence de fonctionnement pour qu'elle soit en dessous de la marge calculée quand on exécute en boucle la paire d'instruction la plus agressive.

Enfin, l'analyse temporelle du processeur AntX est conduite d'abord au niveau fonctionnel. La marge temporelle moyenne est utilisée comme moniteur de dégradation du processeur. Les chemins sensibles sont identifiés *a priori*, dès la phase de simulation fonctionnelle d'une architecture. Au niveau porte logique, les résultats permettent de vérifier les marges temporelles des chemins. Les chemins sensibles sont aussi identifiés ainsi que la fréquence maximale de fonctionnement après une durée d'utilisation choisie. Enfin, les instructions les plus agressives sont identifiées.

Des résultats de cette section, nous tirons deux points importants concernant le chemin critique.

1. D'abord nous confirmons que le chemin critique n'est pas le chemin le plus sensible au HCI pour tous les scénarios, et ce, malgré l'augmentation de son délai.
2. Ensuite, la dégradation de la marge temporelle moyenne est la plus importante pour le benchmark « basicmath ». Or, « bitcount » induit le plus grand délai sur le plus long chemin après dégradation, parmi tous les benchmarks. Le chemin critique n'est donc pas adapté pour représenter la dégradation de la marge temporelle moyenne du circuit.

C'est pourquoi il est important de conserver un ensemble de chemin pour représenter à haut niveau la dégradation du processeur.

4.4 Conclusion

Dans ce chapitre, trois résultats ont été montrés.

1. Nous avons tout d'abord évalué la performance du simulateur fonctionnel. En moyenne, la simulation a une vitesse de 10^7 instructions par seconde.
2. Nous avons évalué la précision par rapport au niveau porte logique. Pour les chemins dont la dégradation relative est faible, il y a proportionnalité entre le niveau porte logique et le niveau fonctionnel. Pour les autres chemins, les différences observées proviennent des trois sources.

Premièrement, les modèles d'instructions contiennent des valeurs pour des cas d'exécution particuliers ; ces valeurs sont obtenues avec les programmes unitaires. D'autre part, les valeurs au niveau porte logique sont calculées à chaque cycle pendant l'exécution effective du benchmark et sont donc plus précis. Ensuite, à chaque cycle d'exécution, la différence entre le niveau porte logique et le niveau fonctionnel se cumule dans le temps. Les simulations des benchmarks comportent des dizaines de milliers de cycles et impliquent une erreur importante en fin de simulation. Enfin, le modèle de dégradation décrit à la section 3, paragraphe 2 utilise une majoration de la dégradation de délai des chemins qui ne tient pas compte des nœuds internes.

3. Nous avons analysé l'effet du logiciel sur les dégradations des marges temporelles du processeur AntX, au niveau porte logique. Les instructions de décalage binaire sont les instructions les plus agressives. Dans le cas du processeur AntX, elles activent les portes logiques contrôlant le bus de résultat de l'ALU et celles utilisées pour calculer l'adresse mémoire.

Ces résultats confirment la validité de notre méthode pour l'exploration d'application sur une architecture de processeur.

CONCLUSIONS ET PERSPECTIVES

De nos jours, la miniaturisation des transistors utilisés dans les procédés de fabrication des MultiProcessor System-on-a-Chip (MPSoC) augmente fortement la probabilité d'apparition des défaillances matérielles. Les méthodes de conception actuelles ajoutent des marges sur les caractéristiques du produit final. Si rien n'est fait au niveau de la conception, les produits deviennent moins compétitifs du fait de ces marges. Il est donc nécessaire d'estimer le vieillissement des MPSoC pendant la conception. Comme les MPSoC contiennent plus d'un milliard de transistors, les techniques de conception des MPSoC se basent sur des simulations à haut niveau d'abstraction et utilisent des simulateurs Instruction Set Simulator (ISS) pour valider leur fonctionnement pendant les étapes de conception.

L'objectif de cette thèse était de définir une méthodologie d'estimation à haut niveau du vieillissement d'un MPSoC. Nous avons proposé une méthode qui est assez rapide pour pouvoir simuler un MPSoC entier et fournir les résultats de vieillissement sans ralentir la simulation fonctionnelle. Elle permet donc l'estimation à haut niveau du vieillissement, soit dès les premières phases de conception d'un Circuit Intégré (CI).

Résumé des contributions

Pour simuler les dégradations matérielles des MPSoC, nous avons répertorié les méthodes actuelles de simulation du vieillissement. Nous avons conclu qu'elles ne sont pas applicables aux MPSoC car la rapidité de simulation est trop faible par rapport au niveau porte logique [Tu+93 ; AE03 ; Hua+11 ; LGS09], ou bien car elles utilisent des modèles qui ne sont pas validés [Sri+04 ; Gup+09a], donc sans assurance sur leur précision.

En particulier, les simulations au niveau transistor et au niveau porte logique sont très précises mais ralentisse trop l'exploration des différentes configurations d'un MPSoC. Elles nécessitent une description physique du circuit, très volumineuse et complexe dans le cas de processeur. Les approches aux niveaux supérieurs permettent d'accélérer notablement le temps de simulation des MPSoC mais ne disposent pas de modèles de dégradation validés. Aussi, elles utilisent des hypothèses fortes sur la dégradation et surestiment sa valeur. À notre connaissance, aucune étude n'a proposé de modèle de dégradation au niveau fonctionnel pour simuler les MPSoC qui prenne en compte les chemins du circuit.

C'est pourquoi nous avons introduit la simulation fonctionnelle du Hot Carrier Injection (HCI) qui permet d'estimer les marges temporelles des MPSoC à haut niveau. Elle prend en compte la

topologie du circuit et elle est validée empiriquement pour plusieurs cas d'exécution. En particulier, une première simulation au niveau porte logique est réalisée à partir de la description *netlist* du circuit. Un modèle de similaire à [TS83] est utilisé pour simuler les dégradations des portes logiques, selon la méthodologie « Bottom-Up ». Ensuite, le modèle de dégradation au niveau fonctionnel est construit, à partir d'application de test ; ceci constitue notre modèle de dégradation empirique. Enfin, une estimation des marges temporelles au niveau fonctionnel est réalisée avec un ISS quelconque. Les marges temporelles du circuit sont estimées après avoir exécuté une application et donc subi un stress HCI.

Dans un second temps, nous avons validé cette méthode en évaluant la précision des estimations pour le processeur AntX. Nous avons tout d'abord évalué la vitesse de simulation fonctionnelle du HCI à 10^7 instructions par seconde. Ensuite, nous avons évalué la précision par rapport au niveau porte logique. Plus précisément, pour les chemins dont la dégradation relative est faible, il y a proportionnalité entre le niveau porte logique et le niveau fonctionnel. Pour les autres chemins, les différences observées proviennent de la méthode de génération des modèles d'instructions, du cumul des erreurs à chaque cycle de simulation et du modèle de délai utilisé au niveau fonctionnel. En appliquant notre méthodologie de simulation fonctionnelle à un processeur, il est possible d'identifier l'unité fonctionnelle la plus vulnérable pour la dégradation HCI. On peut également discriminer les applications les plus agressives pour le vieillissement du circuit.

Finalement, notre méthode d'estimation des dégradations au niveau fonctionnel permet d'estimer les dégradations temporelles des processeurs dues au HCI. Avec notre méthodologie, il est possible de calculer le vieillissement d'un processeur à haut niveau et notre étude pourra être étendue aux autres processeurs d'un MPSoC.

Perspectives à court terme

Utiliser un modèle de dégradation probabiliste. Avec l'augmentation de la variabilité [LSeb], les modèles de dégradation déterministes ne sont plus assez précis. Une valeur probabiliste de marge temporelle peut alors être utilisée pour calculer les dégradations du circuit.

Inclure le phénomène Negative Bias Temperature Instability (NBTI). De nos jours, les défaillances matérielles sont de plus en plus dues au mécanisme NBTI pour les technologies 45 nm et inférieures. En incluant les modèles du NBTI les plus récents, il est possible de construire des modèles de dégradation au niveau fonctionnel pour le NBTI. Le paramètre clé dans la génération du modèle n'est plus la commutation mais le temps de maintien des signaux à la valeur logique '1'. La méthode générale reste identique à celle du chapitre 3.

Perspectives à long terme

Valider la méthodologie en calibrant le modèle de dégradation. Dans cette thèse, les simulations sont basées sur des modèles de dégradation validés auparavant sur silicium [TS83]. Ceci permet de construire une première base pour l'estimation des défaillances. Or, le vieillissement dépend des valeurs de variation de procédé, de tension d'alimentation et de température [Gup+09b], que nous supposons constants dans cette étude. La validation par mesure expérimentale sur une puce apporterait une étape de calibration. Ainsi la température, les dégradations de fréquence d'oscillateurs et la tension d'alimentation peuvent aider à améliorer les modèles de dégradation.

Définir une stratégie d'estimation en ligne du vieillissement. De nos jours, le comportement dynamique des applications implique une difficulté à prévoir au moment de la conception les cas applicatifs à couvrir. Aussi, il est nécessaire d'effectuer de nombreuses simulations pour valider les marges temporelles d'un circuit dans tous les cas applicatifs possibles. Un *monitoring* est alors mis en place pour prendre en compte toutes les conditions d'exécution. Pour cela, il faut ajouter un circuit matériel permettant de surveiller l'exécution et d'estimer les dégradations. Les marges temporelles sont alors disponibles pendant l'exécution. En ajoutant un mécanisme de régulation, on peut imaginer un système adaptant ses performances en fonction de son état de dégradation. Par exemple, un processeur pourra avoir de meilleures performances dans le cas où il est peu soumis aux effets du HCI, tout en ayant un temps de vie constant. Enfin, une telle méthode peut permettre, à terme, de prendre en compte des cas d'exécution non prédictibles et le dynamisme des applications.

Finalement, l'estimation des dégradations est un problème vaste, qui gagne de plus en plus d'attention au sein de la communauté scientifique. La gestion proactive de la fiabilité intéresse aussi tout particulièrement l'industrie des semi-conducteurs car cela permet de transférer les coûts de réparation des produits tombants de plus en plus fréquemment en panne vers des techniques de conception plus raisonnée. Les produits livrés seraient alors plus fiables comme c'est le cas avec les techniques de DfR [5].

5. Design for Reliability

GLOSSAIRE

CI : Circuit Intégré (p. VII, XIII, 1, 2, 6, 20, 21, 25, 27, 38, 41, 53, 91)

ISS : Instruction Set Simulator (p. VIII, 35, 38, 39, 41, 42, 59, 61, 69, 70, 73, 81, 91)

ESL : Electronic System Level (p. VIII, 28, 36, 37)

CMOS : Complementary Metal-Oxide-Semiconductor (p. VII, 2, 11, 14, 15, 18, 24)

MOS : Metal-Oxide-Semiconductor (p. VII, 5, 6, 8, 16, 46)

nMOS : Transistors à effet de champ MOS à canal N (p. 5, 13, 14, 17, 18)

pMOS : Transistors à effet de champ MOS à canal P (p. XI, 5, 14, 17, 18)

SoC : System-on-Chip (p. 5, 8, 25, 27, 28, 30, 36)

MPSoC : MultiProcessor System-on-a-Chip (p. 1, 2, 5, 27, 28, 30, 33–36, 39, 41, 59, 61, 91, 92)

EM : Electromigration (p. 8, 37)

NBTI : Negative Bias Temperature Instability (p. VII, XI, XIII, 1, 2, 8–11, 13, 14, 25, 27, 29–32, 34, 37, 41, 46, 92)

HCI : Hot Carrier Injection (p. VII, VIII, XI, XIII, 1, 2, 6, 8, 11–21, 23–25, 27, 29–32, 34, 37, 41–44, 46–50, 52–54, 56, 58, 60, 66, 79–83, 85–87, 89, 91, 92)

TDDB : Time-Dependent Dielectric Breakdown (p. 8, 30, 35, 37)

MOSFET : Metal-Oxide-Semiconductor Field-Effect Transistor (p. VII, XI, 5, 8, 9, 12, 13, 25)

MTTF : Mean Time-To-Failure (p. 34, 35)

FIT : Failure in Time (p. 8, 34)

VHDL : Very high speed integrated circuit Hardware Description Language (p. 65, 80)

RISC : Reduced Instruction-Set Computer (p. XI, 50, 51, 64)

CFR : Cumulative Failure Rate (p. 33, 34)

MIPS : Microprocessor without Interlocked Pipeline Stages (p. 34, 35, 53)

TSMC : Taiwan Semiconductor Manufacturing Company (p. 34, 64, 65, 80)

RTL : Register Tranfer Level (p. 35, 39, 47, 48, 53, 59, 64, 80, 89)

ALU : Arithmetical and Logical Unit (p. 28, 55, 64, 71, 86, 87, 90)

ADL : Architecture Description Language (p. 27, 61)

BIBLIOGRAPHIE

Références bibliographiques

[Bor+03] S. BORKAR, T. KARNIK, S. NARENDRA, J. TSCHANZ et al. « Parameter variations and impact on circuits and microarchitecture ». Dans : *Design Automation Conference, 2003. Proceedings*. Fév. 2003, p. 338–342 (cf. p. 5).

[Xia+10] Y. XIANG, T. CHANTEM, R. DICK, X. HU et al. « System-level reliability modeling for MPSoCs ». Dans : *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2010 IEEE/ACM/IFIP International Conference on*. Oct. 2010, p. 297–306 (cf. p. 5).

[Gup11] T. GUPTA. « Handling design issues related to reliability in MPSoC at functional level ». Thèse de doct. Université Bordeaux 1, déc. 2011. URL : http://ori-oai.u-bordeaux1.fr/pdf/2011/GUPTA_TUSHAR_2011.pdf (cf. p. 5, 34, 35).

[SH04] J. SEGURA et C. HAWKINS. *Cmos Electronics : How It Works, How It Fails*. IEEE Press, 2004. URL : http://books.google.fr/books?id=Vl8naJHBTDIC (cf. p. 6, 15).

[HP06] J. L. HENNESSY et D. A. PATTERSON. *Computer Architecture, Fourth Edition : A Quantitative Approach*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2006 (cf. p. 6, 51).

[Ste08] K. STEVENS. *Process Scaling*. 2008. URL : http://www.ece.utah.edu/~kstevens/6770/lecture-notes/scaling.pdf (cf. p. 7).

[KK07] I. KOREN et C. KRISHNA. *Fault-Tolerant Systems*. Elsevier/Morgan Kaufmann, 2007. URL : http://books.google.fr/books?id=o_Pjbo4Wvp8C (cf. p. 8).

[Gup+09a] T. GUPTA, C. BERTOLINI, O. HERON, N. VENTROUX et al. « Effects of various applications on relative lifetime of processor cores ». Dans : *Integrated Reliability Workshop Final Report, 2009. IRW '09. IEEE International*. Oct. 2009, p. 132–135 (cf. p. 8, 34, 91).

[DoD98] DOD. *Circuit Design for Reliability Handbook*. Rapp. tech. DoD, 1998 (cf. p. 9).

[JED10] JEDEC. *Failure Mechanisms and Models for Semiconductor Devices*. JEDEC Solid State Technology Association. Nov. 2010 (cf. p. 8, 9, 15, 34, 36).

[Str+09] A. STRONG, E. WU, R. VOLLERTSEN, J. SUNE et al. *Reliability Wearout Mechanisms in Advanced Cmos Technologies*. IEEE Press Series on Microelectronic Systems. John Wiley & Sons, 2009. URL : http://books.google.fr/books?id=pI7I0Mqbjx4C (cf. p. 9–15, 19).

[Hua+03] V. HUARD, F. MONSIEUR, G. RIBES et S. BRUYERE. « Evidence for hydrogen-related defects during NBTl stress in p-MOSFETs ». Dans : *Reliability Physics Symposium Proceedings, 2003. 41st Annual. 2003 IEEE International*. Mar. 2003, p. 178–182 (cf. p. 11, 30).

[Hof+10] K. HOFMANN, H. REISINGER, K. ERMISCH, C. SCHLUNDER et al. « Highly accurate product-level aging monitoring in 40nm CMOS ». Dans : *VLSI Technology (VLSIT), 2010 Symposium on*. Juin 2010, p. 27–28 (cf. p. 11, 13, 15).

[Pag03] M. P. PAGEY. « Hot-Carrier Reliability Simulation in Aggressively Scaled MOS Transistors ». Thèse de doct. Université de Vanderbilt, août 2003 (cf. p. 15, 30).

[Hu+85] C. HU, S. C. TAM, F.-C. HSU, P.-K. KO et al. « Hot-Electron-Induced MOSFET Degradation - Model, Monitor, and Improvement ». Dans : *Solid-State Circuits, IEEE Journal of* 20.1 (fév. 1985), p. 295–305 (cf. p. 15).

[Aga+07] M. AGARWAL, B. PAUL, M. ZHANG et S. MITRA. « Circuit Failure Prediction and Its Application to Transistor Aging ». Dans : *VLSI Test Symposium, 2007. 25th IEEE*. Mai 2007, p. 277–286 (cf. p. 16).

[Sap04] S. SAPATNEKAR. *Timing*. Kluwer Academic Publishers, 2004 (cf. p. 16).

[BC09] J. BHASKER et R. CHADHA. *Static Timing Analysis for Nanometer Designs : A Practical Approach*. Springer, 2009. URL : http://books.google.fr/books?id=N1Zn1RdqPVoC (cf. p. 17, 21–23).

[Sho50] W. SHOCKLEY. *Electrons and holes in semiconductors : with applications to transistor electronics*. Bell Telephone Laboratories series. Van Nostrand, 1950. URL : http://books.google.fr/books?id=ViRTAAAAMAAJ (cf. p. 17).

[Mar+09] J. MARTÍN-MARTÍNEZ, S. GERARDIN, R. RODRÍGUEZ, M. NAFRÍA et al. « Gate dielectric degradation in CMOS inverters ». Dans : *Microelectronic Engineering* 86.10 (2009), p. 2123–2126 (cf. p. 18).

[Hua+11] V. HUARD, N. RUIZ, F. CACHO et E. PION. « A bottom-up approach for System-On-Chip reliability ». Dans : *Microelectronics Reliability* 51.9–11 (2011), p. 1425–1439 (cf. p. 19, 20, 27, 31, 46, 47, 91).

[Tu+93] R. TU, E. ROSENBAUM, W. CHAN, C. LI et al. « Berkeley reliability tools-BERT ». Dans : *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 12.10 (oct. 1993), p. 1524–1534 (cf. p. 20, 30, 39, 91).

[LBS10] D. LORENZ, M. BARKE et U. SCHLICHTMANN. « Aging analysis at gate and macro cell level ». Dans : *Computer-Aided Design (ICCAD), 2010 IEEE/ACM International Conference on*. Nov. 2010, p. 77–84 (cf. p. 20).

[She+10] W. SHEN, Y. CAI, X. HONG et J. HU. « An Effective Gated Clock Tree Design Based on Activity and Register Aware Placement ». Dans : *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 18.12 (déc. 2010), p. 1639–1648 (cf. p. 22).

[LT10] J. LU et B. TASKIN. « Clock Tree Synthesis with XOR Gates for Polarity Assignment ». Dans : *VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on*. Juil. 2010, p. 17–22 (cf. p. 22).

[Arl+06] J. ARLAT, Y. CROUZET, Y. DESWARTE, J.-C. FABRE et al. *Fault Tolerance*. fr. Encyclopedia of Computer Science et Information Systems, 2006 (cf. p. 24).

[Gup+09b] M. GUPTA, J. RIVERS, P. BOSE, G.-Y. WEI et al. « Tribeca : Design for PVT variations with local recovery and fine-grained adaptation ». Dans : *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. Déc. 2009, p. 435–446 (cf. p. 25, 26, 92).

[ITR11] ITRS. *Process Integration, Devices, and Structures*. International Technology Roadmap for Semiconductors. 2011. URL : http://www.itrs.net/Links/2011ITRS/Home2011.htm (cf. p. 25).

[Rig+04] S. RIGO, G. ARAUJO, M. BARTHOLOMEU et R. AZEVEDO. « ArchC : a systemC-based architecture description language ». Dans : *Computer Architecture and High Performance Computing, 2004. SBAC-PAD 2004. 16th Symposium on*. Oct. 2004, p. 66–73 (cf. p. 27, 61).

[Bel+08] G. BELTRAME, C. BOLCHINI, L. FOSSATI, A. MIELE et al. « ReSP : A non-intrusive Transaction-Level Reflective MPSoC Simulation Platform for design space exploration ». Dans : *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*. Mar. 2008, p. 673–678 (cf. p. 27).

[Sch+07] M. de SCHULTZ, A. MENDONCA, F. CARVALHO, O. FURTADO et al. « Automatically-retargetable model-driven tools for embedded code inspection in SoCs ». Dans : *Circuits and Systems, 2007. MWSCAS 2007. 50th Midwest Symposium on*. Août 2007, p. 245–248 (cf. p. 27).

[KN08] N. KAVVADIAS et S. NIKOLAIDIS. « Elimination of Overhead Operations in Complex Loop Structures for Embedded Microprocessors ». Dans : *Computers, IEEE Transactions on* 57.2 (fév. 2008), p. 200–214 (cf. p. 27).

[Mor+11] J. MOREIRA, F. KLEIN, A. BALDASSIN, P. CENTODUCATTE et al. « Using multiple abstraction levels to speedup an MPSoC virtual platform simulator ». Dans : *Rapid System Prototyping (RSP), 2011 22nd IEEE International Symposium on*. Mai 2011, p. 99–105 (cf. p. 27).

[LGS09] D. LORENZ, G. GEORGAKOS et U. SCHLICHTMANN. « Aging analysis of circuit timing considering NBTI and HCI ». Dans : *On-Line Testing Symposium, 2009. IOLTS 2009. 15th IEEE International*. 2009, p. 3–8 (cf. p. 27, 31, 32, 38, 39, 91).

[BES08] C. BESTORY. « Développement de stratégies de conception en vue de la fiabilité pour la simulation et la prévision des durées de vie de circuits intégrés dès la phase de conception ». Thèse de doct. Université de Bordeaux I, 2008 (cf. p. 28, 30).

[Gaj+92] D. D. GAJSKI, N. D. DUTT, A. C.-H. WU et S. Y.-L. LIN. *High Level Synthesis, Introduction to Chip and System Design*. Kluwer Academic Publishers, 1992 (cf. p. 28, 29).

[NP73] L. W. NAGEL et D. PEDERSON. *SPICE (Simulation Program with Integrated Circuit Emphasis)*. Rapp. tech. UCB/ERL M382. EECS Department, University of California, Berkeley, avr. 1973. URL : http://www.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html (cf. p. 29).

[TS83] E. TAKEDA et N. SUZUKI. « An empirical model for device degradation due to hot-carrier injection ». Dans : *Electron Device Letters, IEEE* 4.4 (avr. 1983), p. 111–113 (cf. p. 29, 46, 88, 92).

[Lee+88] P. LEE, M. KUO, K. SEKI, P. LO et al. « Circuit aging simulator (CAS) ». Dans : *Electron Devices Meeting, 1988. IEDM '88. Technical Digest., International*. 1988, p. 134–137 (cf. p. 30).

[Cada] CADENCE. *RelXpert*. http://www.celestry.com/pdf/relxpert.pdf (cf. p. 30).

[Cadb] CADENCE. *Virtuoso Multi-Mode Simulation*. http://www.cadence.com/rl/Resources/datasheets/virtuoso_mmsim.pdf (cf. p. 30, 39).

[Rui+11] N. RUIZ AMADOR, V. HUARD, E. PION, F. CACHO et al. « Bottom-up digital system-level reliability modeling ». Dans : *Custom Integrated Circuits Conference (CICC), 2011 IEEE*. Sept. 2011, p. 1–4 (cf. p. 31, 38, 39).

[Lor12] D. LORENZ. « Aging Analysis of Digital Integrated Circuits ». Thèse de doct. Université Technique de Munich, 2012. URL : http://nbn-resolving.de/urn:nbn:de:bvb:91-diss-00120507-1096635-1-0 (cf. p. 32, 46–48).

[Par+06] C. PARTHASARATHY, M. DENAIS, V. HUARD, G. RIBES et al. « Designing in reliability in advanced CMOS technologies ». Dans : *Microelectronics Reliability* 46.9–11 (2006), p. 1464–1471 (cf. p. 33).

[Sri+04] J. SRINIVASAN, S. ADVE, P. BOSE et J. RIVERS. « The impact of technology scaling on lifetime reliability ». Dans : *Dependable Systems and Networks, 2004 International Conference on*. 2004, p. 177–186 (cf. p. 33, 34, 91).

[Sri+03] J. SRINIVASAN, S. ADVE, P. BOSE, J. RIVERS et al. *RAMP a model for reliability aware microprocessor design*. Rapp. tech. RC23048. IBM, 2003 (cf. p. 33, 38, 39).

[Ska+03] K. SKADRON, M. R. STAN, W. HUANG, S. VELUSAMY et al. « Temperature-aware microarchitecture ». Dans : *In Proceedings of the 30th Annual International Symposium on Computer Architecture*. 2003, p. 2–13 (cf. p. 35, 36).

[BTM00] D. BROOKS, V. TIWARI et M. MARTONOSI. « Wattch : a framework for architectural-level power analysis and optimizations ». Dans : *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*. Juin 2000, p. 83–94 (cf. p. 35, 36).

[Cos+09] A. K. COSKUN, R. STRONG, D. M. TULLSEN et T. SIMUNIC ROSING. « Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors ». Dans : *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*. SIGMETRICS '09. New York, NY, USA : ACM, 2009, p. 169–180 (cf. p. 35, 36, 38, 39).

[Ska04] K. SKADRON. « The importance of computer architecture in microprocessor thermal design ». Dans : *Thermal and Thermomechanical Phenomena in Electronic Systems, 2004. ITHERM '04. The Ninth Intersociety Conference on*. T. 2. Juin 2004, 729–730 Vol.2 (cf. p. 36).

[Per+05] C. PEREIRA, J. LAU, B. CALDER et R. GUPTA. « Dynamic phase analysis for cycle-close trace generation ». Dans : *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. CODES+ISSS '05. Jersey City, NJ, USA : ACM, 2005, p. 321–326. URL : http://doi.acm.org/10.1145/1084834.1084913 (cf. p. 36).

[Cos+06] A. K. COSKUN, T. S. ROSING, Y. LEBLEBICI et G. DE MICHELI. « A simulation methodology for reliability analysis in multi-core SoCs ». Dans : *Proceedings of the 16th ACM Great Lakes symposium on VLSI*. ACM, 2006, p. 95–99 (cf. p. 36).

[RMD07] T. ROSING, K. MIHIC et G. DE MICHELI. « Power and Reliability Management of SoCs ». Dans : *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 15.4 (avr. 2007), p. 391–403 (cf. p. 37, 38).

[Gui11] J. GUILHEMSANG. « Test en ligne pour la détection des fautes intermittentes dans les architectures multiprocesseurs embarquées ». Thèse de doct. Université de Nice Sophia-Antipolis, avr. 2011. URL : http://tel.archives-ouvertes.fr/tel-00640599 (cf. p. 37).

[AVG07] J. ABELLA, X. VERA et A. GONZALEZ. « Penelope : The NBTI-Aware Processor ». Dans : *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*. Déc. 2007, p. 85–96 (cf. p. 38, 39).

[Gup+11a] T. GUPTA, C. BERTOLINI, O. HÉRON, N. VENTROUX et al. « Impact of Power Consumption and Temperature on Processor Lifetime Reliability ». Anglais. Dans : *American Scientific Publishers in JOLPE* (déc. 2011), Vol. 7 N°5. URL : http://hal.archives-ouvertes.fr/hal-00674305 (cf. p. 38, 39).

[MAR10] F. J. MESA-MARTINEZ, E. K. ARDESTANI et J. RENAU. « Characterizing processor thermal behavior ». Dans : *Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*. ASPLOS '10. New York, NY, USA : ACM, 2010, p. 193–204 (cf. p. 47).

[TMW94] V. TIWARI, S. MALIK et A. WOLFE. « Power analysis of embedded software : a first step towards software power minimization ». Dans : *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 2.4 (déc. 1994), p. 437–445 (cf. p. 53).

[MPS98] E. MACII, M. PEDRAM et F. SOMENZI. « High-level power modeling, estimation, and optimization ». Dans : *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 17.11 (nov. 1998), p. 1061–1079 (cf. p. 53).

[Gup+10a] T. GUPTA, C. BERTOLINI, O. HERON, N. VENTROUX et al. « High Level Power and Energy Exploration Using ArchC ». Dans : *Computer Architecture and High Performance Computing (SBAC-PAD), 2010 22nd International Symposium on*. Oct. 2010, p. 25–32 (cf. p. 53).

[Kla98] B. KLASS. *Modeling Inter-instruction Energy Effects in a Digital Signal Processor*. Research report (Carnegie Mellon University. Center for Electronic Design Automation). 1998. URL : http://books.google.fr/books?id=_dajHAAACAAJ (cf. p. 53).

[Lau+04] J. LAURENT, N. JULIEN, E. SENN et E. MARTIN. « Functional level power analysis : an efficient approach for modeling the power consumption of complex processors ». Dans : *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*. T. 1. 2004, 666–667 Vol.1 (cf. p. 53).

[Bec+11] C. BECHARA, A. BERHAULT, N. VENTROUX, S. CHEVOBBE et al. « A Small Footprint Interleaved Multithreaded Processor for Embedded Systems ». Dans : Beirut, Lebanon, déc. 2011 (cf. p. 64).

[Syn] SYNOPSYS. *Design Compiler, v. C-2009.06*. URL : http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DCUltra/Pages/default.aspx (cf. p. 80).

[AE03] W. ABADEER et W. ELLIS. « Behavior of NBTI under AC dynamic circuit conditions ». Dans : *Reliability Physics Symposium Proceedings, 2003. 41st Annual. 2003 IEEE International*. Mar. 2003, p. 17–22 (cf. p. 91).

[LSeb] Q. LIU et S. SAPATNEKAR. « Capturing Post-Silicon Variations Using a Representative Critical Path ». Dans : *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 29.2 (Feb.), p. 211–222 (cf. p. 92).

[Gup+10c] T. GUPTA, C. BERTOLINI, O. HERON, N. VENTROUX et al. « RAAPS : Reliability Aware ArchC based Processor Simulator ». Dans : *Integrated Reliability Workshop Final Report (IRW), 2010 IEEE International*. Oct. 2010, p. 153–156.

[Ber+12] C. BERTOLINI, O. HERON, N. VENTROUX et F. MARC. « Relation between HCI-induced performance degradation and applications in a RISC processor ». Dans : *On-Line Testing Symposium (IOLTS), 2012 IEEE 18th International*. Juin 2012, p. 67–72.

[Her+13] O. HERON, C. BERTOLINI, C. SANDIONIGI, N. VENTROUX et al. « On the Simulation of HCI-Induced Variations of IC Timings at High Level ». Dans : *Journal of Electronic Testing* 29 (2 2013), p. 127–141. URL : http://link.springer.com/article/10.1007/s10836-013-5368-x.

Publications personnelles

Parution dans des journaux scientifiques

[Her+13] O. HERON, C. BERTOLINI, C. SANDIONIGI, N. VENTROUX et al. « On the Simulation of HCI-Induced Variations of IC Timings at High Level ». Dans : *Journal of Electronic Testing* 29 (2 2013), p. 127–141. URL : http://link.springer.com/article/10.1007/s10836-013-5368-x.

Conférences internationales avec comité de lecture

[Gup+10a] T. GUPTA, C. BERTOLINI, O. HERON, N. VENTROUX et al. « High Level Power and Energy Exploration Using ArchC ». Dans : *Computer Architecture and High Performance Computing (SBAC-PAD), 2010 22nd International Symposium on*. Oct. 2010, p. 25–32 (cf. p. 53).

[Ber+12] C. BERTOLINI, O. HERON, N. VENTROUX et F. MARC. « Relation between HCI-induced performance degradation and applications in a RISC processor ». Dans : *On-Line Testing Symposium (IOLTS), 2012 IEEE 18th International*. Juin 2012, p. 67–72.

Workshop avec comité de lecture

[Gup+10c] T. GUPTA, C. BERTOLINI, O. HERON, N. VENTROUX et al. « RAAPS : Reliability Aware ArchC based Processor Simulator ». Dans : *Integrated Reliability Workshop Final Report (IRW), 2010 IEEE International*. Oct. 2010, p. 153–156.

Workshop sans comité de lecture

[Gup+11b] T. GUPTA, C. BERTOLINI, O. HERON, N. VENTROUX et al. « System Level Analysis and Accurate Prediction of Electromigration ». Dans : *European Workshop on CMOS Variability (VARI)*. 2011.

[Gup+10b] T. GUPTA, C. BERTOLINI, O. HERON, N. VENTROUX et al. « Predicting Lifetime using power consumption from Wattch ». Dans : *Proceedings of the 6th International Summer School on Adv. Computer Arch. and Compilation for Embedded Systems (ACACES)*. 2010.

RÉSUMÉ

Actuellement, les circuits numériques nécessitent d'être de plus en plus performants. Aussi, les produits doivent être conçus le plus rapidement possible afin de gagner les précieuses parts de marché. Les méthodes rapides de conception et l'utilisation de MPSoC ont permis de satisfaire à ces exigences, mais sans tenir compte précisément de l'impact du vieillissement des circuits sur la conception. Or les MPSoC utilisent les technologies de fabrication les plus récentes et sont de plus en plus soumis aux défaillances matérielles. De nos jours, les principaux mécanismes de défaillance observés dans les transistors des MPSoC sont le HCI et le NBTI.

Des marges sont alors ajoutées pour que le circuit soit fonctionnel pendant son utilisation, en considérant le cas le plus défavorable pour chaque mécanisme. Ces marges deviennent de plus en plus importantes et diminuent les performances attendues. C'est pourquoi les futures méthodes de conception nécessitent de tenir compte des dégradations matérielles en fonction de l'utilisation du circuit.

Dans cette thèse, nous proposons une méthode originale pour simuler le vieillissement des MPSoC à haut niveau d'abstraction. Cette méthode s'applique lors de la conception du système c.-à-d. entre l'étape de définition des spécifications et la mise en production. Un modèle empirique permet d'estimer les dégradations temporelles en fin de vie d'un circuit. Un exemple d'application est donné pour un processeur embarqué et les résultats pour un ensemble d'applications sont reportés.

La solution proposée permet d'explorer différentes configurations d'une architecture MPSoC pour comparer le vieillissement. Aussi, l'application la plus sévère pour le vieillissement peut être identifiée.

Mots clés : Vieillissement, Injection de porteurs chauds (HCI), Negative bias temperature instability (NBTI), Marges temporelles, MPSoC, Simulation fonctionnelle, Conception

ABSTRACT

Nowadays, more and more performance is expected from digital circuits. What's more, the market requires fast conception methods, in order to propose the newest technology available. Fast conception methods and the utilization of MPSoC have enabled high performance and short time-to-market while taking little attention to aging. However, MPSoC are more and more prone to hardware failures that occur in transistors. Today, the prevailing failure mechanisms in MPSoC are HCI and NBTI.

Margins are usually added on new products to avoid failures during execution, by considering worst case scenario for each mechanism. For the newest technology, margins are becoming more and more important and products performance is getting lower and lower. That's why the conception needs to take into account hardware failures according to the execution of software.

This thesis propose a new methodology to simulate aging at high level of abstraction, which can be applied to MPSoC. The method can be applied during product conception, between the specification phase and the production. An empirical model is used to estimate slack time at circuit's end of life. A use case is conducted on an embedded processor and degradation results are reported for a set of applications.

The solution enables architecture exploration and MPSoC aging can thus be compared. The software with most severe impact on aging can also be determined.

Keywords : Aging, Hot carrier injection (HCI), Negative bias temperature instability (NBTI), Slack time, MPSoC, Functional simulation, Conception