ÉCOLE DE TECHNOLOGIE SUPERIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE
LA MAÎTRISE EN GÉNIE ÉLECTRIQUE
M.Ing

PAR
ABDELKADER BEN LETAIFA

RÉALISATION D'UN PROTOTYPE TESTABLE PAR LA MÉTHODE DE CHAÎNES PARALLÈLES DE COURANT

MONTRÉAL, LE 8 NOVEMBRE 2002

©droits réservés d'Abdelkader Ben Letaifa

REALIZATION OF A TESTABLE PROTOTYPE BY THE METHOD OF CURRENT PARALLEL CHAINS

Abdelkader Ben Letaifa

ABSTRACT

With the exponential growth of the electronic system complexity, the built-in circuit testability got more and more fundamental importance in the microelectronic domain. The scaling from which new fault mechanisms or amplified existing mechanism effects are descended, what forces the revision of test strategies used and the development of new ones. These strategies must be compatible with CAO tools and respect constraints of costs bound to the test.

A new method of test has been developed at the ÉTS to detect the synchronization problems earlier in the process (wafer probing stage). This should permit to detect circuits not respecting specifications of speed more quickly and to avoid packaging uselessly, therefore reducing costs bound to the test.

The objective of this thesis is to design an ASIC integrating the new method and to test the circuit. Then, we want to evaluate the impact of the integration of the method on the conventional stream of conception and to propose the necessary adjustments.

RÉALISATION D'UN PROTOTYPE TESTABLE PAR LA MÉTHODE DE CHAÎNES PARALLÈLES DE COURANT

Abdelkader Ben Letaifa

SOMMAIRE

Avec la croissance exponentielle de la complexité des systèmes électroniques, la testabilité des circuits intégrés revêt plus que jamais une importance capitale dans le domaine de la microélectronique. La réduction à l'échelle dont est issue cette croissance fait apparaître de nouveaux mécanismes de défectuosités ou amplifie l'effet des mécanismes existants. La révision des stratégies de test utilisées et le développement de nouvelles stratégies est alors obligatoire. Une nouvelle méthode de test a été développée à l'ÉTS pour permettre de détecter rapidement les circuits ne respectant pas les spécifications de vitesse et d'éviter de les encapsuler inutilement, réduisant de même coup les coûts liés au test.

L'objectif de ce mémoire vise d'abord la conception, la fabrication et le test d'un circuit intégrant la logique nécessaire pour réaliser la méthodologie de test envisagée. Ensuite, évaluer l'impact de l'intégration de la méthode sur le flot conventionnel de conception et de proposer les ajustements nécessaires.

CE MÉMOIRE A ÉTÉ ÉVALUÉ
PAR UN JURY COMPOSÉ DE :

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

M. Jean Belzile, président du jury
Département de génie électrique à l'École de technologie supérieure

M. René Jr Landry, professeur
Département de génie électrique à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PULIC
LE 10 OCTOBRE 2002
À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

TABLE DES MATIÈRES

Pages

LISTE DES TABLEAUX

Pages

LISTE DES FIGURES

Page

INTRODUCTION

Historique

Au cours des trois dernières décennies, les puces et les systèmes électroniques ont connu un accroissement remarquable de leurs performances et une croissance exponentielle dans leur complexité, croissance jumelée à leur relativement courte durée de vie et aux continuelles exigences sur les deux plans suivants: gain en productivité et qualité des circuits. Plusieurs efforts ont été mis en œuvre pour répondre aux besoins de cette tendance.

Conséquemment, en plus des avancées technologiques, plusieurs domaines rattachés à cette industrie ont également connu une incessante évolution. Mentionnons par exemple le développement des outils de conception. Ce développement a permis aux concepteurs de suivre, tant bien que mal, le rythme effréné dicté par l'apparition des nouvelles technologies. En parallèle, de nouvelles problématiques sont apparues, dont celle liée au test des circuits. Comment peut-on garantir la qualité d'un circuit intégré complexe tout en respectant les différentes contraintes, telles la disponibilité d'outils qui permettent de faire les types de test envisagés, le temps nécessaire pour générer les stimuli à appliquer (vecteurs de test) et tester le circuit, et enfin le coût global engendré par cette opération?

Ces considérations ont mené à l'apparition des méthodes de tests connues sous le nom DFT (Design For Test) [9]. Ces méthodes ont l'avantage de systématiser (automatiser) les opérations visant à rendre plus facilement testable un circuit et à simplifier la génération des vecteurs de test. Durant leur parcours d'existence, ces méthodes ont été sujettes à une évolution permanente pour pouvoir suivre l'évolution des circuits intégrés. Parmi ces méthodes on retrouve celle développée à l'École de technologie supérieure (ÉTS) que nous allons présenter en détail dans le prochain chapitre.

Cette méthode, appelée chaînes parallèles de courant, vise à permettre l'estimation de la vitesse d'opération d'un circuit, sans que celui-ci soit excité à sa fréquence maximale, mais plutôt à basse vitesse. Par le fait même, ce type d'estimation pourrait être effectué très tôt dans le processus de test, à savoir à l'étape de test au niveau de la gaufre, avant même que la gaufre ne soit découpée en dés et que les circuits ne soient encapsulés. Effectuer cette estimation à ce niveau a le potentiel de réduire de manière significative les coûts du test, en éliminant plus rapidement des circuits ne rencontrant pas les spécifications et en évitant de les mettre en boîtier, opération devenue coûteuse avec les techniques sophistiquées d'encapsulation.

Objectifs

L'objectif principal de ce mémoire est de mettre en application la nouvelle méthode de test. Pour ce faire, un prototype a été conçu et fabriqué spécialement pour intégrer la logique nécessaire. Ensuite ce prototype a été validé en laboratoire en utilisant des moyens disponibles tel qu'un oscilloscope, des sources d'alimentation et des générateurs d'horloge. Le circuit intégré a été fabriqué grâce aux facilités offertes par la Société Canadienne de la Microélectronique (SCM).

Contribution

Notre contribution dans ce projet est d'atteindre les objectifs annoncés précédemment, c'est-à-dire concevoir, faire fabriquer et tester un circuit intégré avec les caractéristiques qui répondent aux exigences de la méthodologie de test. De plus une nouvelle stratégie de design fut mise en œuvre pour pouvoir surmonter les contraintes spécifiques à ce circuit.

Contenu du rapport

Dans ce qui suit nous retrouvons les chapitres suivants :

- Méthode de test et prototype envisagé : présentation de la nouvelle méthode de test, ainsi que l'architecture envisagée pour implémenter la logique nécessaire.
- Méthodologie de design : présentation des premières étapes (description VHDL et synthèse) de la conception dans le cas du flot standard, ainsi que les passages particuliers adoptés pour la réalisation du prototype envisagé.
- Dessin des masques (« layout ») : présentation de l'opération de dessin des masques, ainsi que les particularités de notre prototype.
- Tests et résultats : présentation de la stratégie de test, ainsi que les outils requis et les résultats obtenus.

Enfin une conclusion globale qui constitue une synthèse des grands thèmes et idées de ce mémoire, ainsi que des recommandations. Des annexes sont placées à la fin de ce rapport. Elles contiennent le code VHDL et des détails concernant la modélisation en VHDL et la synthèse.

CHAPITRE 1

MÉTHODOLOGIE DE DESIGN ET PROTOTYPE ENVISAGÉ

1.1 Introduction

Au début des années soixante-dix, il fut prédit que la réduction à l'échelle devait permettre de doubler à tous les 12 à 18 mois le nombre de transistors dans un circuit intégré. Cette prédiction, connue sous le nom de loi de Moore [21], s'est avérée juste : la complexité des circuits intégrés a suivi ce rythme exponentiel depuis. Les fréquences d'opération des circuits intégrés ont également connu une progression remarquable, pendant cette même période, passant des KHz au GHz.

Les mécanismes de défectuosités, causant des pertes de rendement, évoluent également au fil des technologies. Certains mécanismes deviennent moins importants, d'autres font leur apparition ou deviennent plus dominants. Ainsi, l'augmentation graduelle du nombre de couches de métallisation, qui varie aujourd'hui entre 5 et 7 couches, a favorisé l'émergence des mécanismes causant des court-circuits ou des circuits ouverts, qui sont aujourd'hui parmi les causes de pertes de rendement les plus courantes. Parmi les conséquences de la présence de ces défectuosités dans un circuit intégré, on note l'augmentation des délais de propagation des signaux affectés, pouvant mener à des problèmes de synchronisation. Ainsi, un circuit ne pourra opérer à la vitesse désirée s'il contient des défectuosités suffisamment importantes pour que le chemin sensibilisé ne viole les spécifications de synchronisation. Une telle violation est appelée panne de type délai (PTD) [9].

Traditionnellement, on détecte les PTD par une combinaison de tests appliqués à la vitesse de fonctionnement anticipée du circuit, aussi appelée vitesse maximale. L'application de tests à vitesse maximale est contraignante à plusieurs niveaux, incluant le fait que certains types de tests ne peuvent pas être utilisés, et que l'application doit être faite après la mise en boîtier.

Le meilleur exemple de test ne pouvant être appliqué à vitesse maximale est le test basé sur des registres à balayage ("scan-based test") [13]. Ce test, qui implique la transformation des bascules d'un circuit afin qu'elles puissent former, en mode test, un ou plusieurs registres à décalage, est seulement appliqué à vitesse réduite.

Certains tests peuvent être utilisés à vitesse maximale, par exemple les tests d'auto vérification ("Built-in self test, BIST") et les tests dits fonctionnels. Les tests d'auto vérification exigent l'ajout dans le circuit de logique supplémentaire permettant la génération de vecteurs pseudo-aléatoires et la compression des résultats internes sous la forme d'une signature. Même si les tests de type BIST sont efficaces pour la détection des PTD, surtout dans les circuits intégrés contenant de la mémoire imbriquée ("embedded memory"), les solutions développées pour les portions des circuits contenant de la logique sont beaucoup moins populaires en raison de l'accroissement inhérent de la surface causé par l'addition des circuits nécessaires à l'implantation de cette fonction d'autotest [13].

Les tests fonctionnels ont également leurs limitations. Nous savons que la génération de vecteurs de tests fonctionnels est facile si nous ne dépassons pas la vitesse limite à laquelle peut opérer le testeur (générateur d'échantillons PG : Pattern Generator; analyseur logique LA : Logic Analyzer), mais dans ce cas le testeur doit être constamment mis à jour pour pouvoir suivre l'évolution des circuits intégrés. À ceci s'ajoute la difficulté d'avoir une couverture satisfaisante de pannes et de la quantifier, ainsi que des séquences d'initialisation très longues, ce qui entraîne une large utilisation de la mémoire du testeur et par conséquent un temps de test important [13]. Le tout rend difficilement justifiable l'utilisation de ce type de test, en particulier pour des circuits plus élaborés.

1.2 Méthodologie proposée

Dans le cadre de ce projet, nous allons présenter une nouvelle méthode de test développé à l'École de technologie supérieure (ÉTS) par le professeur Claude Thibeault. Cette méthode vise la détection des défectuosités causant une

augmentation de délai menant à des violations de synchronisation, ainsi que l'utilisation de testeurs moins récents afin d'estimer la vitesse maximale de fonctionnement du circuit intégré, et ce plus tôt dans le processus des tests, c'est-à-dire, à l'étape de « wafer probing ».
Cette méthode présente deux aspects :

- une méthodologie de design automatisable facilitant le test des circuits intégrés : des convertisseurs voltage-à-courant sont branchés en un ou plusieurs réseaux fournissant un courant activé par le passage de transitions (montantes ou descendantes), et
- une interface entre le circuit à tester et le testeur, qui transforme le courant en signaux binaires qui seront analysés par l'appareil de test utilisé.

La stratégie proposée peut être une alternative efficace et pratique pour les raisons suivantes. Les coûts en logique supplémentaire sont faibles, et elle est facilement automatisable, s'apparentant à la technique d'insertion des chaînes de bascules à balayage ("scan test").

Un autre élément important est le fait que même les testeurs moins rapides sont capables de mesurer des différences de délais entre deux signaux avec une résolution de l'ordre de quelques dizaines de picosecondes. Par exemple, le testeur IMS-XL60 a une fréquence maximale de fonctionnement de 60MHz mais peut mesurer des différences de délais avec une résolution de 100ps (ce testeur va être éventuellement utilisé pour valider le circuit intégré fabriqué spécialement pour la validation de la méthode).

1.3 Principe de fonctionnement.

De manière plus détaillée, les convertisseurs voltage-courant insérés peuvent être de simples inverseurs. Ils seront plus tard appelés des senseurs. Lors du passage d'une transition, il y aura consommation de courant que nous pourrons par la suite exploiter. Ces inverseurs seront insérés aux endroits suivants : à l'entrée des données des bascules, et à l'extrémité de l'arborescence de distribution des signaux d'horloge

(clock tree). Nous créerons au moins deux chaînes d'inverseurs, une pour l'entrée des données des bascules et l'autre pour le signal d'horloge (Figure 1). Le coût d'insertion est d'un inverseur par nœud d'intérêt.

Nous pouvons également sélectionner seulement les bascules situées à la fin des trajectoires combinatoires jugées critiques. Les inverseurs utilisés peuvent être de dimensions minimales, afin de minimiser l'impact de leur insertion (surface, charge).Le courant mesuré aux nœuds des alimentations de deux types de senseurs sera transformé en tension, et ce à l'extérieur du circuit sous test. Ces impulsions seront traitées par le testeur ou un oscilloscope pour déterminer le délai entre les deux signaux VDD_{clk} et VDD_{data} ou entre les deux signaux VSS_{clk} et VSS_{data}. Le délai mesuré entre la transition du signal d'horloge et celle du signal des bascules permet d'estimer la période minimale de l'horloge. En plus, la largeur de l'impulsion créée par les transitions de l'horloge permet d'estimer le biais de synchronisation créé par le réseau de distribution.

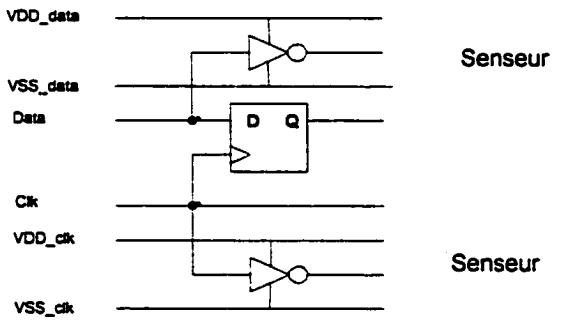

Figure 1 Insertion de senseurs

1.4 Prototype envisagé

Le prototype envisagé est un circuit intégré qui comporte trois modules identiques. Chacun des modules (figure 2) et (figure 3) possède quatre diviseurs de fréquences (« toggle ») dont les signaux d'activation (RST,CLK) sont communs et les sorties indépendantes, quatre chaînes de délai indépendantes constituées exclusivement d'inverseurs, un multiplexeur à quatre entrées, une sortie et 2 signaux de sélection. La sortie du multiplexeur est connectée à une bascule dont la sortie passe directement à la sortie du circuit intégré. Le module comprend aussi deux inverseurs qui agissent comme senseur, l'un pour les données, connecté à l'entrée de la bascule de sortie, et l'autre pour le réseau de l'horloge, connecté à l'entrée d'horloge de la bascule de sortie. Ces deux inverseurs ont chacun leurs propres broches d'alimentation, complètement indépendantes des broches d'alimentation globales. Donc chacun des modules possède quatre entrées (CLK, RST,Sel0 et Sel1) et trois sorties (clk_out, data_out et delay_out) (Figure 4), ajoutés aux quatre broches d'alimentation individuelles (VDD_{clk}, VSS_{clk}, VDD_{data}, VSS_{data}). Le circuit intégré (« TOP »), quant à lui, possède 21 entrées/sorties (figure 5) pour les données et 16 broches d'alimentations (12 pour les senseurs et 4 pour l'alimentation globale).

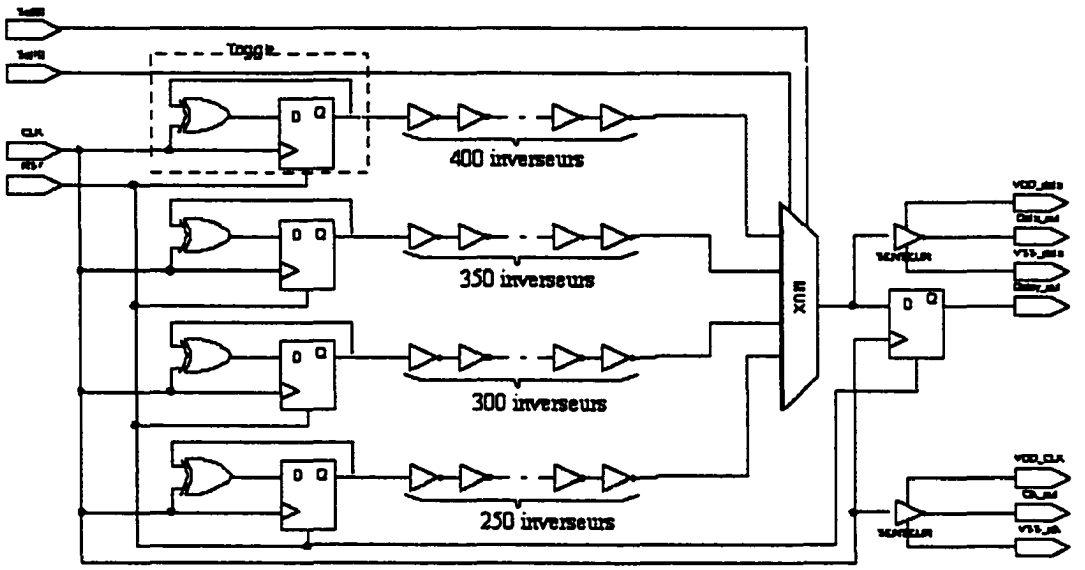

Figure 2 Architecture interne d'un module avant la synthèse

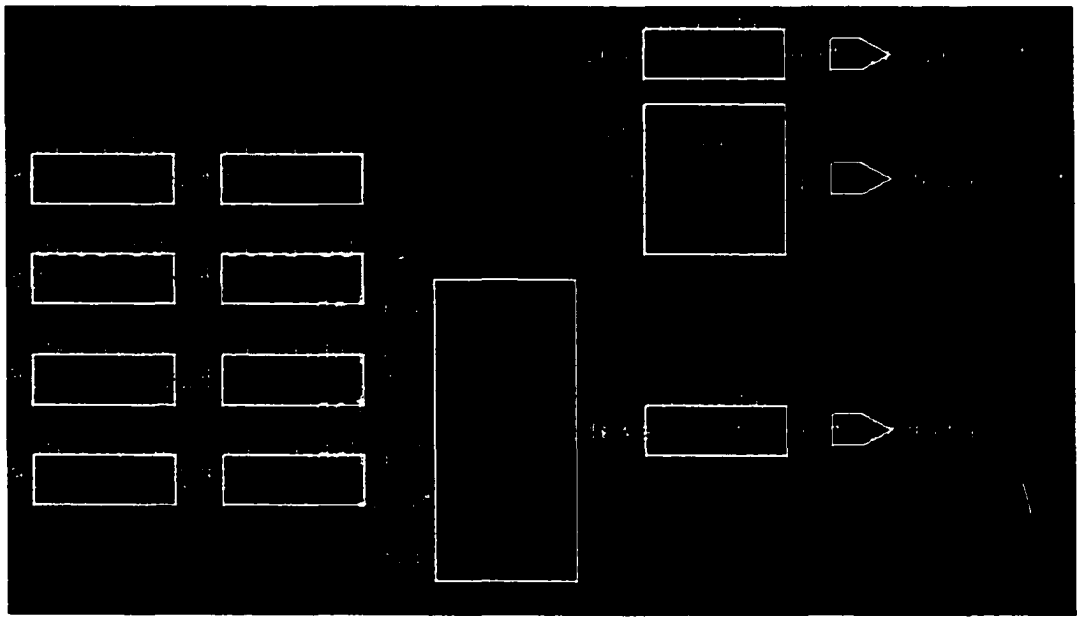

Figure 3 Vue plus détaillée de l'architecture interne d'un module avant la synthèse telle qu'elle apparaît dans Design_Compiler de Synopsys

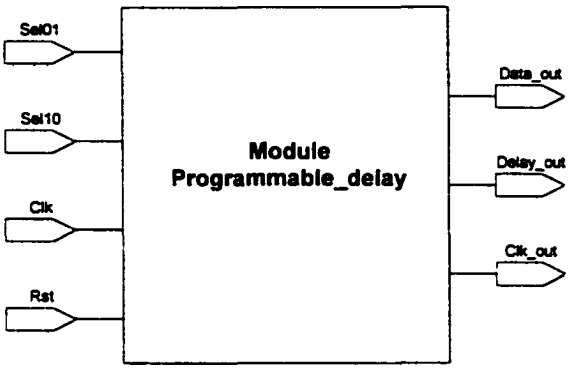

Figure 4 Vue au niveau « Top » d'un module sans les broches d'alimentation

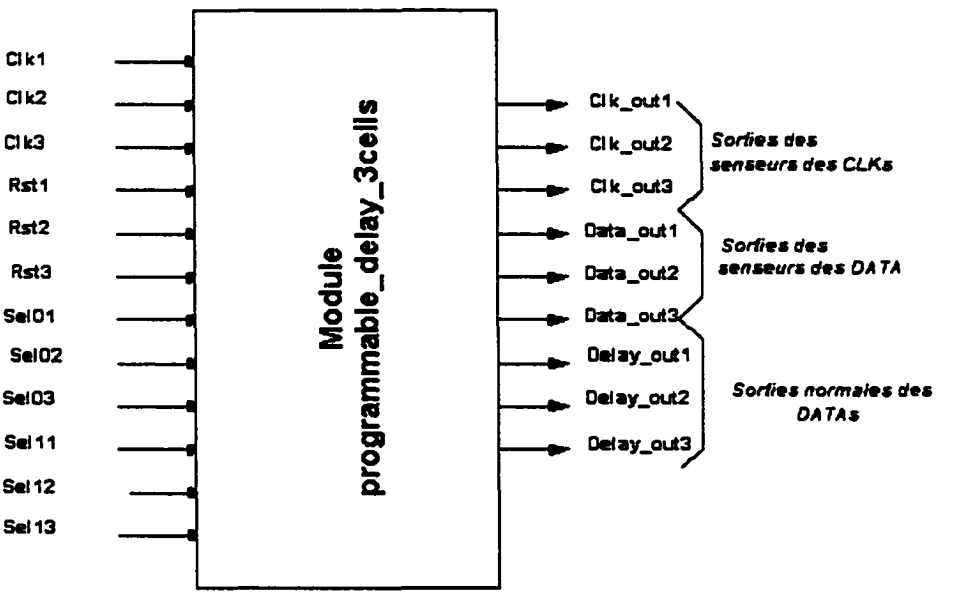

Figure 5 Entrées/sorties du circuit sans les broches d'alimentation

1.5 Conclusion

Dans ce chapitre, nous avons décrit plus en détails la problématique d'intérêt et la solution proposée, à savoir la méthode développée à l'ÉTS pour le test de défectuosités causant une augmentation de délai susceptible de créer des violations de synchronisation. Les avantages de la méthode ont été présentés, ainsi que le prototype envisagé pour mettre en application cette méthode.

Plusieurs modèles de circuit auraient pu être envisagés, mais pour des fins pratiques, nous avons décidé de concevoir le circuit le plus simple possible pour notre preuve de conception, en particulier en prévision des interventions manuelles au niveau du dessin des masques (chapitre 3).

CHAPITRE 2

MÉTHODOLOGIE DE CONCEPTION D'UN CIRCUIT INTÉGRÉ : ENTRÉE VHDL ET SYNTHÈSE

2.1 Introduction

Nous avons décrit au chapitre précédent le prototype envisagé pour mettre en application la nouvelle méthode de test (chaînes parallèles de courant). Dans ce chapitre, nous allons décrire globalement la méthodologie de conception d'un circuit intégré. Le flot standard menant à la réalisation d'un circuit intégré sera présenté, ainsi que les passerelles utilisées dans le cas de notre prototype. Notre objectif est de faire ressortir les changements à apporter afin d'intégrer la méthode de test dans le flot de conception. Ce chapitre portera sur les premières étapes de la conception (entrée VHDL et synthèse), alors que les dernières étapes (dessin de masques et « streamout ») seront décrites au chapitre suivant.

2.2 Conception d'un circuit intégré.

Plusieurs outils sont disponibles sur le marché pour accomplir les étapes de conception d'un circuit numérique. Il s'agit d'un processus qui peut devenir complexe tel que l'illustre la Figure 6. Le flot montré à la Figure 6 est celui adopté par la Société Canadienne de Microélectronique(SCM) pour la technologie CMOS 0.35 μm [11].

Ce flot débute par une entrée de code VHDL décrivant le circuit à réaliser. Habituellement, les circuits sont décrits à des niveaux d'abstraction appelé transfert de registres (« Register Transfer level ») où les parties combinatoires sont clairement distinguables des bascules ou registres. Certains outils de synthèse permettent également de décrire les circuits à un niveau plus élevé, appelé comportemental. Après avoir simulé de façon adéquate le code, nous devons indiquer le style de stratégie d'autotest (« scan test ») que nous comptons utiliser. Nous procéderons par

la suite à la synthèse qui produira une description sous forme d'une liste de portes logiques interconnectées. L'étape suivante consiste en l'insertion des bascules à balayage, selon le type de stratégie préalablement choisie. Le tout est suivi par une simulation au niveau portes logiques pour vérifier le travail de l'outil de synthèse, habituellement en comparant ces nouveaux résultats de simulation avec les résultats précédents obtenus avant la synthèse.

La liste des portes logiques et leurs interconnections est ensuite envoyée à l'environnement des outils utilisées pour la génération automatisée et la vérification du dessin des masques. Les étapes menant à cette génération et cette vérification seront décrites au chapitre suivant.

Un flot typique peut également contenir une étape supplémentaire qui précède la modélisation (entrée schématique) en VHDL, appelée conception au niveau système. À ce niveau, nous nous intéressons d'avantage aux considérations architecturales et systémiques du circuit à concevoir et à son environnement, s'il fait partie d'un système comportant d'autres éléments. À titre d'exemple, s'il agit d'un système de télécommunications, nous nous intéresserons alors à des critères de performance tels le taux de bits en erreurs pour différents rapports signal-à-bruit, la largeur de bande des signaux, etc.

Dans le cadre de ce projet, notre choix a porté sur le langage VHDL pour la modélisation de l'architecture du prototype envisagé, les outils de Synopsys pour la simulation (le simulateur VSS) et la synthèse (l'outil Design Compiler, DC) et les outils de Cadence pour la génération (Silicon-Ensemble et Virtuoso) et la vérification (DRC, LVS) du dessin des masques (Figure 7).

Tel que mentionné précédemment, nous abordons plus en détails dans ce chapitre les deux premières étapes de la conception (description VHDL et synthèse).

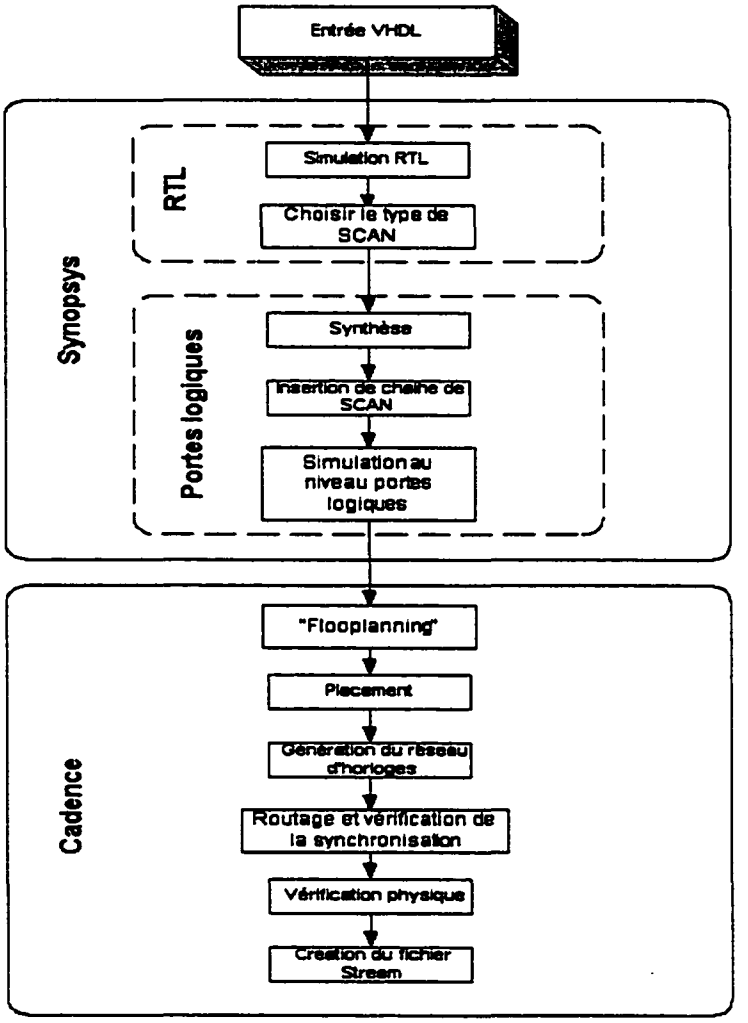

Figure 6 Méthodologie typique en conception VLSI (SCM)

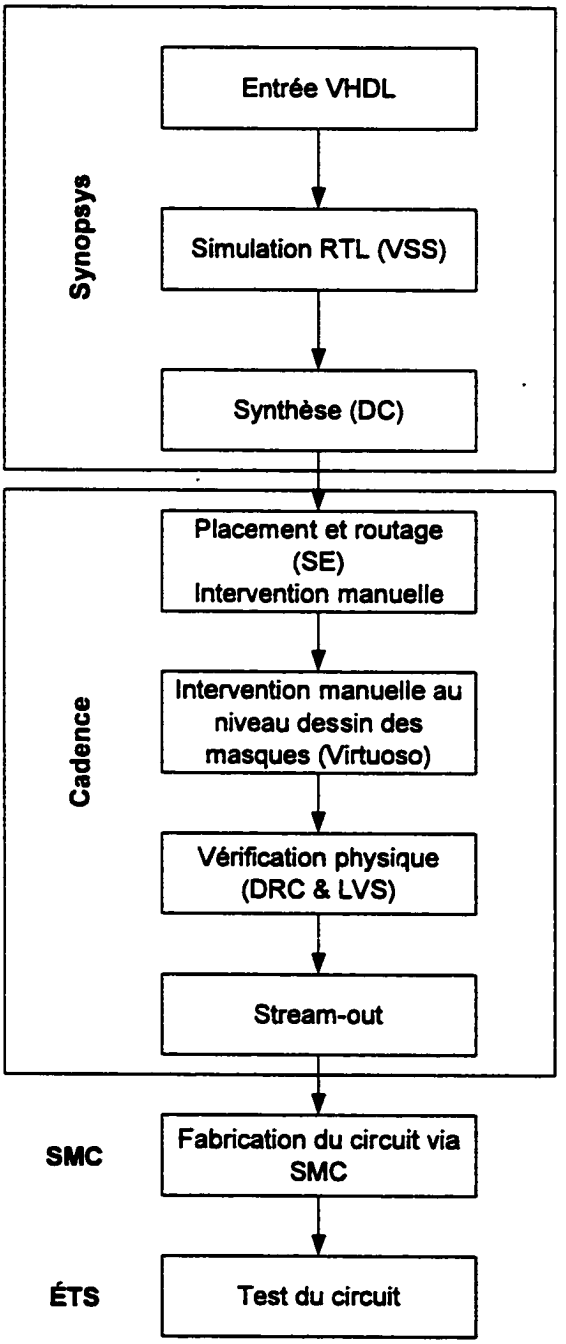

Figure 7 Méthodologie utilisée pour la réalisation du prototype envisagé

2.3 Outils utilisés pour la modélisation et la synthèse

2.2.1 Langages de description hardware [1]

La complexité croissante des circuits intégrés impose l'utilisation de langages de description de matériel de plus en plus haut niveau. Un grand nombre de travaux ont porté sur la recherche d'un langage unique pour décrire différents niveaux d'abstraction.
Deux d'entre eux ont réussi à s'imposer dans l'industrie : le langage Verilog [1, 8] créé par Phil Moore en 1983 et le langage VHDL [1, 2] initié par le DoD (département de la défense des États-Unis d'Amérique) en 1981. Ils permettent de représenter le comportement d'un circuit durant presque toutes les étapes de la conception [1].

En 1987 le langage VHDL est devenu le standard IEEE-1076[1]. Le langage Verilog, placé dans le domaine public en 1990, a été accepté comme un standard en décembre 1995[1]. Il est surtout utilisé dans l'Ouest des États-Unis. D'autres détails sur les particularités du langage VHDL et son utilité en simulation sont fournis à l'Annexe 1.

2.4 VHDL et synthèse

L'une des évolutions récentes du flot de conception est l'utilisation systématique des outils de synthèse : le concepteur écrit une description fonctionnelle de plus ou moins haut niveau dans un de ces langages, il valide cette description par simulation puis il effectue la synthèse.

Le processus de synthèse est une transformation automatique d'un programme écrit dans un langage de description de matériel, en une interconnexion de portes logiques prises dans une bibliothèque prédéfinie. L'une des tâches initiales d'un outil de synthèse est de transposer chaque portion d'une description comportementale ou

autres d'un circuit en un ensemble de fonctions matérielles connues de l'outil et équivalentes vis-à-vis de la simulation [1].

La synthèse à partir d'une description VHDL peut s'avérer un problème, car ce langage a été initialement créé essentiellement pour la simulation et non pour la synthèse [1]. Le langage VHDL a donc été défini indépendamment des contraintes liées à la synthèse. On peut distinguer deux classes de problèmes [1]. Le premier est lié aux types de VHDL. Il existe des types d'objets, des instructions et des fonctions du langage qui trouvent leur utilité dans le cadre de la simulation mais qui ne peuvent pas être implantés de façon matérielle. Un exemple de ces instructions est le *WAIT*, très utilisée pour générer des stimuli. Le concepteur qui vise une description synthétisable ne doit pas se servir de ces fonctionnalités offertes par VHDL. Cet aspect du problème a été résolu puisque les normes IEEE-1164 et IEEE-1076.3 [1] définissent un ensemble de types et de fonctions adaptées au processus de synthèse. Le second problème est lié à la puissance d'expression du langage. En effet VHDL permet de décrire de façon très diverse des éléments matériels aussi simples que des registres ou des portes logiques, mais aussi des objets plus complexes tels que les machines à états finis. L'analyse d'une description VHDL pour identifier les éléments matériels qui doivent être synthétisés est un problème non évident [1].

Les outils industriels actuels sont tous capables de prendre une description VHDL respectant la norme IEEE87 et IEEE93b, et d'en effectuer la synthèse, mais chacun d'eux impose des contraintes sur le style de description à utiliser [1]. Chaque outil de synthèse possède ses propres librairies et sa propre façon pour valider la syntaxe utilisée. Une telle utilisation de VHDL par les outils de synthèse remet en question l'intérêt même de la standardisation du langage [1]. Ceci a bien sûr agi d'une manière directe sur la méthode de conception et le style de codage, puisqu'il devient nécessaire de connaître à priori, l'outil de synthèse qui va être employé avant de pouvoir commencer la description du comportement d'une architecture en VHDL.

La réutilisation de la totalité ou d'une partie du code est soumise aux même contraintes, avec des conséquences qui peuvent s'avérer graves lorsque nous savons

qu'aujourd'hui certaines compagnies décident de ne plus développer ou supporter tel ou tel outil de synthèse. D'ailleurs nous pouvons rencontrer ce problème avec un simple changement d'une version du même outil.

La solution adoptée dans ce projet est de restreindre l'utilisation du langage aux standards. Seules les formats de type STD_LOGIC seront utilisés, avec une légère adaptation du code après synthèse à l'outil de Synopsys.

2.5 Présentation du code

Comme mentionné précédemment, le langage VHDL est adopté pour la modélisation du prototype envisagé. Nous nous sommes imposés deux contraintes pour générer le code VHDL : la simplicité et l'utilisation des librairies standards.

Tant et aussi longtemps qu'il y a des outils compatibles avec les normes IEEE, le code sera réutilisable. Pour les besoins de la méthodologie du test, une architecture spéciale a été envisagée (Figure 8).

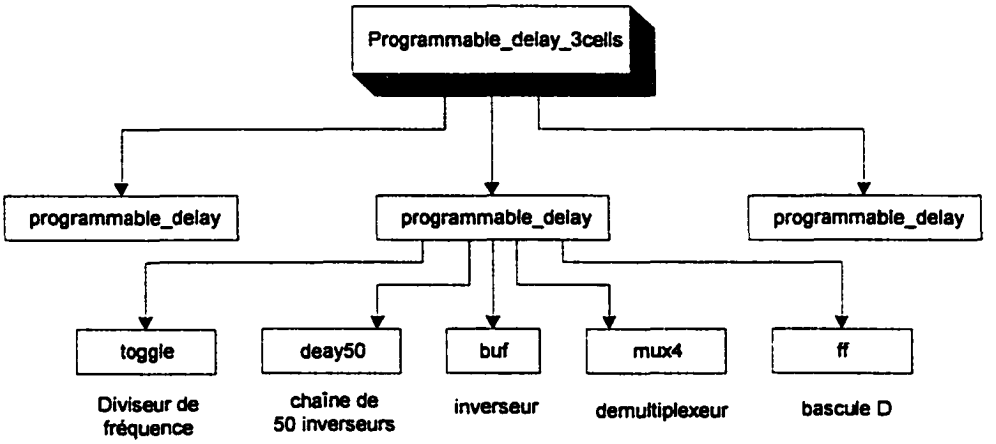

Figure 8 Architecture du modèle VHDL pour le prototype envisagé

1. Unité TOP (programmable_delay_3cells): trois modules indépendants de la même entité (programmable_delay)

2. Sous-unité (programmable_delay): choix entre quatre chaînes de longueurs différentes (nombre d'inverseurs différent)

3. Sous-unité (mux4) : multiplexeur à deux signaux de sélection

4. Sous-unité (buf) : un seul inverseur

5. Sous-unité (delay_50) : chaîne de 50 inverseurs

6. Sous-unité (toggle) : diviseur de fréquence par deux

Donc le circuit comporte trois cellules, chacune présente quatre chaînes de délai différent (400, 350, 300, et 250 inverseurs en série). Le choix du délai se fait au moyen de deux signaux de sélection qui contrôlent le multiplexeur, dont la sortie passe à travers une bascule D. La sortie de cette dernière sera acheminée à la broche de sortie.

- Les entrées du circuit sont (voir Figure 9) : clk1, clk2, clk3, rst1, rst2, rst3, sel01, sel02, sel03, sel11, sel 12, sel12.

- Les sorties sont: clk_out1, clk_out2, clk_out3, data_out1, data_out2, data_out3, delay_out1, delay_out2, delay_out2.
- Les entrées clk* servent comme entrée d'horloge et data pour le circuit. Elles sont complètement indépendantes l'une de l'autre.

- Les entrées rst* permettent la remise à zéro des deux bascules de type D utilisées dans la composante toggle et à la sortie des multiplexeurs. Elles sont complètement indépendantes l'une de l'autre.

- Les entrées sel* servent à contrôler les multiplexeurs. Elles sont complètement indépendantes l'une de l'autre.

- Les sorties clk_out* sont les sorties des senseur (des inverseurs qui n'ont aucun rôle fonctionnel, ils sont ajoutés intentionnellement pour fin de test) connectés à l'horloge. Nous devons voir le signal d'horloge inversée à ce niveau.

- Les sorties delay_out sont les sorties de chaînes de délai. La valeur du délai est proportionnelle à la longueur de la chaîne sélectionnée.

- Les sorties data_out* sont les sorties des senseur connectés à l'entrée des bascules D qui reçoivent le signal de sortie des multiplexeurs . Nous devons voir le signal de data inversé et en avance d'un coup d'horloge par rapport au data.

Le code VHDL complet est présenté à l'annexe 3.

2.6 La synthèse

De manière plus précise, la synthèse est définie comme un ensemble d'itérations visant à transformer un comportement spécifié dans un langage de description de matériel (***Hardware Description Langage***) en une liste de portes logiques interconnectées [1, 5]. Elle correspond donc au passage d'un domaine comportemental ou algorithmique à un domaine structurel.

Les outils de synthèse cherchent tout d'abord à compiler une vue comportementale, trouver son image adaptée à la synthèse indépendamment du langage utilisé [1]. Puis ils optimisent ce modèle en respectant au maximum les contraintes imposées par le concepteur avant d'en effectuer une projection structurelle sur un niveau d'abstraction immédiatement inférieur [1].

Dans le cas particulier de notre design, malgré que les inverseurs utilisés dans la description VHDL portent le même nom d'entité, l'outil a décidé de choisir les inverseurs winv_4 de dimensions plus grandes pour les chaînes de délais et les inverseurs winv_2 avec des dimensions plus petites pour les senseurs. Ce choix est intimement lié aux contraintes imposées sur l'outil, soit par le designer ou par les algorithmes d'optimisation. Un simple calcul de la charge peut conduire à ce résultat.

Deux critères principaux permettent de juger la qualité d'un outil de synthèse : la richesse de son langage d'entrée et la puissance de ses méthodes d'optimisation qui donnent la qualité de l'implémentation. Quelque soit le niveau d'abstraction, la synthèse tient compte de différents objectifs d'optimisation tels que la minimisation du délai, de la surface et de la consommation du circuit. D'autres détails concernant la synthèse sont donnés à l'annexe 2.

2.7 Méthode de test à registre à balayage (« scan test »)

Parmi les méthodes utilisées pour tester des circuits séquentiels (pour les fautes de types collé-à 1 et collé-à 0), l'insertion de bascules en chaîne (« scan chain insertion ») est la plus populaire. Cette méthode de test peut être appliquée de façon séquentielle (série) ou parallèle. Le principe de base demeure cependant le même et repose sur le principe suivant : les registres du circuit sont remplacés par des registres « multiplexés » qui sont reliés ensemble en formant une chaîne. En pratique dans un circuit de grande taille, on pourra utiliser plusieurs chaînes distinctes plutôt qu'une seule. Ceci permettra de réduire la durée des tests. De plus, on peut aussi choisir de ne pas utiliser tous les registres du circuit pour les inclure dans la ou les chaînes de test ; nous parlerons alors de chaîne partielle par rapport à une chaîne complète [10].

Nous entendons par registres multiplexés, des registres pour lesquels nous pouvons choisir leur entrée : soit le chemin de données usuel pour conserver la fonctionnalité du circuit, soit le chemin de données de test (Figure 9). Le choix de l'entrée du registre reposera sur l'activation du signal TE (« test enable »).

Le coût de cette méthode de test est l'augmentation de la surface du circuit principalement dû à l'utilisation de registres multiplexés qui sont nécessairement plus gros que de simples registres. De plus, ces registres ralentissent le circuit. Cette méthode de test requiert également un minimum de deux Entrées/Sorties supplémentaires par chaîne : TE et TI. La sortie TO peut être la sortie Q du dernier registre de la chaîne.

Dans la méthode série, nous forcerons un patron de test par l'entrée TI (« test input »). Cette opération requiert un nombre de coups d'horloge qui dépend de la longueur de la chaîne. Puis, après excitation du patron, nous obtiendrons la réponse du système en récupérant pendant le même nombre de coup d'horloge les bits à la sortie Q du dernier registre de la chaîne.

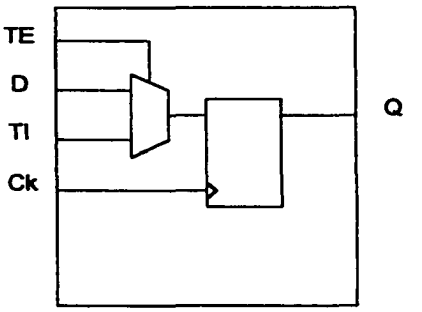

Figure 9 Bascule multiplexée

2.8 Préparation pour le placement et routage

À ce stade, les deux étapes de la conception (modélisation et synthèse) sont accomplies. Le résultat est un fichier « Netlist » qui contient toutes les informations nécessaires pour le placement et routage. Tous les éléments matériels sont définis d'une manière explicite. Ils trouvent leurs équivalences dans les librairies nécessaires pour les outils de placement et routage. Cependant, il est nécessaire, avant d'entamer cette étape, de faire une simulation après synthèse (gate-level simulation) pour s'assurer du bon fonctionnement de notre circuit. Il s'agit de vérifier que le comportement du circuit au niveau portes logiques est le même que celui au niveau RTL. Suite à la confirmation de cette étape, le « netlist » peut être transféré dans un outil de placement et routage. Pour des fins pratiques, le format du fichier à exporter à partir du Design_Compiler est le Verilog. Ce fichier servira comme point d'entrée pour l'outil Cadence.

Mentionnons que l'insertion des instances des entrées/sorties, qui aurait pu être effectué pendant la modélisation en VHDL ou après la synthèse, est laissé pour l'étape de placement et routage, car plusieurs passages particuliers seront effectués spécialement pour les broches d'alimentations.

2.9 Conclusion

Dans ce chapitre, nous avons présenté une description détaillée des premières étapes de la méthode de conception d'un circuit intégré en utilisant le langage VHDL. Nous avons insisté sur l'aspect particulier de ce langage de modélisation. De leur côté, les outils de simulation et de synthèse sont de plus en plus complets. Ils utilisent des algorithmes pour la compilation, la vérification et l'interprétation d'une description VHDL à partir duquel ils génèrent une projection structurelle. Malgré le niveau très élevé et la sophistication de ces outils, l'intervention du concepteur est nécessaire et ne peut être remplacée par des algorithmes.

Finalement, pendant cette première phase de conception, nous pouvons dire que notre flot spécial de conception est un sous-ensemble du flot standard, vu que certaines étapes ont été ignorées, tel que l'insertion de chaînes de « scan » et l'ajout de broches d'entrées/sorties. D'ailleurs c'est un des avantages de la nouvelle méthodologie pour cette étape. Dans le prochain chapitre, nous poursuivrons la description du flot de conception.

CHAPITRE 3

PLACEMENT ET ROUTAGE ET DESSIN DES MASQUES

3.1 Introduction

Nous avons vu au chapitre précédant les deux premières étapes de la conception d'un circuit numérique (description du circuit en VHDL et la synthèse avec Synopsys). Dans ce chapitre, il est surtout question de la phase finale dans le flot de conception d'un circuit électrique (figure 10) : la génération et la vérification du dessin des masques qui incluent le placement et routage, la vérification des règles de dessin (DRC) et de la cohérence entres les masques et la représentation schématique (LVS) et la génération du fichier «stream ».

Pour accommoder les particularités de la nouvelle méthode de test, des passages spéciaux sont adoptés. Ils seront décrits avec le flot standard de placement et routage pour la réalisation du prototype envisagé.

3.2 Placement et routage

Pendant le placement et routage, l'outil cherche d'une part à trouver l'arrangement optimal des modules ou cellules qui composent le circuit intégré et d'autre part, à effectuer l'interconnexion des signaux entre les différents modules/cellules et broches d'entrées/sorties.

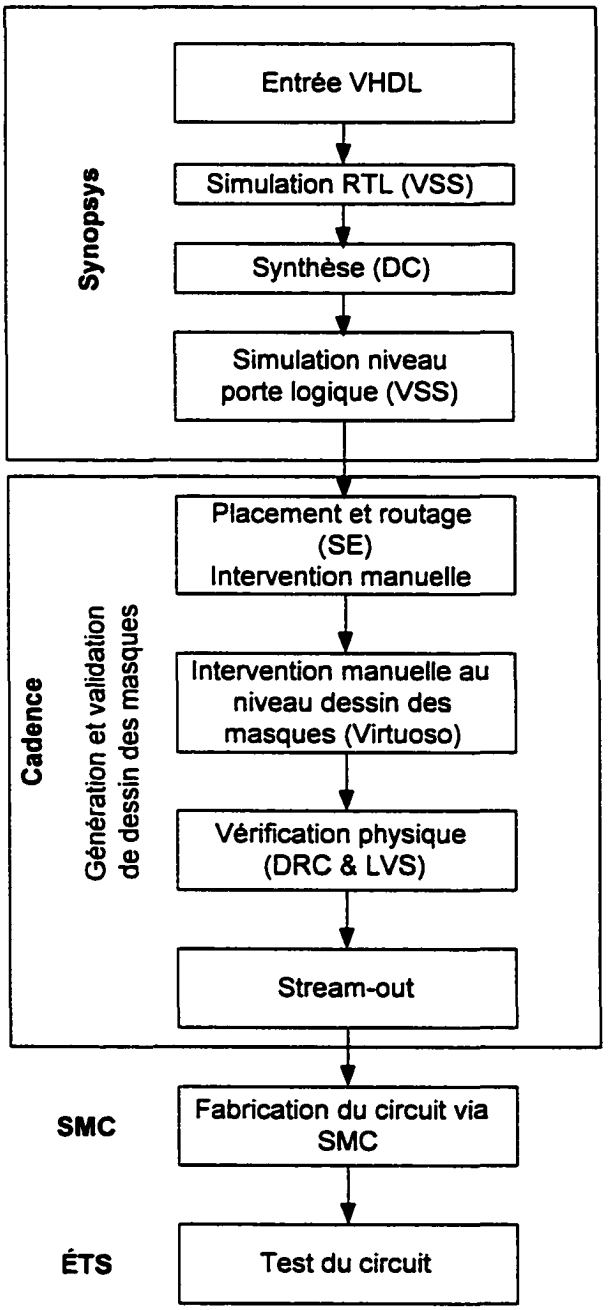

Figure 10 Méthodologie utilisée pour la réalisation du prototype envisagé

Il s'agit d'un processus dont les entrées est la liste d'éléments constituant le circuit, et la liste de signaux définissant les interconnexions. La sortie est la liste des mêmes éléments (cellule de base, modules et broches d'entrées/sorties) et des interconnexions, pour lesquels les paramètres suivants seront connus : coordonnées, surface et orientation.

Les deux processus, le placement et le routage, sont intimement liés et interdépendants car le placement dépend de la faisabilité du routage et la longueur des interconnexions dépend du placement. En pratique, les algorithmes traitent les deux problèmes séparément et d'une manière séquentielle. Des itérations sont nécessaires pour l'ajustement des résultats dans le sens optimal en respectant au maximum les contraintes imposées par le concepteur.

Il existe un certain nombre d'algorithmes de placement et routage dont les plus connus sont [7] :

- Placement constructif : production d'un placement initial complet. Technique de partitionnement globale, reposant sur la technique de « branch&bound » et « cluster growth »
- Placement itératif : modification du résultat d'un placement pour en obtenir un meilleur. Ce processus est réitéré jusqu'à l'atteinte d'une ou plusieurs conditions d'arrêt. Elle repose sur la technique de simulation Monte-Carlo.
- Steiner tree : utilisé par les algorithmes de placement pour déterminer le meilleur score du routage.
- MIN-CUT : utilisé pour déterminer le nombre de lignes de connections qui coupent la ligne fictive établie par l'algorithme.

Les problèmes de placement et de routages sont des problèmes de complexité en $O(n^2)$ (NP-complete), ce qui signifie qu'avec l'augmentation du nombre de cellules, il devient impossible de couvrir toutes les combinaisons possibles [21]. Malgré la croissance dramatique de la vitesse que peuvent atteindre les microprocesseurs, les

outils de placement et routage adoptent des algorithmes capables de produire des solutions heuristiques. La raison derrière ce choix est de limiter le nombre d'itérations nécessaires pour aboutir à une solution. Il s'agit tout simplement de trouver une solution acceptable, et non nécessairement la solution optimale.

3.2.1 Outils utilisés pour le placement et routage

Le choix de l'outil de placement et routage est porté sur l'outil de Cadence : Silicon-Ensemble. Ce dernier est défini comme un outil de synthèse physique, il représente un outil de placement et routage très puissant pour les circuits de type ASIC. Nous nous sommes servis du Silicon-Ensemble pour faire le placement et routage de notre circuit en visant la technologie CMOS 0.35 µm. Le mécanisme de transfert entre différents outils est à la fois complexe et exigeant, car souvent les logiciels proviennent de différents fournisseurs, ce qui augmente le risque de d'incompatibilité.

Dans le cadre de notre projet, il est sujet d'un transfert entre Synopsys et Cadence. Le "netlist" généré par Design-Compiler en format Verilog servira d'entrée pour Silicon-Ensemble. Des bibliothèques et des fichiers de configuration sont nécessaires pour accomplir cette étape, tel qu'illustré aux figures 11, 12et 13.

La figure 11 montre l'interface graphique de Silicon-Ensemble qui permet l'importation des fichiers de configuration nécessaires. La figure 12 montre l'interface graphique de Silicon-Ensemble qui permet l'importation du fichier Verilog et la figure 13 montre le statut de l'importation des différents fichiers avec une statistique de différentes composantes du circuit.

Pour les besoins de la méthode de test, des interventions manuelles sont alors nécessaires. En un premier temps, il s'agit de faire un estimé de la surface globale du circuit. Il est très important de bien choisir la géométrie (carré ou rectangulaire) du circuit, car cette étape est manuelle et l'outil de "floorplanning" nécessite des valeurs initiales pour déterminer la surface du circuit. Cependant si le circuit est

complexe, nous pouvons nous permettre de faire quelques itérations manuelles jusqu'à ce que nous trouvions une valeur acceptable comme le montre la figure 14. Le statut de l'opération est indiqué dans la fenêtre « Expected Results ». La valeur acceptable est celle qui est à la fois la plus petite possible, tout en répondant à nos contraintes budgétaires. La surface du circuit est étroitement liée au nombre d'entrées/sorties et au nombre de portes logiques utilisées dans le circuit. Cet estimé est nécessaire, d'abord pour des fins administratives (pour déterminer le coût de la fabrication), ensuite pour l'outil de "floorplanning".

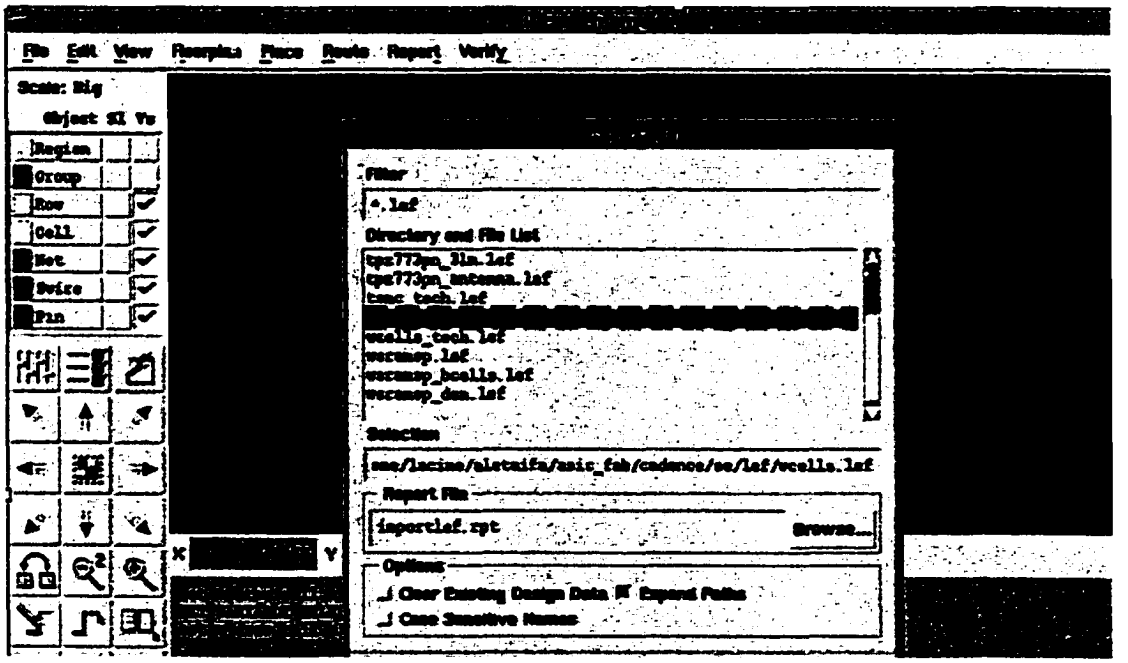

Figure 11 Importation des fichiers en format LEF (Library Exchange Format)

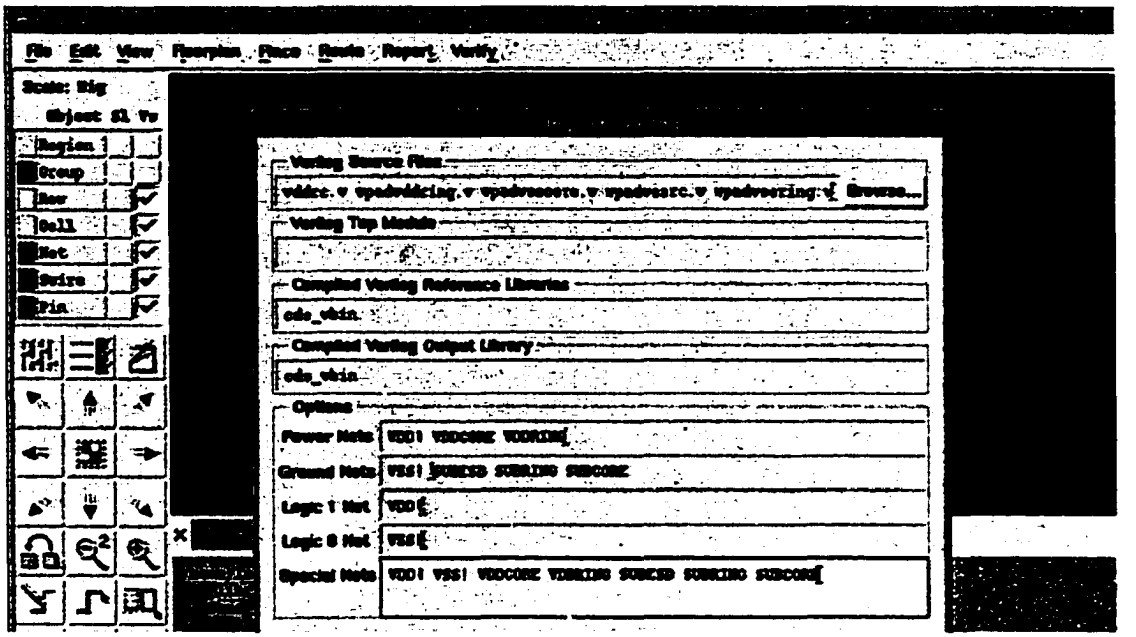

Figure 12 Importation des fichiers en format Verilog

Figure 13 Résultat de l'importation

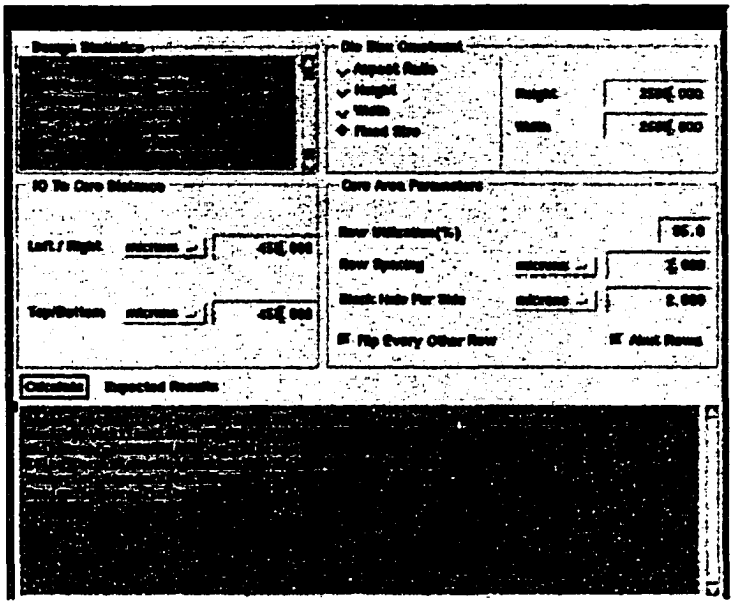

Figure 14 Initialisation du "FLOORPLANNER"

Le résultat du "floorplanning" est présenté à la figure 15, allouant l'espace nécessaire pour les E/S et l'espace pour le noyau du circuit. Le fichier obtenu sera modifié manuellement pour accommoder la nouvelle méthode de test. Ces modifications représentent une déviation par rapport au flot standard du placement et routage. Elles sont nécessaires pour accomplir les tâches suivantes :

1. Réorganiser les connections des anneaux d'alimentation (VDDRING, VDDCORE, SUBESD, SUBRING et SUBCORE), car la façon dont ils sont connectés automatiquement créera un conflit lors du routage.

2. Ajouter les broches (« pins ») spéciales d'alimentation pour l'alimentation des senseurs. Ces broches peuvent être également ajoutées dans le fichier de configuration pow.def, cependant il est toujours nécessaire d'intervenir manuellement pour réorganiser les connections.

Le fichier modifié sera importé de nouveau dans Silicon-Ensemble pour faire le placement. Le résultat obtenu est présenté à la figure 16. Cette figure montre

l'emplacement de chaque composante à l'intérieur des limites allouées. Ce résultat subira lui aussi des modifications manuelles. Les plots d'interconnections (entrées, sorties et alimentations globales) sont placés originalement en périphérie, les anneaux d'alimentation, eux, sont plus vers l'intérieur des entrées/sorties , et enfin au centre, nous trouvons le noyau (« core ») du circuit, c'est-à-dire toutes les autres cellules (winv_2, winv_4, wand_2 ...) avec les lignes d'alimentation secondaire.

Pour simplifier le routage des broches d'alimentation des senseurs, nous avons déplacé ces derniers vers l'extrémité du noyau du circuit, le plus proche possible des broches d'alimentation correspondantes en modifiant leurs coordonnées respectives. Normalement, toutes les cellules sont alimentées à partir de deux anneaux d'alimentation, soit directement ou à partir des segments dérivés (« stripes »).

Lorsque les étapes précédentes sont accomplies avec succès, nous pouvons entamer la procédure du routage. Cette étape est cruciale car c'est suivant cette étape que toutes les cellules et les broches d'alimentations seront interconnectées. Dans le cas particulier de notre prototype, les contraintes imposées sur la surface ont été rencontrées et l'opération du routage est accomplie avec succès à l'exception de la broche de VDD_{global} qui n'est pas connectée à l'anneau VDDRING. Nous avons choisi de résoudre ce problème par une intervention manuelle au niveau dessin de masques.

La sortie de l'outil Silicon_Ensemble est un « netlist » qui définit les cellules qui composent le circuit avec leurs coordonnées, leurs surfaces et leurs orientations, ainsi que leurs interconnexions. Pour les besoins de l'outil d'édition de dessin de masques (Virtuoso), le fichier sera exporté sous format DEF (« Design Exchange Format »).

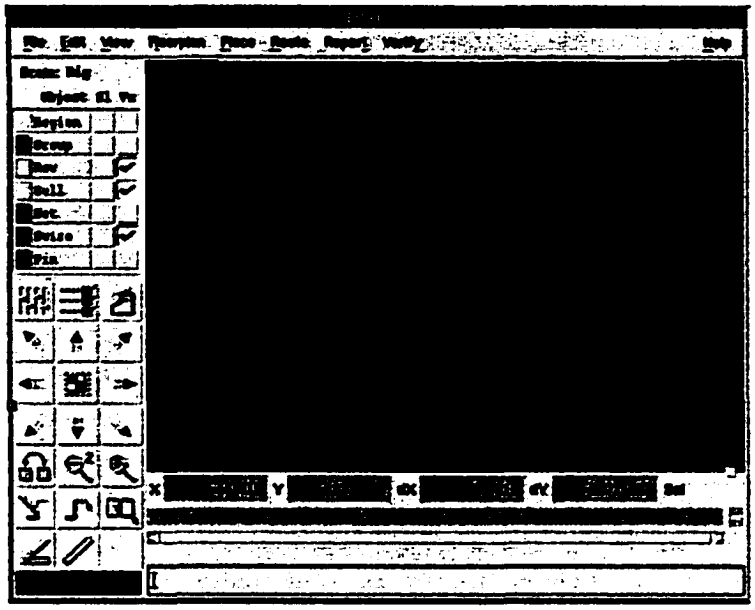

Figure 15 État initial du « floorplanning »

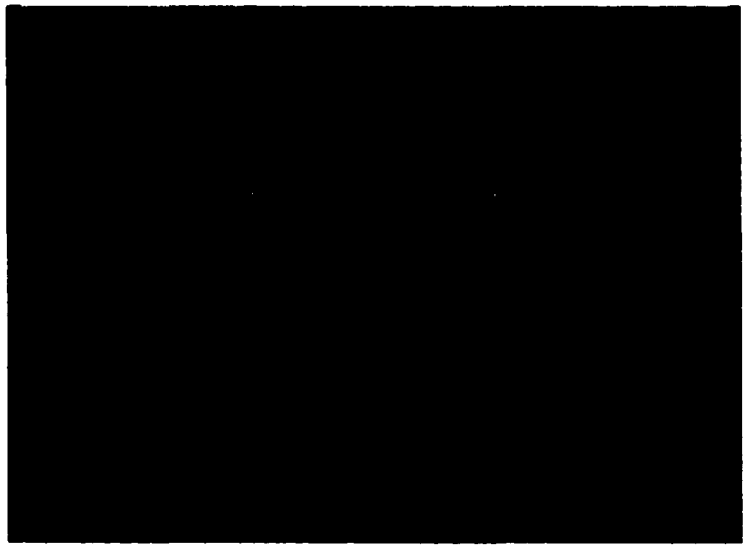

Figure 16 Résultat du placement de toutes les cellules

3.3 Transfert dans Virtuoso et dessin de masques (« layout »)

3.3.1 Visualisation et édition du dessin de masques

Le dessin de masques définit la position et les interconnexions de différentes couches de semi-conducteurs, POLYs, métaux, les VIAs et CONTACTs utilisés. Pour compléter la conception du circuit, nous devons utiliser un environnement de visualisation et d'édition du dessin des masques, supportant les opérations de vérification du respect des règles de dessin (ex. Dimensions et distances minimales) et de la cohérence avec le « netlist ». Ces vérifications doivent en effet être effectuées avant d'envoyer le circuit à la fabrication. Ces vérifications sont d'autant plus nécessaires dans notre cas puisque des interventions manuelles (édition) seront effectuées.

3.3.2 Virtuoso

L'éditeur de dessin de masques Virtuoso fournit un environnement d'édition de dessin de masque et un environnement de vérification qui supporte toutes les techniques de design des circuits intégrés. Il peut optimiser les espaces entre les différentes traces du dessin. Il possède une interface graphique très conviviale et facile à manipuler. De plus, il est supporté par de très puissants logiciels de vérification, comme DIVA, DRACULA, etc.

3.3.3 Synthèse du dessin des masques

L'entrée pour Virtuoso sera le fichier exporté de Silicon-Ensemble, un fichier en format DEF. Comme dans le cas des autres logiciels, des bibliothèques et des fichiers de configuration sont nécessaires : **wcells_dsm** et **pads_dsm.**Lorsque l'importation est complétée avec succès (DEFin : completed), le résultat est un « layout » dont les vues sont abstraites. La synthèse de dessin de masques (ne pas confondre avec celle effectuée à partir de la description VHDL, chapitre3) est le passage de la vue

abstraite à la vue « layout ». Tel que mentionné précédemment, deux vérifications seront faites. La première est une vérification des règles de dessin (DRC : Design Rules Check) et la deuxième est une comparaison entre la vue schématique et celle du dessin de masques (LVS : Layout Versus Schematic).

3.4 Vérification des règles de dessin (niveau « layout »)

Avant de procéder à la vérification proprement dite, nous devons, si nécessaire, ajouter les traces de métal manquantes. Dans notre cas, nous avons ajouté une trace de métal reliant la broche « VDDCORE » (VDD_{global}) à l'anneau de métal qui correspond au « VDDRING », ainsi que les traces nécessaires pour les broches d'alimentations spéciales (VDD_2, VDD_3, VDD_4, VDD_5, VDD_6, VDD_7, VSS_2, VSS_3, VSS_4, VSS_5, VSS_6 et VSS_7). Comme toutes les cellules sont dans leur vue « abstract », nous devons les remplacer par leur vue « layout » pour pouvoir modifier le dessin de masques.

Une fois cette opération achevée avec succès, nous pouvons voir le dessin des masques de chacune des cellules de la bibliothèque de la technologie wcells, CMOS 0.35 μm (figure 17). Par la suite, nous procédons à la vérification des règles de dessin de masques (DRC), pour laquelle il existe certaines contraintes. Premièrement, l'outil de vérification de dessin de masques n'effectue pas la vérification en temps réel. Deuxièmement, les stations de travail auxquelles nous avons accès ne possèdent pas les capacités requises pour effectuer une vérification globale. Un des dangers est que l'exécution du programme soit stoppée et que nous perdions nos données. Fort heureusement, il est possible de faire une vérification DRC partielle (figure 18), ce qui nous permet d'éviter la perte de temps et d'effectuer la vérification seulement sur la partie modifiée du design. Nous pouvons aussi nous servir du service de vérification de DRC de la Société Canadienne de Microélectronique pour effectuer une vérification globale de tout le circuit. Leurs outils sont plus à jour, plus sophistiqués et dotés d'un support professionnel.

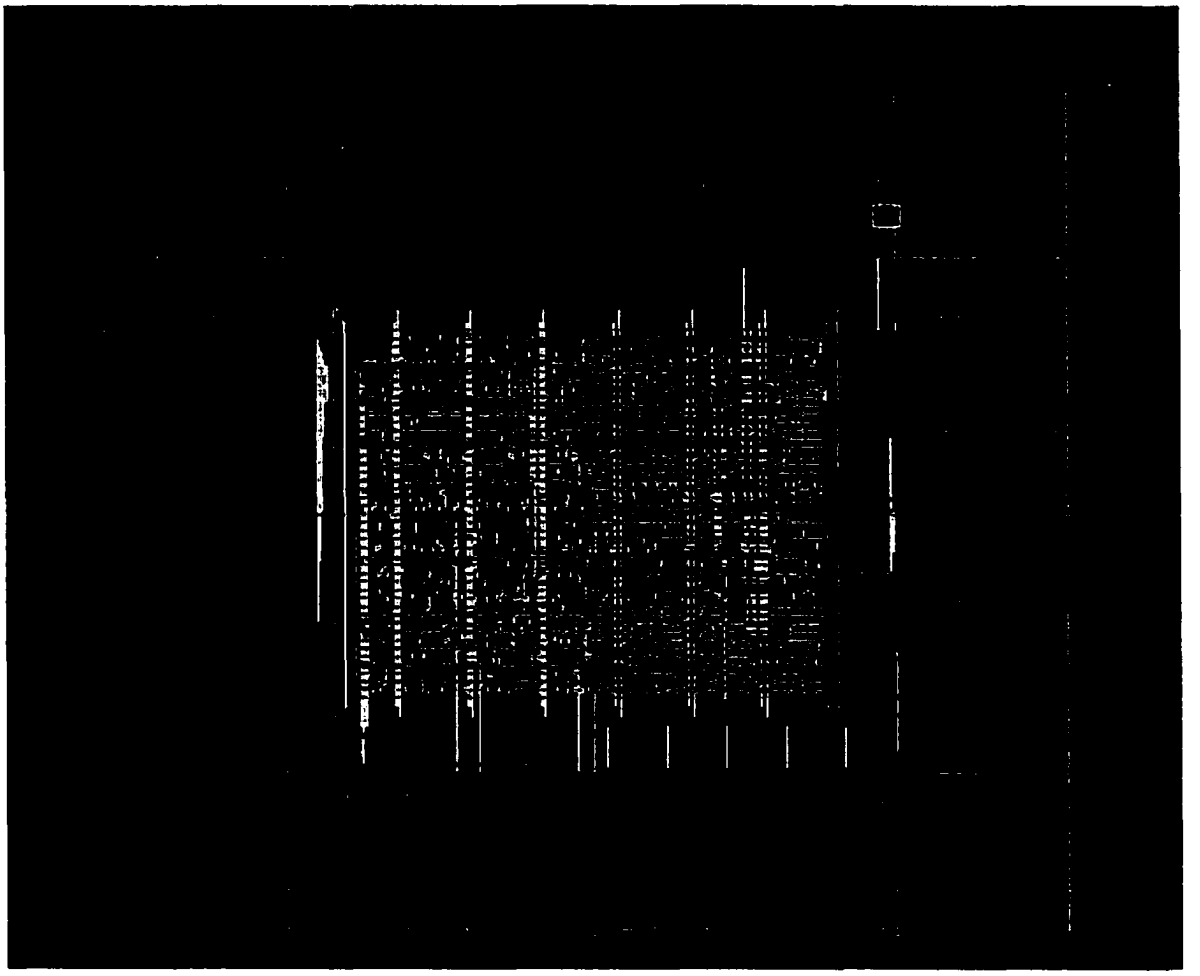

Figure 17 Vue « layout » du circuit

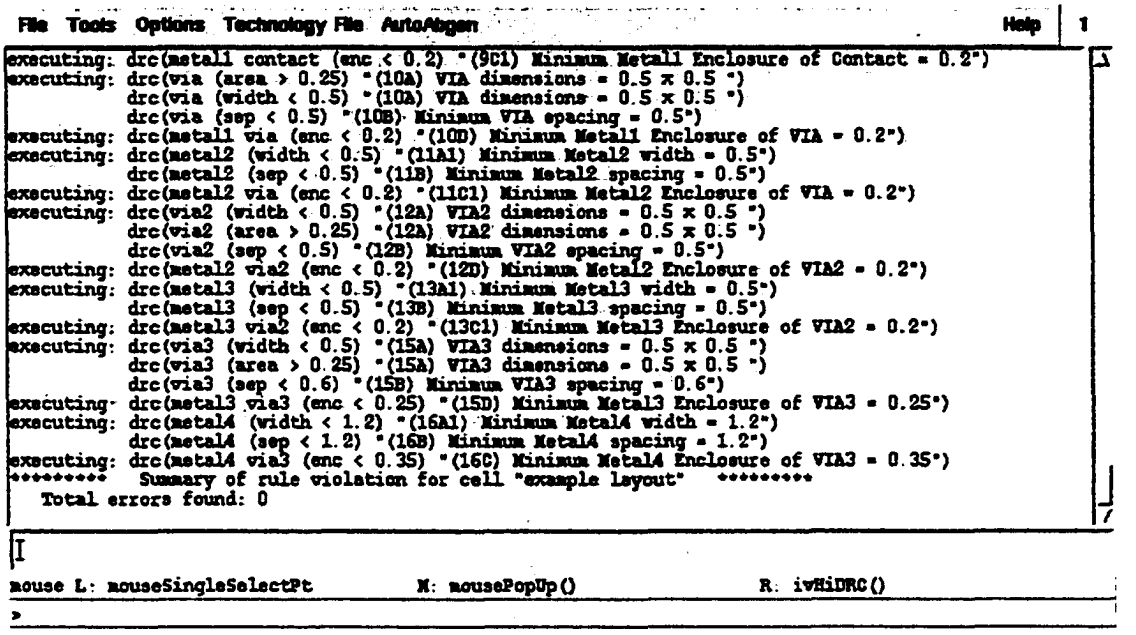

Figure 18 Exemple d'une vérification partielle de règles de dessin

3.5 Comparaison entre schématique et dessin des masques (LVS)

Dans le but de procéder à une comparaison entre le circuit au niveau schématique et le « layout » produit, nous allons utiliser la procédure « LVS ». Cette vérification est nécessaire pour éviter les mauvaises surprises. Il s'agit des dernières manipulations avant de procéder à la fabrication. Cette étape vérifie si le dessin de masques généré correspond bien à notre circuit d'un point de vue électrique. Avant de procéder à cette vérification, deux étapes sont nécessaires :

1. Extraire les composantes du layout. Le résultat de cette opération est la vue EXTRACTED (figure 19).

2. Préparer une vue schématique à partir du « netlist » fourni par **Design_Compiler** de Synopsys (figure 20).

Figure 19 Vue « EXTRACTED » du Circuit

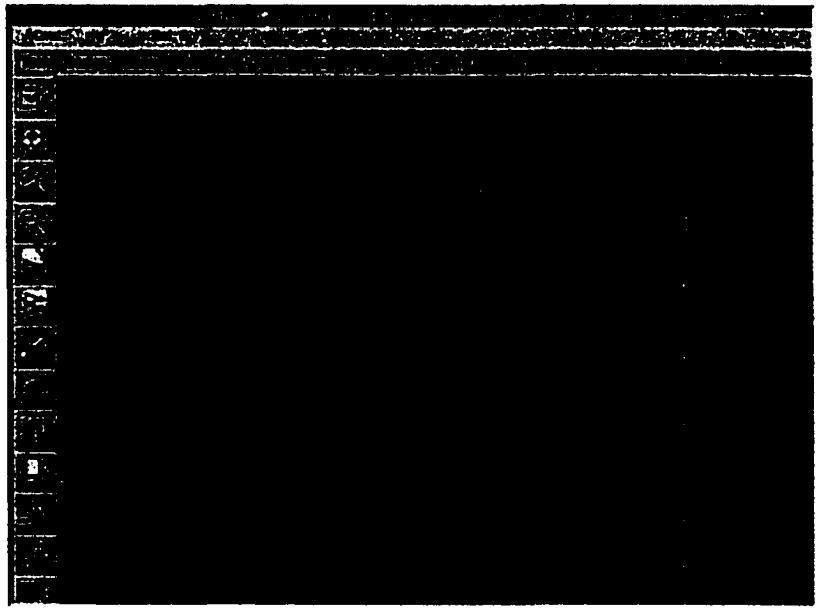

Figure 20 Vue partielle du schématique du circuit

Par la suite, nous pouvons exécuter la procédure « LVS » qui procède à la comparaison entre les deux vues (schématique et extraite) et produira un rapport dans lequel nous pouvons trouver la liste des nœuds pour les deux vues.

Lorsque l'outil arrive à faire la comparaison, indépendamment que les deux listes soient identiques ou non, l'outil donne le message que la comparaison est réussie (« Comparaison program completed successfully »).

3.6 Soumission du design à la fabrication(stream)

La procédure que nous allons décrire ci-après n'est pas générale pour tous les fabricants de circuits intégrés. Il s'agit d'une procédure propre à la Société Canadienne de Microélectronique.

Lorsque le design passe avec succès les deux étapes de DRC et LVS, il ne reste plus qu'à insérer le LOGO et à transformer le circuit en format « stream » (GDSII).

Dans le cas présent, le prototype sera fabriqué via les facilités offertes par la Société Canadienne de Microélectronique. Pour cette raison, la soumission pour la fabrication se fait, généralement en deux étapes (ou plus) :

1. Vérification préliminaire du DRC : Cette étape consiste à faire une vérification globale du circuit et déterminer le pourcentage des différents éléments présents (figure 21). Chaque élément mentionné sur la figure 21 représente une couche dans le circuit ; c'est à dire, un masque sera réservé pour chacun des éléments présents. Cette étape permet de détecter les erreurs DRC que nous ne pouvons pas détecter avec nos propres outils. Elle devrait être répétée jusqu'à l'obtention d'un rapport indiquant l'absence des erreurs non acceptables. En effet selon la technologie utilisée, il y a certaines erreurs peuvent être ignorées. Comme nous le montre la figure 22, il y a deux violations de règle de dessin (ODC163 et ODC263) qui sont toujours présentes, mais elles sont acceptées dans le cas de technologie CMOS 0.35 µm.

2. Deuxième vérification du DRC : Cette vérification est effectuée après la correction des erreurs de DRC non acceptables pour la technologie 0.35 µm et l'ajout des quantités manquantes des différents éléments présents dans le circuit.

Des pourcentages minimum de chacune des quantités d'éléments présents dans le design sont exigés (figure 23). Par exemple, dans le cas de la TSMC (la compagnie qui a fabriqué notre circuit) un minimum de 30% pour chacune des couches de métal et 14% pour chacune de couches de poly sont nécessaires pour la technologie 0.35 µm. Cette exigence vient du fait que les manufacturiers utilisent ce qu'ils appellent CMP (Chemical-Mechanical Polishing) pour avoir des surfaces parfaitement planes. Cette étape permet de générer les rapports finaux nécessaires pour le service de la fabrication pour décider d'envoyer ou non le circuit à la fabrication.

Enfin si le circuit répond à tous les critères, il sera envoyé pour fabrication et un échéancier sera communiqué aux usagés de ce service à travers le site Internet de la SCM, ce qui fut fait dans le cas de notre prototype.

layers present in ICDTS012.strm which will be processed

2 nwell

7 pplus

8 nplus

11 active

13 poly1

15 contact

16 metal1

17 via12

18 metal2

19 pad 27

via23

31 metal3

59 text

Figure 21 Éléments présents dans le circuit

---------- ***OUTPUT CELL SUMMARY*** ----------

CELL-NAME LAYER # ---------- W I N D O W ----------

DATATYPE # OF POLYGONS TEXTS (LINE SEGMENTS)

ODC163 63/ 0 -1279.48 -1177.40 1278.73 1176.65 58 0

ODC263 63/ 0 -1279.48 -1177.40 1278.73 1176.65 58 0

OUTDISK PRIMARY CELL : OUTICDTS012

WINDOW : -1279.48 -1177.40 1278.73 1176.65

ENDED AT TIME =17:12:57 DATE =12-DEC-2001 1

****** *PROBLEM GEOMETRY ERROR LISTING* ******

***** *END OF PROBLEM GEOMETRY LISTING* *****

NUMBER OF ACUTE ANGLE INPUT POLYGONS = 12

Figure 22 Exemple d'erreurs acceptables dans le cas de la technologie 0.35 µm

Areas are given in square um:
metal1 area is 2008438.7 of 6257970.825 for 32.0 percent
metal2 area is 2021093.0 of 6257970.825 for 32.2 percent
metal3 area is 2189917.9 of 6257970.825 for 34.9 percent
poly1 area is 886783.6 of 6257970.825 for 14.1 percent

Figure 23 Pourcentage de chaque couche présente dans le design

3.7 Conclusion

Malgré l'amélioration remarquée des outils utilisés de placement et routage, nous avons été obligés d'intervenir manuellement pour corriger des défauts de design. En effet nous avons dû ajouter des traces de métal pour connecter la patte VDD_{global} à l'anneau VDDRING, même si cette opération est effectuée par le flot standard du design.

Les modifications requises dans le cadre de notre prototype, par rapport au flot standard, sont minimes d'un point de vue complexité, c'est à dire, qu'elles peuvent être intégrées facilement dans le flot standard. Les outils sont capables d'effectuer des opérations complexes. Il est donc permis de penser que les interventions manuelles pourraient être automatisées et intégrées au flot standard avec un simple changement de configuration, ou avec l'ajout d'une entrée pour pouvoir mettre des contraintes sur la façon dont le logiciel manipule les objets.

CHAPITRE 4

TESTS ET RÉSULTATS

4.1 Introduction

Le but ultime de ce travail est de réaliser un prototype testable, qui intègre la logique nécessaire pour mettre en application la nouvelle méthode de test. Une fois fabriqué, le circuit devrait être validé. Pour ce faire, un environnement de test a été développé. Dans ce chapitre, nous allons montrer que les circuits sont fonctionnels et que les résultats obtenus coïncident avec ceux de la simulation. Deux types de tests s'imposent. Le premier représente un ensemble de tests fonctionnels et le deuxième représente les tests de type DFT (« Design For Testability ») suivant la nouvelle méthode.

4.2 Environnement de test

Traditionnellement, l'ensemble des outils utilisés pour tester un circuit numérique comporte un générateur d'échantillons (« pattern generator »,PG), un analyseur logique (« logic analyzer »,LA) et un autre outil pour analyser les résultats. Étant donné la faible complexité du circuit à tester, nous avons pu simplifier au maximum l'environnement de test, tel qu'illustré à la figure 24.

Dans notre cas, nous avons une seule entrée dynamique (horloge), et les autres étant statiques (signal de la remise à zéro : Rst1, les signaux de contrôle : Sel01, Sel11). Par conséquent, le générateur du signal d'horloge peut être un simple générateur de fréquence. En ce qui concerne les signaux statiques, des sources de tensions continues seront suffisantes pour accomplir cette tâche. Le voltage pour les broches de VDD est 3.3 V, celui pour les broches de VSS est 0 V.

Un oscilloscope servira à visualiser le comportement temporel des différents signaux et prendre les mesures nécessaires. Pour transformer le courant en tension, nous

avons placé des résistances de 100 Ω ou 1kΩ entres les broches d'alimentation des senseurs et les sources d'alimentation (Figure 25). Due à certaines difficultés techniques, nous n'avons pas utilisé le testeur IMS, tel qu'initialement prévu.

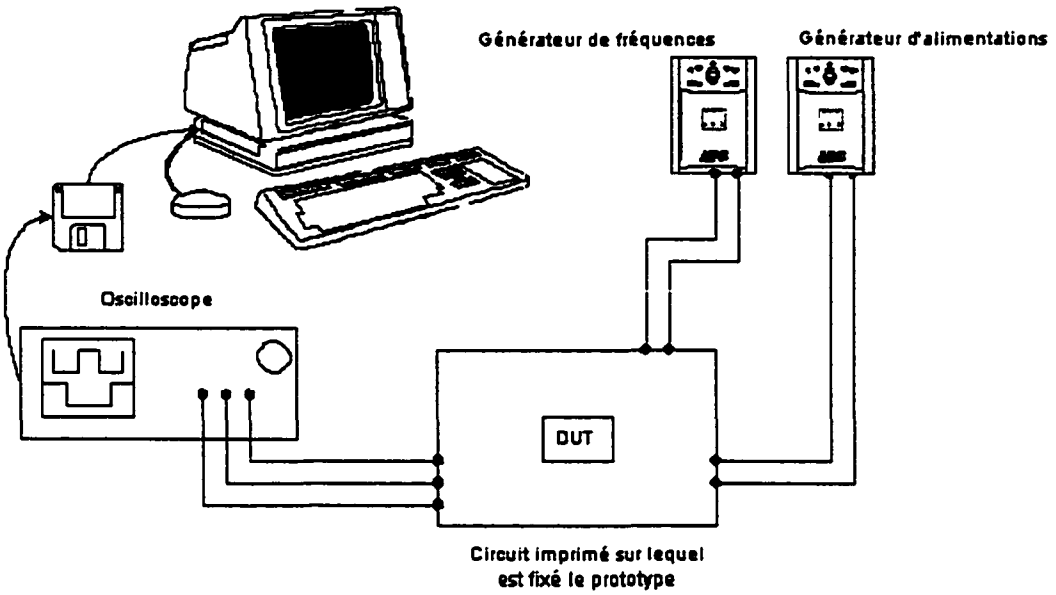

Figure 24 Environnement de test pour valider le circuit

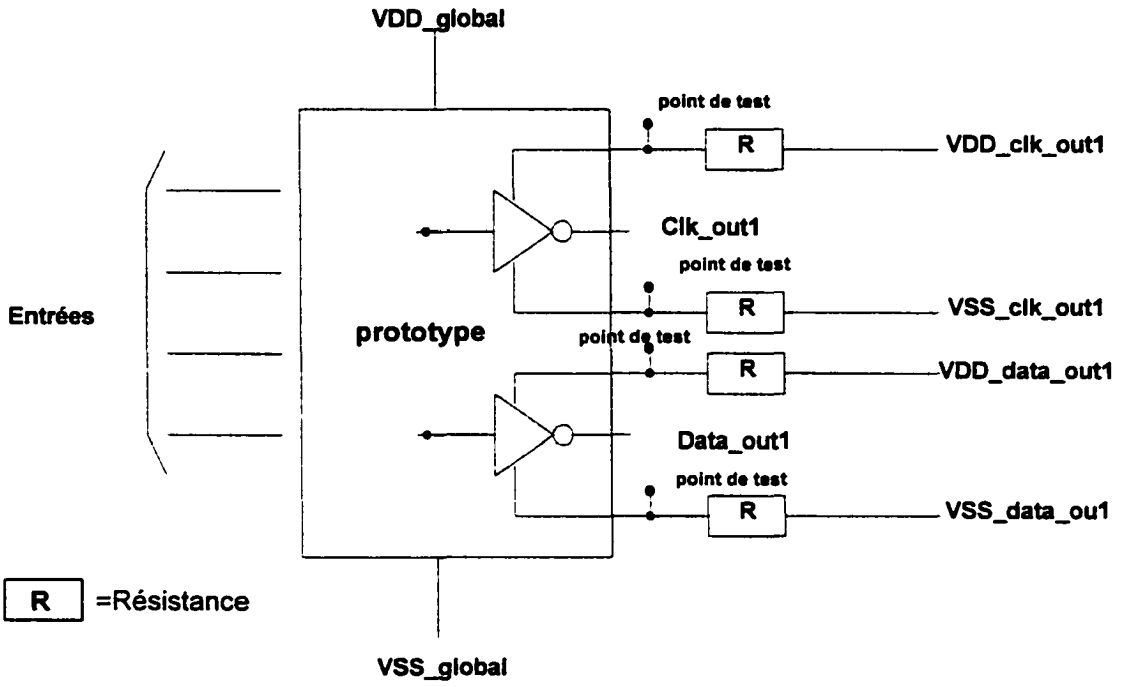

Figure 25 Module avec le circuit de conversion *courant-tension*

4.3 Tests fonctionnels

Ces tests consistent à vérifier la fonctionnalité du circuit. En d'autres termes, nous vérifions si les sorties de données correspondent aux résultats attendus. Pour exécuter cette tâche, et puisque le prototype contient trois modules identiques et indépendants, il suffit de stimuler un seul module au départ et par la suite répéter cet exercice pour les deux qui restent. Les vecteurs de test générés pour cette fin devraient exercer toutes les entrées du circuit avec toutes les combinaisons possibles. Cette opération, qui est habituellement complexe ou même impossible, est faisable dans notre cas étant donné la faible complexité du circuit à tester. Ces tests fonctionnels consistent à comparer les résultats de la simulation (niveau porte logiques avec délais) à ceux

obtenues avec l'environnement de test. Il faut noter que les délais simulés sont estimés depuis une technologie légèrement différente de celle utilisée par le circuit, puisqu'il n'était pas possible d'avoir accès aux informations pertinentes pour le procédé CMOSP35. Le tableau I résume les vecteurs utilisés. Chacun des vecteurs est répété sur plusieurs périodes d'horloge. Tous les cas, sauf celui où le signal Rst1=1, sont illustrés dans les figures suivantes.

Tableau I

Vecteurs utilisés pour les tests fonctionnels

Définitions des vecteurs de test			
Entrées			
Rst1	Clk1	Sel01	Sel11
1	X	X	X
0	actif	0	0
0	actif	0	1
0	actif	1	0
0	actif	1	1

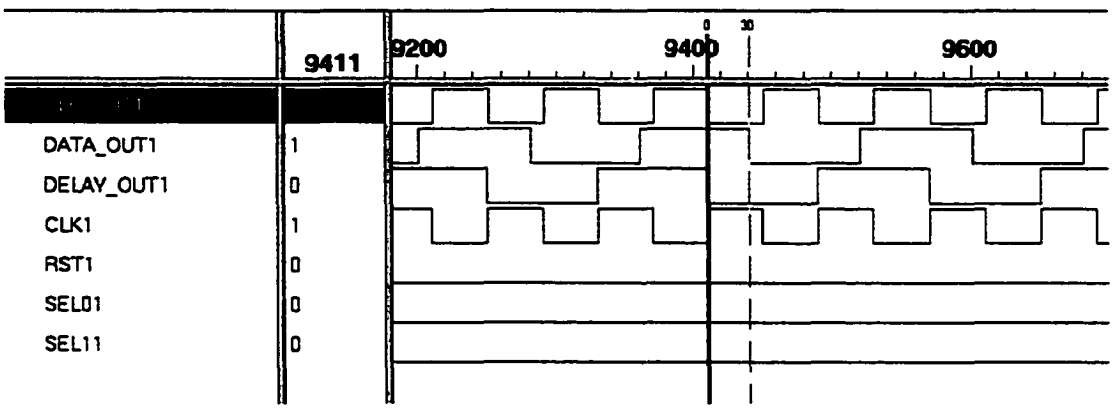

Figure 26 Simulation au niveau porte logique avec délais

Sel01 = 0 Sel11 = 0

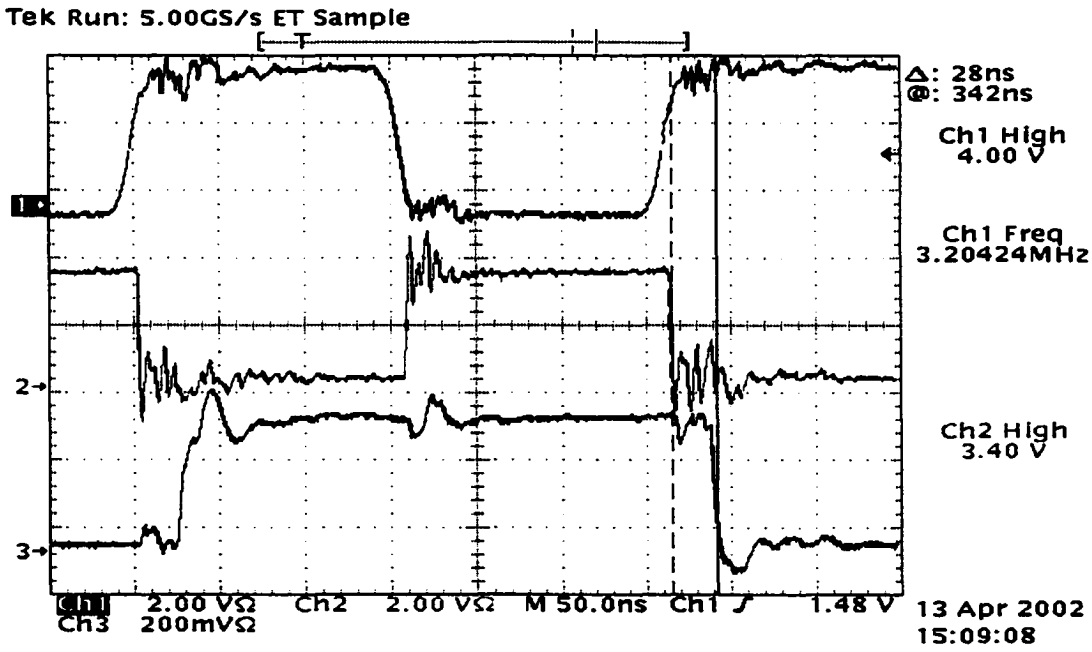

Figure 27 Délai entre Clk_out1 et Data_out1 avec Sel01 = 0 Sel11 = 0
Les signaux sont dans l'ordre : Clk1, Clk_out1 et Data_out1

Les figures 26 et 27 illustrent le cas du vecteur 01 répété sur plusieurs périodes d'horloge. La figure 26 montre les résultats de simulation. Ces résultats montrent que le signal Clk_out1 est inversé par rapport à celui du Clk1 (l'entrée de l'horloge), et

que le signal data_out1 a une période égale à la moitié de celle de l'horloge d'entrée. De plus, le délai entre Clk_out1 (descente) et Data_out1 (descente) est estimé à 30 ns. Le délai est similaire à celui mesuré (Figure 27), qui est de 28 ns, ce qui correspond à celui de la chaîne des inverseurs la plus longue. La figure 27 permet également de vérifier le comportement logique du circuit pour ce vecteur.

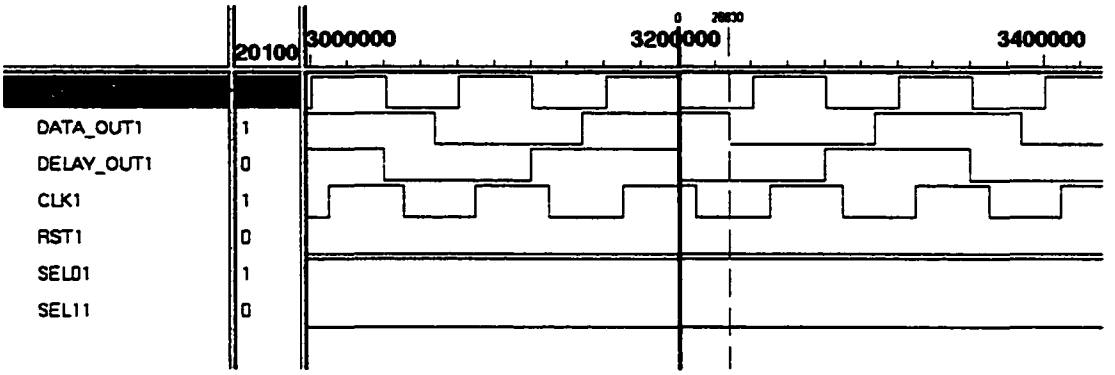

Figure 28 Simulation au niveau porte logique avec Sel01 = 1 et Sel11 = 0

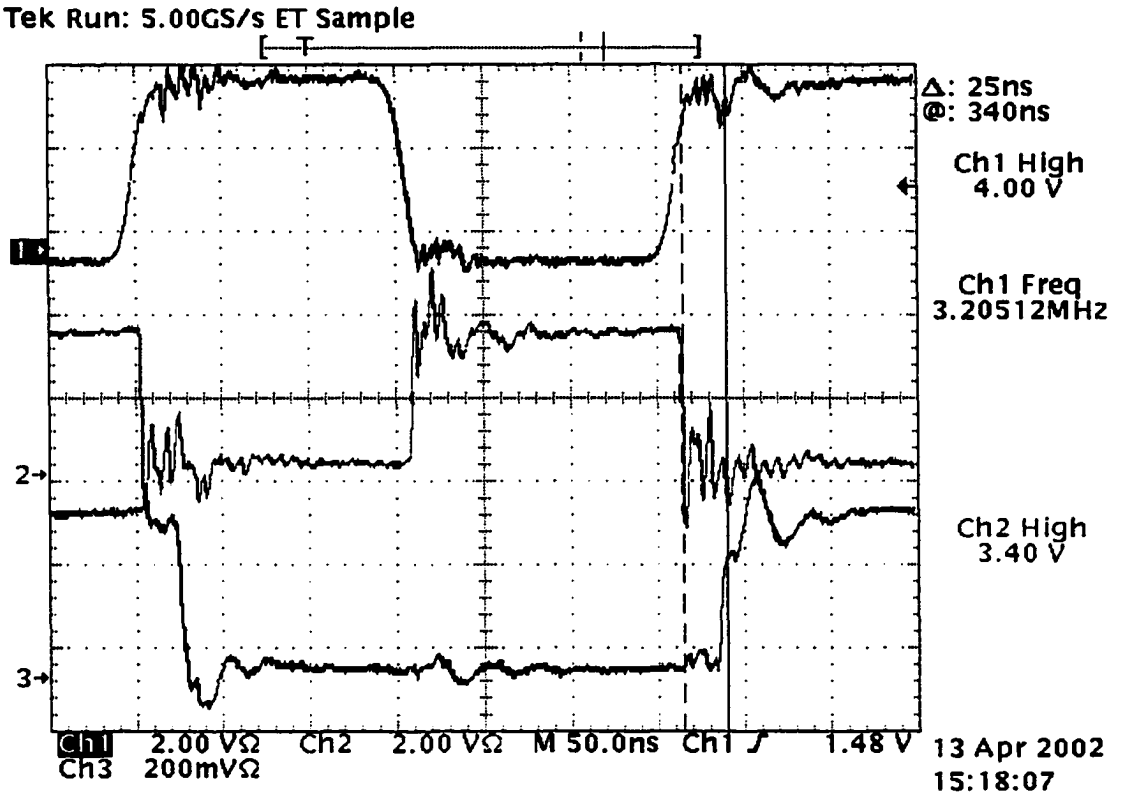

Figure 29 Délai entre Clk_out1 et Data_out1 avec Sel01 = 1 Sel11 = 0
Les signaux sont dans l'ordre : Clk1, Clk_out1 et Data_out1

Les figure 28 et 29 illustrent le cas du vecteur 2 (répété). Les signaux Clk_out1 et Data_out1 ont ici aussi le comportement prévu. Le délai simulé de 26.83 ns (niveau porte logique) et celui de 25 ns (mesuré au laboratoire) correspondent à la deuxième plus longue chaîne des inverseurs.

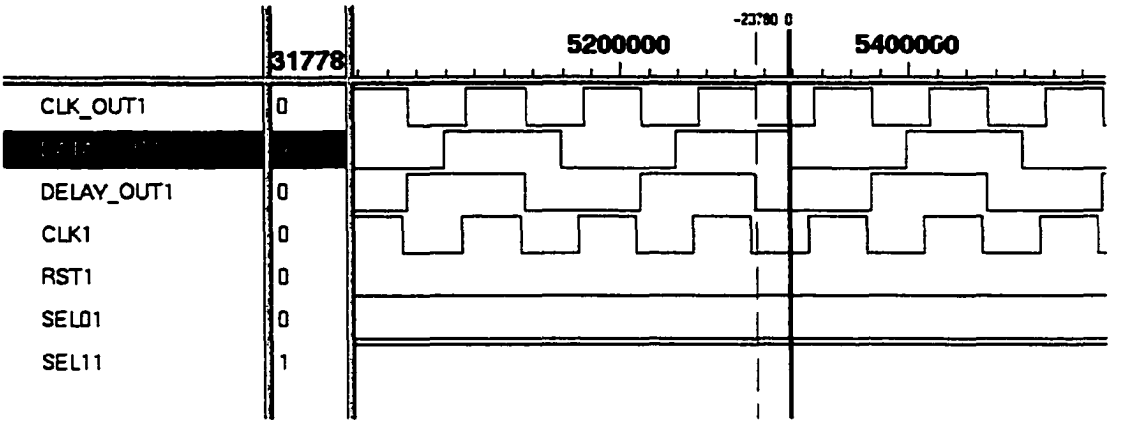

Figure 30 Simulation au niveau porte logique avec Sel01= 0 et Sel11=1

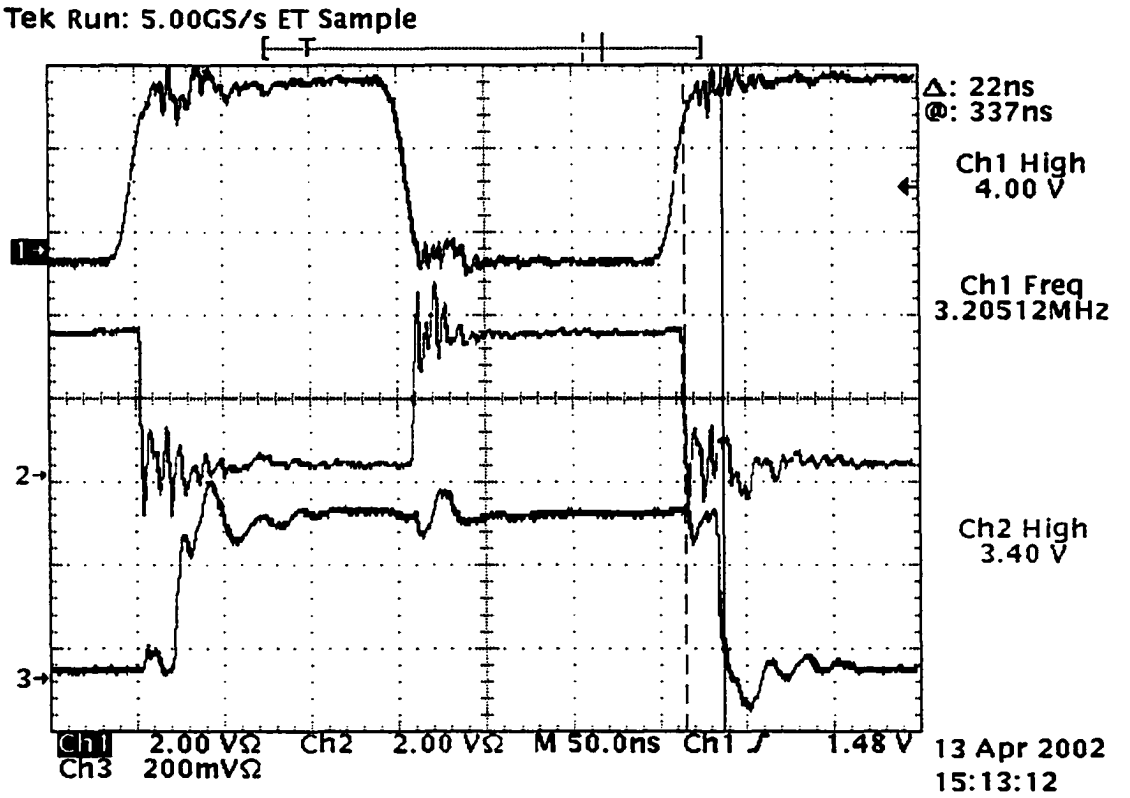

Figure 31 Délai entre Clk_out1 et Data_out1 avec Sel01 = 0 Sel11 = 1
Les signaux sont dans l'ordre : Clk1, Clk_out1 et Data_out1

Les figures 30 et 31 illustrent le cas du vecteur 3 (répété) et montrent que les signaux Clk_out1 et Data_out1 ont toujours le même comportement désiré. Les délais entre Clk_out1 et Data_out1 de 23.78 ns (simulation niveau portes logiques) et 22 ns (mesuré au laboratoire) correspondent à la chaîne des inverseurs la troisième plus longue.

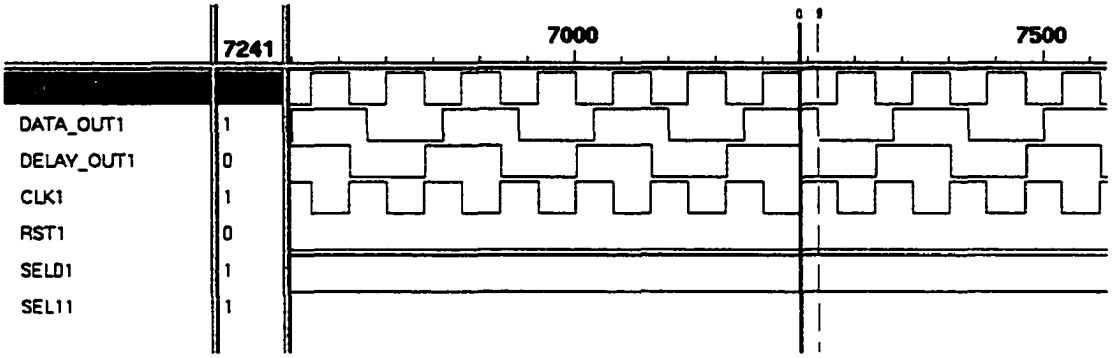

Figure 32 Simulation au niveau portes logiques avec Sel01=1 et Sel 11=1

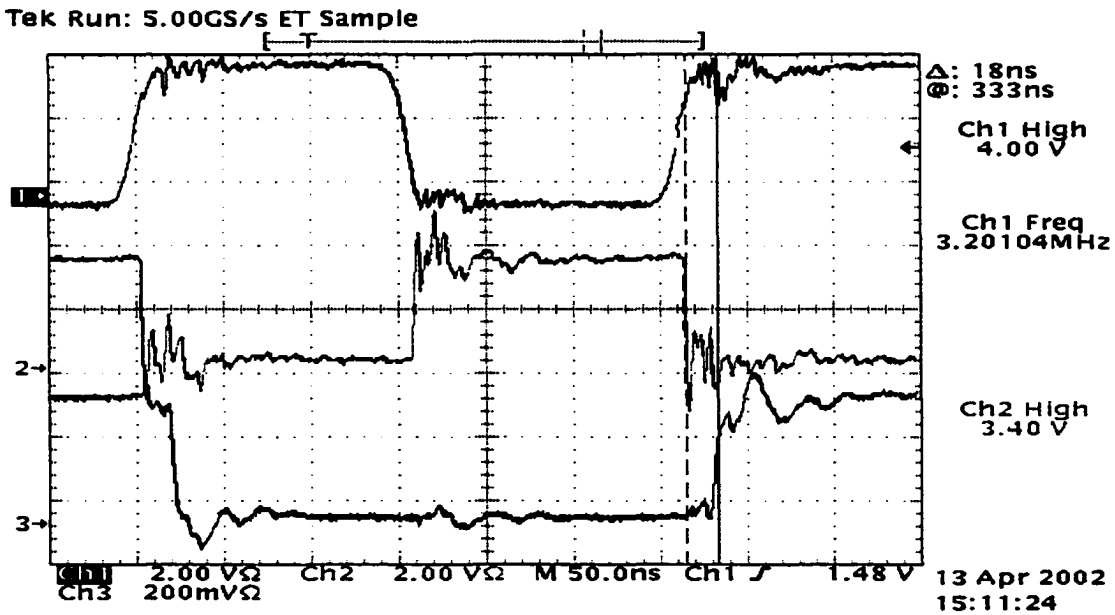

Figure 33 Délai entre Clk_out1 et Data_out1 avec Sel00 = 1 Sel11 = 1
Les signaux sont dans l'ordre : Clk1, Clk_out1 et Data_out1

Finalement, Les figures 32 et 33 illustrent le dernier cas (vecteur 4 répété). Une fois de plus, les signaux du Clk_out1 et Data_out1 sont corrects. Les délais entre Clk_out1 et Data_out1 de 19 ns (simulation niveau porte logique) et de 18 ns (mesuré au laboratoire) correspondent à celui de la chaîne des inverseurs la plus courte.

Tableau II

Résultats de simulation au niveau portes logiques et délais réels

Sommaire des délais calculés au niveau porte logique et de ceux mesurés au laboratoire				
Configuration	**Niveau porte logique**		**Mesures au laboratoire**	
Sel1-Sel0	Délai (ns)	Figure	Délai (ns)	Figure
00	30	Figure 26	28	Figure 27
01	26.83	Figure 28	25	Figure 29
10	23.78	Figure 30	22	Figure 31
11	19	Figure 32	18	Figure 33

Le tableau II fait un sommaire des résultats des tests fonctionnels. On remarque que les délais simulés au niveau porte logique sont pessimistes par rapport au délais mesurés. De plus il existe une claire corrélation entre les délais simulés et ceux mesurés. Les délais mesurés serons utilisés plus loin comme référence pour estimer la précision de la méthode proposée.

4.4 Tests de DFT

Ces tests consistent à utiliser la logique ajoutée pour la validation de la méthode de test (chaînes parallèles de courant). La stratégie utilisée est celle présentée à la figure 25; nous avons inséré des résistances entre les points de test et les sources de tension pour transformer le courant en tension. Au cours de ces tests, nous allons au départ estimer le délai entre le front montant de Clk_out1 et le début de la validité du signal Data_out1 à partir des signaux d'alimentation des senseurs, afin d'établir la corrélation entre ce délai et celui mesuré entre Clk_out1 et Data_out1. Originalement, nous avions prévu utiliser le testeur IMS pour mesurer la vitesse maximale du circuit, en augmentant graduellement la fréquence d'opération jusqu'à ce que la sortie Data_out1 soit erronée. Cette mesure devait être notre référence ultime. Malheureusement, le testeur était non opérationnel au moment de prendre ces mesures. C'est la raison pour laquelle nous devions utiliser le délai mesuré entre Clk_out1 et Data_out1 comme référence. Par la suite, nous allons estimer les différentes marges de bruit sur les signaux d'alimentation des senseurs, dans notre quête d'estimation des délais.

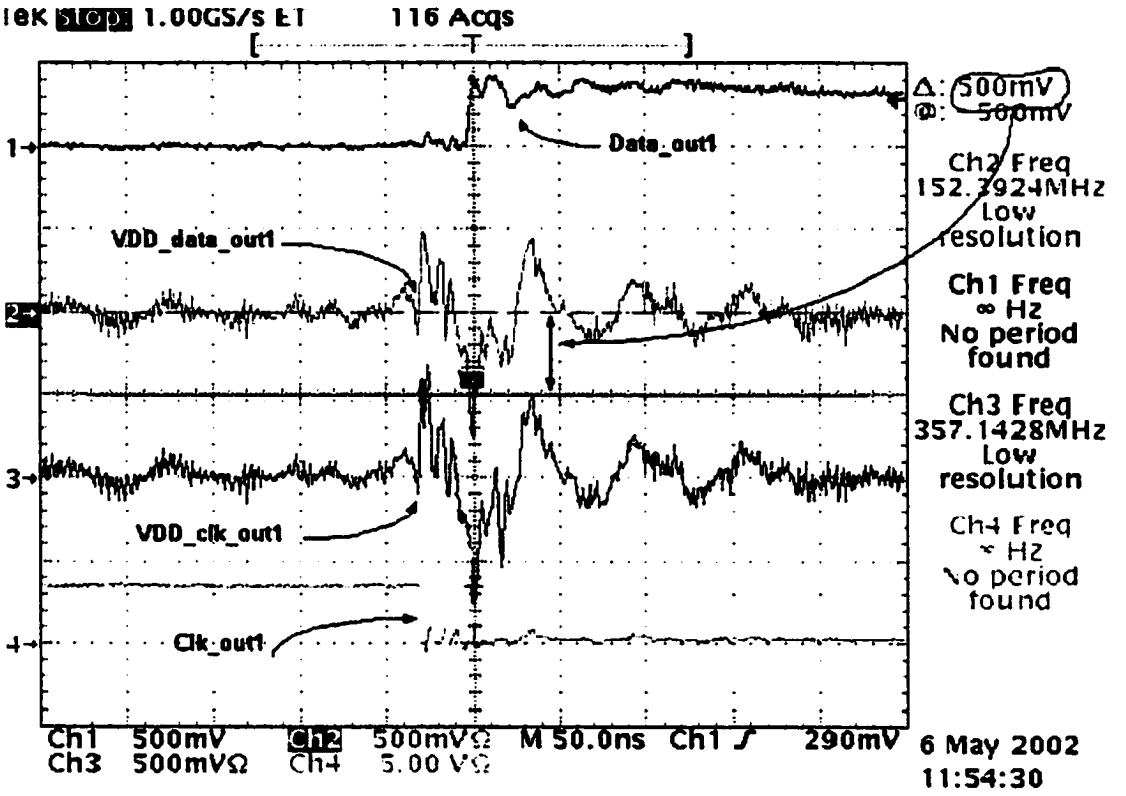

Figure 34 Seuil limite utilisé pour la détection du signal VDD_{data}

La figure 34 nous montre le seuil limite considéré pour la détection du signal VDD_{data}. Ce seuil est négatif et sa valeur absolue est fixée à 500 mV. Le signal VDD-data_out1 doit franchir le seuil (i.e. sa valeur doit être inférieur à 500 mV) pour considérer le signal VDD_{data} valide. Tous les sommets possédant une amplitude supérieur à cette valeur sont considérés comme étant du bruit.

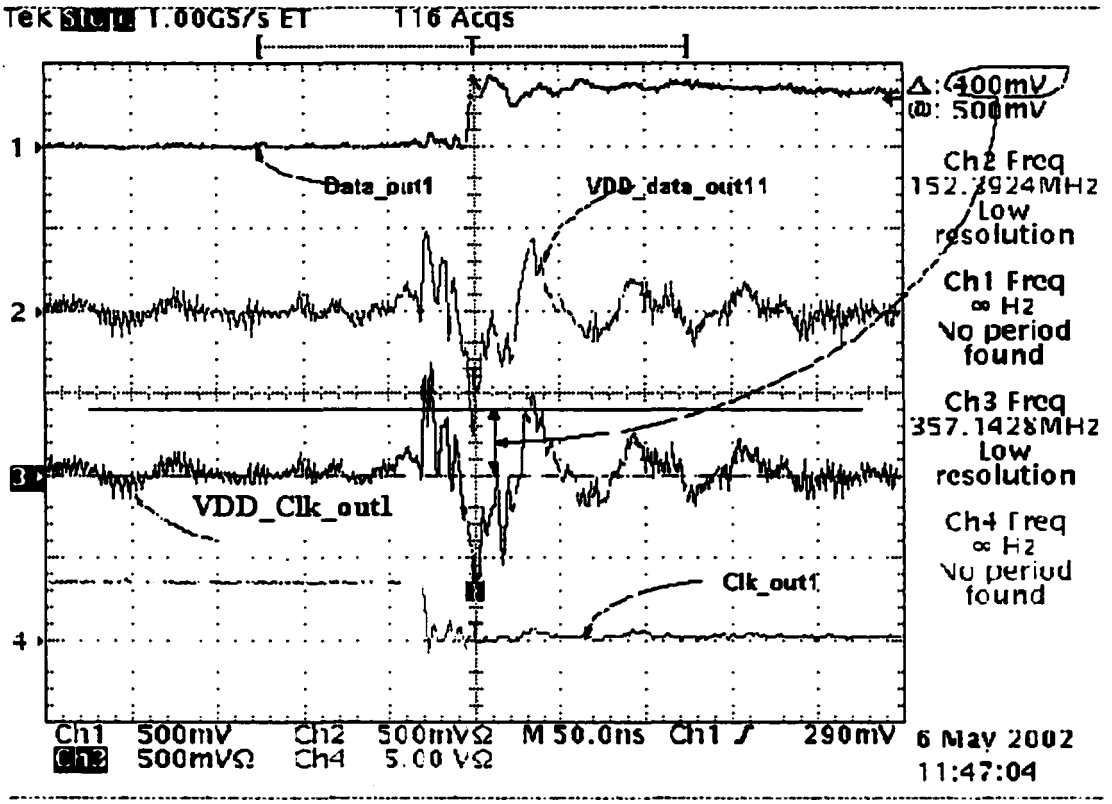

Figure 35 Seuil limite utilisé pour la détection du signal VDD_{clk}

La figure 35 nous montre le seuil limite considéré pour la détection du signal VDD_{clk}. Dans ce cas, le seuil est positif et sa valeur est fixée à 400 mV. Le signal VDD_Clk_out1 doit être supérieur à 400 mV pour considérer le signal VDD_{clk} valide. Tous les sommets possédant une amplitude inférieure à cette valeur sont considérés comme étant du bruit .

4.4.1 Procédure pour mesurer le délai entre VDD_{data} et VDD_{clk}

Dans le cadre de ce projet, la mesure est réalisée à partir du graphique obtenu à l'écran d'un oscilloscope. Nous parlons donc ici d'une mesure quasi idéale exemptée par exemple des délais inhérents de l'utilisation de comparateurs pour les seuils. D'après les figures 34 et 35, le délai entre VDD_{data} et VDD_{clk} débute par le passage du signalVDD_{clk} au dessus du seuil de 400 mV et se termine par le passage du signal

VDD_{data} sous le seuil de 500 mV. L'élaboration d'un circuit transformant le signal du $VDD_{senseur}$ en onde rectangulaire fait partie des travaux futurs.

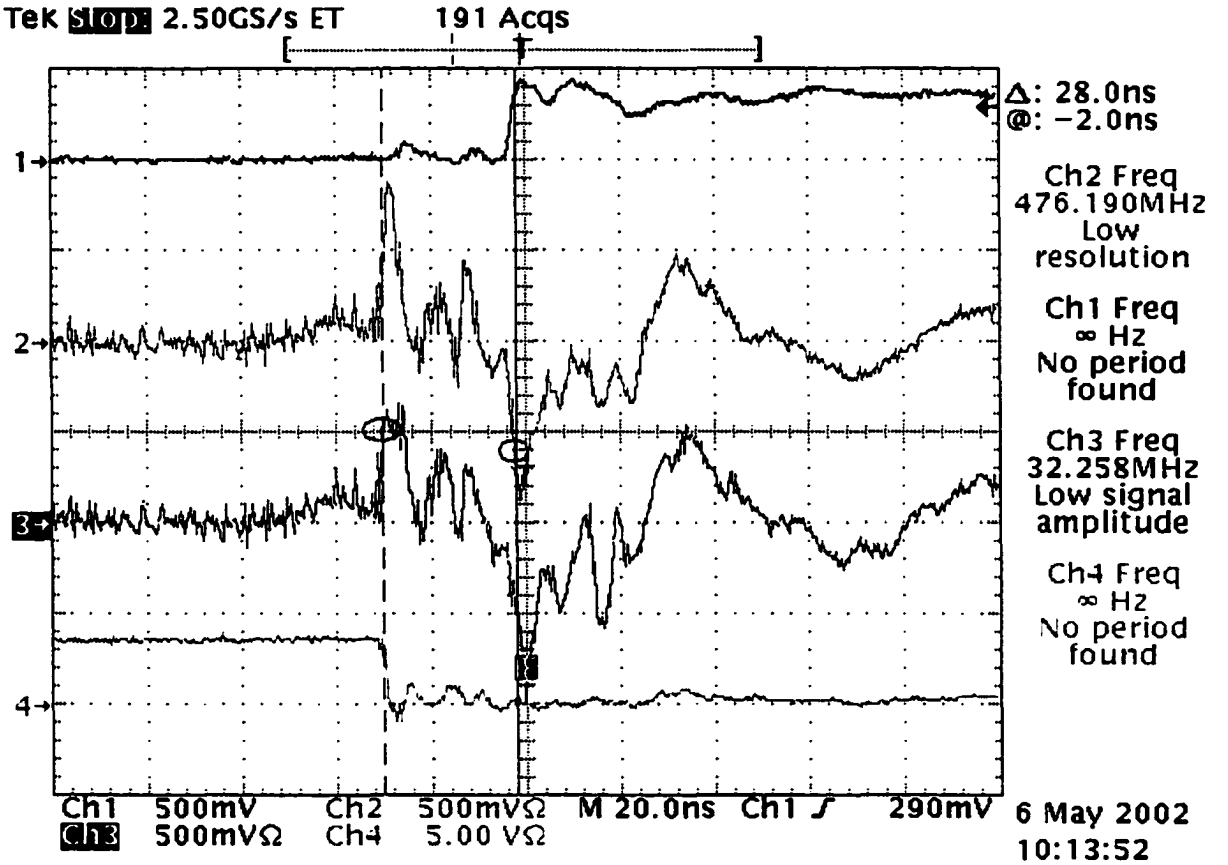

Figure 36 Mesure du délai entre VDD_{data} et VDD_{clk} (Sel01 = 0 Sel11 = 0 R = 100 Ω). Les signaux sont dans l'ordre : Data_out1, VDD_{data}, VDD_{clk} et Clk_out1

La figure 36 illustre le cas du vecteur 1 (répété). Le délai mesuré entre les points de test VDD_{data} et VDD_{clk} est de 28 ns, ce qui correspond à celui mesuré entre les deux broches de sortie Data_out1 et Clk_out1(~28 ns, Figure 27).

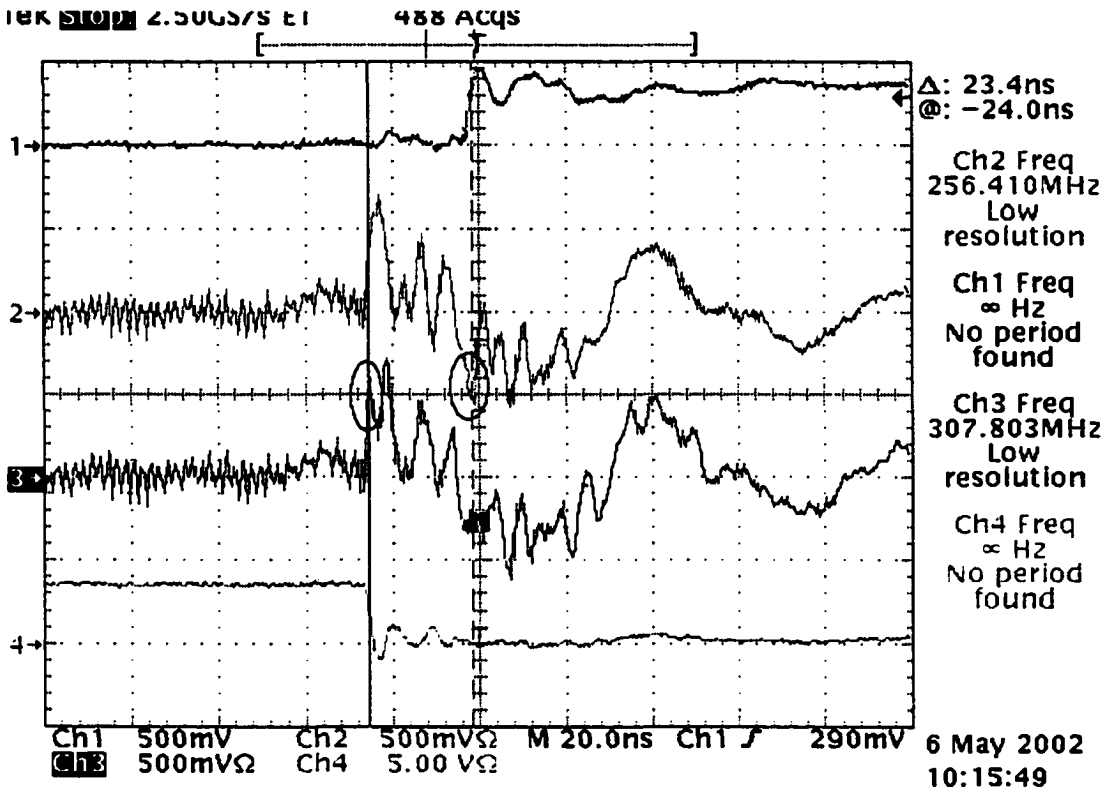

Figure 37 Mesure du délai entre VDD_{data} et VDD_{clk}
(Sel01 = 1 Sel11 = 0 R = 100 Ω).
Les signaux sont dans l'ordre : Data_out1, VDD_{data}, VDD_{clk} et Clk_out1

La figure 37 illustre le cas du vecteur 2 (répété). Le délai mesuré entre les points de test VDD_{data} et VDD_{clk} (~23.52 ns) est à peu près égal à celui mesuré entre les deux broches de sortie Data_out1 et Clk_out1(~24 ns, Figure 29).

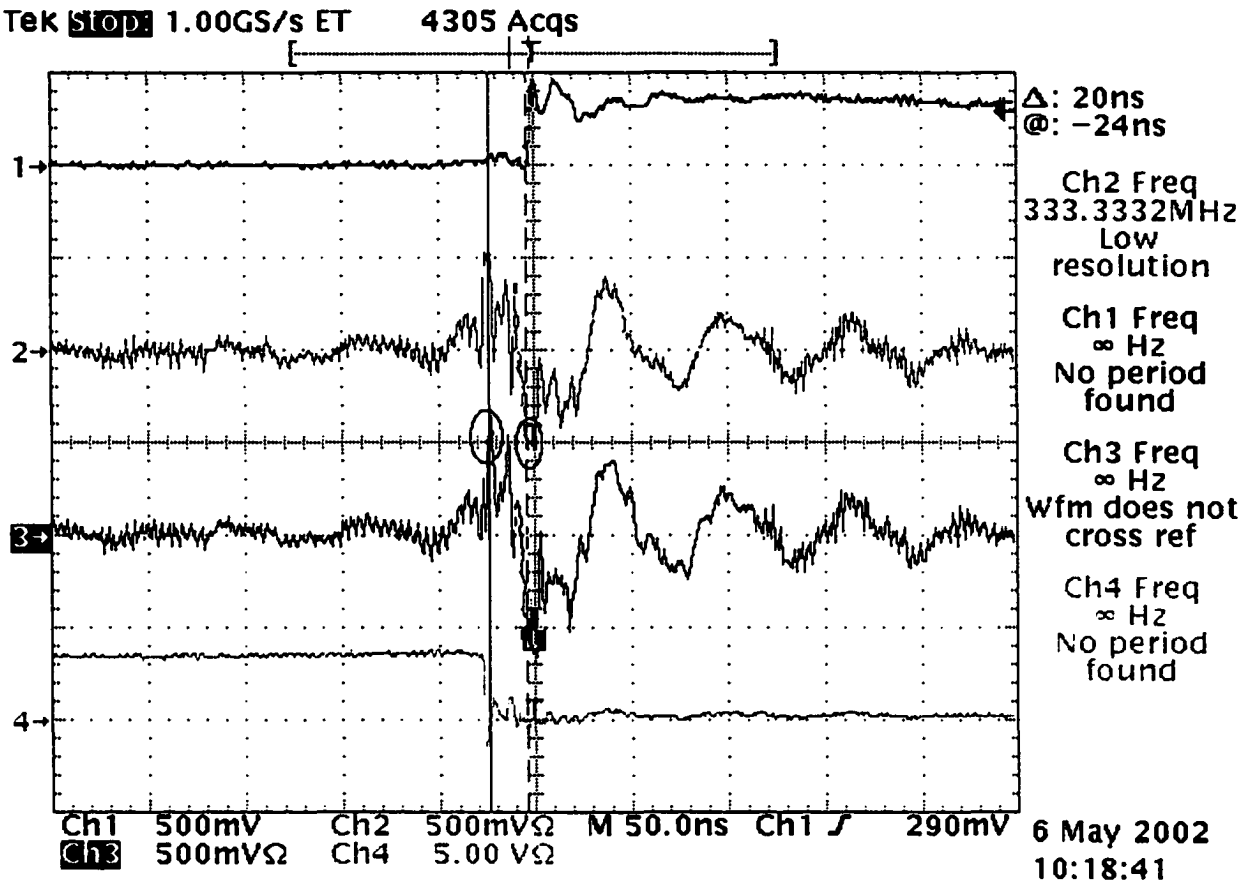

Figure 38 Mesure du délai entre VDD_{data} et VDD_{clk}
(Sel01 = 0 Sel11 = 1 R = 100 Ω).
Les signaux sont dans l'ordre : Data_out1, VDD_{data}, VDD_{clk} et Clk_out1

La figure 38 illustre le cas du vecteur 3 (répété). Le délai mesuré entre les points de test VDD_{data} et VDD_{clk} (~20 ns) est similaire à celui mesuré entre les deux broches de sortie Data_out1 et Clk_out1(~18 ns, Figure 31).

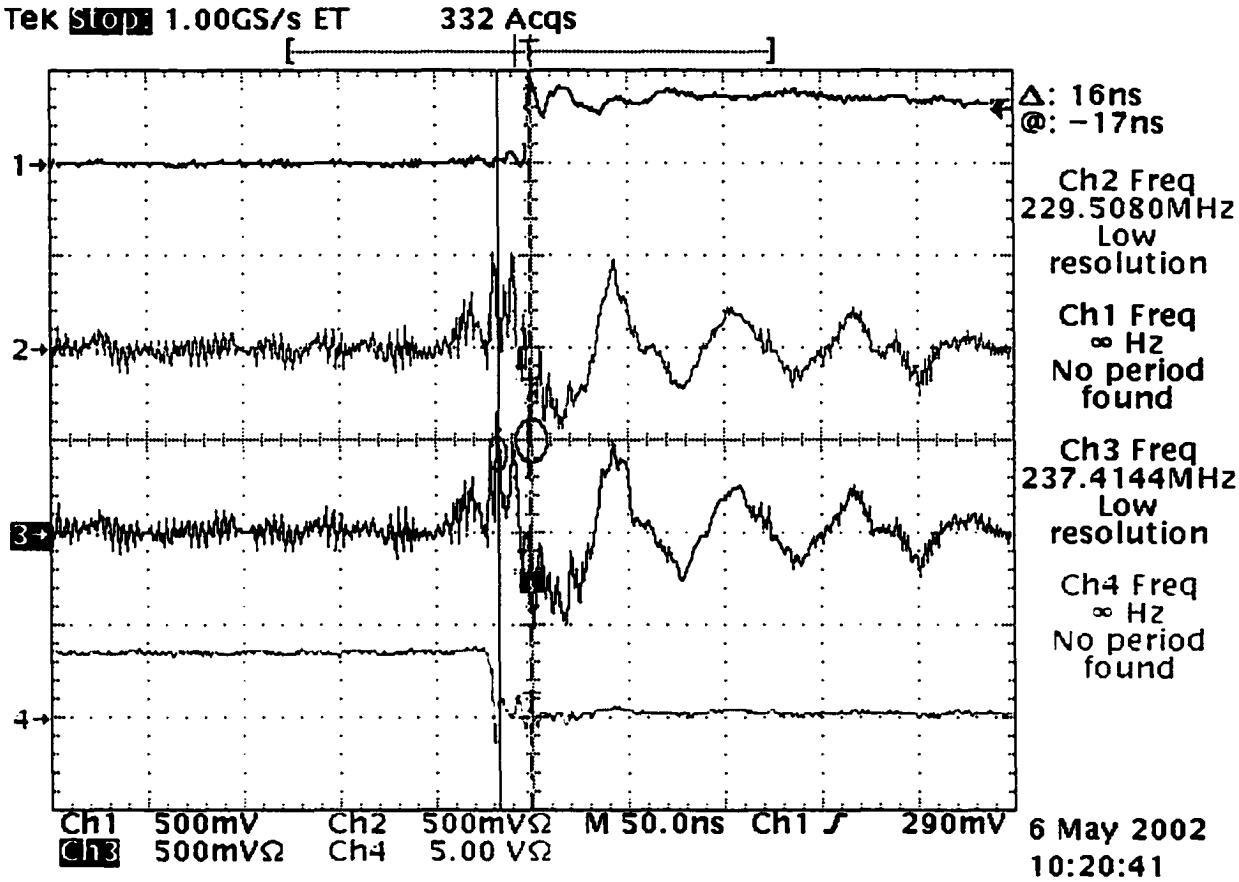

Figure 39 Mesure du délai entre VDD_{data} et VDD_{clk}
(Sel01 = 0 Sel11 = 1 R = 100 Ω).
Les signaux sont dans l'ordre :
Data_out1, VDD_{data}, VDD_{clk} et Clk_out1

Finalement, la figure 39 illustre le cas du vecteur 4 (répété). Le délai mesuré entre les points de test VDD_{data} et VDD_{clk} (~16 ns), correspond à celui mesuré entre les deux broches de sortie Data_out1 et Clk_out1(~16 ns, Figure 33). Un agrandissement de ce graphique apparaît à la figure 40.

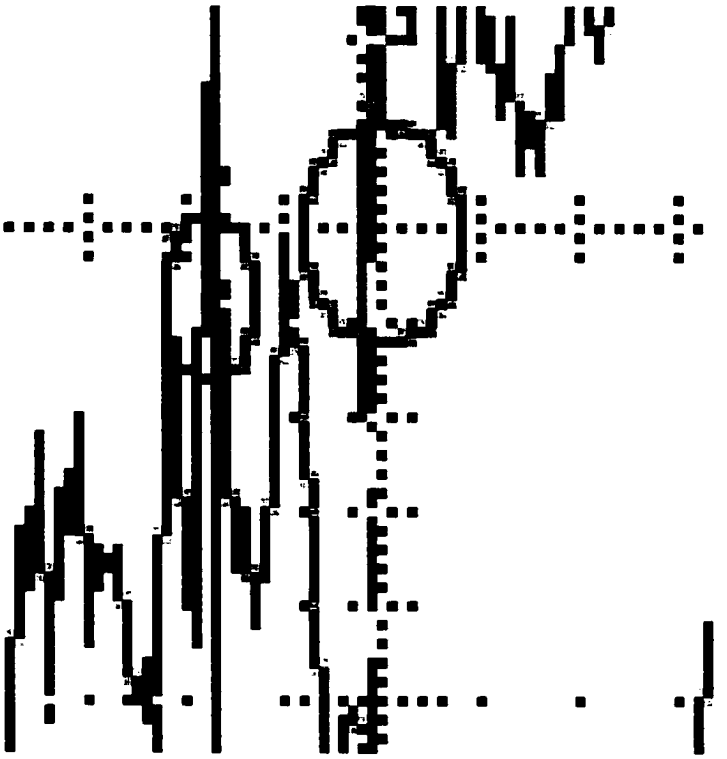

Figure 40 Agrandissement de la figure 39

Tableau III

Résultats de mesures au niveau des signaux de données et ceux aux points de test

Sommaire des délais mesurés au niveau des signaux de données et de ceux mesurés au points de test (VDD_{data} et VDD_{clk})				
Configuration	**DATA_out1-CLK_out1**		**VDD_{data} - VDD_{clk}**	
Sel1 1-Sel0 1	Délai (ns)	Figure	Délai (ns)	Figure
00	28	Figure 27	28	Figure 36
01	25	Figure 29	23.4	Figure 37
10	22	Figure 31	20	Figure 38
11	18	Figure 33	16	Figure 39

4.4.2 Mesure des marges pour VDD_{data} et VDD_{clk}

Nous nous intéressons ici aux marges d'erreur disponibles autour des seuils pour VDD_{data} et VDD_{clk}. Ces marges permettent d'éviter des déclenchements manqués. Pour ce qui est de la protection contre les faux déclenchements dus au bruit, nous considérons que la marge est à peu près égale au seuil, puisqu'il est possible de limiter la prise de mesures dans des fenêtres bien définies. Ceci a pour effet par exemple d'éviter les impulsions causées par front descendant de l'horloge Clk1.
Les valeurs des marges sont définies comme suivant :

Pour VDD_{data}

$$\textbf{Marge = valeur absolue (valeur minimale de } VDD_{data}\textbf{) - 500 mV.} \qquad (4.1)$$

et pour VDD_{clk}

$$\textbf{Marge = valeur maximale de } VDD_{clk}\textbf{ – 400 mV} \qquad (4.2)$$

Dans ce qui suit, nous allons présenter les mesures faites lors de l'application des vecteurs 1 et 4 (cas limites). Pour chaque marge, 2 mesures sont illustrées, chacune avec des valeurs de résistance R différentes (100 Ω et 1k Ω).

4.4.2.1 Marge pour VDD_{data}

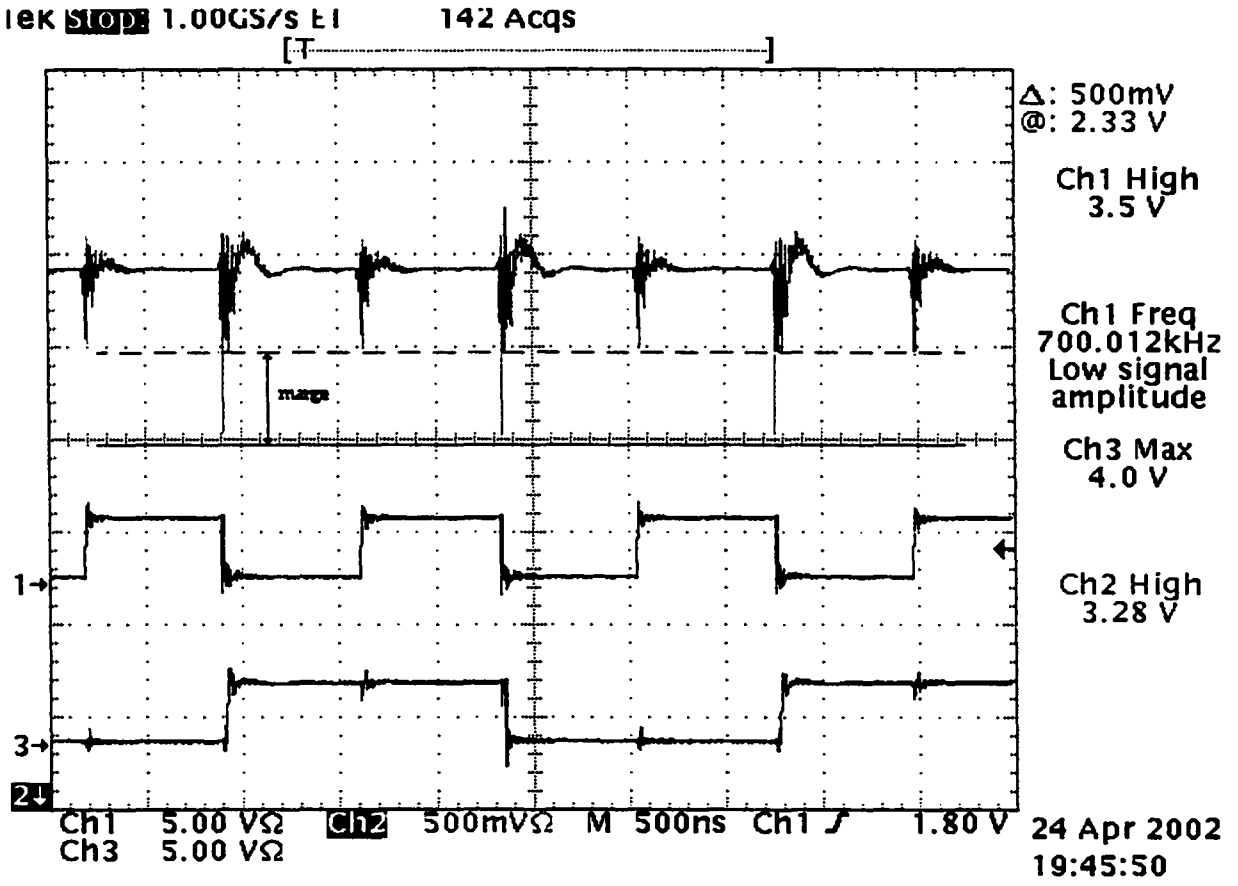

Figure 41 Première mesure de la marge du seuil pour VDD_{data} , vecteur1.
(Sel00 = 0 Sel11 = 0 R = 100 Ω)
Les signaux sont dans l'ordre : VDD_{data}, Clk_out1 et Data_out1

La figure 41 illustre le cas du vecteur 1 (répété) et de la première mesure de la marge disponible à partir du seuil pour VDD_{data} au vecteur 1, avec R=100 Ω. La valeur mesurée est d'environ 500 mV.

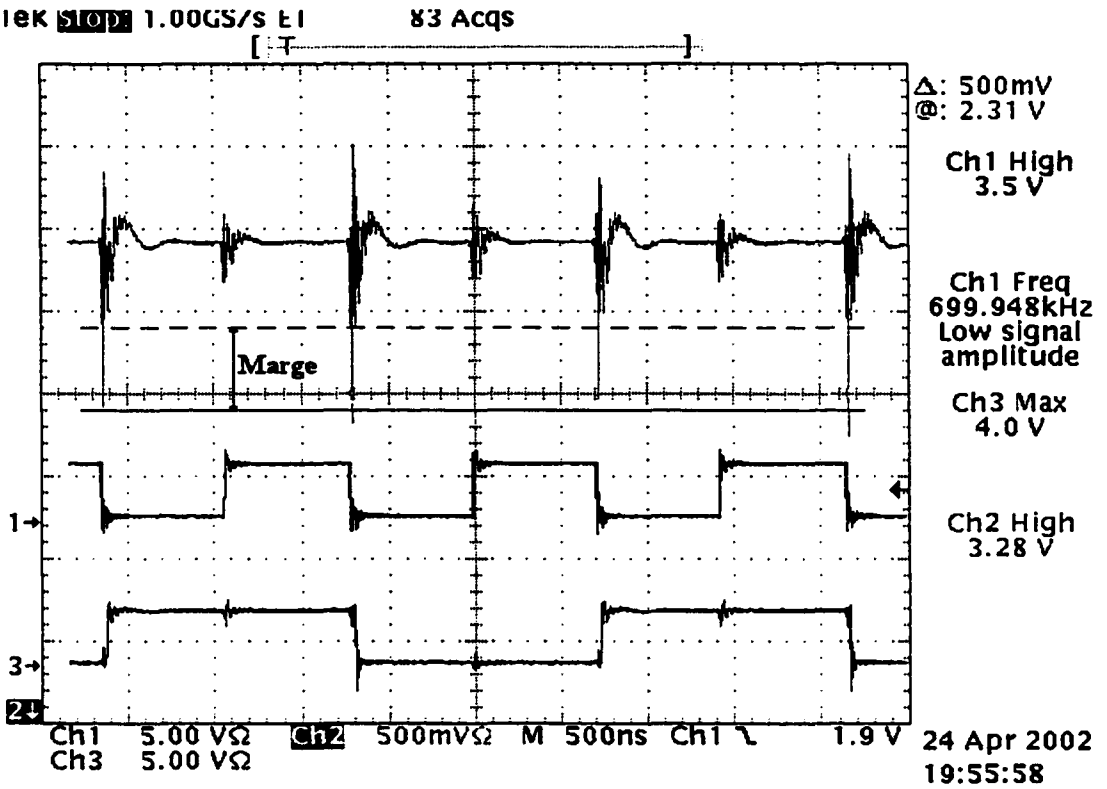

Figure 42 Deuxième mesure de la marge pour VDD_{data}, vecteur 1.
(Sel00 = 0 Sel11 = 0 R = 1 kΩ)
Les signaux sont dans l'ordre : VDD_{data}, Clk_out1 et Data_out1

La figure 42 illustre la deuxième mesure faite pour la même marge avec R= 1kΩ. La valeur mesurée est d'environ 500mV.

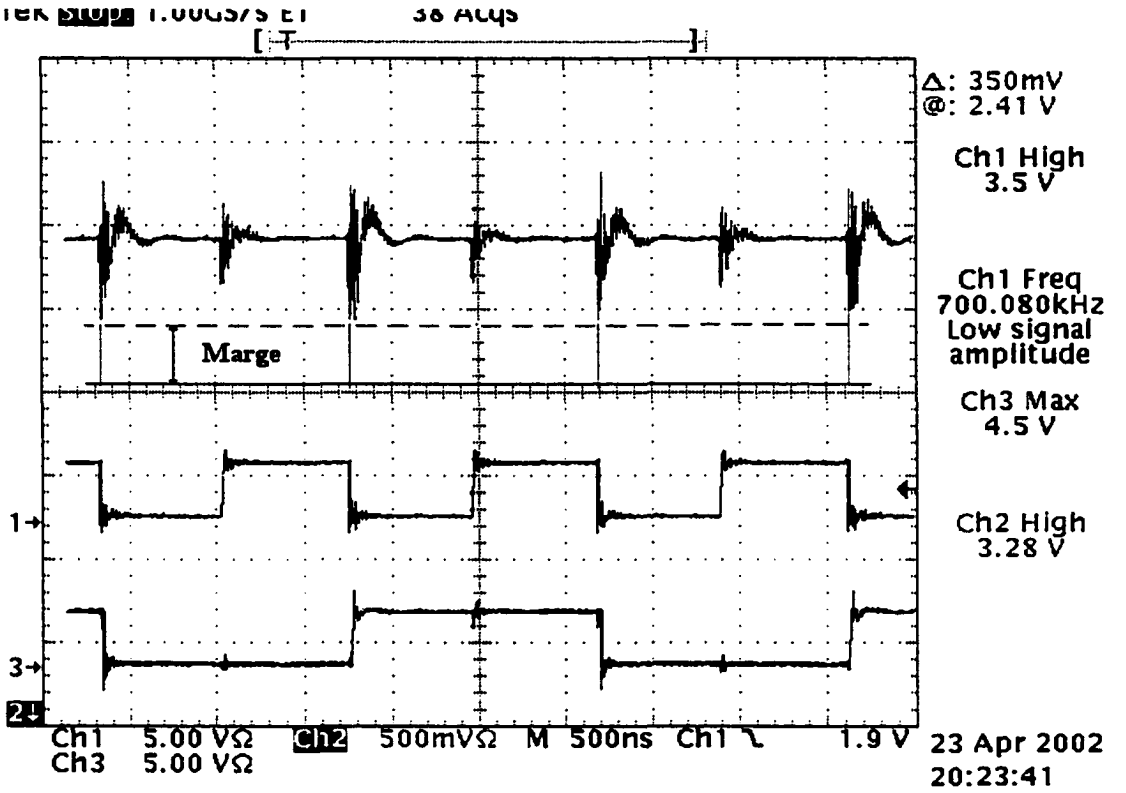

Figure 43 Première mesure de la marge pour VDD_{data}, vecteur 4.
(Sel00 = 1 Sel11 = 1 R = 100 Ω).
Les signaux sont dans l'ordre : VDD_{data}, Clk_out1 et Data_out1

La figure 43 illustre la première mesure de la marge du seuil pour VDD_{data} avec le vecteur 4 et R=100 Ω. La valeur mesurée est d'environ 350 mV.

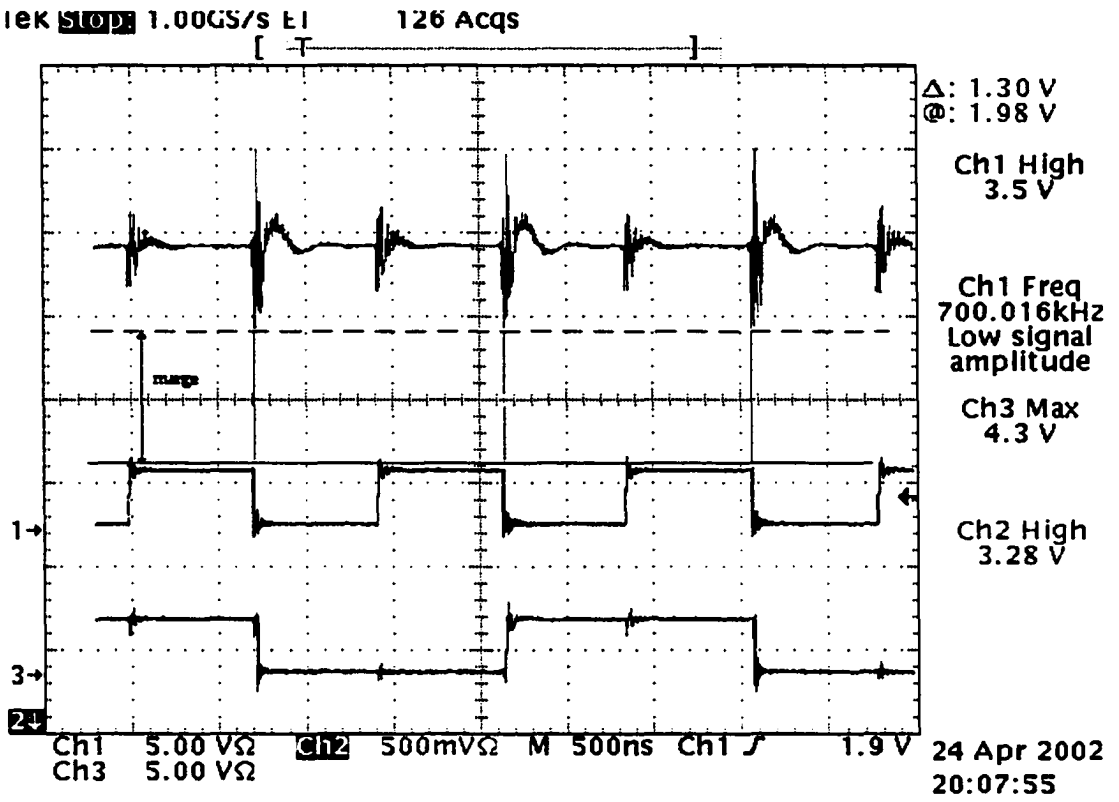

Figure 44 Deuxième mesure de la marge pour VDD_{data}, vecteur4.
(Sel00 = 1 Sel11 = 1 R = 1 kΩ)
Les signaux sont dans l'ordre : VDD_{data}, Clk_out1 et Data_out1

La figure 44 montre la deuxième mesure de la marge du seuil pour VDD_{data} au vecteur 4 avec R= 1k Ω . La valeur mesurée est d'environ 800 mV.

Dans l'ensemble, nous pouvons noter une certaine variation de la marge qui, au minimum, est de 350 mV, ce qui est nettement suffisant.

4.4.2.2 Marge pour VDD_{clk}

Nous répétons ici l'exercice pour le signal VDD_{clk}.

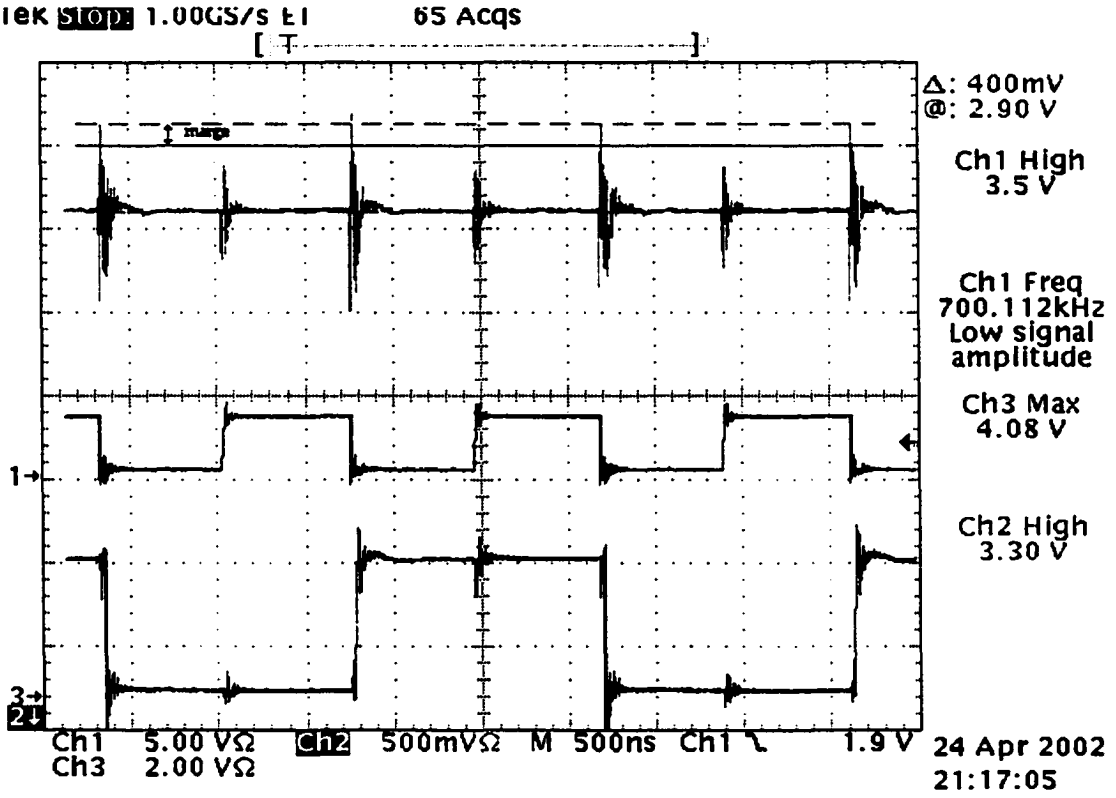

Figure 45 Première mesure de la marge pour VDD_{clk}, vecteur1.
(Sel00 = 0 Sel11 = 0 R = 100 Ω)
Les signaux sont dans l'ordre : VDD_{clk}, Clk_out1 et Data_out1

La figure 45 illustre la première mesure de la marge du seuil pour VDD_{clk} au vecteur1 (R= 100 Ω). La valeur mesurée est d'environ 130 mV.

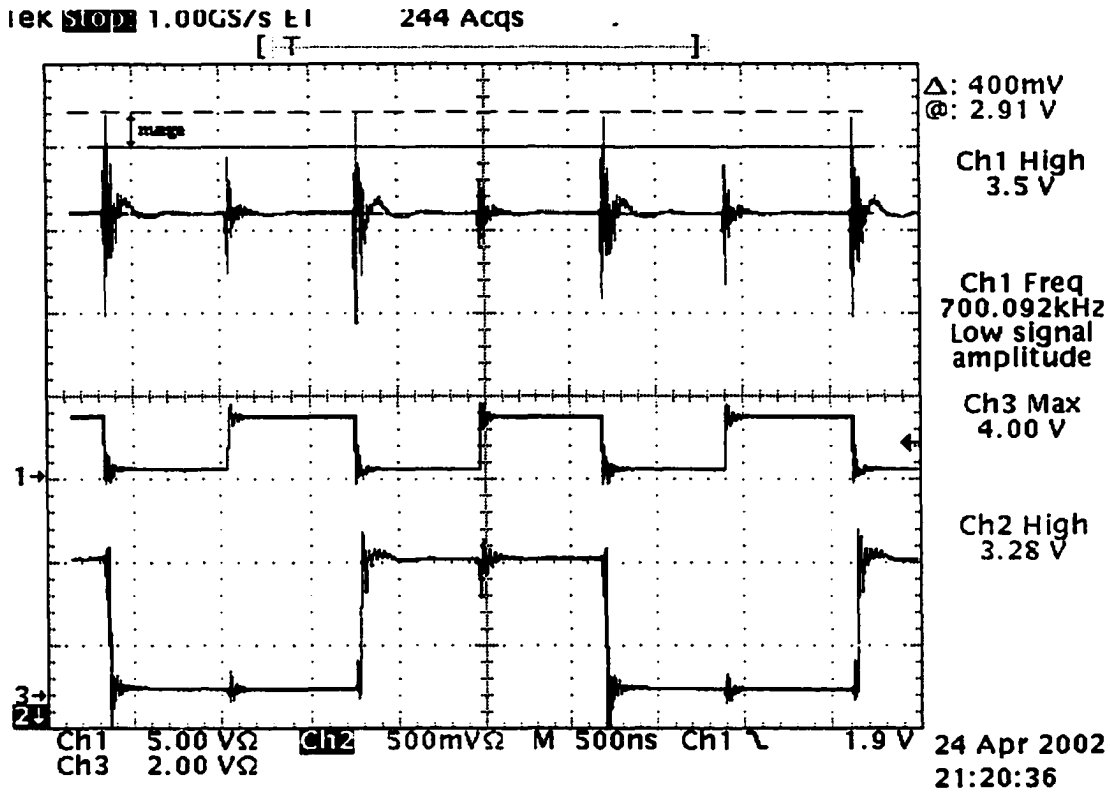

Figure 46 Deuxième mesure de la marge pour VDD_{clk}, vecteur1.
(Sel00 = 0 Sel11 = 0 R = 1 kΩ)
Les signaux sont dans l'ordre : VDD_{clk}, Clk_out1 et Data_out1

La figure 46 illustre la deuxième mesure de cette marge au vecteur 1 (R= 1 kΩ). La valeur mesurée est d'environ 200 mV.

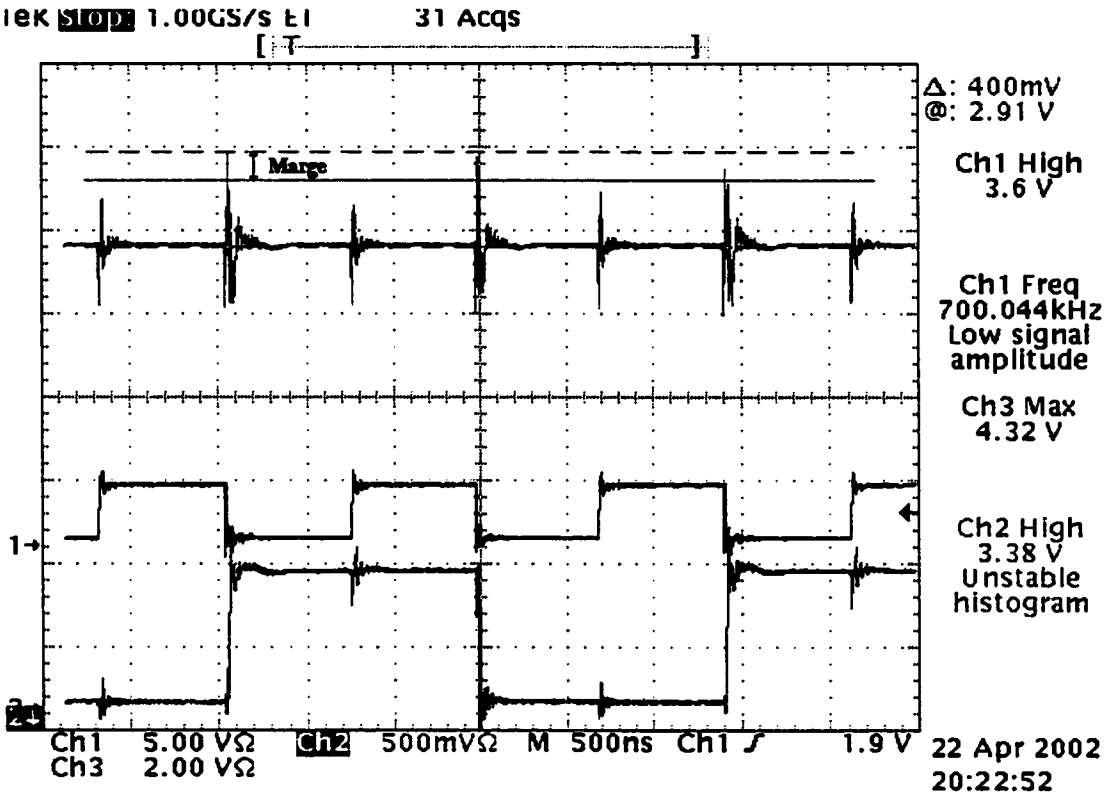

Figure 47 Première mesure de la marge pour VDD_{clk}, vecteur 4.
(Sel00 = 1 Sel11 = 1 R = 100 Ω)
Les signaux sont dans l'ordre : VDD_{clk}, Clk_out1 et Data_out1

La figure 47 illustre la première mesure de la marge du seuil pour VDD_{clk} au vecteur 4 (R= 100 Ω). La valeur mesurée moyenne est d'environ 100 mV.

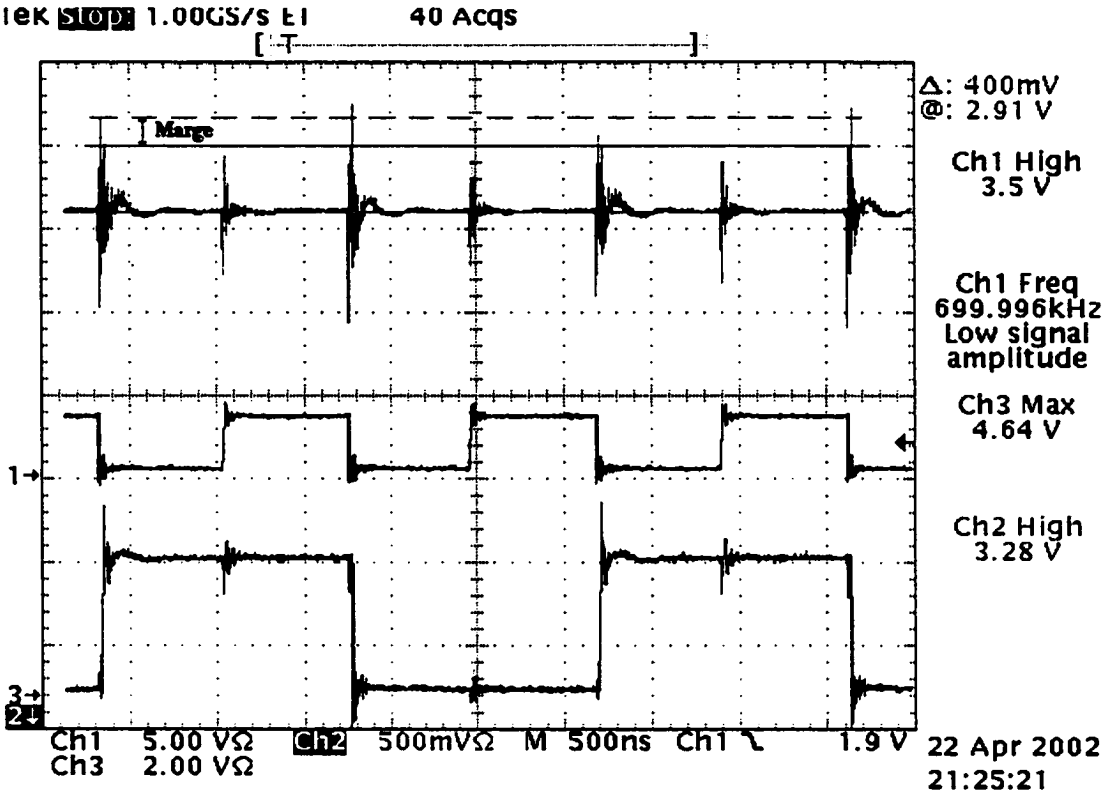

Figure 48 Deuxième mesure pour la marge de VDD_{clk} , vecteur 4.
Sel00 = 1 Sel11 = 1 R = 1 kΩ
Les signaux sont dans l'ordre : VDD_{clk}, Clk_out1 et Data_out1

La figure 48 illustre la deuxième mesure de la marge du seuil pour VDD_{clk} au vecteur 4 (R= 1 kΩ). La valeur moyenne mesurée est d'environ 150 mV. Dans l'ensemble, la marge pour VDD_{clk} est plus stable mais plus faible que celle pour VDD_{data}. Elle demeure néanmoins suffisante car la consommation de courant provenant des senseurs branchés sur le signal de l'horloge devrait être par définition stable et à peu près constante peu importe le vecteur appliqué.

Tableau IV

Sommaire des marges pour VDD_{data} et VDD_{clk}

Mesures des marges	R=100		R=1k	
	VDD_data	VDD_clk	VDD_data	VDD_clk
Sel00	500 mV	130 mV	350 mV	100 mV
Sel11	500 mV	200 mV	800 mV	150 mV

Le tableau IV fait le sommaire de la marge. Les deux valeurs de résistance R utilisées donnent des résultats somme toute assez similaires, ce qui est une bonne indication de la marge de manœuvre en ce qui concerne la conception de la logique supplémentaire pour la détection et la transformation des pulses de courant en pulses de tension.

4.4 Discussion

Sans être une preuve de concept finale, cette expérimentation fournit des résultats qui montrent qu'il est possible d'utiliser le courant comme forme de transport de l'information ciblée pour l'amener et le traiter à l'extérieur du circuit. Les résultats souffrent quelque peu du montage expérimental et de l'inductance du filage, dont l'effet est mis en évidence par la résonance observable dans les signaux. Malgré tout, il est devient clair qu'on pourrait gagner en précision si une partie ou la totalité du traitement était fait à l'intérieur même du circuit sous test.

4.5 Conclusion

D'après les résultats obtenus, nous avons rencontré les objectifs décrits au premier chapitre de ce mémoire. D'une part, les résultats de tests fonctionnels correspondent à ceux obtenu lors de la simulation. D'autre part, les tests du DFT, ont permis de

déterminer la période minimale à laquelle peut fonctionner le circuit suivant une configuration bien déterminée. Ces mêmes tests ont permis d'établir la corrélation entre les délais mesurés aux points de test et celui mesuré entre Clk_out1 et Data_out1. Finalement, à partir des mesures de marges nous avons vérifier la conformité à la façon dont nous avions mesuré les délais entre VDD_{data} et VDD_{clk}.

CONCLUSION

L'objectif de ce travail est de réaliser un circuit intégré de type ASIC. La technologie CMOS 0.35 µm a été ciblée pour la fabrication de ce circuit. Ce prototype devait intégrer la logique nécessaire pour implémenter une méthodologie de test récemment proposée. Pour accomplir cette tâche, nous avons élaboré une architecture qui répond aux besoins de la nouvelle méthode de test. De plus, nous avons adopté une stratégie de design qui nous a permis de concevoir le prototype envisagé avec les moyens disponibles à l'École de technologie supérieure.

Le flot spécial proposé consiste à implémenter les besoins particuliers et nécessaires de la méthodologie de test, avec un niveau d'intervention minimum par rapport au flot standard de design. Certaines étapes, non nécessaires dans le cadre de notre prototype ont été ignorées. L'utilisation de ce flot a permis la réalisation du prototype envisagé.

Tous les passages utilisés dans la conception du prototype ont été bien exécutés et d'une manière relativement facile, ce qui prouve l'efficacité de la méthodologie du design et la possibilité d'automatiser le processus avec un minimum d'efforts. L'autre élément important est le fait que les outils utilisés sont parmi les plus puissants qui existent dans le domaine de la conception des circuits intégrés.

Enfin, les résultats obtenus lors de la validation du prototype sont concluants. Ils coïncident bien avec les résultats attendus de point de vu fonctionnel. À partir des tests de DFT, nous avons pu estimer la période minimale à laquelle peut fonctionner le circuit et établir la corrélation entre les mesures prises aux points de test et celles prises à partir des sorties normales du circuit. D'où l'efficacité de la méthode de test.

RECOMMANDATIONS

Certes, il reste des améliorations à apporter à la méthodologie utilisée pour réaliser ce prototype, afin de la rendre plus conviviale pour des circuits qui intègrent la méthode de test (chaînes parallèles de courant) dans des circuits plus complexes ou qui ciblent une technologie autre que celle utilisée pour la réalisation de ce prototype. D'abord elle est intimement liée à la technologie CMOS 0.35 μm, il faudrait donc rendre cette méthodologie moins dépendante de la technologie. Ensuite, il faudrait l'automatiser pour qu'elle devienne efficace et moins risquée, car vue la complexité des circuits intégrés, il est toujours recommandé de ne pas faire des interventions manuelles. Une solution possible est d'adopter le même principe utilisé par les algorithmes d'insertion des chaînes de bascules à balayage «scan test ».

ANNEXE 1

VHDL ET LA SIMULATION

Ce qui suit est fortement inspiré de la thèse de doctorat de Jacomme, L., (1999).Analyse sémantique de descriptions VHDL synchrones en vue de la synthèse. Thèse de doctorat de l'université de Paris VI

Les particularités du langage VHDL

Le VHDL est basé sur la notion d'exécution parallèle de processus séquentiels qui communiquent à travers des signaux. Il permet de décrire tous les aspects temporels liés aux communications entre processus. Il donne ainsi la possibilité de spécifier précisément le comportement intrinsèquement parallèle des circuits numériques, ainsi que leurs caractéristiques temporelles.

Le langage VHDL offre au concepteur la possibilité de définir de nouveaux types de données. Ceci permet notamment de rendre une spécification beaucoup plus lisible. En effet, il est beaucoup plus pratique d'employer un type énuméré plutôt qu'un type entier. La définition d'un type de données implique la mise en oeuvre de fonctions de manipulation et de conversion. Ces opérations peuvent entraîner le ralentissement du processus de la compilation et de dépannage («debugging»). Comme c'est le cas pour plusieurs langages de haut niveau, il est possible en VHDL de surcharger des fonctions existantes, pour les adapter à d'autres types de données. La vérification de la bonne utilisation des types et de leurs fonctions de manipulation est effectuée lors de la compilation. C'est un des avantages de VHDL, puisque les risques d'utilisation incorrecte sont détectés très tôt dans le processus, ce qui constitue un gain considérable.

VHDL et la Simulation

Le mécanisme d'élaboration défini par la norme assure en particulier la transformation des instructions concurrentes en processus. La phase d'élaboration assure également la création de tous les signaux et de toutes les variables de la description et elle leur donne une valeur initiale bien définie.

La simulation VHDL est pilotée par l'apparition d'évènements sur les signaux de communication entre processus. Le cycle de simulation appelé delta-cycle est découpé en trois phases distinctes ***EXECUTE***, ***UPDATE*** et ***RESUME*** définies ainsi :

- La phase ***EXECUTE*** correspond à une exécution en parallèle des processus jusqu'à leur prochain point de suspension. Au cours de cette phase, les variables internes des processus réveillés sont modifiées au moment de leur affectation et les transactions effectuées sur les signaux de communication sont enregistrées dans un échéancier;
- Au cours de la phase ***UPDATE***, les valeurs et les différents attributs des signaux sont mis à jour à partir de l'échéancier;
- La phase ***RESUME*** établit la liste des processus à réveiller en fonction des évènements intervenus dans l'échéancier.

Avant d'entrer dans la boucle de simulation, il est nécessaire d'initialiser la description. Cette étape préliminaire est définie dans le manuel de référence du langage VHDL et consiste notamment à exécuter une fois chacun des processus de la description jusqu'à la rencontre d'un point de suspension. Durant cette exécution, les variables et les signaux peuvent changer de valeur. Ces différentes initialisations n'expriment pas un comportement matériel réel, elles sont simplement le support d'un mécanisme de simulation.

ANNEXE 2

LA SYNTHÈSE

Ce qui suit est fortement inspiré de la thèse de doctorat de Jacomme, L., (1999).Analyse sémantique de descriptions VHDL synchrones en vue de la synthèse. Thèse de doctorat de l'université de Paris VI

La synthèse comportementale

La synthèse comportementale, aussi appelée synthèse de haut niveau, est la transformation d'une description de type algorithmique en une description de type transfert entre registres : ***Register Transfert Level (RTL).***

Une description algorithmique ne comporte aucune information architecturale et temporelle. Elle représente simplement le comportement d'un circuit en le considérant comme une boite noire.

Les outils de synthèse de haut niveau doivent:

- Effectuer une allocation efficace d'un ensemble de ressources matérielles permettant d'implanter les opérations nécessaires à l'exécution de l'algorithme. Cela consiste en général à interconnecter des registres avec des opérateurs complexes afin d'élaborer un ou plusieurs chemins de données.
- Construire un ou plusieurs séquenceurs capables de piloter dans le temps les opérateurs des différents chemins de données.

La synthèse *RTL*

Le point d'entrée de la synthèse ***RTL*** est une description comportementale synchrone spécifiant le comportement d'un circuit cycle par cycle. Les états du circuit sont complètement définis et l'ensemble des transitions d'un état à un autre est décrit sous la forme d'un ensemble de fonctions de transfert entre des registres. Cependant les registres matériels ne sont pas nécessairement identifiés.

La description comportementale est généralement constituée d'un ensemble d'affectations concurrentes et d'un ensemble de processus séquentiels communicants. Elle définit le comportement d'éléments matériels interconnectés, tels que des machines à états finis, des mémoires, des blocs combinatoires, des opérateurs arithmétiques et des registres. Un outil de synthèse ***RTL*** peut cependant modifier l'allocation de ces ressources matérielles. C'est le niveau d'entrée des outils industriels tels que : ***Design Compiler*** de ***Synopsys***, ***Synergy*** de ***Cadence***, ***Asic Synthesizer*** de ***Compass*** , ***Symplify*** de ***Symplicity*** ou encore ***Autologic*** de ***Mentor Graphics***.

La tâche d'un outil de synthèse ***RTL*** est de transformer une telle description comportementale en une interconnexion d'éléments matériels connus de l'outil, tout en respectant des contraintes d'optimisation. Ces éléments matériels sont par exemple des portes logiques, des bascules ou bien encore des mémoires ou des opérateurs arithmétiques.

Lors de la phase d'optimisation, un outil de synthèse ***RTL*** peut modifier profondément la structure du circuit sans changer son comportement global. Par exemple il peut appliquer un nouveau codage à une machine à états finis. Il peut également modifier la position de certains multiplexeurs pour partager efficacement des ressources matérielles. Dans certains cas il est nécessaire d'effectuer des

transformations encore plus importantes comme l'ajout de registres pour réduire la taille de certains chemins combinatoires critiques. La synthèse ***RTL*** est constituée d'une phase de compilation, d'une phase d'optimisation et d'une phase de projection structurelle.

La phase de compilation

La phase de compilation de la description d'un circuit spécifie cycle par cycle son comportement. Le résultat de cette compilation est un modèle adapté à la synthèse et indépendant du langage de description utilisé. La phase de compilation de la synthèse doit ramener tous les symboles manipulés à un niveau booléen, identifier précisément les éléments matériels particuliers tel que les registres et les portes logiques et extraire l'ensemble des fonctions de transfert entre les registres.

Cette étape a pour rôle essentiel de matérialiser tout ce qui n'est pas explicitement identifié dans la description initiale. La complexité de cette phase est complètement dépendante du langage de description de matériel qui est utilisé en entrée de la synthèse. En effet, plus ce langage permet de décrire de façon différente et de façon implicite des éléments matériels, plus les algorithmes d'analyse sont complexes.

La phase d'optimisation

Après avoir achevé la compilation d'une description et extrait un ensemble de registres et de fonctions de transfert entre registres, l'outil doit alors appliquer différentes techniques d'optimisation propre à la synthèse. Ces opérations peuvent être au besoin dirigées par le designer, en imposant par exemple des contraintes pour le partage de ressources matérielles.

Dans le cas particulier de notre prototype, nous avons eu recours à des attributs « attributes » de Design Compiler pour l'empêcher d'optimiser les chaînes des

inverseurs, car d'un point de vue fonctionnel, avoir deux inverseurs successifs sans aucune dérivation est l'équivalent d'avoir un fil. L'attribut utilisé pour empêcher une telle optimisation est « set_dont_touch ».

La phase de la projection structurelle

La phase de projection structurelle succède à la phase d'optimisation. Elle consiste à trouver une implantation matérielle aux registres et aux fonctions de transferts précédemment optimisées. Lors de cette transformation, un outil de synthèse peut par exemple:

- Ajouter des amplificateurs en sortie d'une porte logique (Output buffer)
- Construire un arbre de distribution pour un signal particulier (Clock tree)

Le résultat de cette projection structurelle est une interconnexion de portes logiques, de registres, de mémoires et d'opérateurs arithmétiques, etc.

ANNEXE 3

CODE VHDL

```
-------------------------------------------------
-- Date        : Thu Feb  1 14:48:53 2001
-- Author      : Ben_letaifa
-- Company     : ETS
-------------------------------------------------
-- Root of Design:
-- ---------------
--    Unit   Name  :  programmable_delay
--    Library Name  :  ETS
--    Creation Date :  Tue Feb 27 15:16:27 2001
--    Version       :  1.2
-- -------------
--    Target
--      HDL       :  VHDL
---------------------------------------
---------------------------------------
-- Library Name :  ETS
-- Unit   Name :  mux4
-- Unit   Type :  Text Unit
--
---------------------------------------
library ieee; use ieee.std_logic_1164.all;
library ieee; use ieee.STD_LOGIC_UNSIGNED.all;
library ieee; use ieee.std_logic_arith.all;

entity  mux4  is
port (delay1    : in std_logic ;
    delay2    : in std_logic ;
    delay3    : in std_logic ;
    delay4    : in std_logic ;
    delay_out : out std_logic ;
    sel0      : in std_logic ;
    sel1      : in std_logic );
end;

architecture  rtl  of  mux4  is

begin
process(delay1, delay2, delay3, delay4, sel0, sel1)
begin
 if   (sel0= '0' and sel1= '0') then delay_out <=  delay1;
 elsif (sel0= '1' and sel1= '0') then delay_out <=  delay2;
 elsif (sel0= '0' and sel1= '1') then delay_out <=  delay3;
 else  delay_out <=  delay4;
 end if;
end process;
end;
```

```
-------------------------------------------------
-- Library Name : ETS
-- Unit   Name : hard_wxor
-- Unit   Type : Text Unit
-- -----------------------------------------------
library ieee; use ieee.std_logic_1164.all;
library ieee; use ieee.std_logic_arith.all;
library ieee; use ieee.STD_LOGIC_UNSIGNED.all;

entity hard_wxor is
port ( a:        in std_logic ;
b:        in std_logic ;
       xor_out: out std_logic );
end;
architecture RTL of hard_wxor is
begin
xor_out <= a xor b;
end;
-------------------------------------------------
-- Library Name : ETS
-- Unit   Name : inverter
-- Unit   Type : Text Unit
-- -----------------------------------------------
library ieee; use ieee.std_logic_1164.all;
library ieee; use ieee.std_logic_arith.all;
library ieee; use ieee.STD_LOGIC_UNSIGNED.all;

entity inverter is
port    (in_inv  : in std_logic ;
    out_inv  : out std_logic );
end;
architecture RTL of inverter is
begin
out_inv <= not in_inv;
end;

-------------------------------------------------
-- Library Name : ETS
-- Unit   Name : delay50
-- Unit   Type : Text Unit
-- -----------------------------------------------

library ieee; use ieee.std_logic_1164.all;
library ieee; use ieee.std_logic_arith.all;
library ieee; use ieee.STD_LOGIC_UNSIGNED.all;
```

```
entity delay50 is
port (  i50 : in std_logic ;
        o50 : out std_logic);
end;
architecture RTL of delay50 is

signal  inter0,inter1,inter2,inter3,inter4,inter5,inter6,inter7,inter8,inter9,
       inter10,inter11,inter12,inter13,inter14,inter15,inter16,inter17,inter18,
       inter19,inter20,inter21,inter22,inter23,inter24,inter25,inter26,inter27,
       inter28,inter29,inter30,inter31,inter32,inter33,inter34,inter35,
          inter36,inter37,inter38,inter39,inter40,inter41,inter42,inter43,inter44,
       inter45,inter46,inter47,inter48,inter49 : std_logic;
BEGIN
inter0  <= not i50;
inter1  <= not inter0;
inter2  <= not inter1;
inter3  <= not inter2;
inter4  <= not inter3;
inter5  <= not inter4;
inter6  <= not inter5;
inter7  <= not inter6;
inter8  <= not inter7;
inter9  <= not inter8;
inter10 <= not inter9;
inter11 <= not inter10;
inter12 <= not inter11;
inter13 <= not inter12;
inter14 <= not inter13;
inter15 <= not inter14;
inter16 <= not inter15;
inter17 <= not inter16;
inter18 <= not inter17;
inter19 <= not inter18;
inter20 <= not inter19;
inter21 <= not inter20;
inter22 <= not inter21;
inter23 <= not inter22;
inter24 <= not inter23;
inter25 <= not inter24;
inter26 <= not inter25;
inter27 <= not inter26;
inter28 <= not inter27;
inter29 <= not inter28;
inter30 <= not inter29;
inter31 <= not inter30;
inter32 <= not inter31;
inter33 <= not inter32;
```

```
inter34 <= not inter33;
inter35 <= not inter34;
inter36 <= not inter35;
inter37 <= not inter36;
inter38 <= not inter37;
inter39 <= not inter38;
inter40 <= not inter39;
inter41 <= not inter40;
inter42 <= not inter41;
inter43 <= not inter42;
inter44 <= not inter43;
inter45 <= not inter44;
inter46 <= not inter45;
inter47 <= not inter46;
inter48 <= not inter47;
inter49 <= not inter48;
o50    <= not inter49;

end RTL;
```

```
library ieee; use ieee.std_logic_1164.all;
library ieee; use ieee.STD_LOGIC_UNSIGNED.all;
library ieee; use ieee.std_logic_arith.all;
--module wdp_2 (vdd, vss, q, qb, d0, ck);

entity  ff  is
port (  clk      : in std_logic;
d        : in std_logic;
rst      : in std_logic ;
         q       : out std_logic
    );
end;

architecture  rtl  of  ff  is
begin
process(clk, rst)
begin
         if rst ='1' then q<='0';
         elsif (clk'event and clk ='1') then
         q <= d;
         end if;
end process ;
end;
```

```
-------------------------------------------------
-- Library Name : ETS
-- Unit   Name : toggle
-- Unit   Type : Text Unit
-- -----------------------------------------------------
library ieee; use ieee.std_logic_1164.all;
library ieee; use ieee.STD_LOGIC_UNSIGNED.all;
library ieee; use ieee.std_logic_arith.all;

entity  toggle  is
        port    (clk     : in std_logic;
rst     : in std_logic ;
                T_data : out std_logic
            );
end;

architecture rtl of toggle is

signal int_clk, int_d, hard_one : std_logic ;
component ff
 port ( clk,d, rst : in std_logic;
      q        : out std_logic
    );
end component ;
component hard_wxor
 port ( a,b    : in std_logic;
      xor_out       : out std_logic
    );
end component ;
begin

hard_one <='1';

inst_hard_wxor : hard_wxor
port map ( a => hard_one ,
        b => int_clk ,
        xor_out  => int_d
       );

inst_ff : ff
port map ( clk => clk , d => int_d , rst => rst , q=> int_clk);

T_data <= int_clk ;

end;
```

```
------------------------------------------------
--
-- Library Name :  ETS
-- Unit   Name :  progammable_delay
-- Unit   Type :  Block Diagram
--
------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.STD_LOGIC_UNSIGNED.all;
use ieee.std_logic_arith.all;

entity progammable_delay is
 port (
     clk,rst              : in std_logic;
     delay_out,clk_out, data_out: out std_logic;
     sel0                 : in std_logic;
     sel1                 : in std_logic
     );


end progammable_delay;

architecture rtl of progammable_delay is

 signal a1    : std_logic;
 signal a2    : std_logic;
 signal a3    : std_logic;
 signal a4    : std_logic;
 signal a5    : std_logic;
 signal a6    : std_logic;
 signal a7    : std_logic;
 signal b1    : std_logic;
 signal b2    : std_logic;
 signal b3    : std_logic;
 signal b4    : std_logic;
 signal b5    : std_logic;
 signal b6    : std_logic;
 signal d1    : std_logic;
 signal d2    : std_logic;
 signal d3    : std_logic;
 signal d4    : std_logic;
 signal d5    : std_logic;
 signal delay1 : std_logic;
 signal delay2 : std_logic;
 signal delay3 : std_logic;
```

```
signal delay4 : std_logic;
signal e1     : std_logic;
signal e2     : std_logic;
signal e3     : std_logic;
signal e4     : std_logic;
signal T_data1 : std_logic;
signal T_data2 : std_logic;
signal T_data3 : std_logic;
signal T_data4 : std_logic;
signal out_mux4: std_logic;
component toggle
   port (
        clk,rst : in std_logic;
        T_data  : out std_logic
        );
end component;
component delay50
   port (
        i50 : in std_logic;
        o50 : out std_logic
        );
end component;
component mux4
   port (
        delay1 : in std_logic;
        delay2 : in std_logic;
        delay3 : in std_logic;
        delay4 : in std_logic;
        delay_out : out std_logic;
        sel0 : in std_logic;
        sel1 : in std_logic
        );
end component;
component ff
  port( clk,d,rst : in std_logic;
        q         : out std_logic);
end component;
component inverter
port (
     in_inv      : in std_logic;
     out_inv     : out std_logic);
end component ;

 -- Start Configuration Specification
 -- ++ for all : toggle
 -- ++     use entity work.toggle(rtl);
 -- ++ for all : delay50
```

```
-- ++      use entity work.delay50(RTL);
-- ++ for all : mux4
-- ++      use entity work.mux4(rtl);
-- End Configuration Specification


begin
 C0: toggle
  port map (
         clk => clk,
         rst => rst,
         T_data => T_data1);

 C1: toggle
  port map (
         clk => clk,
         rst => rst,
         T_data => T_data2);

 C2: toggle
  port map (
         clk => clk,
         rst => rst ,
         T_data => T_data3);

 C3: toggle
  port map (
         clk => clk,
         rst => rst ,
         T_data => T_data4);
-----&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&------
clk_sensor :  inverter
port map  ( in_inv => clk ,
         out_inv => clk_out);
-----&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&------
 C5: delay50
  port map (
         i50 => T_data1 ,
         o50 => a1);

 C6: delay50
  port map (
         i50 => a1,
         o50 => a2);
```

```
C7: delay50
 port map (
        i50 => a2,
        o50 => a3);

C8: delay50
 port map (
        i50 => T_data2,
        o50 => b1);

C9: delay50
 port map (
        i50 => b1,
        o50 => b2);

C10: delay50
 port map (
        i50 => a3,
        o50 => a4);
C11: delay50
 port map (
        i50 => b2,
        o50 => b3);

C12: delay50
 port map (
        i50 => a4,
        o50 => a5);

C13: delay50
 port map (
        i50 => b3,
        o50 => b4);

C14: delay50
 port map (
        i50 => a5,
        o50 => a6);

C15: delay50
 port map (
        i50 => a6,
        o50 => a7);
```

```
C16: delay50
 port map (
        i50 => a7,
        o50 => delay1);
C17: delay50
 port map (
        i50 => b4,
        o50 => b5);

inst_18: delay50
 port map (
        i50 => b5,
        o50 => b6);

C20: delay50
 port map (
        i50 => T_data3,
        o50 => d1);

C21: delay50
 port map (
        i50 => d1,
        o50 => d2);

C22: delay50
 port map (
        i50 => d2,
        o50 => d3);

C23: delay50
 port map (
        i50 => d3,
        o50 => d4);
inst_24: delay50
 port map (
        i50 => d4,
        o50 => d5);

C25: delay50
 port map (
        i50 => d5,
        o50 => delay3);

C26: delay50
 port map (
        i50 => T_data4,
        o50 => e1);
```

```
C27: delay50
 port map (
        i50 => e1,
        o50 => e2);

C28: delay50
 port map (
        i50 => e2,
        o50 => e3);

C29: delay50
 port map (
        i50 => e3,
        o50 => e4);
C30: delay50
 port map (
        i50 => e4,
        o50 => delay4);

C32: delay50
 port map (
        i50 => b6,
        o50 => delay2);

inst_mux4: mux4
 port map (
        delay1 => delay1,
        delay2 => delay2,
        delay3 => delay3,
        delay4 => delay4,
        delay_out => out_mux4,
        sel0 => sel0,
        sel1 => sel1);

inst_ff : ff
port map ( clk => clk , d => out_mux4 ,rst => rst, q => delay_out );

-----&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&------
data_sensor : inverter
port map ( in_inv => out_mux4 ,
        out_inv => data_out);
-----&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&------
end rtl;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
----------TOP----------
-------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.STD_LOGIC_UNSIGNED.all;
use ieee.std_logic_arith.all;
entity progammable_delay_3cells is
 port (
     clk1     : in  std_logic ;
     clk2     : in  std_logic ;
     clk3     : in  std_logic ;

     rst1     : in  std_logic ;
     rst2     : in  std_logic ;
     rst3     : in  std_logic ;

     delay_out1: out std_logic ;
     delay_out2: out std_logic ;
     delay_out3: out std_logic ;

     clk_out1,clk_out2,clk_out3     : out std_logic;
     data_out1, data_out2, data_out3 : out std_logic ;

     sel01    : in  std_logic ;
     sel11    : in  std_logic ;

     sel02    : in  std_logic ;
     sel12    : in  std_logic ;

     sel03    : in  std_logic ;
     sel13    : in  std_logic
     );

end progammable_delay_3cells;

architecture rtl of progammable_delay_3cells is

component progammable_delay
 port (
     clk,rst  : in std_logic;
     delay_out,clk_out,data_out : out std_logic;
     sel0     : in std_logic;
     sel1     : in std_logic
     );

 end component;
```

```
begin

inst_cell1 : progammable_delay
port map (
      clk      => clk1,
      rst      => rst1,
      delay_out => delay_out1,
      clk_out  => clk_out1,
      data_out  => data_out1,
      sel0     => sel01,
      sel1     => sel11
inst_cell2 : progammable_delay
port map (
      clk      => clk2,
      rst      => rst2,
      delay_out => delay_out2,
      clk_out  => clk_out2,
      data_out  => data_out2,
      sel0     => sel02,
      sel1     => sel12
      );

inst_cell3 : progammable_delay
port map (
      clk      => clk3,
      rst      => rst3,
      delay_out => delay_out3,
      clk_out  => clk_out3,
      data_out  => data_out3,
      sel0     => sel03,
      sel1     => sel13
      );
end rtl ;
```

```
-------------------------------------------------
--
-- Library Name :  ETS
-- Unit   Name :  TB_programmable_delay_3cells
-- Unit   Type :  Text Unit
--
-------------------------------------------------
----------------------------------------
----------------------------------------
--
-- Author     : Ben_letaifa  Abdelkader
--
-- Description :
--
----------------------------------------
----------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.STD_LOGIC_UNSIGNED.all;
use ieee.std_logic_arith.all;
entity TB_programmable_delay_3cells is
 port (
      clk_out1 : out std_logic;
      clk_out2 : out std_logic;
      clk_out3 : out std_logic;
      data_out1 : out std_logic;
      data_out2 : out std_logic;
      data_out3 : out std_logic;
      delay_out1 : out std_logic;
      delay_out2 : out std_logic;
      delay_out3 : out std_logic
      );

end TB_programmable_delay_3cells;
 use work.all;
architecture rtl of  TB_programmable_delay_3cells  is

  signal clk1 : std_logic;
  signal clk2 : std_logic;
  signal clk3 : std_logic;
  signal rst1 : std_logic;
  signal sel01 : std_logic;
  signal sel11 : std_logic;
```

```
component stim_gen
   port (
        clk1 : out std_logic;
        clk2 : out std_logic;
        clk3 : out std_logic;
        rst1 : out std_logic;
        sel01 : out std_logic;
        sel11 : out std_logic
        );
end component;
component progammable_delay_3cells
   port (
        clk1 : in std_logic;
        clk2 : in std_logic;
        clk3 : in std_logic;
        rst1 : in std_logic;
        rst2 : in std_logic;
        rst3 : in std_logic;
        delay_out1 : out std_logic;
        delay_out2 : out std_logic;
        delay_out3 : out std_logic;
        clk_out1 : out std_logic;
        clk_out2 : out std_logic;
        clk_out3 : out std_logic;
        data_out1 : out std_logic;
        data_out2 : out std_logic;
        data_out3 : out std_logic;
        sel01 : in std_logic;
        sel11 : in std_logic;
        sel02 : in std_logic;
        sel12 : in std_logic;
        sel03 : in std_logic;
        sel13 : in std_logic
        );
end component;

-- Start Configuration Specification
-- ++ for all : stim_gen
-- ++     use entity work.stim_gen(rtl);
-- ++ for all : progammable_delay_3cells
-- ++     use entity work.progammable_delay_3cells(rtl);
-- End Configuration Specification
```

```
begin

 inst_stim_gen: stim_gen
  port map (
        clk1 => clk1,
        clk2 => clk2,
        clk3 => clk3,
        rst1 => rst1,
        sel01 => sel01,
        sel11 => sel11);

 DUT: progammable_delay_3cells
  port map (
        clk1 => clk1,
        clk2 => clk2,
        clk3 => clk3,
        rst1 => rst1,
        rst2 => rst1,
        rst3 => rst1,
        delay_out1 => delay_out1,
        delay_out2 => delay_out2,
        delay_out3 => delay_out3,
        clk_out1 => clk_out1,
        clk_out2 => clk_out2,
        clk_out3 => clk_out3,
        data_out1 => data_out1,
        data_out2 => data_out2,
        data_out3 => data_out3,
        sel01 => sel01,
        sel11 => sel11,
        sel02 => sel01,
        sel12 => sel11,
        sel03 => sel01,
        sel13 => sel11);
end rtl;
```

```
-----------------------------------------
-- Library Name : ETS
-- Unit   Name : stim_gen
-- Unit   Type : Text Unit

-----------------------------------------
-- Author     : Ben_letaifa  Abdelkader
-----------------------------------------
library ieee; use ieee.std_logic_1164.all;
library ieee; use ieee.STD_LOGIC_UNSIGNED.all;
library ieee; use ieee.std_logic_arith.all;

entity  stim_gen  is
port (clk1 : out std_logic ;
    clk2 : out std_logic ;
    clk3 : out std_logic ;
    rst1 : out std_logic ;
--     rst2 : out std_logic ;
--     rst3 : out std_logic ;
    sel01 : out std_logic :='0';
    sel11 : out std_logic :='0'
--     sel02 : out std_logic:='0' ;
--     sel12 : out std_logic ;
--     sel03 : out std_logic ;
--     sel13 : out std_logic
);
end;

architecture rtl of stim_gen is
signal sel01_int , sel11_int : std_logic :='0';
begin

process_clk1 : process
begin
        clk1 <='0';
        wait for 40 ns;
        clk1 <='1';
        wait for 40 ns;
end process;

process_clk2 : process
begin
        clk2 <='0';
        wait for 50 ns;
        clk2 <='1';
        wait for 50 ns;
end process;
```

```
process_clk3 : process
begin
        clk3 <='0';
        wait for 60 ns;
        clk3 <='1';
        wait for 60 ns;
end process;

process_sel01 : process
begin
        sel01 <='0';
        wait for 2000 ns;
        sel01 <='1';
        wait for 2000 ns;
end process;

process_sel11 : process
begin
        sel11<='0';
        wait for 4000 ns;
        sel11 <='1';
        wait for 4000 ns;
end process;

process_reset : process
begin
        rst1 <='1';
        wait for 100 ns;
        rst1 <='0';
        wait for 2000 ns;
        rst1 <='1';
        wait for 100 ns;
        rst1 <='0';
        wait for 2000 ns;
        rst1 <='1';
        wait for 100 ns;
        rst1 <='0';
        wait for 2000 ns;
        rst1 <='1';
        wait for 100 ns;
        rst1 <='0';
        wait for 2000 ns;
        wait;
    end process;
end;
```

BIBLIOGRAPHIE

[1] Jacomme, L., (1999). *Analyse sémantique de descriptions VHDL synchrones en vue de la synthèse*. Thèse de doctorat de l'université de Paris VI

[2] Perry,D., (1998). *VHDL* (3ème éd.). New York : McGraw-Hill.

[3] Chang, K. C., (1997). *Digital design and modeling with VHDL and synthesis*. Loa Alamitos, Californie.: IEEE Computer Society Press.

[4] Zoran, S., (1998). VHDL and FPLDs *in digital systems design, prototyping and customization*. Massachusetts: Kluwer Academic Publishers.

[5] Kurup, P., Abbasi T., (1995). *logic synthesis using synopsys* (2ème éd.). Norwell, Massachussetts: kluwer Academic Publishers.

[6] Savaria, Y., (1998). *Conception et verification des circuits VLSI*. Montréal Québec : éditions de l'École Polytechnique de Montréal.

[7] Sebastien, M. J., Smith. (1997). *Application-Specific integrated circuits*. Addisson-Wesley.

[8] Ciletti, M. D., (1999*). Modeling, synthesis, and rapid prototyping with the VERILOG HDL*. New Jersey: Prentice-Hall, Inc.

[9] Tendolkar, N., Molyneaux, R., Pyron, C., and Raina, R., (2000) At-speed testing of delay faults for motorola's MPC7400, a PowerPC™ Microprocessor, *in 18th IEEE VLSI test symposium*, Montréal, Québec, Canada.

[10] Canadian Microelectronics Corporation (August 1996*). Tutorial on design flow for scan-based DFT: from RTL synthesis to layout using 0.8-micron BiCMOS technology.*

[11] Canadian Microelectronics Corporation (December 2000*). Tutorial on CMC's digital IC design flow: Instructions for taking a sample design through the digital flow (0.35-micron and 0.18-micron CMOS technologies with black-box libraries).*

[12] Press, R., Illman, R., (October 2001) ATPG Pattern Compaction The Next Wave. Mentor Graphics Publication, [En ligne]. http://www.mentor.com/dft (Page consultée en juillet 2002)

[13] Tendolkar, N., Woltenberg, R., Raina, R., Lin, X., Swanson, B., Aldrich, G., (june 2001). Scan-Based At-Speed Testing for the Fastest Chips. Mentor Graphics Publication. , [En ligne]. http://www.mentor.com/dft (Page consultée en juillet 2002)

[14] Eide, G., (October 2001) Embedded deterministic test-DFT technology for low-cost IC manufacturing test. Mentor Graphics Corporation, [En ligne]. http://www.mentor.com/ (Page consultée en juillet 2002)

[15] Press, R., Illman, R., (October 2001) ATPG pattern compaction tne next wave. Mentor Graphics Publication, [En ligne]. http://www.mentor.com/dft (Page consultée en juillet 2002)

[16] Bartley, M., (2001) *verification-it's all about confidence*. Mentor Graphics Publication, [En ligne]. http://www.mentor.com/ (Page consultée en juillet 2002)

[17] Mentor Graphics Corporation (August 2001). Testing large-capacity CAM with MBISTArchitect and fastscan macro test. Mentor Graphics Publication, [En ligne]. http://www.mentor.com/ (Page consultée en juillet 2002)

[18] Mentor Graphics Corporation (March 2001). Designs with multiple clock domains: avoiding clock skew and reducing pattern count using DFTAdvisor™ and fastscan™. Mentor Graphics Publication, [En ligne]. http://www.mentor.com/ (Page consultée en juillet 2002)

[19] Mentor Graphics Publication (January 2000). Solving the challenges of testing small embedded cores and memories using fastscan macro test. Mentor Graphics Publication, [En ligne]. http://www.mentor.com/ (Page consultée en juillet 2002)

[20] *Tutorial du cours SYS-834: Technologie VLSI et ses applications, Automne 2000*. École de Technologie Supérieure.

[21] *Note du cours VLSI (Hiver 97)* : Université du Québec à Montréal

[22] Michaël alias, (2000). Le Microprocesseur. Publication Informatech, [En ligne]. http://informatech.online.fr/articles/cpu/ (Page consultée en septembre 2002)

[23] LogicVision, Logic BIST. [En ligne]. http://www.logicvision.com/products/logic.htm (Page consultée en septembre 2002)

[24] The University of Texas at Dallas. [En ligne]. http://www.utdallas.edu/~grinnell/ee7v81/ (Page consultée en juin 2002)

[25] Cadence, Automated Custom Physical Design Methodology Service (ACPD), [En ligne]. http://www.cadence.com/infosheets/is_acpd.html(Page consultée en septembre 2002)