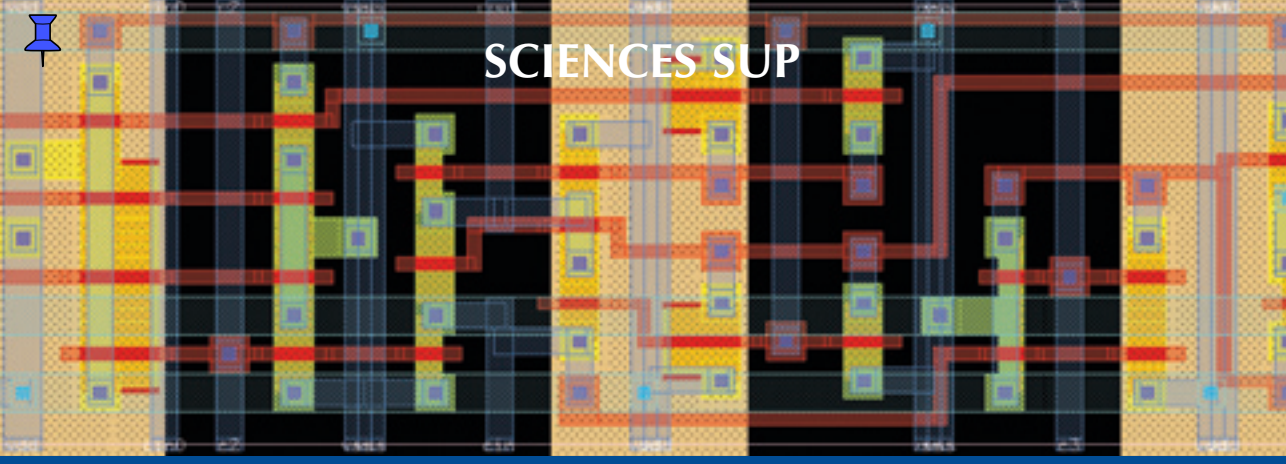


SCIENCES SUP

Cours et exercices corrigés

Master • Écoles d'ingénieurs

CONCEPTION DES CIRCUITS VLSI

DU COMPOSANT AU SYSTÈME

François Anceau
Yvan Bonnassieux

DUNOD

CONCEPTION DES CIRCUITS VLSI

Consultez nos catalogues sur le Web

www.dunod.com

CONCEPTION DES CIRCUITS VLSI

Du composant au système

François Anceau
Professeur au Conservatoire National des Arts et Métiers

Yvan Bonnassieux
Maître de conférences à l'École Polytechnique

DUNOD

Dessin des masques d'une Unité Arithmétique et Logique (UAL) - Voir chapitre 6
Développé par le laboratoire SOC/Lip6 de l'université Pierre et Marie Curie - Paris VI

ISBN 978-2-10-050036-9

Table des matières

Chapitre **1**

Introduction

L'objectif de cet ouvrage est de présenter les techniques de conception des circuits intégrés CMOS complexes. La conception de circuits électroniques logiques a débuté dans les années 1950 avec le développement du RADAR et des premiers ordinateurs. Elle s'est ensuite développée, surtout aux États-Unis, dans l'industrie et dans les départements d'Electrical Engineering des grandes universités américaines, au cours des années 1970 et 80 avec l'apparition des microprocesseurs VLSI nMOS puis CMOS. Les centres de recherche se sont intéressés à cette discipline à partir du milieu des années 1970 avec l'implication d'informaticiens qui souhaitaient réaliser des microprocesseurs [ANC86]. La publication internationale de l'ouvrage de Mead et Conway [MEA80] joua le rôle d'un véritable détonateur. Du jour au lendemain, de nombreuses équipes de recherche se mirent à dessiner des circuits. Des organisations furent mises sur pied pour permettre la réalisation de circuits universitaires. Malheureusement, tous les pays concernés ne surent pas retirer le même profit de cet élan.

La maîtrise de la conception des circuits intégrés VLSI est une condition nécessaire au développement d'une industrie électronique performante. Alors qu'une seule chaîne de fabrication permet la réalisation de nombreux circuits différents, la conception de ces circuits (souvent spécifiques aux besoins de l'industrie) nécessite de nombreux concepteurs, ce qui ouvre de larges perspectives professionnelles dans cette discipline.

L'étude de la circuiterie VLSI fait appel à tout un ensemble de notions qui ne font pas partie des cursus habituels de physique et d'informatique. Certains ne sont pas, ou plus, enseignés. Ils font partie du savoir-faire des concepteurs de circuits intégrés complexes, comme par exemple la technique des circuits polyphasés. Une étude approfondie de l'ensemble des sujets abordés nécessiterait d'écrire une encyclopédie. Nous avons donc choisi de ne traiter que les éléments importants de chaque domaine

et de négliger les domaines qui ne comportent pas de spécificité microélectronique, pour pourvoir disposer d'un point de vue global sur cette discipline et permettre ainsi son approfondissement ultérieur.

1.1 L'ÉVOLUTION TECHNOLOGIQUE

Depuis une cinquantaine d'années, l'évolution de la complexité des circuits intégrés double tous les 18 mois (loi de Moore [MOO65]). Cette évolution exponentielle a permis de réaliser, de manière monolithique, des organes électroniques de plus en plus complexes qui étaient auparavant réalisés sous la forme d'armoires (par exemple : des processeurs, des mémoires, des commutateurs téléphoniques…).

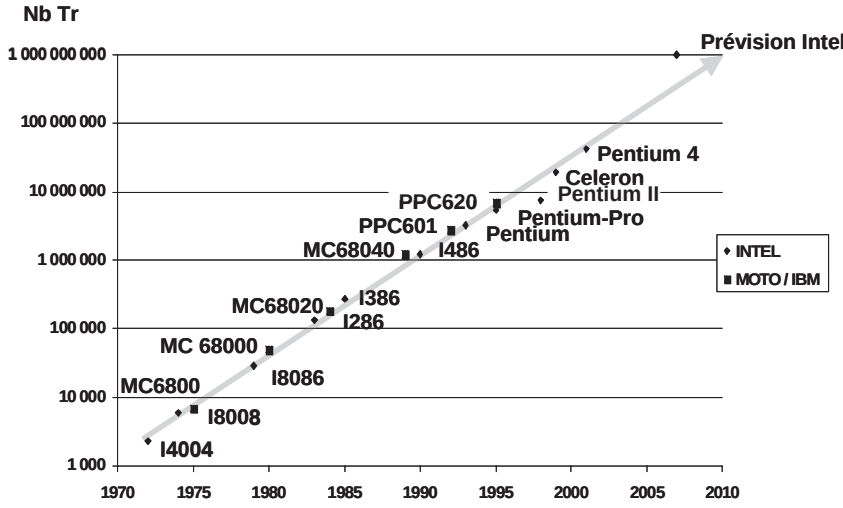


Figure 1.1 Évolution exponentielle de la complexité des microprocesseurs (illustration de la loi de Moore)

Le principal moteur de cette évolution réside dans la diminution régulière de la taille des motifs de dessin des circuits intégrés. Partis de quelques dizaines de microns dans les années 1960, ceux-ci sont maintenant de 60 nm en 2006, et tout montre que cette évolution n'est pas terminée.

La capacité de l'industrie microélectronique à poursuivre cette évolution est proprement incroyable. Elle surprend tout le monde, y compris les experts. Ceux-ci, réunis au sein d'une organisation appelée SIA (*Semiconductor Industry Association*), publient régulièrement des prédictions (appelées ITRS pour *International Technology Roadmap for Semiconductors*) qui s'avèrent systématiquement sous-évaluées pour un futur qui dépasse trois ans, c'est-à-dire l'horizon de leurs recherches. La meilleure prédiction est encore le simple prolongement du passé, aussi incroyable qu'il puisse être.

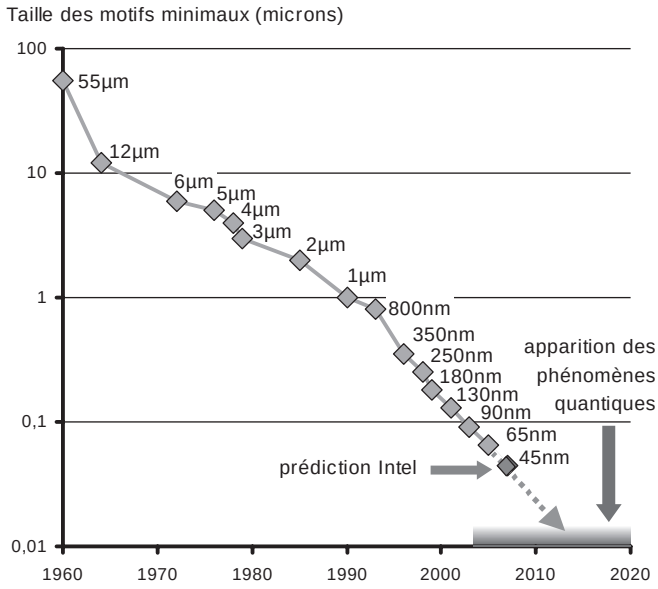


Figure 1.2 Évolution de la taille des motifs minimaux de la technologie

1.2 L'ÉVOLUTION DES CIRCUITS INTÉGRÉS

L'évolution des circuits intégrés est certainement l'aventure technologique la plus fabuleuse de l'histoire humaine. L'ampleur des progrès réalisés dépasse de loin tout ce qui a été fait dans les autres domaines, y compris l'aviation et le spatial.

Les circuits intégrés utilisent deux types de composants actifs, appelés « *transistors* » :

– Les transistors à *effet de champ*, proposés par J.E. Lilienfeld en 1925-1928 [LIL33], mais pratiquement réalisés par M.M. Atalla, D. Kahng et E. Labate fin 1959. L'idée maîtresse de ces composants était la transposition à l'état solide d'une triode. Ceux-ci ont été appelés FET puis MOS-FET, puis MOS. Ils ont successivement été réalisés avec des grilles métalliques (technologie PMOS grille alu) puis avec des grilles en polysilicium (technologie nMOS) puis sous forme complémentaire (technologie CMOS).

– Les transistors dits « *bipolaires* », découverts sous une première forme (transistors à pointes) par J. Bardeen et W.H. Brattain aux laboratoires Bell le 23 décembre 1947 [BAR50], puis sous leur forme définitive (transistors à jonctions) en 1948 par W. Shockley [SHO76] au terme d'une étude théorique. Contrairement aux transistors à effet de champ, dont le débit est commandé par une tension, les transistors bipolaires se comportent comme des amplificateurs de courant.

La plus grande facilité de fabrication et d'utilisation des transistors à effet de champ leur a permis de devenir les composants fondamentaux des circuits complexes à partir de 1975.

Les premiers circuits intégrés furent réalisés, quasi simultanément par Jack Kilby chez Texas Instrument (le 12 septembre 1958) [KIL64] et par Robert Noyce chez Fairchild [NOY61]. Après un combat juridique, la paternité de cette première réalisation fut attribuée à Jack Kilby.

Les premiers circuits intégrés étaient réalisés en technologie bipolaire. Ils comportaient quelques dizaines de transistors. Cette technologie fut principalement utilisée pour créer les premières familles de composants logiques. En particulier, la famille TTL (pour *Transistor-Transistor Logic*) ou 74xx qui a perduré jusqu'à maintenant transposée en technologie MOS sous la forme de composants discrets et de cellules de circuits intégrés complexes. Vers 1970 les circuits intégrés complexes en technologie PMOS grille alu commencèrent à apparaître. Leur lenteur limitait leur domaine d'application à des mémoires puis aux premiers microprocesseurs. Ted Hoff conçut le premier microprocesseur commercial (Intel 4004) en 1972. L'arrivée de la technologie nMOS vers 1974 permit la mise sur le marché des premiers microprocesseurs de grande diffusion 8 bits puis 16 bits (Intel 8080, 8085, 8086 ; Motorola 6800, 6809, 68000 ; Zilog Z80, Z8000 ; MOS 6502), et de leurs composants associés. Pour lutter contre l'augmentation de la dissipation thermique, la technologie nMOS fut remplacée par la CMOS au début des années 1980, ce qui permit de poursuivre l'aventure jusqu'aux microprocesseurs géants actuels regroupant plusieurs centaines de millions de transistors et fonctionnant à des fréquences de plusieurs gigahertz.

Le passage d'une technologie à la suivante n'est pas aussi immédiat qu'il y paraît. Une fois que la faisabilité des nouveaux composants est établie, il faut encore apprendre à utiliser cette technologie. Par exemple, prise naïvement, la conception manuelle de circuits CMOS se solde par beaucoup de connexions qui occupent de la place et réduisent fortement la densité. Lorsque de bons principes d'organisation topologique furent trouvés, cette technologie permit de réaliser des circuits presque aussi denses que ceux réalisés avec la technologie nMOS précédente.

La saga des circuits intégrés se heurte actuellement à une nouvelle épreuve : la dissipation des circuits les plus puissants devient prohibitive. En effet, le passage d'une génération de produits à la suivante (par exemple dans le cas des microprocesseurs) nécessite un gain de performance qui résulte de trois facteurs indépendants :

– une nouvelle architecture plus performante ;
– une circuiterie plus rapide ;
– une technologie plus performante.

Or, l'augmentation de performance due aux deux premiers facteurs se traduit par une augmentation exponentielle de la consommation. Celle due à l'amélioration technologique se fait à consommation constante (voire décroissante). Si l'on recherche la performance maximale, le résultat se solde toujours par une augmentation de la consommation. Par unité de surface, cette dissipation a dépassé en 1996 celle des plaques de cuisson et elle se dirigeait résolument vers celle du cœur des centrales nucléaires.

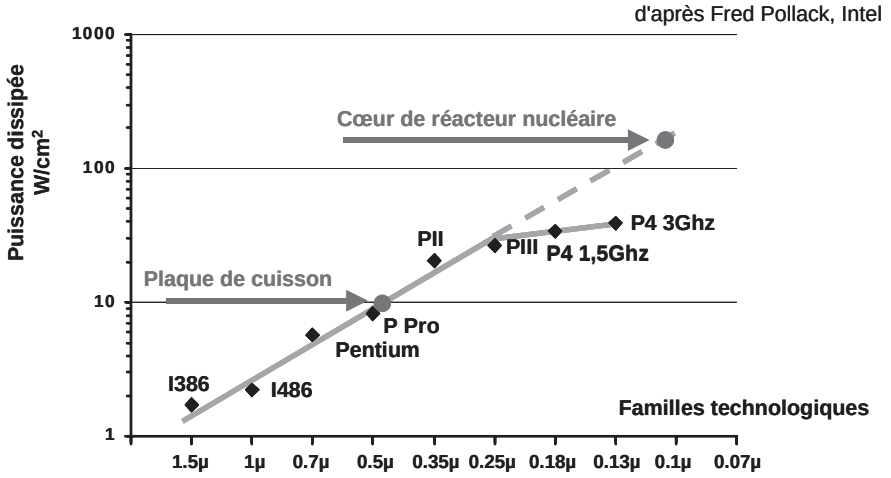


Figure 1.3 Évolution de la puissance dissipée par unité de surface des processeurs Intel X86 (d'après [POL99])

La poursuite de cette course aurait nécessité l'utilisation de dispositifs de refroidissement très coûteux. Heureusement, l'avenir de ces circuits complexes se situe en grande majorité dans des appareils portables (téléphone, gadgets électroniques, ordinateurs, baladeurs musicaux et vidéo…), ce qui a permis de changer de cible commerciale et de bénéficier d'un sursis momentané.

1.3 PETIT HISTORIQUE DE LA CIRCUITERIE LOGIQUE

Les premiers circuits logiques furent des circuits asynchrones à relais (l'additionneur Model K de G.R. Stibitz en 1937). Cette technologie issue de la téléphonie se développa entre les années 1930 et 1950. Des montages à tube à vide furent aussi utilisés pour le traitement en temps réel des images Radar dans les années 1940 et 1950. La différence importante de tension entre l'entrée (la grille) et la sortie (l'anode) des tubes à vide compliquait beaucoup la réalisation de couplages directs entre les étages successifs. Cela conduisit à utiliser des couplages capacitifs et une représentation des grandeurs logiques sous la forme d'impulsions qui véhiculaient simultanément les valeurs et les instants d'occurrence. Les montages électroniques de cette époque se comportaient comme un réseau de fonctions parcourues par des impulsions d'activation. La grande nouveauté des ordinateurs s'est située dans la réutilisation d'un petit ensemble de fonctions pour effectuer une grande variété de tâches.

Les ordinateurs conçus au début des années 1960 étaient asynchrones et continuaient à utiliser la logique à impulsions, malgré le fait que l'utilisation de transistors permit de réaliser des couplages directs (comme par exemple dans la circuiterie RTL (pour *Resistor-Transistor Logic*)). Les durées allouées aux opérateurs pour fonctionner étaient déterminées par des batteries de monostables dont le réglage nécessitait un

doigté certain. Pour « assainir » le fonctionnement de ces machines, la logique synchrone à niveaux fut utilisée à partir du milieu des années 1960. Elle déboucha sur la création de grandes familles logiques comme la TTL qui perdure encore jusqu'à maintenant après son passage en CMOS.

L'arrivée des circuits VLSI « complexes » dans les années 1970 posa un nouveau problème. Leur technologie, complètement nouvelle, n'était pas réalisée avec une grande précision et des variations importantes de leurs caractéristiques étaient fréquentes entre les lots de circuits. Pour s'affranchir de ce problème, une logique polyphasée, basée sur le principe des écluses, fut utilisée. De plus, comme le nombre de transistors disponibles était toujours inférieur aux désirs des concepteurs, ceux-ci inventèrent des astuces pour en utiliser le moins possible. La logique dynamique et celle d'interrupteurs firent leur entrée (ou plutôt leur retour !). Pendant ce temps, la logique monophasée était devenue le classique des électroniciens qui l'utilisaient pour réaliser des cartes.

L'augmentation constante de la complexité des circuits intégrés laissait prévoir la limite de la logique synchrone dès 1980 [ANC82]. Deux familles de solutions furent proposées :

– La logique asynchrone « moderne » fut proposée dès 1980 sous la forme de l'utilisation de signaux impulsionnels transmis de manière différentielle par un protocole de poignée de main. Après plus de 25 ans d'efforts, cette proposition académique reste un sujet d'avenir…
– Des techniques de synchronisation des circuits très complexes furent mises au point dans les années 1990 et conduisirent au développement des microprocesseurs modernes. Ces circuits sont constitués de blocs synchrones qui reçoivent une horloge centrale localement resynchronisée.

Les contraintes de test dues à la complexité des circuits et aux risques de parasitage entre les lignes métalliques, associé au fait que le nombre de transistors n'est plus une contrainte, conduisirent à l'abandon de la logique polyphasée ainsi que des techniques de logique dynamique et d'interrupteurs qui posaient des problèmes de test, mais surtout qui n'entraient pas dans la culture classique des électroniciens malgré les avantages importants qu'elles auraient pu apporter.

L'avenir de l'électronique est la réalisation de petits appareils très complexes, mais portables et de grande autonomie. Cet objectif ne peut être atteint que par de nouvelles technologies, mais aussi par de nouvelles techniques de circuiterie dont beaucoup restent à inventer.

1.4 LE MONDE INTÉRIEUR AUX CIRCUITS INTÉGRÉS

L'intérieur d'un circuit intégré complexe est un univers très différent de celui de la carte de circuit imprimé sur lequel il est monté. Toutes les échelles sont réduites par des facteurs importants, de l'ordre de plusieurs milliers. Par exemple, les dimensions avec lesquelles un circuit intégré est dessiné s'expriment, actuellement, en dizaines de nanomètres, alors que le dessin du circuit imprimé est réalisé au dixième de millimètre.

Les temps de montée, de descente, de transfert sont mesurés en picosecondes à l'intérieur du circuit intégré et en nanosecondes sur la carte. Il en est de même pour les capacités, comptées en fentofarads dans le circuit intégré et en picofarads sur la carte. Les courants s'expriment en microampères dans le circuit intégré en en milliampères sur la carte. Par contre, les signaux logiques internes à un circuit intégré peuvent atteindre des fréquences qui sont actuellement de plusieurs gigahertz, alors qu'il est difficile d'atteindre plusieurs centaines de mégahertz sur la carte.

Cette énumération pourrait être prolongée, mais il en ressort que tout est réduit dans le monde interne à un circuit intégré (sauf la complexité et les performances !) par rapport à l'électronique « traditionnelle ». En revanche, un circuit intégré représente un monde gigantesque à son échelle. Pour risquer une comparaison, il faudrait la faire soit avec une carte routière de toute l'Europe (avec une définition d'une dizaine de mètres), soit avec une tapisserie géante de 500 mètres de côté, tissée avec quatre points par centimètre. Tout cela pour montrer que les distances relatives sont gigantesques à l'échelle du circuit. L'organisation topologique d'un tel circuit relève de techniques qui sont à comparer avec l'organisation territoriale des pays.

Figure 1.4 Le dessin d'un circuit intégré complexe peut être comparé à une carte routière de l'Europe (avec des détails de 10 m).

Le passage entre le monde de l'intérieur du circuit intégré et celui de la carte sur laquelle il est monté nécessite l'utilisation de dispositifs adaptateurs importants pour réaliser les correspondances électriques et géométriques. Ces dispositifs se composent de deux parties :

– Un premier niveau d'adaptation est réalisé par une couronne d'amplificateurs et des plots de connexion qui occupent généralement toute la périphérie du circuit intégré. Cette couronne réalise l'adaptation électrique et géométrique entre le « cœur » du circuit intégré et le boîtier.

– Le boîtier réalise un second niveau d'adaptation, surtout géométrique, avec la carte.

La couronne comporte des amplificateurs multi-étages pour les sorties, des dispositifs de protection électrique pour les entrées, et des plots de connexion suffisamment gros pour que l'on puisse se connecter dessus. Vue de l'intérieur du circuit intégré, la commande d'un dispositif sur la carte nécessite une amplification et un ralentissement des signaux comparables à ceux nécessités par la commande de dispositifs électromécaniques à partir d'une carte électronique. Cela signifie que le dialogue entre deux circuits intégrés complexes est comparable au fait de passer par des relais et des servomoteurs pour interconnecter deux cartes d'électronique.

Ces importantes limitations sont des facteurs qui poussent à mettre toutes les différentes fonctions dans le même circuit intégré en augmentant ainsi sa complexité. Cette tendance est d'ailleurs incitée par l'évolution technologique qui permet une augmentation régulière de la complexité et des performances en diminuant régulièrement la taille des motifs qui permettent le dessin des circuits intégrés.

1.5 L'EFFORT DE CONCEPTION DES CIRCUITS INTÉGRÉS

La conception des circuits intégrés se fait principalement *via* deux approches. La première consiste à considérer la technologie VLSI comme une simple évolution technologique de la conception des cartes électroniques. Ces techniques de conception des circuits intégrés peuvent être vues comme une transposition de celles utilisées pour la conception des cartes électroniques. De telles approches sont utilisées par les techniques de conception « rapide » telle que celle des circuits prédiffusés et des circuits programmables (FPGA, EPLD). La seconde approche consiste à se rendre compte que la technologie VLSI possède ses propres spécificités qui peuvent être exploitées pour réaliser des circuits beaucoup plus optimisés et plus performants. La circuiterie des circuits VLSI devient alors une nouvelle discipline qui s'appuie sur de nouveaux concepts (logique d'interrupteurs, rétention dynamique de l'information, transparence des bascules, circuits polyphasés, et qui intègre les aspects topologiques qui conduisent au dessin des masques). Cette relation entre la fonctionnalité des blocs et leur topologie ouvre une dimension tout à fait nouvelle dans la conception des circuits électroniques.

La conception optimisée des circuits intégrés complexes pose un problème majeur : dessiner à la main un circuit intégré complexe revient à fournir un travail de type artisanal, c'est-à-dire dont le coût « tout compris » est à peu près d'un élément par heure. Cela signifie que le temps de dessin d'un microprocesseur moderne serait de l'ordre de 100 millions d'heures de travail ! Un tel coût pharaonique limiterait sérieusement le nombre de modèles disponibles et en augmenterait fortement le prix de vente. Les façons de réduire ce coût consistent :

– à chercher une organisation du circuit intégré qui multiplie les cellules identiques qui n'ont qu'à être dessinées une seule fois ;
– à chercher à récupérer des sous-ensembles les plus gros possibles d'un circuit à l'autre. Cette approche est très utilisée. D'une part, les constructeurs de circuits

intégrés disposent de bibliothèques de cellules qui sont à la base de leur approches de conception automatique. D'autre part, il existe un marché pour des sous-ensembles importants, appelés IP (pour *Intellectual Property*) tels que des processeurs, des mémoires, des modules systèmes, etc.

– Cette thésaurisation de dessins se heurte à l'évolution des règles technologiques qui changent tout les six mois et qui demande l'adaptation constante des dessins. Les constructeurs de circuits intégrés disposent d'équipes qui adaptent en permanence les dessins de leurs cellules de base à l'évolution de leur technologie.
– Le troisième volet de cette réduction de coût consiste à dessiner automatiquement les circuits à l'aide d'outils informatiques « intelligents ». Les outils de CAO (pour *Conception Assistée par Ordinateur*) sont parmi les outils informatiques les plus avancés. Ils sont actuellement capables de partir de la description du comportement souhaité pour le circuit et de créer niveau par niveau des descriptions de plus en plus précises de la structure du futur circuit. Toutefois, les dernières étapes sont difficiles à franchir et actuellement, le processus de conception automatique s'arrête à l'assemblage et à l'interconnexion de cellules standard dessinées manuellement. La thésaurisation des IP se fait d'ailleurs beaucoup plus au niveau de descriptions « synthétisables » qu'à celui des dessins de leurs masques.

Malheureusement, ces techniques de conception automatique des circuits intégrés découlent de celles initialement développées pour les cartes électroniques et n'utilisent pratiquement pas (pour ne pas dire pas du tout) les spécificités de la technologie des circuits intégrés. Cette option a été prise pour au moins deux raisons :

– Les spécificités de la technologie microélectronique sont assez différentes de celles de l'électronique traditionnelle et demandent une refonte profonde des techniques utilisées.
– Le marché visé par ces outils se veut plus large que celui des concepteurs de circuits intégrés. Il présente la microélectronique comme une simple évolution technologique capable de réaliser les mêmes montages que ceux réalisés sur carte. Il vise donc à permettre aux électroniciens d'utiliser la technologie microélectronique en conservant leur « espace culturel ».

Avec ces outils, un circuit intégré peut être dessiné pour un coût très inférieur à celui de son dessin manuel, mais comme toute médaille a son revers, ses performances seront inférieures (d'environ 50 %) et sa taille sera supérieure (environ doublée) par rapport à une conception manuelle. Ce compromis n'a rien de surprenant, il est comparable à celui obtenu pour le logiciel pour lequel il a été tranché, depuis déjà longtemps, en faveur de l'automatisme.

La quasi-totalité des circuits intégrés est maintenant compilée. Restent les microprocesseurs géants dont environ la moitié de la surface est encore dessinée manuellement pour maintenir les performances aux limites de ce qui est possible.

Avec les outils modernes, la réalisation électronique d'un circuit intégré devient quasiment invisible pour son concepteur qui n'est concerné que par la description du comportement escompté. L'utilisation de techniques plus spécifiques à la micro-électronique redeviendrait donc possible sans perturber l'espace cognitif des concepteurs.

Toutefois, le fait que les outils actuels permettent d'obtenir des résultats acceptables et que le coût des efforts de recherche et de développement nécessaires pour mettre en œuvre ces nouvelles techniques est important, n'incite pas les fabricants de logiciels de conception à faire cet effort.

Pour justifier notre effort dans ce contexte morose, nous pensons que des notions de conception manuelle sont utiles à toute personne impliquée dans la conception (même automatique !) de circuits intégrés, un peu comme la connaissance de la programmation en assembleur peut fournir un contexte culturel à tout programmeur d'applications proches des couches basses d'un système informatique.

1.6 LES LANGAGES DE CONCEPTION

Dès la fin des années 1960, l'étude de langages d'aide à la conception des montages électroniques fut entreprise. Deux approches furent adoptées :

– La première a consisté à chercher à décrire le comportement souhaité pour de futurs circuits.
– La seconde fut liée à la recherche de la compréhension des notions de base des différents niveaux d'abstraction de l'électronique.

Les objectifs de ces langages ont été multiples dès le départ :

– spécification (contractuelle) des futurs circuits ;
– vérification de leur comportement (par simulation puis par vérification formelle) ;
– entrée des outils de synthèse automatique des circuits ;
– détermination des tests de fabrication de ces nouveaux circuits.

Les premiers langages ont été développés dans des contextes de centres de recherche. Comme les techniques de synthèse à partir du comportement n'étaient pas opérationnelles, ceux-ci se sont appuyés sur des descriptions abstraites des montages projetés. Toutefois, la compréhension de ces niveaux ne fut jamais satisfaisante. Les notions dégagées ne purent jamais se débarrasser complètement du modèle informatique sous-jacent. Ce type de langage finit par être standardisé sous la forme de VHDL et de Verilog vers le milieu des années 1980.

Le développement de méthodes de synthèse à partir de descriptions comportementales relança les efforts dans la première voie et consacra l'abandon de la seconde. Les nouveaux langages sont maintenant très proches des langages informatiques. C'est le cas par exemple de SystemC. On peut toutefois penser que la connaissance des notions de base des différents niveaux d'abstraction de l'électronique serait bien utile…

Les langages de conception ont toujours donné une base conceptuelle aux concepteurs de circuits intégrés. Leur évolution vers l'informatique est une résultante d'une certaine diminution de l'intérêt pour l'électronique au profit de l'informatique.

BIBLIOGRAPHIE

[LIL33] J.E. Lilienfeld, *Device for controlling electric current*, brevet US n° 1 900 018, 7 mars 1933.

[BAR50] J. Bardeen et W.H. Brattain, *Three electrodes circuit element utilizing semiconductive material*, brevet US n°2 524 035, 3 octobre 1950.

[NOY61] R.N. Noyce, *Semiconductor Device-and-lead Structure*, brevet US n° 2 981 877, 25 avril 1961.

[KIL64] J.S. Kilby, *Miniaturized Electronic Circuits*, brevet US n° 3 128 743, 23 juin 1964.

[MOO65] G. Moore, Cramming more components onto integrated circuits, *Electronics*, Vol. 38, n° 8, 19 avril 1965.

[SHO76] W. Shockley, The path of Junction Transistor, *IEEE Transaction on Electron Devices* (reprise de 1976), nov. 1984.

[MEA80] C. Mead et L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980, traduction : *Introduction aux systèmes VLSI*, InterEditions, 1983.

[ANC82] F. Anceau, A Synchronous Approach for Clocking VLSI Systems, *IEEE Journal of Solid-State Circuits*, Vol. SC17 n° 1, février 1982.

[ANC86] F. Anceau, *The Architecture of Micro-Processors*, Addison-Wesley, 1986.

[POL99] F. Pollack, New Microarchitectures Challenges in the Coming Generations of CMOS Process Technologies, *Micro32*, 1999.

Chapitre **2**

Du silicium à l'inverseur CMOS

2.1 SEMICONDUCTEURS

Les *semiconducteurs* sont des matériaux qui se situent entre les métaux et les métalloïdes dans le tableau de Mendeleïev. Ils sont caractérisés par le fait que leur couche électronique superficielle contient quatre électrons (ils sont de valence 4). Les principaux semiconducteurs sont le germanium, le silicium et le carbone. Actuellement, seuls le germanium et le silicium sont utilisés en microélectronique. Les alliages de métaux-métalloïdes sont aussi utilisés comme matériaux semiconducteurs (tel l'arséniure de gallium).

Le silicium est un métalloïde. C'est le principal constituant du sable. Il fut isolé en 1823 par Jöns Jacal Berzelius. Sa densité est de 2,33, son point de fusion de 1 410 °C. Comme le diamant, le silicium cristallise suivant un réseau cubique. Un cristal de silicium contient $5\ 10^{22}$ atomes au cm^3.

Pour faciliter la compréhension des mécanismes au niveau du cristal, nous utiliserons une représentation planaire du cristal de silicium.

2.1.1 Semiconducteur intrinsèque

Les semiconducteurs très purs (ayant une proportion d'impuretés inférieure à 10^{-12}) et monocristallins sont appelés *intrinsèques*. Ils sont naturellement isolants car tous les électrons de leur couche périphérique sont engagés dans les liaisons chimiques du cristal. Un très faible courant peut néanmoins les traverser car l'agitation thermique libère quelques électrons (dans une proportion de $3\ 10^{-13}$ à température ambiante) qui créent un courant. Les places libérées par les électrons libérés (appelés *trous*) se comportent comme des charges positives mobiles. Les trous se déplacent par des mouve-

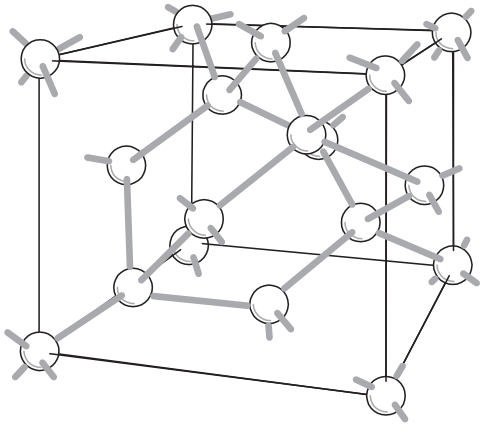


Figure 2.1 Cristal de silicium

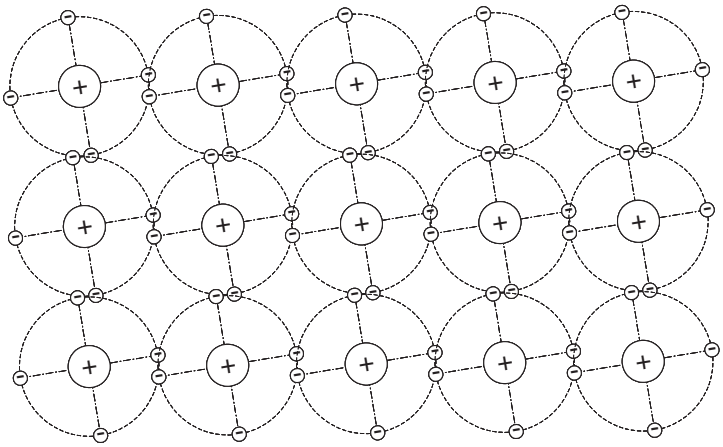


Figure 2.2 Représentation planaire d'un cristal de silicium

ments d'électrons en sens opposé. Le déplacement des trous participe aussi au courant électrique. Contrairement aux conducteurs, le courant qui traverse les semiconducteurs croît avec la température qui libère d'autant plus d'électrons qu'elle augmente. On appelle *mobilité* μ, d'un électron ou d'un trou, la vitesse moyenne qu'il acquiert sous l'effet d'un champ électrique.

$$\mu = \frac{v_{m/s}}{E_{v/m}}$$

La mobilité des électrons est environ le double de celle des trous.

2.1.2 Semiconducteur de type N

L'adjonction d'une faible proportion (10^{-7} à 10^{-4}) de matériaux de valence 5 (arsenic ou phosphore) appelé *dopant*, dans un cristal semiconducteur fait qu'un certain nombre de ses atomes sont remplacés par des atomes du dopant (figure 2.3). Ceux-ci engagent quatre de leurs électrons périphériques dans les liaisons chimiques du cristal tandis que leur cinquième électron se retrouve libéré, laissant l'atome de dopant chargé positivement, donc *ionisé*. Ces électrons mobiles rendent le matériau d'autant plus conducteur qu'il contient plus de dopant. La mobilité du silicium dopé N dépend donc du taux de concentration du dopant. Couramment, elle est d'environ 750.

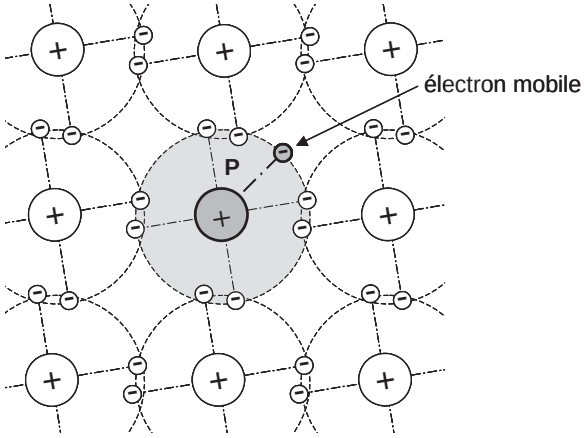


Figure 2.3 Semiconducteur de type N

2.1.3 Semiconducteur de type P

Si le matériaux dopant est maintenant de valence 3 (bore dans une proportion de 10^{-19} à 10^{-4}). Ses atomes vont s'engager dans trois liaisons chimiques du cristal (figure 2.4). La quatrième place est inoccupée. Celle-ci peut être occupée par un électron libéré par l'agitation thermique qui crée un trou qui se comporte comme une charge positive mobile. L'atome de dopant devient alors un ion négatif. Les trous mobiles rendent le matériau d'autant plus conducteur qu'il contient plus de dopant. La mobilité du silicium dopé P dépend donc du taux de concentration du dopant. Couramment, elle est d'environ 380.

2.2 DIODE/JONCTION

La coexistence, au sein d'un même cristal d'une zone de type N et d'une autre de type P produit une *jonction* à leur interface (figure 2.5). Cette jonction produit un effet redresseur. En effet, l'établissement d'une différence de polarisation dite *directe* entre ces deux zones (+ sur la zone de type P et – sur celle de type N) repousse leurs charges

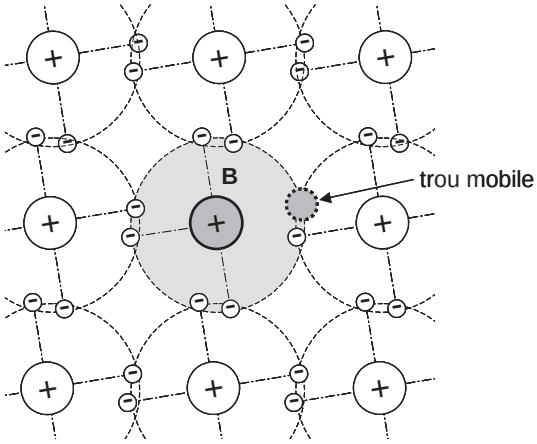


Figure 2.4 Semiconducteur de type P

mobiles respectives vers la jonction où elles s'annihilent provoquant leur réalimentation, d'où un courant (figure 2.6). Si la polarité de cette tension est inversée, chaque zone dopée va attirer ses charges mobiles qui vont s'écarter de la jonction, provoquant une zone vide de charges (dite *déplétée*), donc isolante (figure 2.7). Au repos, une certaine quantité d'électrons et de trous s'annihilent au travers de la jonction provoquant une polarisation spontanée positive de la zone de type N et négative de celle de type P. Cette polarisation repousse les charges mobiles de la jonction bloquant l'annihilation des charges. Pour qu'un courant puisse circuler, une polarisation directe de la diode doit contrecarrer cette polarisation spontanée. Cette tension, de 0,8 v pour le silicium, est appelée le seuil de la diode. Elle varie avec la température et les caractéristiques des matériaux.

La zone déplétée qui apparaît au niveau de la jonction lorsque la diode est bloquée ou polarisée à une tension directe inférieure à son seuil de diode, provoque l'apparition d'une capacité au niveau de la jonction. La valeur de cette capacité peut être assez importante. Elle dépend de la tension de polarisation.

2.3 CAPACITÉ MOS

Une capacité MOS (*Metal Oxide Semiconductor*) est constituée d'une armature métallique, d'une couche isolante et d'une autre armature en semiconducteur, appelée *substrat*, supposé ici de type P (figure 2.8).

Au repos, le substrat contient des charges négatives fixes (les atomes de semiconducteur ionisés) et des charges positives mobiles (les trous).

Supposons que le substrat soit maintenu à un potentiel fixe (0 v) et que l'électrode métallique soit portée à un potentiel négatif. Les trous vont être attirés et vont venir s'accumuler sous l'électrode métallique renforçant localement la densité des charges mobiles, d'où le type P du semiconducteur (figure 2.9).

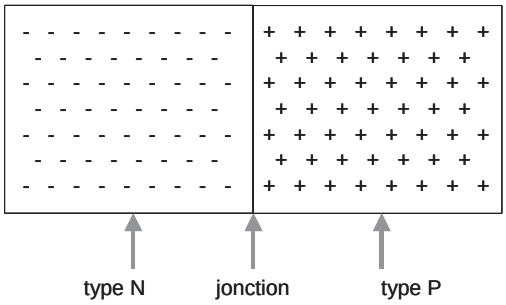


Figure 2.5 Jonction N/P

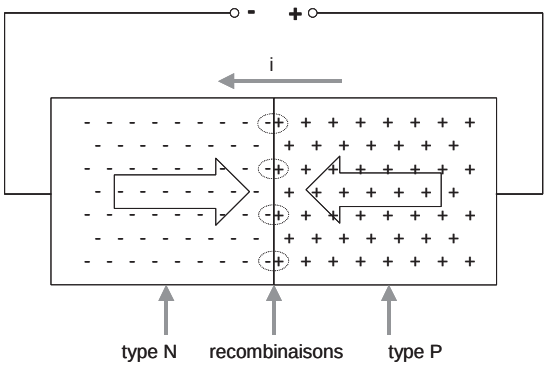


Figure 2.6 Courant direct au travers une jonction

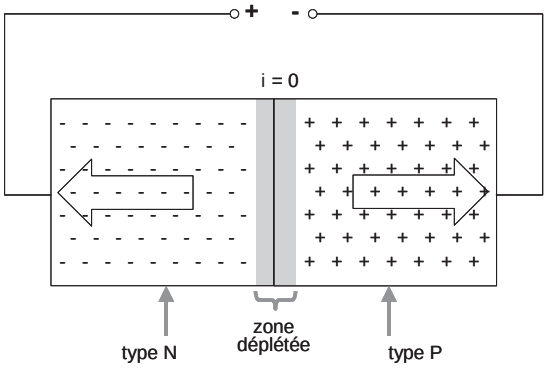


Figure 2.7 Jonction polarisée en inverse

Dans un second temps, portons l'électrode métallique à un potentiel positif. Les trous vont alors être repoussés. Ils vont quitter la zone sous l'électrode métallique.

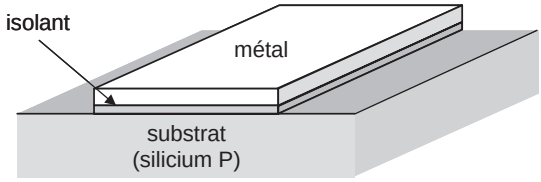


Figure 2.8 Capacité MOS

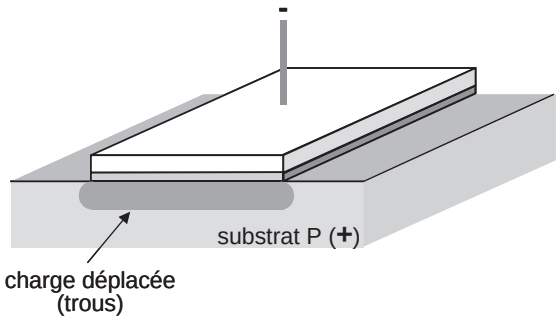


Figure 2.9 Capacité MOS (subtrat P) polarisée négativement

Lorsque le potentiel de cette dernière croît, et si le potentiel positif n'est pas trop fort, le semiconducteur sous l'électrode métallique va devenir vide de charges mobiles. Il sera dit *déplété*. Lorsque le potentiel positif de l'électrode métallique croît, celle-ci va attirer les électrons qui se libèrent par effet thermique. La densité de ceux-ci sous l'électrode métallique va devenir suffisamment importante pour inverser le type du semiconducteur qui va devenir localement de type N puisque ses charges mobiles deviennent des électrons (figure 2.10).

La tension nécessaire pour inverser le type du semiconducteur est appelée la tension de *seuil*. Elle est notée *vt*.

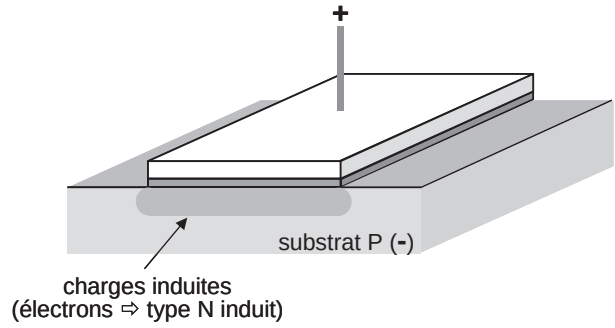


Figure 2.10 Capacité MOS (subtrat P) polarisée positivement

2.4 TRANSISTOR MOS

Un transistor MOS (aussi appelé transistor FET pour *Field Effect Transistor*) est constitué d'une capacité MOS dont l'armature métallique est appelée la *grille* (figure 2.11). Deux zones de contact en semiconducteur, de type opposé au substrat et appelées *source* et *drain* sont disposées, sur le substrat, de part et d'autre de la grille. Au repos, la source est isolée du drain par deux diodes tête-bêche. Si la grille est portée à un potentiel tel que le type du semiconducteur du substrat s'inverse, alors le drain se trouve relié à la source par un pont résistif, appelé *canal*, qui se trouve être du même type de matériaux semiconducteur que la source et le drain. Le potentiel de grille commande donc le passage du courant entre le drain et la source du transistor.

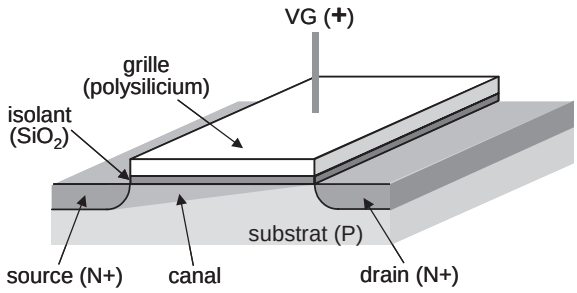


Figure 2.11 Transistor MOS de type N

Ce transistor est physiquement symétrique, toutefois son fonctionnement électrique ne l'est pas.

L'utilisation d'un substrat de type opposé permet de réaliser des transistors complémentaires qui deviendront conducteurs pour des potentiels opposés. On parlera alors de transistors N ou P suivant le type du semiconducteur induit pour constituer leurs canaux.

Ces transistors sont schématisés par les symboles suivants :

Transistor N *Transistor P*

Figure 2.12

Les symboles utilisés pour représenter les transistors reflètent leur symétrie physique. Celle-ci sera d'ailleurs utilisée dans certains montages dans lesquels la même électrode passera alternativement du rôle de source à celui de drain et inversement.

Dans les schémas électroniques, l'électrode de substrat des transistors MOS est souvent omise compte tenu du caractère systématique de sa polarisation (0 v pour les transistors N et Vdd pour les transistors P).

2.4.1 Courant dans un transistor MOS

La source est prise comme origine de tous les potentiels. Celui de la surface du semiconducteur varie à peu près linéairement le long du canal, ce qui fait que la tension de seuil locale *vt*(*x*) du semiconducteur varie de vt_0 à la source à vt_d au niveau du drain (figure 2.13).

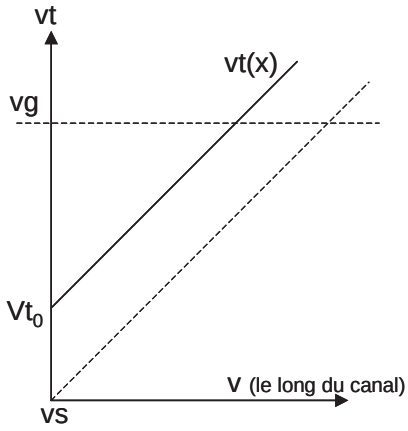


Figure 2.13 Diagramme de l'évolution de la tension *vt* le long du canal

La tension aux bornes d'une tranche *dx* de cette capacité est : (*vg* – *vt*(*x*))

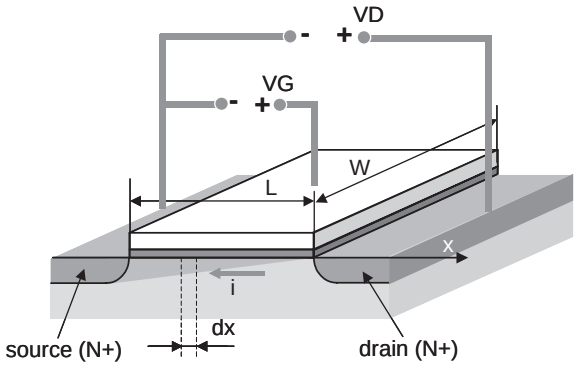


Figure 2.14 Polarisation d'un transistor MOS de type N

La charge électrique contenue dans cette tranche sera :

$$dq = \frac{\varepsilon}{e} W dx(vg - vt(x))$$

Le courant qui circule du drain à la source correspond à un déplacement de cette charge élémentaire pendant dt :

$$dq = i\,dt$$

La vitesse de déplacement des charges (électrons) est déterminée par leur mobilité :

$$dt = \frac{dx}{v} = dx\frac{dx}{\mu dv}$$

d'où :

$$i\frac{dx_2}{\mu dV} = \frac{\varepsilon}{e} W dx(vg - vt(x))$$

$$i\,dx = \mu\varepsilon\frac{W}{e}(vg - vt(x))dv$$

en intégrant le long du canal et en prenant directement *vt* comme variable, on obtient :

$$i\int_0^L dx = \mu\varepsilon\frac{W}{e}\int_{vs}^{vd}(vg - vt)\,dv$$

$$i = \mu\frac{\varepsilon}{e}\frac{W}{L}\int_{vs}^{vd}(vg - vt)\,dv$$

$\frac{\varepsilon}{e} = c_{ox}$ est la capacité d'un carré unitaire de la grille vis-à-vis du substrat, $\frac{W}{L}$ est le facteur de forme du transistor.

Cette intégration peut se faire graphiquement. Nous supposerons que *vt* varie linéairement le long du canal (figure 2.15) :

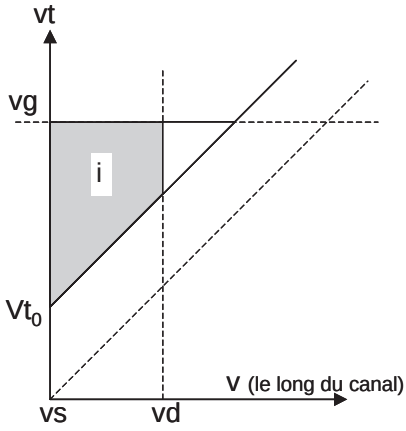


Figure 2.15

Plusieurs cas sont à considérer :

➤ *vd* petit (figure 2.16) :

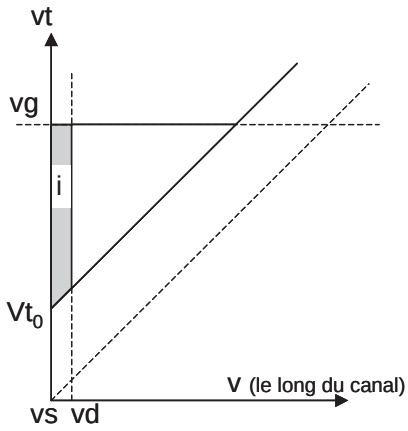


Figure 2.16

Le courant est alors de la forme :

$$i = \mu c_{ox} \frac{W}{L} (vg - Vt_0) vd$$

Cette zone de la caractéristique du transistor est appelée *linéaire*.

➤ $vd \leq (vg - Vt_0)$ (figure 2.17)

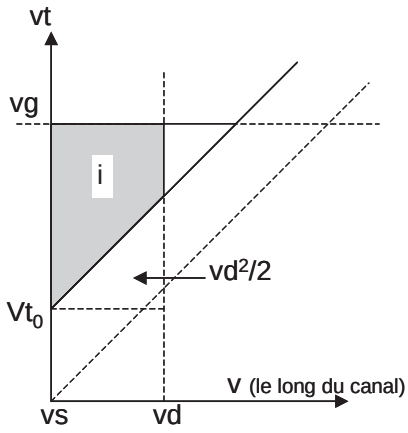


Figure 2.17

Le courant est alors de la forme :

$$i = \mu c_{ox} \frac{W}{L}\left((vg - Vt_0)vd - \frac{vd^2}{2}\right)$$

Cette zone de la caractéristique du transistor est appelée *quadratique*.

➤ $vd \geq (vg - Vt_0)$ (figure 2.18)

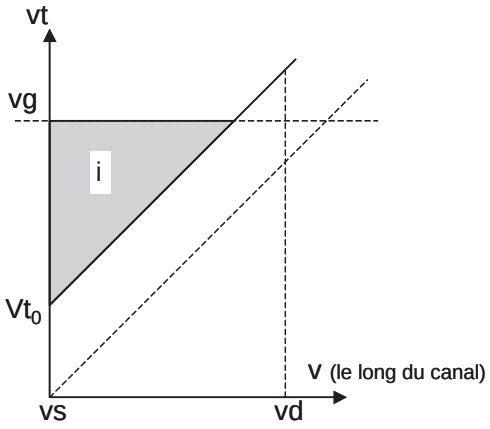


Figure 2.18

Le courant est alors constant :

$$i = \mu c_{ox} \frac{W}{L} \frac{(vg - Vt_0)^2}{2}$$

Cette zone de la caractéristique du transistor est appelée *saturée*.
La caractéristique $i(vd)$ du transistor est donc :

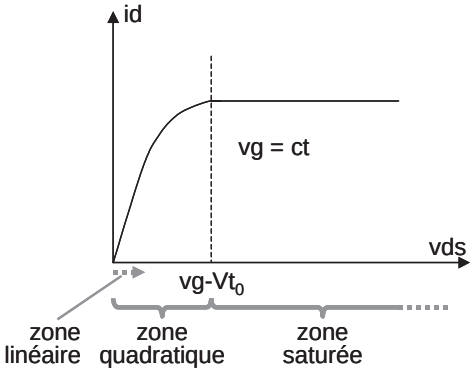


Figure 2.19

Dans la partie saturée de sa caractéristique, un transistor MOS se comporte comme un générateur de courant.

La valeur du courant dans les différentes zones de fonctionnement dépend de la capacité unitaire *cox* de la grille. Cela signifie que l'épaisseur *e* de l'isolant doit être la plus faible possible. Dans les technologies modernes, cette épaisseur n'est que d'une fraction de nm, c'est-à-dire de seulement quelques couches atomiques.

2.4.2 Cas des transistors P

Les équations qui fournissent la valeur du courant dans des différents modes de fonctionnement d'un transistor ne provoquent pas l'inversion du signe de ce courant lors de l'inversion du signe des potentiels. Elles doivent donc être considérées comme travaillant sur des valeurs absolues.

2.4.3 Effets du second ordre

La caractéristique d'un transistor réel diffère légèrement de celle calculée (figure 2.23).

– La polarisation locale de la capacité grille ne varie pas linéairement le long du canal (figure 2.20).

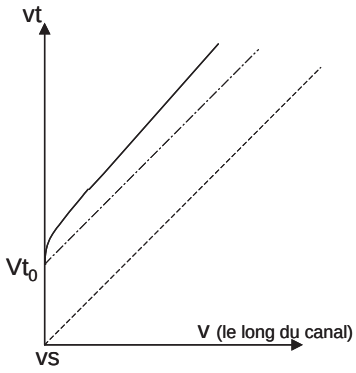


Figure 2.20

– La mobilité n'est pas constante. Elle dépend du champ électrique. C'est ce qui provoque une pente des caractéristiques *i*(*vd*) en mode saturé. Ce phénomène provient de la formation d'une zone deplétée, appelée « pinch-off », entre l'extrémité du canal et le drain (figures 2.22 et 2.23).
– La tension de seuil varie lorsque le canal est court ou étroit. Elle diminue si le canal est court et augmente s'il est étroit.

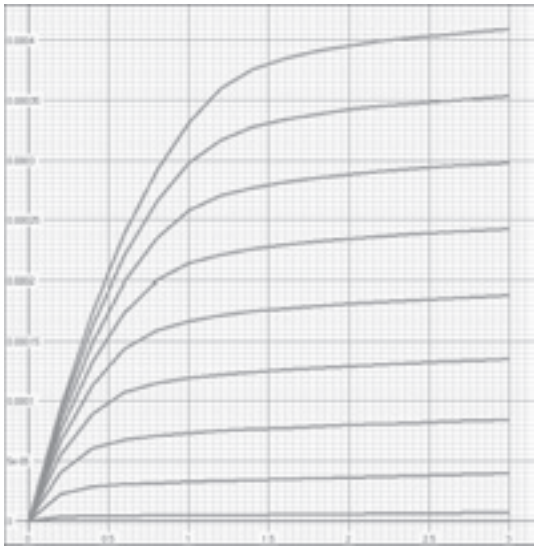


Figure 2.21 Caractéristiques d'un transistor N (L = 0,6 µm, W = 1,5 µm)

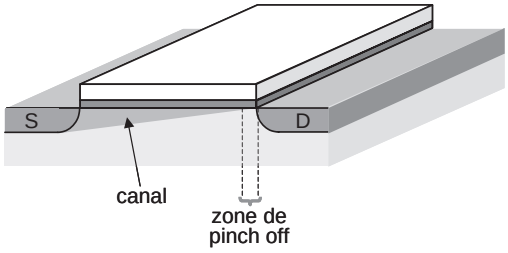


Figure 2.22 Zone de pinch-off

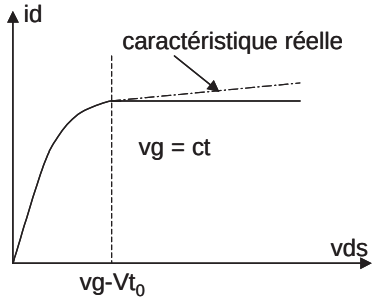


Figure 2.23

2.5 L'INVERSEUR CMOS

Un inverseur est obtenu en montant un transistor N et un transistor P tête-bêche. Les deux grilles de ces transistors sont reliées entre elles et à l'entrée du montage.

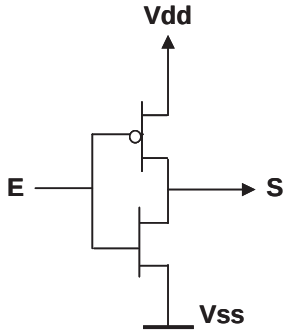


Figure 2.24

2.5.1 Caractéristique de transfert de l'inverseur

Lorsque l'on fait varier le potentiel d'entrée V_{in} du montage de 0 v à la tension d'alimentation, notée *Vdd*, sa tension de sortie V_{out} varie de *Vdd* à 0 V. On constate que lorsque V_{in} est proche de 0 v ou de *Vdd*, un seul transistor conduit et l'autre est bloqué. Cela fait que ce type de montage possède la caractéristique intéressante de ne dissiper aucune énergie au repos. Avec les technologies modernes (inférieures à 100 nm), des courants de fuite viennent obscurcir ce tableau idyllique.

Nous remarquons que les deux transistors sont toujours parcourus par le même courant et que la somme de leurs tensions *vds* est égale à *Vdd* (figure 2.25).

Considérons les différentes zones de la caractéristique de transfert (figure 2.26).

Nous noterons V_T la tension de seuil Vt_0 rendue commune aux transistors N et P.

– Zone A : TrN bloqué ($vgs < V_T$) ; $i = 0$; TrP quadratique ($Vds = 0$ v) ; $V_{out} = Vdd$.
– Zone B : TrN saturé ($vds \approx Vdd$) ; TrP quadratique ($vds \approx 0$ v).
– Zone C : TrN saturé ($vds = Vdd/2$) ; TrP saturé ($vds = Vdd/2$).
– Zone D : TrN quadratique ($Vds \approx 0$ v) ; TrP saturé ($vds \approx Vdd$).
– Zone E : TrN quadratique ($vds = 0$ v) ; $i = 0$; TrP bloqué ($vgs < V_T$) $V_{out} = 0$ v.

a) Points caractéristiques

➤ Point α

L'expression du courant commun est :

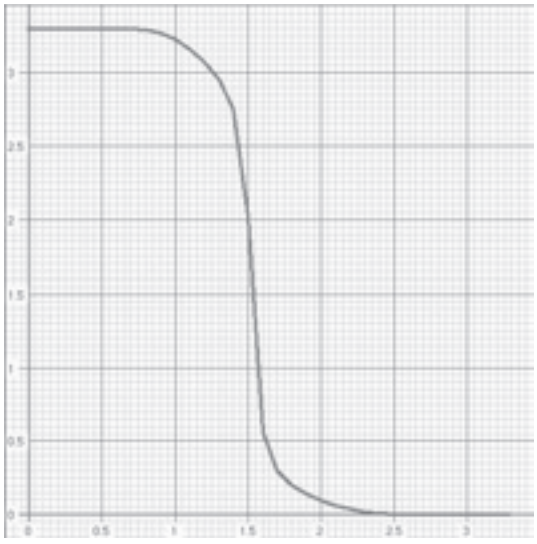


Figure 2.25 Caractéristique de transfert $V_{out} = f(V_{in})$ d'un inverseur réel

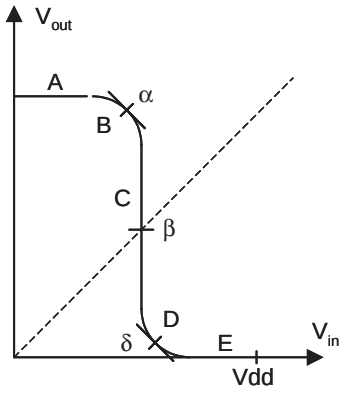


Figure 2.26

$$\alpha = \frac{1}{2}\mu_n c_{ox}\frac{W_n}{L_n}(V_{in} - V_T)^2$$

$$= \frac{1}{2}\mu_p c_{ox}\frac{W_p}{L_p}[2((Vdd - V_{\text{in}}) - V_T)(Vdd - V_{\text{out}}) - (Vdd - V_{\text{out}})^2]$$

en supposant que l'inverseur est équilibré, c'est-à-dire que :

$$\mu_n c_{ox}\frac{W_n}{L_n} = \mu_p c_{ox}\frac{W_p}{L_p}$$

si $L_n = L_p$ = dimension minimale, alors :

$$\boxed{\frac{W_p}{W_n} = \frac{\mu_n}{\mu_p}}$$

l'égalité des courants se simplifie en :

$$(V_{in} - V_T)^2 = 2(Vdd - V_{in} - V_T)(Vdd - V_{out}) - (Vdd - V_{out})^2 \qquad (1)$$

que l'on dérive par rapport à V_{in} et V_{out} :

$$2(V_{in} - V_T)dV_{in} = -2[(Vdd - V_{out})dV_{in} + (Vdd - V_{in} - V_T)dV_{out}] + 2(Vdd - V_{out})dV_{out}$$

$$V_{in} - V_T) = -\left((Vdd - V_{out}) + (Vdd - V_{in} - V_T)\frac{dV_{out}}{dV_{in}}\right) + (Vdd - V_{out})\frac{dV_{ou}}{dV_{in}}$$

Le point α est choisi de manière que pour toute tension $< V_{in\alpha}$ l'inverseur n'amplifie pas le bruit qui pourrait être superposé à son signal d'entrée. Cela correspond à la relation :

$$\frac{dV_{out}}{dV_{in}} = -1$$

Ce qui donne :

$$(V_{in} - V_T) = -2(Vdd - V_{out}) + (Vdd - V_{in} - V_T)$$

$$(Vdd - V_{out}) = \frac{Vdd}{2} - V_{in} \qquad (2)$$

Reportons dans (1) pour éliminer V_{out} :

$$(V_{in} - V_T)^2 = 2(Vdd - V_{in} - V_T)\left(\frac{Vdd}{2} - V_{in}\right) - \left(\frac{Vdd}{2} - V_{in}\right)^2$$

$$= \left(\frac{Vdd}{2} - V_{in}\right)\left(\frac{3}{2}Vdd - V_{in} - 2V_T\right)$$

$$V_T^2 = 4V_{in}V_T - 2VddV_{in} + \frac{3}{4}Vdd^2 - VddV_T$$

$$2V_{in}(Vdd - 2V_T) = (Vdd - 2V_T)\left(\frac{3}{4}Vdd + \frac{V_T}{2}\right)$$

d'où :

$$\boxed{V_{in\alpha} = \frac{1}{8}(3Vdd + 2V_T)}$$

Calcul de la tension de sortie : de 2) on déduit :

$$V_{out\alpha} = \frac{Vdd}{2} + V_{in\alpha}$$

en tenant compte de l'expression de $V_{in\alpha}$:

$$V_{out\alpha} = \frac{Vdd}{2} + \frac{1}{8}(3Vdd + 2V_T)$$

$$\boxed{V_{out\alpha} = \frac{1}{8}(7Vdd + 2V_T)}$$

qui se trouve être supérieur à $Vdd - V_T$ pour la technologie considérée (3,03 v contre 2,7 v). Cela assure que le transistor P de la porte suivante sera bien bloqué.

Calcul du courant :

$$i_\alpha = \frac{1}{2}\mu_n c_{ox}\frac{W_n}{L_n}(V_{in} - V_T)^2$$

$$= \frac{1}{2}\mu_n c_{ox}\frac{W_n}{L_n}\left(\frac{3}{4}\left(\frac{Vdd}{2} - V_T\right)\right)^2$$

$$\boxed{i_\alpha = \frac{9}{32}\mu_n c_{ox}\frac{W_n}{L_n}\left(\frac{Vdd}{2} - V_T\right)^2}$$

Le point α sera donc choisi comme le niveau maximum du niveau logique « 0 ». La pente de – 1 de la caractéristique de transfert à cet endroit assure que le niveau de bruit en sortie sera au plus égal à celui en entrée.

➤ Point β

Ce point est situé, par symétrie, au milieu de la caractéristique :

$$V_{in\beta} = V_{out\beta} = \frac{Vdd}{2}$$

d'où :

$$\boxed{i_\beta = \frac{1}{2}\mu_n c_{ox}\frac{W_n}{L_n}\left(\frac{Vdd}{2} - V_T\right)^2}$$

comme il peut aussi s'écrire :

$$i_\beta = \frac{1}{2}\mu_n c_{ox}\frac{W_n}{L_n}(V_{out} - V_T)^2$$

qui ne dépend pas de V_{in}. Le gain de l'inverseur, considéré comme un amplificateur analogique, est donc théoriquement infini. Pratiquement, les effets parasites le ramènent à une valeur voisine de 10.

➤ Point δ

L'expression du courant commun est :

$$_{\delta} = \frac{1}{2}\mu_n c_{ox}\frac{W_n}{L_n}[2(V_{in} - V_T)V_{out} - V_{out}^2] = \frac{1}{2}\mu_p c_{ox}\frac{W_p}{L_p}(Vdd - V_{in} - V_T)^2$$

Un raisonnement identique conduit à :

$$2(V_{in} - V_T)V_{out} - V_{out}^2 = (Vdd - V_{in} - V_T)^2 \quad (3)$$

que nous dérivons :

$$V_{out} + (V_{in} - V_T)\frac{dV_{out}}{dV_{in}} - V_{out}\frac{dV_{out}}{dV_{in}} = -(Vdd - V_{in} - V_T)$$

Le point δ est choisi avec des contraintes semblables à celles du point α, d'où :

$$\frac{dV_{out}}{dV_{in}} = -1$$

on obtient :

$$V_{out} = V_{in} - \frac{Vdd}{2} \quad (4)$$

que nous reportons dans (3) :

$$2(V_{in} - V_T)\left(V_{in} - \frac{Vdd}{2}\right) - \left(V_{in} - \frac{Vdd}{2}\right)^2 = (Vdd - V_{in} - V_T)^2$$

Ce qui donne, après développement :

$$2V_{in}(Vdd - 2V_T) = (Vdd - 2V_T)\left(\frac{5}{4}Vdd - \frac{V_T}{2}\right)$$

d'où :

$$\boxed{V_{in\delta} = \frac{1}{8}(5Vdd - 2V_T)}$$

Calcul de la tension de sortie : on reporte la valeur de $V_{in\delta}$ dans (4) :

$$V_{out\delta} = \frac{1}{8}(5Vdd - 2V_T) - \frac{Vdd}{2}$$

$$\boxed{V_{out\alpha} = \frac{1}{8}(Vdd - 2V_T)}$$

qui se trouve être inférieur à V_T pour la technologie considérée (0,26 v contre 0,6 v). D'où le point δ sera choisi comme le niveau minimum du niveau logique « 1 ».

Calcul du courant :

$$i_{\delta} = \frac{1}{2}\mu_n c_{ox}\frac{W_n}{L_n}[2(V_{in} - V_T)V_{out} - V_{out}^2]$$

De (3) éliminons V_{out} :

$$i_\delta = \frac{1}{2}\mu_n c_{ox}\frac{W_n}{L_n}\left[2(V_{in}-V_T)\left(V_{in}-\frac{Vdd}{2}\right)-\left(V_{in}-\frac{Vdd}{2}\right)^2\right]$$

Remplaçons V_{in} par sa valeur au point δ :

$$i_\delta = \frac{1}{2}\mu_n c_{ox}\frac{W_n}{L_n}\left[2\left(\frac{1}{8}(5Vdd-2V_T)-V_T\right)\left(\left(\frac{1}{8}(5Vdd-2V_T)\right)-\frac{Vdd}{2}\right)\right.$$
$$\left.-\left(\left(\frac{1}{8}(5Vdd-2V_T)\right)-\frac{Vdd}{2}\right)^2\right]$$

qui se réécrit en :

$$\boxed{i_\delta = \frac{9}{32}\mu_n c_{ox}\frac{W_n}{L_n}\left(\frac{Vdd}{2}-V_T\right)^2}$$

On remarque que $i_\alpha = i_\delta = \frac{9}{16}i_\beta = 0{,}56 i_\beta$, ce qui est évident par symétrie.

Exemple numérique :
Pour l'inverseur donné en exemple (Vdd = 3,3 V et V_T = 0,6 V) :

$V_{in\alpha}$ = 1/8(9,9 + 1,2) = 1,39 v, mesuré à 1,22 v,

$V_{in\delta}$ = 1/8(16,5 – 1,2) = 1,91 v, mesuré à 1,77 v.

Le courant qui traverse cet inverseur est donné par la courbe suivante. Le courant maximum (i_β) est d'environ 80 µA.

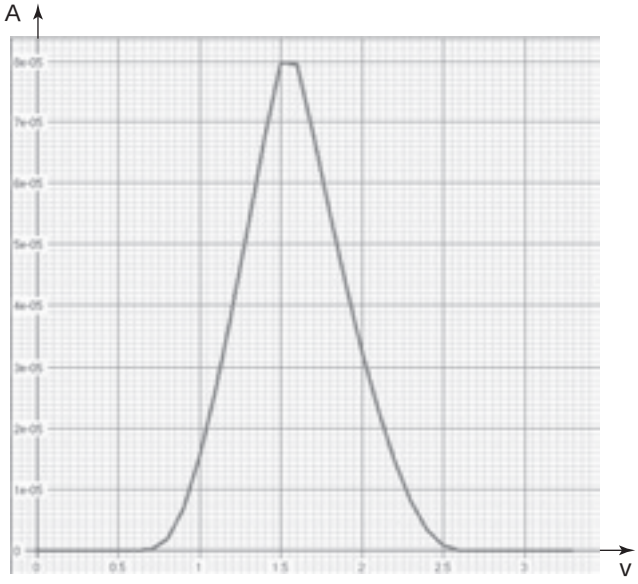


Figure 2.27 Courant dans un inverseur

2.5.2 Niveaux logiques

L'étude et l'utilisation du fonctionnement des portes se fait sur une abstraction logique de leurs niveaux d'entrée et de sortie qui représentent des niveaux de tension.

Les *niveaux logiques* $V0$ et $V1$ sont choisis de manière à assurer simultanément :

– Un « bon » blocage du ou des transistors qui doivent l'être, c'est-à-dire un niveau $V0$ inférieur à V_T et un niveau $V1$ supérieur à $Vdd - V_T$, soit pour notre exemple : $V0 < 0{,}6$ v et $V1 > 3{,}3$ v $- 0{,}6$ v $= 2{,}7$ v.

– Une « bonne » limitation de la propagation des *bruits* (parasites) dans le circuit, c'est-à-dire un niveau $V0$ inférieur à $V_{in\delta}$ et un niveau $V1$ supérieur à $V_{in\alpha}$, soit, pour notre exemple : $V_0 < 1{,}22$ v et $V_1 > 1{,}77$ v.

La contrainte du bon blocage des transistors est toujours supérieure à celle de la limitation de la propagation du bruit.

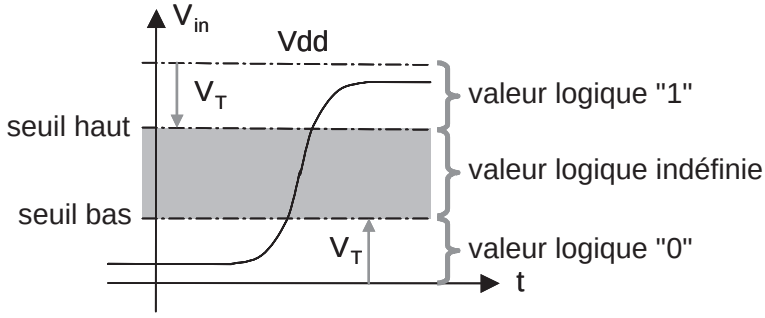


Figure 2.28

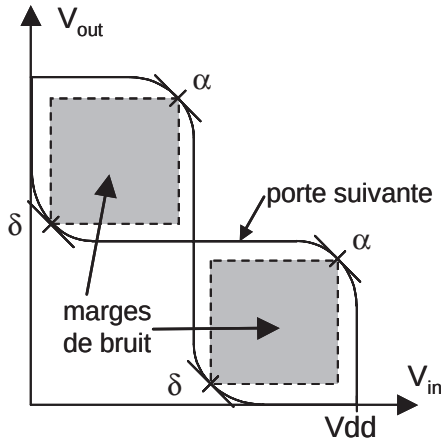


Figure 2.29 Marges de bruit

Les marges de bruit montrent quel bruit peut être superposé au niveau de sortie $V_{out\delta}$ pour qu'il n'atteigne pas le niveau d'entrée $V_{in\alpha}$ de la porte suivante, et de même pour les niveaux $V_{out\alpha}$ et $V_{in\delta}$.

2.5.3 Inverseur « minimal »

Nous appellerons « inverseur minimal » celui qui est dessiné avec les valeurs minimales des règles technologiques : la longueur de ses transistors est égale à taille du motif minimal de la technologie (ici 0,6 µm) et la largeur du transistor N égale au côté de la zone de débordement (de la zone active) d'un contact entre la zone *active* et le premier niveau de métal (ici 1,5 µm). La largeur du transistor P sera calculée de manière que :

$$\mu_n c_{ox} \frac{W_n}{L_n} = \mu_p c_{ox} \frac{W_p}{L_p}$$

soit, en supposant C_{ox} et L identiques pour les deux transistors :

$$\boxed{\frac{W_p}{W_n} = \frac{\mu_n}{\mu_p}} \text{ voisin de 2}$$

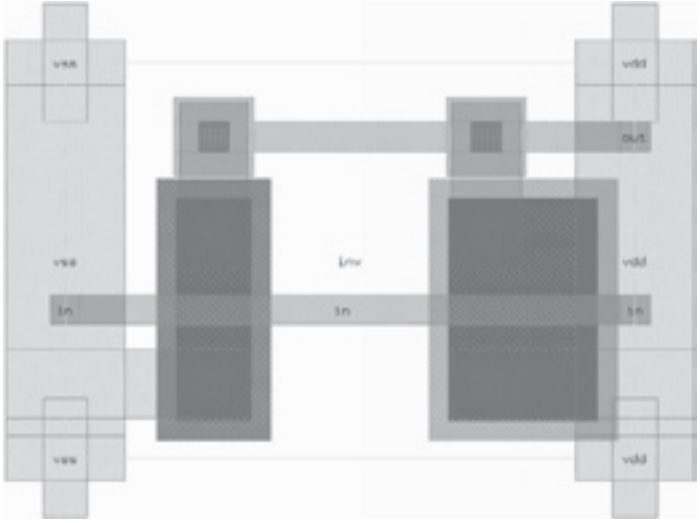


Figure 2.30 Dessin des masques d'un inverseur minimal en technologie CMOS 0,6 µm

Nous verrons que cet inverseur minimal peut jouer un rôle de référence dans le calcul des portes logiques.

2.5.4 Caractérisation dynamique de l'inverseur minimal

La caractérisation dynamique de cet inverseur consiste à déterminer ses temps de montée, de descente et de transit. Pour cela, il doit être mis en situation, c'est-à-dire attaqué par un signal identique à celui qu'il produit et chargé de la même manière que ce qu'il présente en entrée (ou par un multiple, pour l'étude de la sortance). Cette situation est obtenue en simulant une chaîne d'inverseurs et en étudiant celui disposé au centre (figure 2.31). La construction d'un modèle mathématique crédible pour déterminer ces valeurs est beaucoup trop complexe.

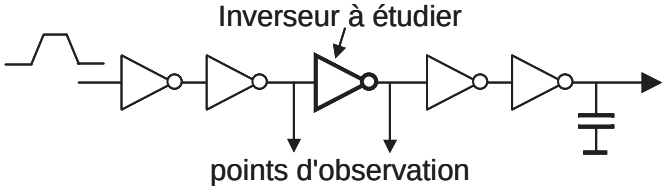


Figure 2.31 Montage de test d'un inverseur

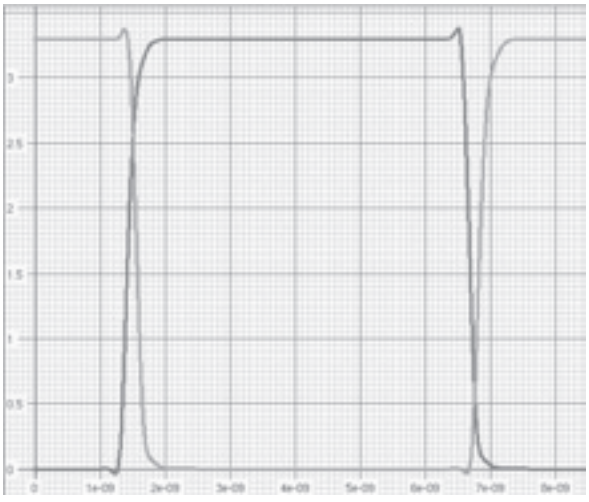


Figure 2.32 Signaux d'entrée et de sortie de l'inverseur minimal

Sur ce chronogramme, nous pouvons mesurer les temps de montée et de descente égaux à 260 ps environ et le temps de propagation égal à 150 ps environ. Une capacité de charge de 15 ff sur le dernier inverseur l'amène à générer un signal très voisin de celui des inverseurs centraux. Cette capacité correspond donc à celle d'entrée de l'inverseur.

a) Déséquilibrage des transistors

Lorsque nous nous écartons du rapport μ_n/μ_p pour dimensionner la largeur des transistors P et N, l'équilibre entre les temps de monté et de descente est perturbé. Cet effet peut être utilisé pour certaines applications spéciales.

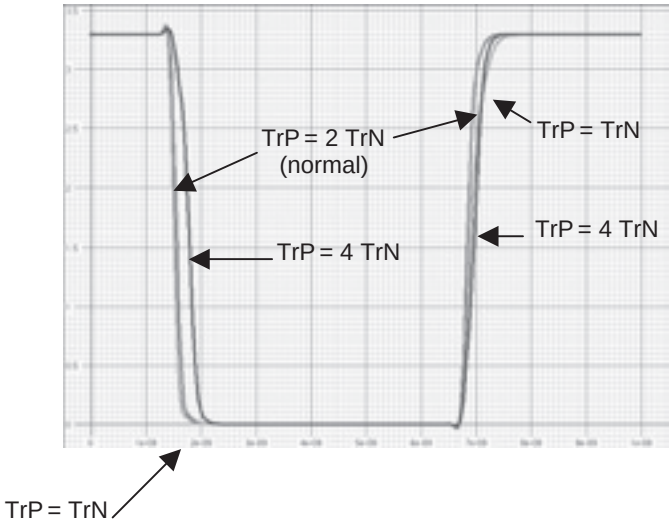


Figure 2.33 Influence de la largeur relative des transistors N et P

L'augmentation de la largeur du transistor P finit par ralentir la descente du signal de sortie de la porte modifiée car cet élargissement augmente la capacité de sortie de la porte.

b) Temps de propagation en fonction de la charge

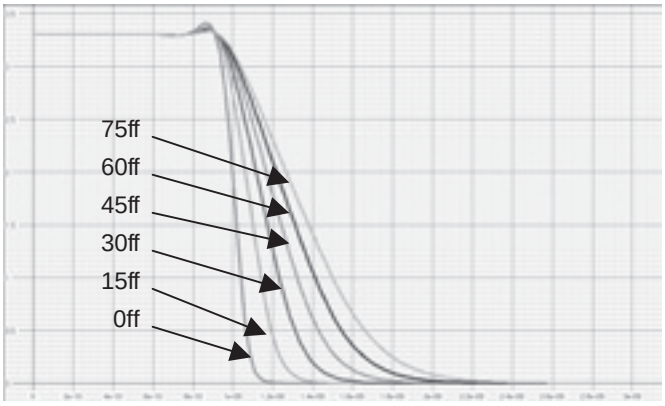


Figure 2.34 Signal de sortie d'un inverseur minimal en fonction de sa charge capacitive

Nous constatons que le temps de propagation d'un inverseur est proportionnel à la somme de sa charge capacitive et de sa capacité de sortie.

$$\tau = k(Cs + Cch) \tag{3}$$

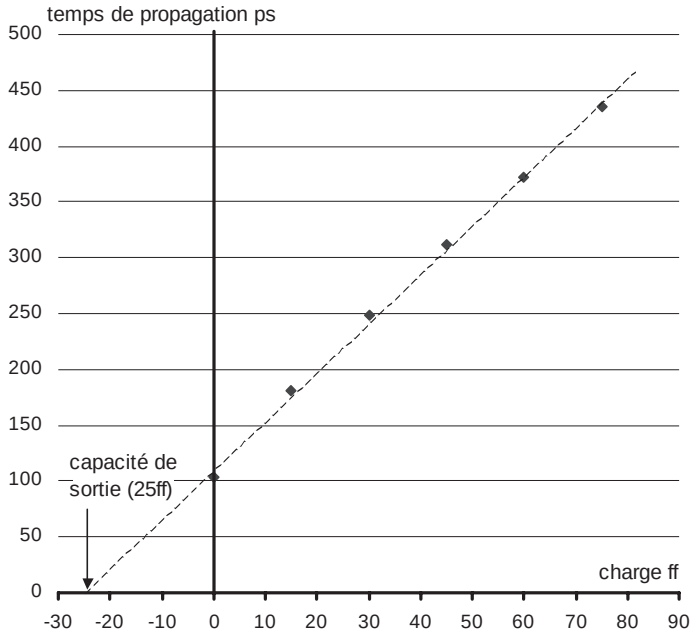


Figure 2.35 Évolution du temps de propagation de l'inverseur en fonction de sa charge

En prolongeant cette courbe, nous voyons que l'impédance de sortie de l'inverseur contient une capacité d'environ 25 ff reliée à *Vss*.

c) Détermination de la sortance

La sortance correspond à la capacité d'un inverseur à en piloter plusieurs de même type. Ce paramètre se détermine de plusieurs manières :

– Par l'étude de la déformation du signal de sortie :

Au fur et à mesure qu'il est chargé, le signal de sortie se déforme. La charge maximale correspond à la déformation maximale et au temps de propagation qui restent acceptables.

– Par l'étude de la capacité de *reformatage* de l'inverseur, ou de la porte suivante :

Le reformatage est la capacité d'une porte à régénérer un signal correct à partir de la sortie déformée de la porte précédente. La limite de cette capacité fournit une mesure de la sortance de l'inverseur étudié.

Pour l'inverseur étudié à titre d'exemple, la déformation de son signal de sortie et l'évolution de son temps de propagation deviennent importantes pour une charge

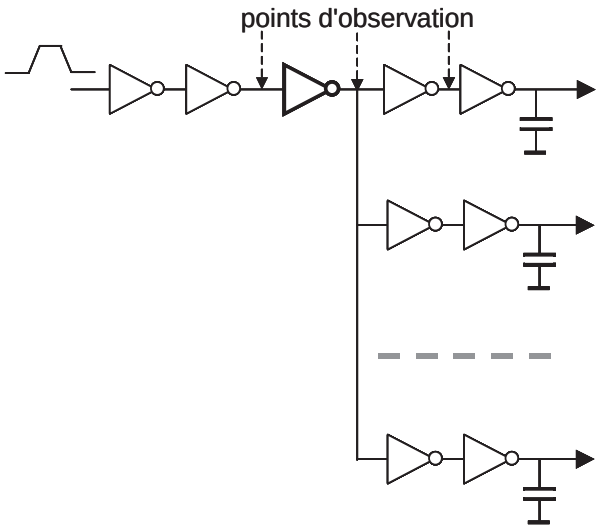


Figure 2.36

constituée de 8 inverseurs du même type. Toutefois, les inverseurs qui constituent la charge sont encore tout à fait capables de reformater le signal. La sortance de cet inverseur s'avère donc être d'au moins 8.

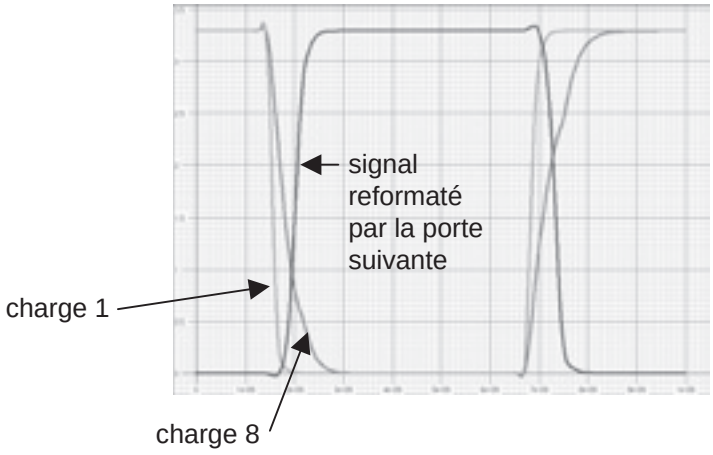


Figure 2.37 Déformation et reformatage des signaux pour une charge de 8 inverseurs de même type.

d) Influence de la taille des transistors

Si nous dessinons l'inverseur avec des transistors élargis par un certain facteur, nous multiplions leur conductante par ce facteur ainsi que les capacités parasites. Nous constatons que les signaux de sortie de l'inverseur agrandi (chargé par des inverseurs également agrandis par le même facteur) ne changent pas ! En effet, nous multiplions la sortance absolue des inverseurs par ce coefficient, mais aussi leur charge, ce qui produit des signaux de sortie identiques. Vouloir augmenter la vitesse d'une pièce de circuit en augmentant la taille de ses transistors est donc illusoire…

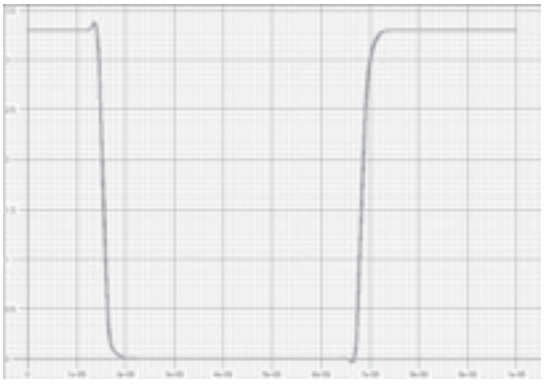


Figure 2.38 Les signaux de sortie d'un inverseur minimal et d'un inverseur dans lequel les transistors sont quatre fois plus larges, sont strictement superposables

Ce raisonnement n'est valable que pour un morceau de circuit qui ne comporte pas de charges capacitives importantes et fixes, comme des connexions très longues ou des capacités imposées. Dans ce cas, il faut évidemment adapter la taille des transistors pour que ces charges restent dans les limites de sortance acceptables pour que les portes aient des temps de propagation acceptables.

Cela montre que les performances de l'inverseur minimal constituent une limite inférieure que le dimensionnement des transistors ne permettra que d'approcher, mais pas de dépasser. L'inverseur minimal devient une sorte « d'unité de conception ». Ses performances constituent donc des « limites technologiques » pour les circuits intégrés réalisés avec cette technologie. Le dimensionnement des transistors des portes plus complexes se fera donc par référence avec ceux de l'inverseur minimal.

e) Adaptation de charge

Pour piloter des charges capacitives importantes comme des bus ou des charges externes au circuit, il est nécessaire d'utiliser plusieurs inverseurs en série pour réaliser une adaptation progressive entre les capacités de charge internes à la circuiterie et celle imposée.

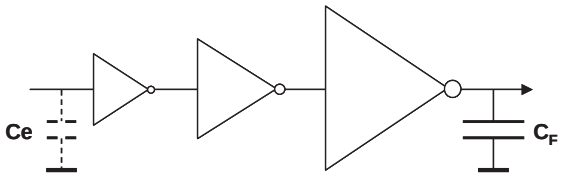


Figure 2.39 Chaîne d'adaptation de charge

Nous pouvons écrire une formule qui donne le temps de propagation d'un inverseur en fonction du rapport entre sa capacité de charge *Cch* et celle de son entrée *Ce*. Cette écriture tient compte de la linéarité de la dépendance du temps de propagation de cette porte vis-à-vis de la valeur de sa capacité de charge (formule (3) de §2.5.4.b) et de l'indépendance de ce retard vis-à-vis d'une modification homothétique de la chaîne de portes (§2.5.4.d).

$$\tau = K\left(\frac{Cs}{Ce} + \frac{Cch}{Ce}\right)$$

Posons $\gamma = Cch/Ce$ pour représenter le coefficient de charge relative de l'inverseur (appelée sortance). Dans cette formule, Cs/Ce est une constante pour un type de portes. Dans le cas de notre exemple, ce rapport vaut $25/15 = 1{,}67$ et $K = 66$ ps.

L'attaque de la capacité C_F nécessite n étages d'adaptation. S'ils ont tous le même coefficient de charge γ, leurs temps de propagation individuels sont tous identiques et égaux à :

$$\tau = K\left(\frac{Cs}{Ce} + \gamma\right)$$

D'où le temps de propagation de la chaîne d'adaptation est de :

$$T = n\tau = nK\left(\frac{Cs}{Ce} + \gamma\right) \tag{4}$$

le fait que tous les inverseurs de la chaîne aient le même coefficient de charge signifie que :

$$\frac{C_F}{Ce} = \gamma^n$$

soit :

$$\log\left(\frac{C_F}{Ce}\right) = n\log(\gamma)$$

Remplaçons n dans (4) par sa valeur :

$$T = \frac{\log\left(\frac{C_F}{Ce}\right)}{\log\gamma} K\left(\frac{Cs}{Ce} + \gamma\right)$$

$$dT = K \log\left(\frac{C_F}{Ce}\right)\left(\frac{1}{(\log\gamma)^2}\left(-\frac{1}{\gamma}\right)\left(\frac{Cs}{Ce}+\gamma\right)+\frac{1}{\log\gamma}\right)d\gamma$$

La valeur minimale de T est obtenue pour :

$$\frac{dT}{d\gamma} = K \log\left(\frac{C_F}{Ce}\right)\left(\frac{1}{(\log\gamma)^2}\left(-\frac{1}{\gamma}\right)\left(\frac{Cs}{Ce}+\gamma\right)+\frac{1}{\log\gamma}\right) = 0$$

d'où :

$$-\frac{Cs}{Ce}\frac{1}{\gamma}+(\log\gamma-1) = 0$$

soit :

$$\boxed{\log\gamma = \frac{Cs}{Ce}\frac{1}{\gamma}+1}$$

Pour l'inverseur donné en exemple, une résolution numérique donne une valeur de γ de **4,08** pour l'inverseur donné en exemple. Toutefois, une valeur de **4** peut être considérée comme utilisable pour un large ensemble de technologies.

Chapitre **3**

Fabrication des circuits intégrés

3.1 INTRODUCTION

La technologie PLANAR est aujourd'hui utilisée de manière quasiment exclusive pour la fabrication des circuits intégrés.

Les figures 3.1 et 3.2 montrent de manière rapide les différentes étapes de cette technologie. Celles-ci seront plus détaillées dans les chapitres suivants.

Partant d'une « tranche » de cristal de silicium appelée *wafer*, une première étape d'oxydation permet d'obtenir une couche de silice (SiO_2). Une seconde étape de photolithographie permet de définir la géométrie du motif à réaliser. Elle consiste en un dépôt de résine photosensible, puis son exposition à un rayonnement ultraviolet au travers d'un masque. Le développement de cette résine par une dissolution chimique permet de l'enlever là où elle a été exposée.

Une seconde attaque chimique ou physique appelée *gravure* permet d'enlever la silice là où elle n'est plus protégée par la résine. Une nouvelle dissolution permet d'enlever la résine restante.

Le dopage du substrat dans cette ouverture ce fait par une implantation ionique ou par la diffusion d'impuretés. La silice joue alors un rôle de barrière qui empêche l'implantation de ces impuretés ailleurs que dans la zone gravée.

La photolithographie est également utilisée pour graver un dépôt de métal pour réaliser les contacts et les interconnections.

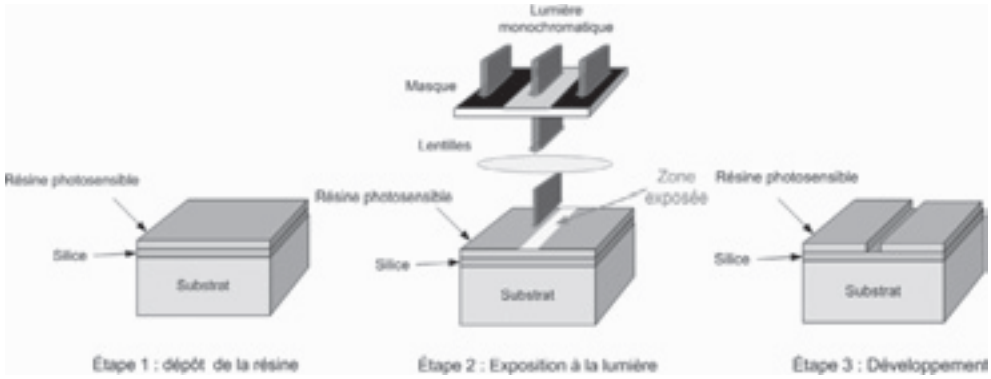


Figure 3.1 Technologie Planar (1re partie)

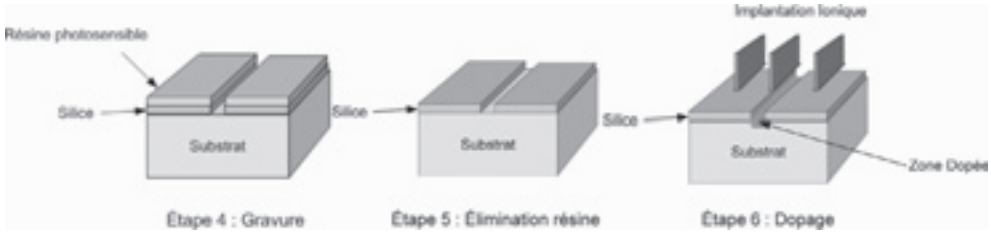


Figure 3.2 Technologie Planar (2e partie)

3.1.1 Photolithographie optique

La photolithographie est le procédé qui permet le transfert de motifs géométriques d'un masque vers une fine couche de résine photosensible qui recouvre une tranche de semi-conducteur. Ces motifs définissent les différentes régions d'un circuit intégré telles que : les zones de dopage, les connections métalliques, les points de contacts, etc. Les motifs de résine définis par la photolithographie ne sont pas permanents. Ils servent seulement de masques protecteurs pour les gravures des dispositifs qui constituent les circuits.

a) Le masquage

La plupart des équipements de lithogravure utilisés pour la fabrication des circuits intégrés utilisent des lampes ultraviolettes (de longueur d'onde 0,2 à 0,4 µm). Les étapes de lithographie doivent être réalisées dans des conditions de très grande propreté. Celles-ci sont réalisées dans des *salles blanches*. Dans ces dernières, la quantité de particules et leur taille par unité de volume est parfaitement maîtrisée ainsi que la température et l'humidité.

La photolithographie optique est réalisée par des machines appelées *masqueurs*. La résine photosensible, déposée en une fine couche sur les tranches est insolée par

son exposition, pendant un temps donné, par une lampe ultraviolette à travers une lentille de collimation.

Il existe trois méthodes de masquage :

– Le masquage par *contact* qui nécessite un contact direct entre la tranche recouverte de résine photosensible et le masque comme le montre la figure 3.3. Cette méthode peut entraîner des dégradations des masques surtout en présence de poussières.

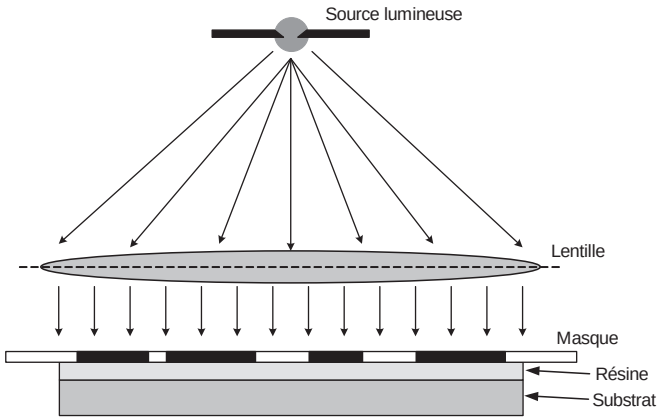


Figure 3.3 Masquage par contact

– Le masquage de *proximité*, dans lequel une faible distance (50 µm) est laissée entre le masque et la résine. Cette méthode engendre des phénomènes de diffraction qui dégradent la résolution des motifs. Elle est utilisable pour transférer des motifs de taille minimum de 2 µm.
– Le masquage par *projection* (figure 3.4). La réalisation d'un masque qui permette le transfert de motifs submicroniques sur une tranche entière est pratiquement impossible. Pour cela, les masqueurs modernes (appelés *photorépéteurs*) projettent sur la tranche l'image d'un masque d'un seul circuit, en la réduisant grâce à une optique. Cette projection sera répétée pour chacun des circuits réalisés. À chaque pas, un alignement est nécessaire avec les motifs déjà réalisés sur la tranche.

Les lampes utilisées par les masqueurs sont en général des lampes à arc à vapeur de mercure car elles fournissent une grande intensité lumineuse avec une grande stabilité, elles permettent d'atteindre une résolution maximale de 300 nm. Pour atteindre des résolutions supérieures on utilise des lasers solides (de type EXIMER) qui permettent, grâce à des longueurs d'onde de 157 nm d'atteindre une résolution de 70 nm. Pour atteindre des résolutions encore supérieures, de nouvelles approches restent à inventer.

La longueur d'onde de la lumière ultraviolette utilisée est supérieure à la taille des motifs à transférer. Cela introduit des déformations dans les motifs projetés. Pour compenser cet effet, le dessin des masques est modifié par calcul.

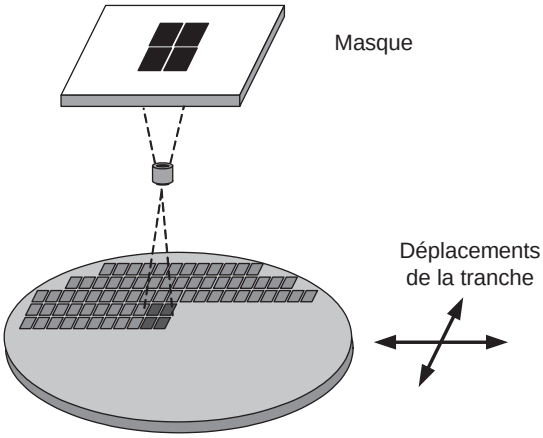


Figure 3.4 Masquage par un photorépéteur

Les dessins des masques, issus du système de conception (CAO), sont utilisés pour piloter un système de lithographie par faisceau d'électrons pour réaliser les masques physiques qui seront projetés sur la tranche. Chacun de ces masques est constitué d'un substrat de silice recouvert d'une couche de chrome.

Il existe deux types de résines photosensibles dites *positives* et *négatives*. Celles-ci se caractérisent par leur différence de réponse aux radiations ultraviolettes. Pour les résines positives, seules les parties protégées par le masque ne seront pas éliminées par le développement. Les résines négatives ont un comportement inverse.

L'étalement de la résine sur la tranche se fait par force centrifuge, à l'aide d'une machine appelée *tournette*.

3.2 SÉQUENCE DE FABRICATION D'UN INVERSEUR CMOS

Nous décrirons, dans ce chapitre, les principales étapes de la réalisation d'un inverseur CMOS. Nous présenterons en parallèle les éléments clefs des technologies utilisées.

Nous décrirons, pour des raisons pédagogiques, le procédé de fabrication dit LOCOS (*LOCal Oxydation of Silicon*) qui se caractérise principalement par le fait que de l'oxyde épais (appelé FOX pour *Thick field OXyde*) est déposé sur toutes les régions non actives du circuit.

3.2.1 Fabrication des tranches de silicium

La fabrication des tranches de silicium est une activité distincte de celle des circuits intégrés. Elle est réalisée dans des usines spécialisées.

a) Croissance du monocristal de silicium

Le silicium est le matériau de loin le plus utilisé pour la réalisation de circuits intégrés. Le matériau de départ est un type particulier de sable très pur (SiO_2) nommé *quartzite*. Après divers traitements thermiques et chimiques, on obtient du silicium de qualité électronique. Celui-ci se caractérise par une concentration d'impuretés inférieure à une partie par milliard. Un monocristal géant (actuellement de 300 mm de diamètre pour un mètre de longueur) est ensuite réalisé par la technique dite de *Czochralski* qui utilise un appareil nommé « extracteur de cristal » (figure 3.5).

De manière synthétique, on peut décrire le processus de croissance du cristal de la manière suivante :

– On place dans le creuset du silicium de qualité électronique que l'on chauffe jusqu'à sa température de fusion.
– Un petit cristal de silicium (le germe) est suspendu sur un support tournant. Ce germe est ensuite partiellement plongé dans le silicium en fusion.
– On commence alors à retirer lentement le germe du silicium en fusion (tirage). Un refroidissement progressif permet la croissance d'un grand cristal ayant la même orientation cristalline que le germe.

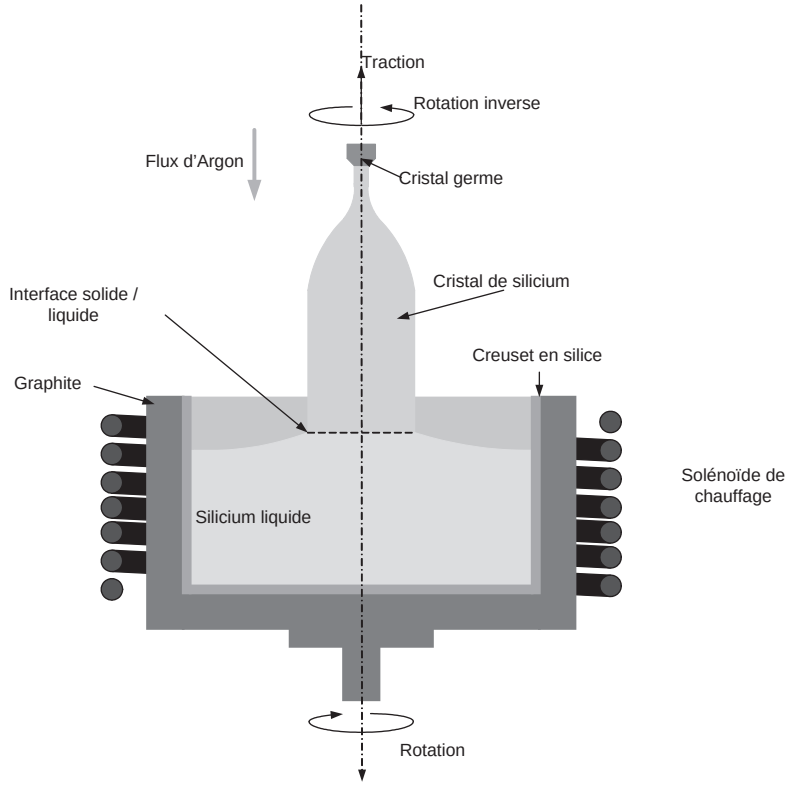


Figure 3.5 Croissance d'un monocristal de silicium par la méthode de Czochralski

La vitesse de croissance classique du monocristal est de l'ordre de quelques millimètres par minute. Pour obtenir des cristaux de silicium de grand diamètre, on ajoute un champ magnétique, qui permet le contrôle de la concentration des défauts, des impuretés et de l'oxygène. Lors de la croissance du cristal, une quantité connue de dopant (bore ou phosphore) est mélangée au silicium en fusion pour obtenir un cristal ayant le dopage désiré. Le monocristal est ensuite épuré par une technique de fusion localisée parcourant le cristal du germe à la base. Celle-ci est ensuite séparée par sciage. Le monocristal est ensuite scié en tranches d'une fraction de millimètre d'épaisseur. Une fois polies et oxydées, ces tranches constitueront le matériau de départ de la réalisation collective des circuits intégrés.

3.2.2 Étape 1 : réalisation du caisson N

La réalisation d'un circuit intégré CMOS démarre à partir d'une tranche de silicium P (substrat), c'est-à-dire d'une tranche de silicium contenant des impuretés constituées d'atomes de bore. La réalisation des transistors MOS de type P nécessite, au préalable, la réalisation de zones dopées N que l'on nomme *caissons N*. Il est à noter que dans les technologies submicroniques il est d'usage de faire aussi des caissons P pour les transistors de type N (pour améliorer le contrôle du dopage du canal de ces transistors).

La réalisation des caissons N se décompose en cinq sous-étapes dont les trois premières constituent la base de la photolithographie.

– Dépôt de la résine photosensible.
– Masquage de cette résine.
– Gravure de l'oxyde de silicium.
– Élimination de la résine.
– Implantation ionique de la zone N et recuit.

a) Masquage de la résine

Les tranches de silicium sont livrées oxydées. Cet oxyde sera gravé par *photolithographie* pour servir de masque pour l'implantation ionique qui formera les caissons. On commence par déposer une couche de résine photosensible sur la tranche à l'aide d'une tournette. Cette résine sera insolée à travers le masque du caisson N (figure 3.6).

b) Développement de la résine

Le développement enlève la résine qui n'a pas été insolée. Seule la zone correspondant au futur caisson N laisse apparaître la silice (figure 3.7).

c) Gravure de la silice

La silice est ensuite gravée par attaque chimique ou par plasma (figure 3.8). On obtient une zone de silicium mis à nu correspondant au caisson N. Le reste de la tranche reste recouvert de silice, donc protégé. La résine est ensuite enlevée par dissolution.

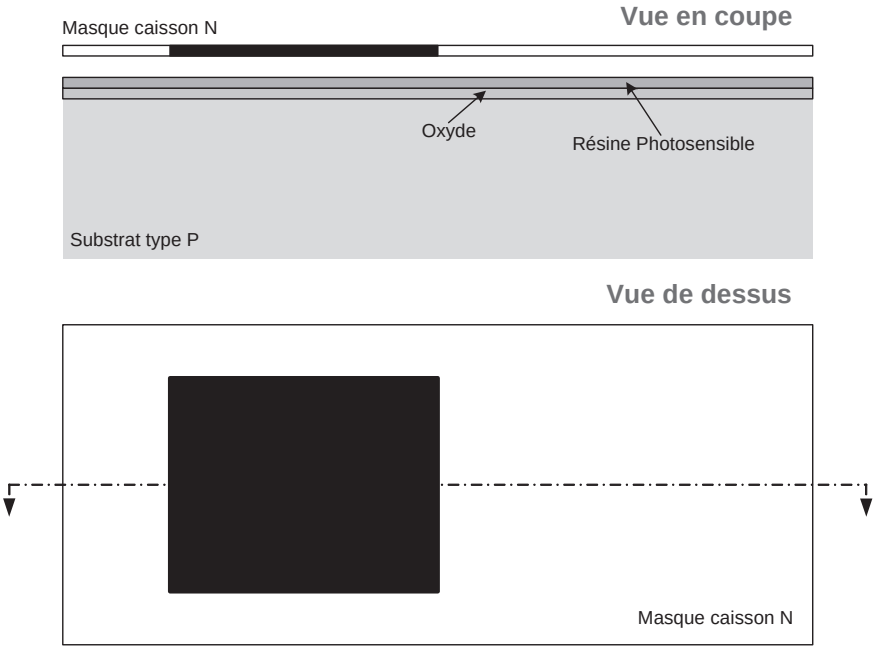


Figure 3.6 Étape 1 (masquage de la résine)

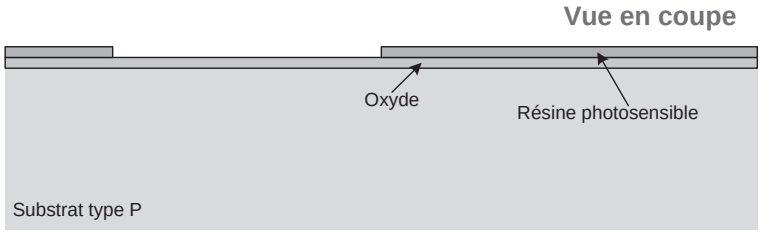


Figure 3.7 Étape 1 (développement de la résine)

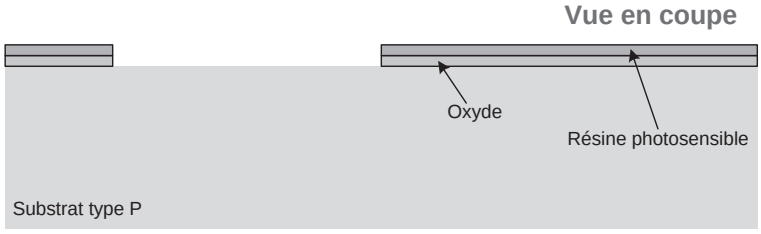


Figure 3.8 Étape 1 (gravure de la silice)

d) Implantation ionique de phosphore

Le caisson est maintenant réalisé par implantation ionique de phosphore (figure 3.9). Seule la zone non protégée par la silice est dopée. Cette dernière réalise donc un masque qui défini proprement la zone du caisson N.

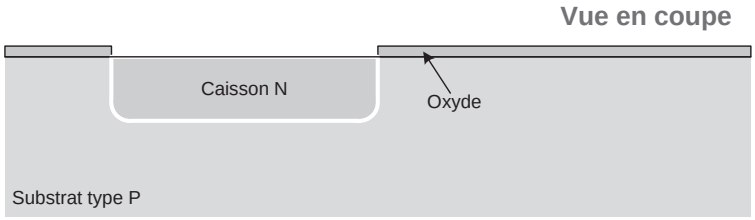


Figure 3.9 Étape 1 (implantation de phosphore)

e) Technologies associées à l'étape 1

➤ Gravures

La gravure va permettre de reproduire les motifs des masques sur les couches minces situées en dessous. En fonction des matériaux à graver et des dessins à réaliser, on trouve deux types de gravure :

– une gravure humide qui utilise un produit chimique ;
– une gravure sèche qui utilise les plasmas.

L'un des problèmes de la gravure est sa sélectivité. En effet, on remarque que les solutions chimiques utilisées ne gravent pas qu'un seul matériau mais les attaquent tous à des vitesses différentes. La sélectivité d'une gravure se mesure par le rapport des vitesses de gravure des différents matériaux. Les gravures au plasma présentent une directivité qui permet une grande précision dans la reproduction des motifs.

➤ Implantation ionique

Le dopant est ici implanté grâce à flux d'ions. Sa concentration est fonction de la masse des ions et de l'énergie d'implantation. Classiquement, on travaille avec des énergies comprises entre 1 KeV et 1 MeV ce qui donne des profondeurs d'implantation comprises entre 10 nm et 10 µm avec une densité de dopant comprise entre 10^{12} et 10^{18} ions/cm^2. Le principal intérêt de cette méthode est sa précision et sa répétitivité. Elle peut, en outre, se faire à température ambiante.

La source d'un implanteur ionique contient un filament permettant de casser les molécules de gaz de dopant (PH_3, BF_3…). Un potentiel de 40 kV appliqué par des électrodes permet d'accélérer ces ions et de créer le flux. Un champ magnétique permet de trier les ions désirés. Un tube d'accélération avec un champ réglable de 180 kV permet de choisir le niveau d'énergie voulu pour réaliser l'implantation. Des plaques de déflexion et de focalisation permettent de diriger le flux sur la tranche.

La collision des ions avec le cristal de silicium engendre des dégradations de la structure cristalline qu'un recuit permet de compenser en grande partie. Lorsqu'ils

pénètrent dans le semi-conducteur, les ions perdent leur énergie par une série de collisions avec les électrons et les noyaux des atomes de silicium. Ces collisions peuvent engendrer des déplacements d'atomes de silicium qui se propagent sous la forme de défauts dans la structure du réseau cristallin.

3.2.3 Étape 2 : préparation des zones actives

La seconde étape consiste en la préparation des zones actives, c'est-à-dire des zones qui seront ultérieurement dopées N ou P et qui correspondent aux drains et aux sources des transistors MOS (N et P) ainsi qu'aux contacts des caissons.

La préparation des zones actives se décompose en sept sous-étapes :

– Dépôt de nitrure de silicium (SiN).
– Dépôt de la résine photosensible.
– Masquage et développement de cette résine.
– Gravure du SiN.
– Élimination de la résine.
– Croissance de l'oxyde de champ.
– Élimination du SiN.

a) Dépôt et gravure du nitrure de silicium

La première de ces sous-étapes consiste à réaliser un dépôt par plasma de nitrure de silicium Si_3N_4. Celui-ci servira de masque pour la future croissance de l'oxyde de champ. La seconde sous-étape consiste à déposer une couche de résine photosensible à l'aide d'une tournette. Cette couche de résine sera ensuite insolée à travers le masque des zones actives puis développée (figure 3.10).

Le nitrure de silicium est ensuite gravé par plasma. Le silicium est mis à nu à l'extérieur des zones qui correspondent aux futures régions actives et aux contacts de caisson. Celles-ci sont recouvertes, donc protégées, par du nitrure de silicium (figure 3.11).

b) Croissance de l'oxyde de champ

Les zones non protégées par le SiN sont oxydées par oxydation thermique. La couche épaisse de silice ainsi obtenue isole électriquement entre eux les différents éléments du circuit (figure 3.12). Cette couche est appelée *oxyde de champ*.

Le nitrure de silicium est ensuite enlevé pour mettre à nu les zones actives et permettre ainsi la réalisation des étapes suivantes.

c) Technologies associées à l'étape 2

➤ Dépôt de diélectriques

Le dépôt de couches minces de diélectrique est utilisé principalement pour la protection des composants.

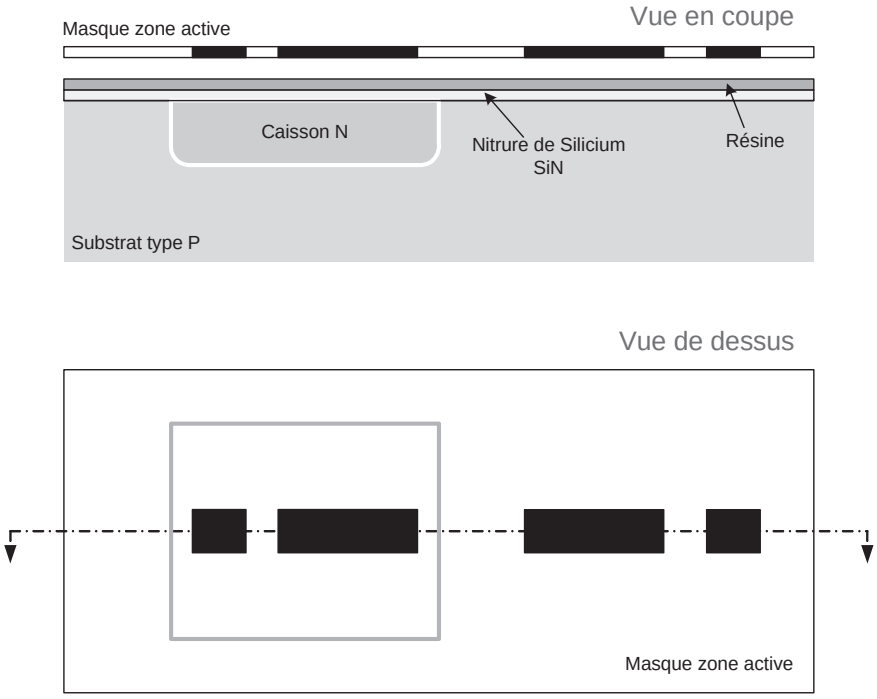


Figure 3.10 Étape 2 (masquage de la résine)

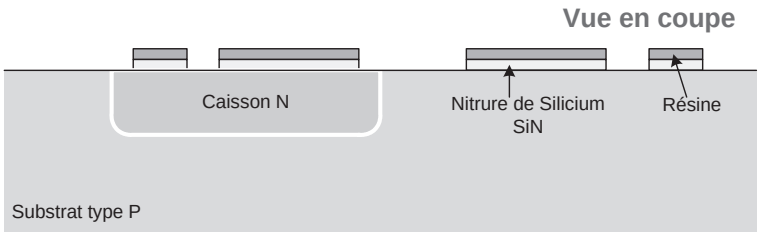


Figure 3.11 Étape 2 (gravure du SiN)

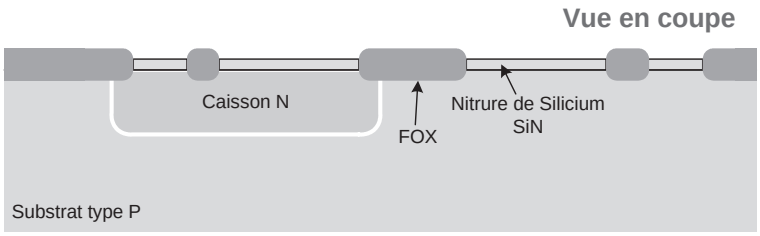


Figure 3.12 Étape 2 (croissance de l'oxyde de champ)

Il existe trois méthodes classiques de dépôt en phase vapeur (CVD pour *Chemical Vapor Deposition*) :

– Dépôt chimique en phase vapeur à pression atmosphérique (APCVD pour *Atmospheric CVD*).
– Dépôt chimique en phase vapeur à basse pression (LPCVD pour *Low Pressure CVD*).
– Dépôt en phase vapeur avec assistance par plasma (PECVD pour *Plasma Enhanced CVD*).

Pour la PECVD, l'énergie du plasma s'ajoute à l'énergie thermique d'une CVD classique, ce qui permet de travailler à plus basse température (100 à 400 °C). Par contre, la capacité est limitée au niveau de la taille des tranches.

La silice obtenue par CVD est de qualité inférieure à celle obtenue par oxydation thermique. Ainsi, cette silice est utilisée de manière complémentaire. Non dopée, elle est utilisée comme isolant entre les différentes couches de métallisation ou comme masque dans une opération d'implantation ionique.

Le dépôt de nitrure de silicium peut être obtenu soit par une LPCVD vers 750 °C ou encore par une PECVD à 350 °C.

La LPCVD permet d'obtenir un nitrure de bonne qualité et de grande densité qui est utilisé comme masque d'oxydation.

Le nitrure obtenu par PECV est de moins bonne qualité et de plus faible densité. Il est utilisé comme passivation finale des circuits intégrés et comme protection contre les rayures et l'humidité.

3.2.4 Étape 3 : réalisation des grilles

La troisième étape consiste en la réalisation des grilles des transistors MOS (N et P).

La réalisation des grilles se décompose en sept sous-étapes :

– Croissance de l'oxyde de grille.
– Dépôt du polysilicium.
– Dépôt de la résine photosensible.
– Masquage et développement de cette résine.
– Gravure du polysilicium.
– Gravure de l'oxyde de grille.
– Élimination de la résine.

a) Croissance de l'oxyde mince

Cette étape commence par la croissance thermique de l'oxyde mince sur toute la surface de la tranche (figure 3.13). Cet oxyde doit d'être de très bonne qualité diélectrique tout en étant extrêmement mince (< 4 nm pour la technologie 90 nm) car il va servir d'isolant pour les grilles.

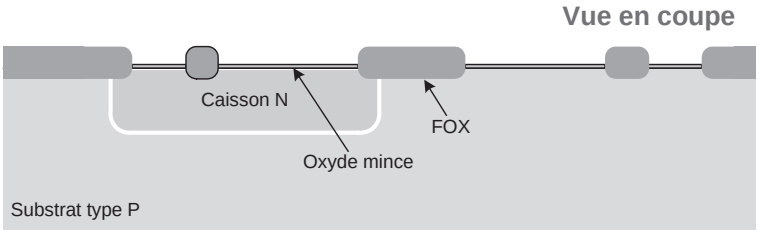


Figure 3.13 Étape 2 (croissance de l'oxyde mince)

b) Dépôt et gravure du polysilicium

Une couche de polysilicium est déposée par LPCVD puis gravée par photolithographie pour obtenir les grilles des transistors (et les connexions en polysilicium) (figures 3.14 et 3.15).

c) Gravure de l'oxyde mince

L'oxyde mince est éliminé par une gravure plasma sauf sous les zones protégées par le polysilicium (concept d'auto-alignement) (figure 3.16). On a ainsi réalisé les grilles de transistors MOS (P et N).

d) Technologies associées à l'étape 3

➤ Oxydation thermique du silicium

La silice obtenue par oxydation du silicium est de très bonne qualité. Elle est utilisée pour l'isolation électrique et la protection de surface mais aussi pour sa facilité de fabrication. Cette caractéristique est la principale raison de l'utilisation du silicium dans les composants actuels.

Le réacteur d'oxydation se compose d'un tube de quartz horizontal chauffé par effet joule. Dans ce tube on place les tranches verticalement. Un flux continu d'oxygène pur ou de vapeur d'eau (pour les couches épaisses) parcours ce tube pendant le chauffage.

Une autre propriété intéressante de la silice est sa capacité à être utilisée comme masque sélectif contre la diffusion des dopants à haute température. En effet, la vitesse de diffusion dans la silice est 2 à 3 ordres de grandeur plus faible que dans le silicium. Cette caractéristique est très utilisée dans le processus de fabrication des circuits intégrés.

➤ Dépôt du polysilicium

L'utilisation du polysilicium pour réaliser les grilles des transistors MOS s'est généralisé car il surpasse les métaux précédemment utilisés, du point de vue de la fiabilité.

Le polysilicium est obtenu par LPCVD à 600/650 °C par pyrolyse du silane sous une pression de 25 à 130 Pa.

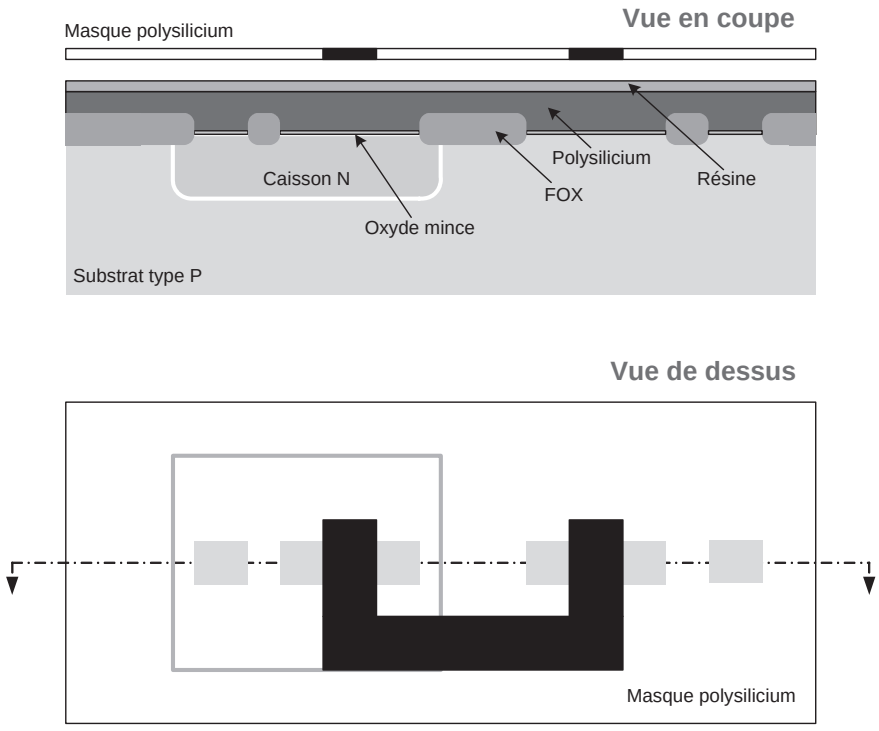


Figure 3.14 Étape 3 (masquage du polysilicium)

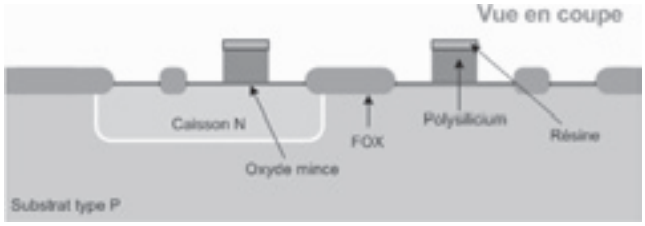


Figure 3.15 Étape 3 (gravure du polysilicium)

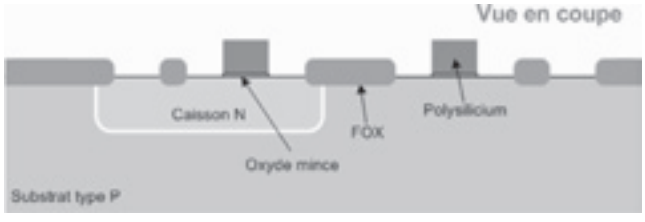


Figure 3.16 Étape 3 (Gravure de l'oxyde mince)

3.2.5 Étape 4 : dopage des zones actives

La quatrième étape consiste en la réalisation des zones actives des transistors. Celles-ci sont dopées respectivement N+ pour les MOS N et P+ pour les MOS P.

Le dopage des zones actives se décompose en huit sous-étapes :

– Dopage des zones P+ :
  • Dépôt de résine photosensible.
  • Masquage et développement de cette résine.
  • Implantation ionique de la zone P+ et recuit.
  • Élimination de la résine.
– Dopage des zones N+ :
  • Dépôt de résine photosensible.
  • Masquage et développement de cette résine.
  • Implantation ionique de la zone N+ et recuit.
  • Élimination de la résine.

a) Masquage et développement de la résine pour les zones P+

La résine est insolée à travers le masque zone P+ (figure 3.17). Son développement laisse à découvert les zones qui doivent être dopées P+.

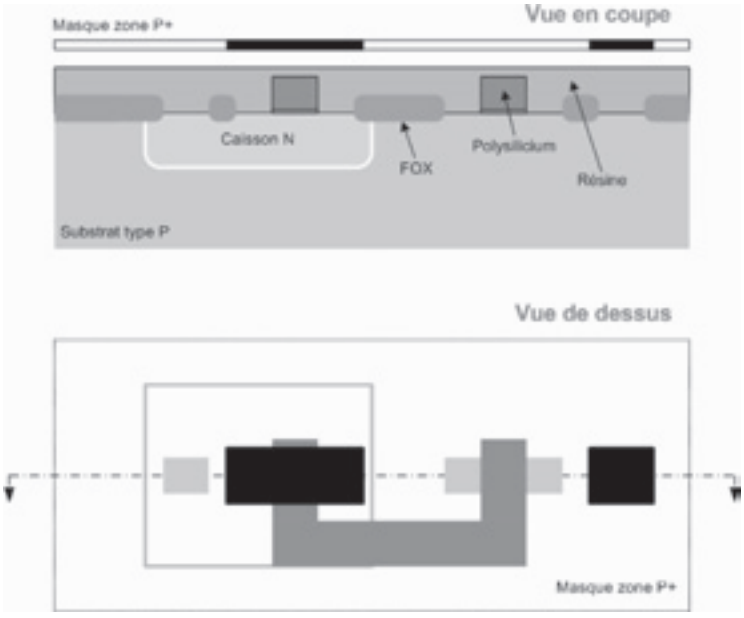


Figure 3.17 Étape 4 (masquage des zones P+)

b) Réalisation des zones P+

Les zones P+ sont réalisées par implantation ionique de bore, c'est-à-dire les sources et les drains des transistors P mais aussi les contacts du substrat dopé P (figure 3.18). La grille en polysilicium joue encore le rôle d'un masque pour délimiter précisément les sources et les drains des transistors et éviter tout recouvrement (auto-alignement).

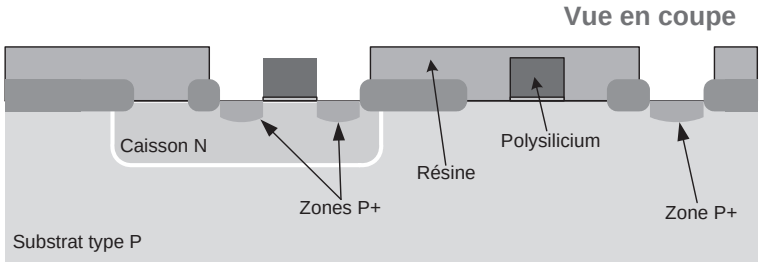


Figure 3.18 Étape 4 (développement résine et dopage P+ par implantation ionique)

c) Réalisation des zones N+

La réalisation des zones actives N+ suit exactement le même processus que celle des zones P+, si ce n'est l'utilisation d'un masque délimitant les sources et drain des transistors N et les contacts des caissons N ainsi que l'utilisation d'une implantation ionique de phosphore (figure 3.19).

3.2.6 Étape 5 : réalisation des via des contacts

La cinquième étape consiste en la réalisation des via des contacts, c'est-à-dire des trous de connexion entre les zones dopées ou le polysilicium, et le métal 1.

La réalisation des via des contacts se décompose en six sous-étapes :

- Dépôt de silice.
- Planéarisation.
- Dépôt de la résine photosensible.
- Masquage et développement de cette résine.
- Gravure de la silice.
- Élimination de la résine.

a) Dépôt de silice

Une couche de silice d'isolation est déposée sur toute la tranche par CVD.

b) Planéarisation

Pour permettre la réalisation de connexions métalliques fiables et la maîtrise des couplages capacitifs, il est important que la surface sur laquelle sera déposée le métal 1

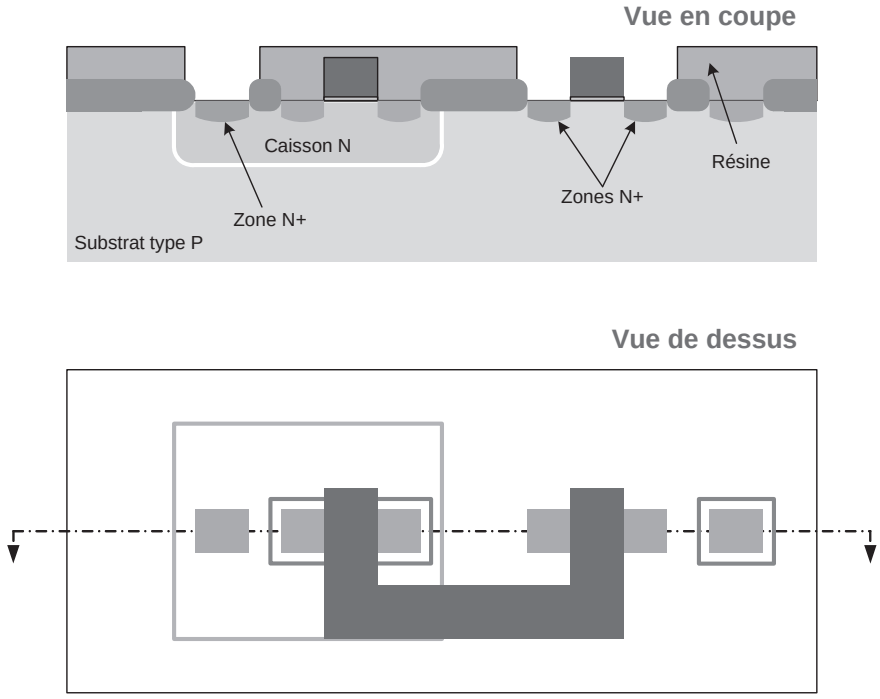


Figure 3.19 Étape 4 (développement de la résine et dopage N+ par implantation ionique)

soit parfaitement plane. Cela est obtenu par un polissage mécanique appelé *planéarisation.*

c) Masquage et développement de la résine

Voir figure 3.20.

d) Gravure de la silice

La silice est gravée par plasma jusqu'au zones actives ou jusqu'au polysilicium, dans les ouvertures de la résine (figure 3.21).

3.2.7 Étape 6 : réalisation des connexions en métal 1

La sixième étape consiste en la réalisation des connexions en métal 1.

La réalisation des connexions en métal 1 se décompose en cinq sous-étapes :

– Dépôt du métal 1.
– Dépôt de la résine photosensible.
– Masquage et développement de cette résine.
– Gravure du métal 1.
– Élimination de la résine.

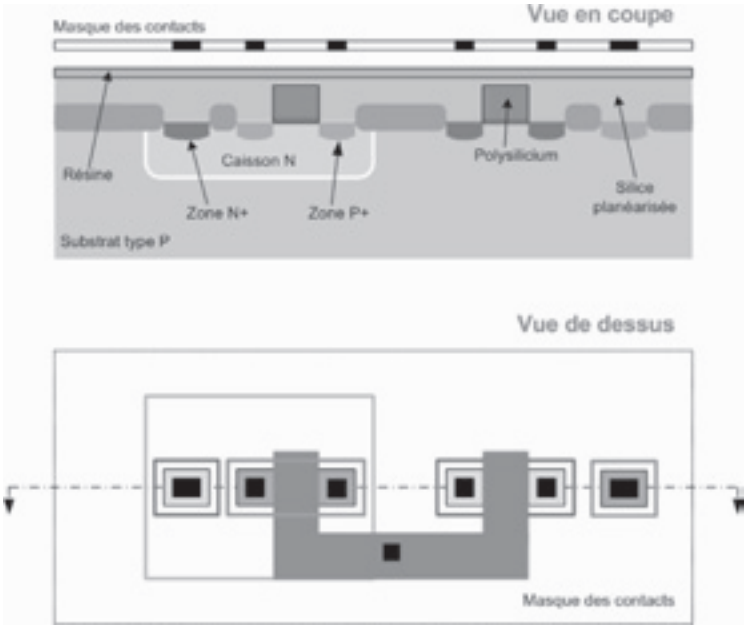


Figure 3.20 Étape 5 (masquage des contacts)

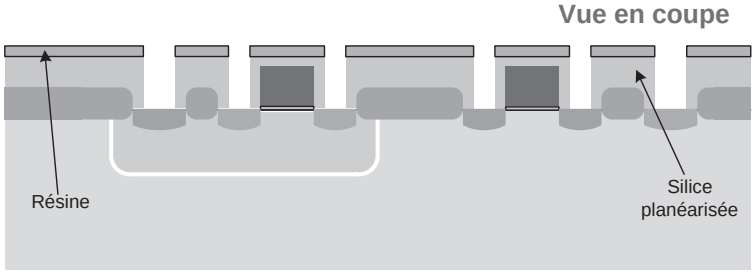


Figure 3.21 Étape 5 (développement de la résine et gravure des contacts)

a) Gravure du métal 1

Le métal 1 est déposé par pulvérisation sur l'ensemble de la tranche. Il est ensuite gravé par photolithographie à partir du masque des connexions à réaliser en métal 1 (figure 3.22).

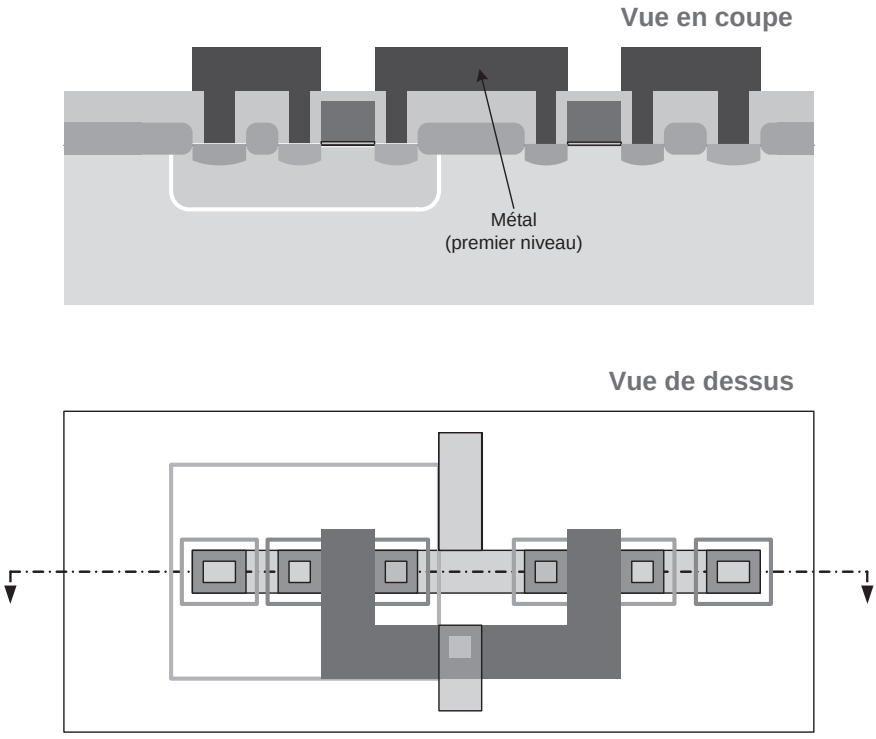


Figure 3.22 Étape 6 (gravure du métal 1)

b) Technologies associées à l'étape 6

➤ Dépôt de métal

Le métal à déposer est soit un alliage d'aluminium-tungstène, soit du cuivre plus conducteur, mais plus difficile à déposer. Le métal doit bien pénétrer dans les via pour réaliser de bons contacts.

– La pulvérisation cathodique ou « sputtering » est réalisée dans une chambre à vide. Un flux d'ions bombarde une cible de métal (Ti, Al, Cu, TiN) qui libère ses atomes qui viennent se déposer sur les tranches.
– L'évaporation se fait aussi, dans une chambre à vide, par sublimation du métal (sous un très fort courant dans un creuset de tungstène) ou grâce à un flux d'électron.

3.2.8 Étape 7 : réalisation des via métal 1 – métal 2

La septième étape consiste en la réalisation des via entre les couches de métal (ici métal 1 – métal 2).

La réalisation des via entre les couches de métal se décompose en cinq sous-étapes :

– Dépôt de silice.
– Dépôt de la résine photosensible.

– Masquage et développement de cette résine.
– Gravure de la silice.
– Élimination de la résine.

➤ Gravure des via

Cette gravure s'arrête sur le métal 1 (figure 3.23).

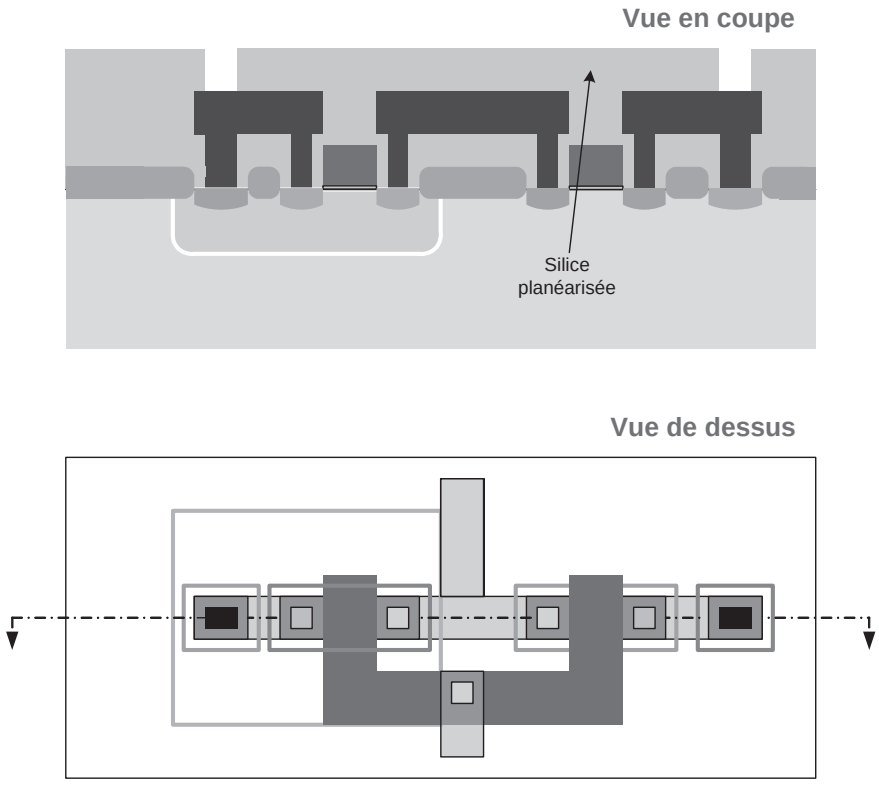


Figure 3.23 Étape 7 (gravure des via métal 1-métal 2)

3.2.9 Étape 8 : réalisation des connexions en métal 2

La huitième étape consiste en la réalisation des couches de connexions en métal (ici métal 2).

La réalisation des connexions en métal 2 se décompose en cinq sous-étapes :

– Dépôt du métal 2.
– Dépôt de la résine photosensible.
– Masquage et développement de cette résine.
– Gravure du métal 2.
– Élimination de la résine.

a) Gravure du métal 2

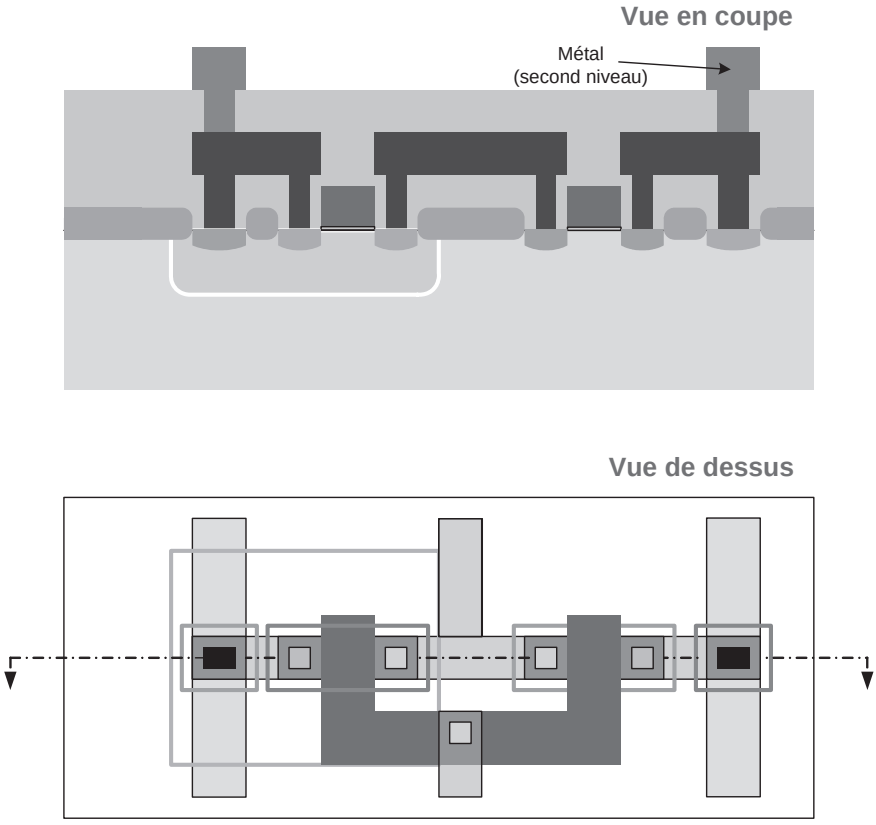


Figure 3.24 Étape 8 (gravure du métal 2)

b) Finition du circuit

La finition du circuit comprend :

– Réalisation des couches d'interconnexion métalliques suivantes.
– Passivation du circuit.
– Réalisation des plots de connexion.
– Test des circuits.
– Découpage de la tranche.
– Montage en boîtier et marquage des circuits.

c) Réalisation des couches d'interconnexion métalliques suivantes

Les circuits modernes comportent jusqu'à huit couches de métal qui sont obtenues en répétant les étapes 7 et 8 du processus.

d) Passivation du circuit

Le circuit terminé est recouvert d'une couche de nitrure de silicium pour le protéger contre les agressions mécaniques et chimiques.

e) Réalisation des plots de connexion

Les plots de connexion sont des surfaces carrées de métal à nu, relativement importantes (d'environ 100 µm de côté) qui recevront les fils de connexion avec le boîtier. Ces surfaces doivent résister à la pression des thermocompressions. Pour cela, elles sont réalisées par des empilements de couches judicieusement choisies.

f) Tests des circuits

Chaque circuit de la tranche est testé par une machine munie d'une *carte à pointes* qui lui permet de se connecter successivement sur les plots de chaque circuit. Une séquence d'excitation est envoyée pour savoir si le circuit est bon ou mauvais. Les circuits mauvais sont marqués avec une tache d'encre et seront éliminés.

g) Découpage de la tranche

La tranche testée est collée sur une mince feuille de plastique adhésive, puis la tranche est découpée avec une scie diamantée, sans couper la feuille de plastique. L'étirement de celle-ci permet la séparation des circuits.

h) Montage en boîtier et marquage des circuits

Les circuits bons sont saisis par une pipette sur la feuille de plastique étirée pour être montés dans des boîtiers dans lesquels ils sont soudés. Les connexions sont ensuite réalisées par de fins fils d'or thermo-compressés entre les plots des circuits et ceux de leurs boîtiers. Un test final permet de s'assurer du bon fonctionnement des circuits montés.

3.3 PRINCIPES DE DÉFINITION DES RÈGLES DE DESSIN

La réalisation des motifs constituants un circuit doit respecter les différentes contraintes physiques et technologiques du procédé. Ces contraintes sont exprimées par un ensemble de règles qui définissent les conditions pour qu'un circuit fonctionne convenablement et qu'il soit fiable. Les règles de dessin constituent une simplification pratique de la complexité de ces contraintes technologiques.

Les circuits deviennent de plus en plus complexes, ce qui nécessite l'augmentation du nombre de couches de métal pour réaliser leurs interconnexions. La réalisation de ces couches entraîne une augmentation de la complexité du procédé de fabrication, d'où le nombre des masques et des règles de dessin. Ainsi, la conception d'une simple mémoire DRAM NMOS de 256 Ko nécessite l'utilisation de près d'une centaine de règles de conception et treize niveaux de masques.

Nous commencerons, dans ce paragraphe, par décrire succinctement les différentes contraintes qui sont utilisées pour le calcul des règles de dessin. Puis dans un second temps, nous présenterons quelques cas concrets de règles.

3.3.1 Les différents types de contraintes

En général, les conditions à respecter lors de la conception expriment des contraintes pouvant être classées en trois types :

– contraintes dues au processus technologiques utilisé ;
– contraintes physiques ;
– contraintes de rendement et de fiabilité.

a) Contraintes dues au processus technologique

La complexité des circuits intégrés fait qu'il est important d'exercer un contrôle attentif du processus de photolithographie par lequel le dessin du circuit est transféré du masque sur le substrat. Ainsi, la taille d'un motif gravé est différente de celle qui a été dessinée. Cela est dû au fait qu'à chaque étape, une ou plusieurs sources de distorsion interviennent, engendrant des variations dans la taille des motifs réalisés. Dans un processus technologique, pour une séquence d'alignement donnée, de nombreuses sources d'incertitudes peuvent être identifiées. Elles comprennent le désalignement des niveaux de masquages, les erreurs de dimensionnement sur les masques, leur expansion thermique et l'incertitude sur les longueurs et les largeurs des motifs causée par le processus (par exemple la surgravure et la diffusion latérale).

Chaque technologie est caractérisée par un certain nombre de paramètres. Lorsque la taille des motifs ou des structures atteint ces limites, des phénomènes parasites apparaissent. Citons par exemple les trois cas suivants :

– Dans le cas de dessins submicroniques, la taille des motifs est maintenant inférieure (90 à 65 nm en 2006) à la longueur d'onde des lampes utilisées pour l'insolation de la résine (157 nm). Pour compenser les déformations des motifs projetés, dues aux phénomènes de diffraction, la géométrie des masques est volontairement modifiée.
– Les effets des canaux courts ou étroits. Les caractéristiques des transistors (tension de seul) varient lorsque canal devient court ou étroit.
– L'épaisseur des couches de métal devient prépondérante vis-à-vis de la largeur des lignes, ce qui engendre des phénomènes de capacités latérales.

L'évolution technologique consiste à repousser ces paramètres par l'utilisation de nouveaux procédés, comme par exemple le passage à la lithographie X.

b) Contraintes physiques

Elles découlent des limitations intrinsèques sur les composants dues aux phénomènes physiques comme, par exemple :

– L'apparition d'un courant de grille par effet tunnel, lorsque l'épaisseur de l'oxyde mince atteint quelques niveaux atomiques.

– La dispersion des caractéristiques des transistors, lorsque la longueur effective des transistors se mesure en quelques milliers d'atomes du cristal. Le très faible nombre d'atomes d'impuretés fait alors sortir le comportement du transistor des moyennes statistiques habituelles.
– L'augmentation de la résistance sources-drain des transistors lorsque leurs dimensions diminuent (due à la résistivité des matériaux). Idem pour les connexions métalliques.
– Le maintien de la valeur de la capacité parasite des connexions lorsque les dimensions technologiques diminuent (augmentation de la capacité surfacique compensée par une diminution de la largeur des connexions).

c) Contraintes de rendement et de fiabilité

Cette dernière famille de contraintes, a pour objectifs d'améliorer le rendement de fabrication et d'homogénéiser et d'accroître la durée de vie des circuits. Parmi les principales contraintes, citons plus particulièrement :

– L'électromigration qui arrache des atomes des couches de métal lorsque la densité de courant dépasse un certain seuil amenant une rupture des connexions.
– La diffusion lente des impuretés due à l'échauffement du circuit modifie les caractéristiques des transistors.
– La tenue aux radiations qui décroît lorsque les dimensions technologiques diminuent (contraintes des circuits durcis).

d) Incidences sur la conception système

Les effets des phénomènes physiques peuvent remonter dans les niveaux de conception et influencer la conception système du circuit. Par exemple, la répartition de la dissipation thermique sur le circuit, la génération des bruits, la longueur des connexions et les éventuels effets micro-onde qui apparaissent avec l'augmentation des fréquences de fonctionnement.

3.3.2 Exemples de règles de dessin

L'objectif ne peut être ici de décrire de manière exhaustive l'ensemble des règles de dessin d'une technologie particulière mais d'illustrer cette problématique par deux exemples.

a) Règles de dessin des contacts

Les règles de dessin des contacts métalliques sont représentées, sur la figure 3.25.

– La dimension de l'ouverture (notée R1 sur la figure) qui est contrôlée par la résolution de la photolithographie (on accepte l'arrondissement de l'ouverture). Dans les technologies actuelles, la taille des contacts est fixe.
– Le débordement des métaux (notée R2 sur la figure) qui tient compte du mésalignement maximal des masques contacts et métaux.

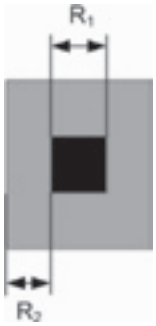


Figure 3.25 Règles de dessin d'un contact

b) Règles de dessin Poly & zone active

Les garde entre une connexion en polysilicium et la zone active d'un transistor, telle que représentée, sur la figure suivante, impose que le polysilicium soit sur l'oxyde épais (FOX) donc à une distance minimale de la zone active (règle notée R3 sur la figure 3.26). Ainsi, le polysilicium de connexion forme un transistor parasite dont la tension de seuil est supérieure à la tension d'alimentation du circuit. Ce transistor toujours bloqué pourra être alors négligé.

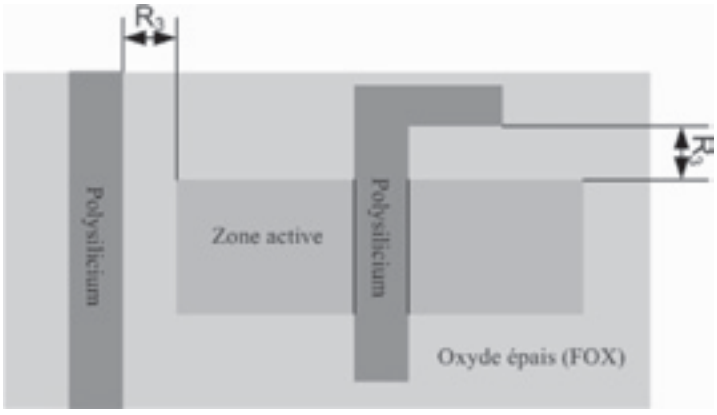


Figure 3.26 Règles de dessin connexion polysilicium et zone active

Pour terminer nous présentons ici un exemple de règles de dessin d'une technologie hypothétique de 0,6 µm (figure 3.27).

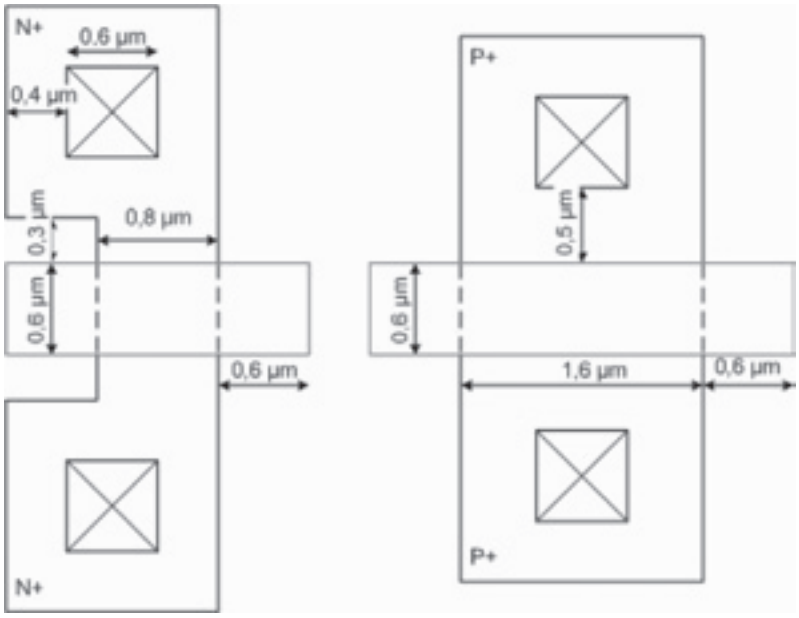


Figure 3.27 Exemple de règles 0,6 µm

Chapitre **4**

Réseaux de conduction et portes

4.1 REPRÉSENTATION SYMBOLIQUE DES SIGNAUX

Lorsque l'on regarde un montage électronique avec le niveau d'abstraction logique, les groupements de transistors sont vus comme des *portes* et les signaux qu'ils échangent comme des *signaux logiques*.

4.1.1 Signaux logiques

Les *signaux logiques* représentent des niveaux de tension. Une valeur logique 1 représente une tension au-dessus d'un seuil haut, tandis qu'une valeur logique 0 représente une tension en dessous d'un seuil bas. Une tension entre ces deux seuils ne correspond pas à une valeur logique définie.

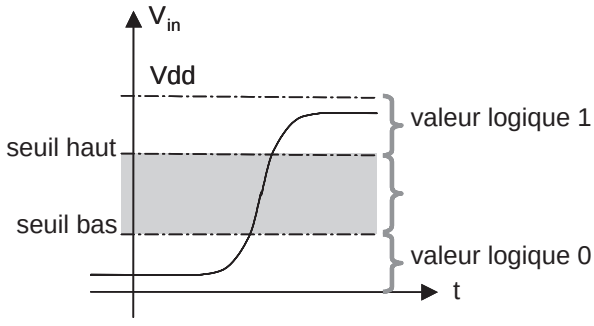


Figure 4.1 Niveaux logiques

4.1.2 Chronogrammes

Les variations et les états des signaux logiques sont représentés de manière symbolique par les graphismes présentés figures 4.2 à 4.6.

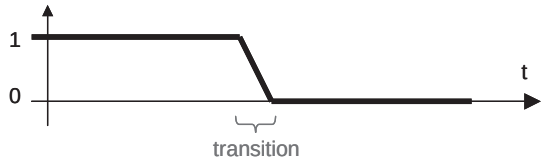


Figure 4.2 Transition entre une valeur 1 suivie d'une valeur 0

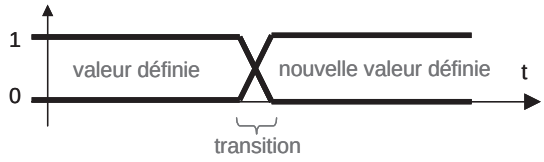


Figure 4.3 Transition entre deux valeurs définies non connues

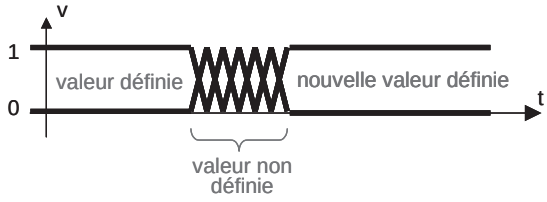


Figure 4.4 Valeur non définie entre deux valeurs définies non connues

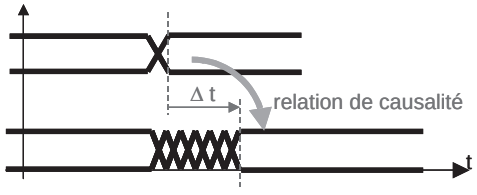


Figure 4.5 Relation de causalité
entre une transition et le début d'une valeur définie non connue

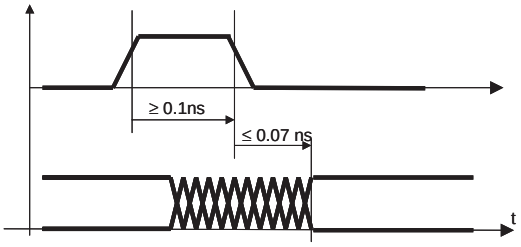


Figure 4.6 Contraintes temporelles sur la durée d'une impulsion et sur l'instant d'apparition d'une valeur non définie connue

Les valeurs définies non connues peuvent avoir une valeur (stabilisée) 1 ou 0 suivant le fonctionnement particulier du montage. Les valeurs non définies peuvent prendre n'importe quelle valeur logique ou non et comporter des transitions quelconques.

4.1.3 Signaux événementiels et de valeur

Les signaux logiques véhiculent deux types d'informations :

– des *événements* liées aux transitions de ces signaux, par exemple dans le cas de signaux d'horlogerie, de chargement de registres, de formatage…
– des *valeurs* liées à leur état électrique, par exemple la sortie de registres, les entrées et les sorties de réseaux combinatoires.

Dans la plupart des cas, les signaux véhiculent uniquement l'un ou l'autre type d'information, rarement les deux. Les contraintes liées à chaque type d'information sont très différentes. Le fait de véhiculer les deux types d'information additionne les contraintes et est de ce fait peu recommandé. Toutefois, de tels signaux existent (par exemple les voies séries de télécommunication…).

Les signaux qui véhiculent des événements sont appelés des *signaux événementiels* (ou encore *signaux de temps* ou *horloges*) (figure 4.7). Certains événements sont liés à des transitions de la valeur portée par un tel signal, par exemple ses transitions négatives. Celles-ci doivent donc être définies sans ambiguïté. Le signal doit donc être exempt de toute transition parasite qui créerait des événements non souhaités.

Figure 4.7 Signal événementiel

La combinaison des événements suit une logique particulière dans laquelle le repère temporel doit être bien défini.

Les signaux qui véhiculent des valeurs sont appelés des *signaux de valeur*. Ces valeurs sont représentées par l'état logique de ces signaux. Ceux-ci doivent être stables dans l'*intervalle d'examen* du signal. En dehors de cet intervalle, le signal peut avoir n'importe quelle valeur (figure 4.8).

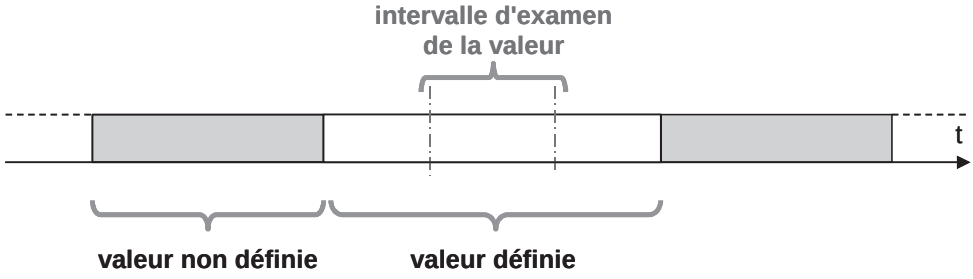


Figure 4.8 Signal de valeur

4.1.4 Propreté d'un signal

Nous dirons qu'un signal est *propre*, sur un certain intervalle temporel, s'il est exempt de parasites. Cette propriété est particulièrement importante pour les signaux événementiels. En effet, tout parasite introduit des transitions inopportunes qui génèrent des événements indésirables qui peuvent modifier l'état du système (figure 4.9).

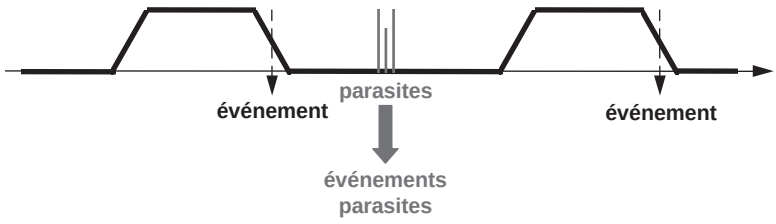


Figure 4.9 Les parasites sur un signal de temps créent des événements non désirés

La propreté d'un signal de valeur est moins importante. En effet, ce signal ne doit être exempt de parasites que pendant son intervalle d'examen.

Les parasites sur un signal peuvent provenir de différentes sources (ponts de résistances, couplages capacitif, selfiques…), mais la source la plus importante provient de l'établissement des réseaux combinatoires. Ces parasites sont alors appelés des *aléas*. Ils proviennent du fait que ces réseaux sont constitués de plusieurs branches en parallèle dans lesquelles les transitions des signaux internes se propagent différemment. Au fur et à mesure de l'arrivée des signaux internes la ou les sorties vont afficher des valeurs transitoires incorrectes avant d'afficher la valeur définitive lorsque tout le réseau sera établi. Le chemin minimum entre les entrées qui ont varié et la

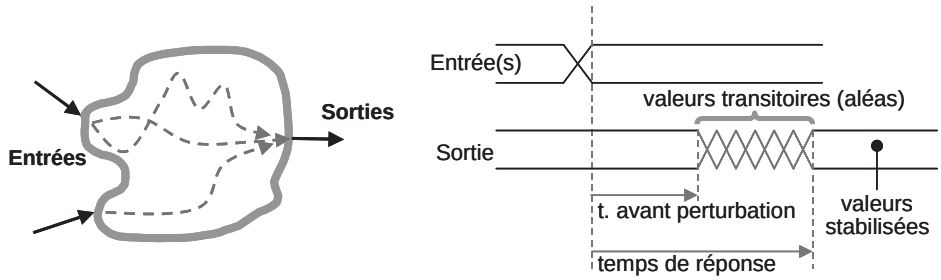


Figure 4.10 Chronogramme des variations d'une sortie d'un réseau combinatoire

sortie considérée permettra de déterminer l'instant à partir duquel la sortie va commencer à varier, que nous appellerons *temps avant perturbation*, tandis que l'examen du chemin maximum permettra de déterminer l'instant d'occurrence de la valeur finale, que nous appellerons le *temps de réponse* du circuit combinatoire (figure 4.10). L'intervalle entre ces deux instants correspond à des valeurs indéterminées constituant l'aléa.

En équilibrant la longueur de ces deux chemins, il est possible de concevoir des réseaux combinatoires qui ne génèrent pas d'aléas. L'inconvénient est qu'ils sont beaucoup plus complexes.

4.1.5 Validation des signaux temporels (horloges)

Il est déconseillé de générer ou même de valider, un signal temporel à l'aide d'un circuit combinatoire complexe sans précautions très particulières. Par exemple, il est déconseillé d'utiliser directement les sorties d'un décodeur comme des signaux temporels.

Une bonne technique pour conditionner un signal d'horloge consiste à utiliser une porte ET spéciale, alimentée par cette horloge (figure 4.11). Lorsque la porte n'est pas alimentée, aucun signal ne peut apparaître en sortie. L'inconvénient de cette technique est que l'énergie de l'horloge conditionnelle provient directement de celle de l'horloge primaire.

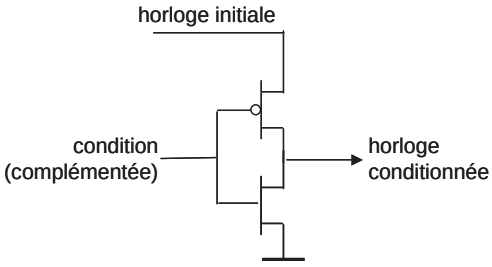


Figure 4.11 Conditionnement d'une horloge

4.2 LE TRANSISTOR VU COMME UN INTERRUPTEUR

Les transistors MOS peuvent se comporter comme des interrupteurs (figure 4.12). Pour cela, il suffit de les faire travailler entre leurs états *bloqués* (Vgs < Vt, pour un transistor N) et *saturés* (c'est-à-dire Vgs suffisant pour que Vds = Vsat).

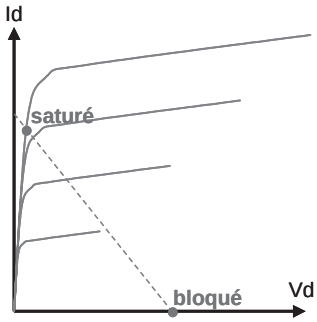


Figure 4.12 Un transistor vu comme un interrupteur

Dans l'état bloqué (Vgs < Vt, pour un transistor N), un transistor peut être comparé à un contact ouvert, tandis que dans l'état saturé (Vgs = Vdd, pour un transistor N), il peut être comparé à un contact fermé (présentant une résistance Ron) (figure 4.13).

Figure 4.13 Transistor N et son contact équivalent

4.2.1 Imperfections

L'assimilation d'un transistor à un interrupteur n'est toutefois pas parfaite. Il s'avère que si un transistor N doit transmettre un signal à Vdd, alors sa tension de commande Vgs va devenir trop faible pour maintenir le transistor saturé et la tension de sortie de l'interrupteur à transistor n'atteint, au mieux, que Vdd-Vt (figure 4.14). Cela est particulièrement visible lorsque l'interrupteur est chargé par une pure capacité, ce qui est le cas général. Par contre, un transistor N transmet bien les tensions proches de 0 v.

L'effet sera symétrique pour un transistor P utilisé comme un interrupteur. Celui-ci transmet bien une tension Vdd, mais, par contre, une tension de 0v sera transmise comme Vt.

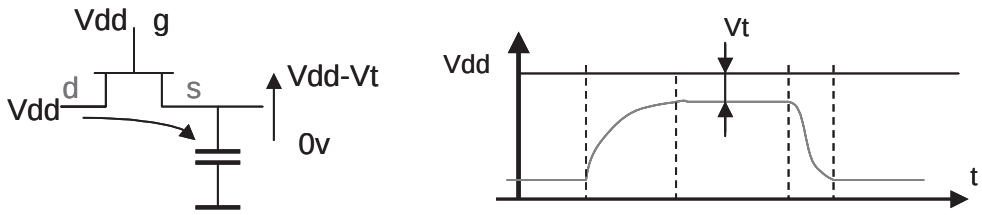


Figure 4.14 Mauvaise transmission des « 1 » par un transistor N

4.3 RÉSEAUX DE CONDUCTION

4.3.1 Logique de conduction

Depuis longtemps, on réalise des réseaux complexes en assemblant des interrupteurs et des relais. Ces réseaux ont pour propriété que la signification de leurs entrées, dont les valeurs de vérité correspondent aux positions des interrupteurs et aux tensions qui alimentent les relais, est différente de celle de leur « sortie » dont les valeurs de vérité représentent la *conduction* du réseau qui peut être conducteur ou non (figure 4.15).

Il est facile de voir que la mise en série d'interrupteurs ou de relais correspond à réaliser un ET de leurs *fonctions de conduction*, tandis que leur mise en parallèle correspond à réaliser un OU de leurs *fonctions de conduction*.

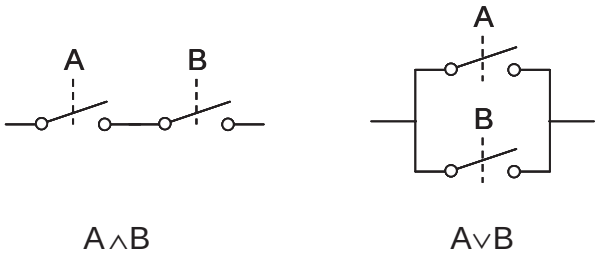


Figure 4.15

a) Fonctions de conduction des transistors

La fonction de conduction d'un transistor N est triviale puisque ce transistor conduit lorsque la valeur logique du signal qui excite sa grille est vraie. Celle d'un transistor P est inverse, c'est-à-dire que ce transistor conduit lorsque la valeur logique du signal qui excite sa grille est fausse.

b) Fonction de conduction des réseaux constitués de transistors d'un seul type

La fonction de conduction d'un réseau constitué de transistors d'un seul type est obtenue par la composition des fonctions de conduction de ses transistors et des

sous-réseaux qui le composent. Les fonctions de conduction de ces éléments en parallèle sont combinées par des `OU` et celles de ceux en série par des `ET`.

– Dans le cas de transistors N, la fonction de conduction du réseau est identique à celle d'un réseau d'interrupteurs ou de relais (figure 4.16).

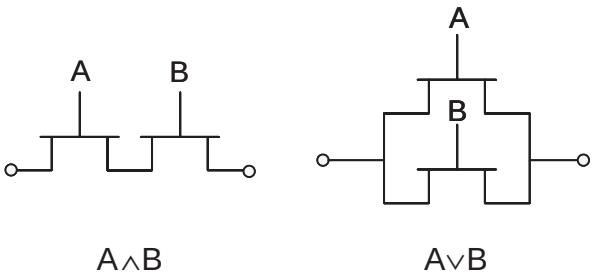


Figure 4.16

– Dans le cas de l'assemblage de transistors P, comme leur fonction de conduction est inverse de celle des transistors N, la fonction de conduction de deux transistors P en série sera $\overline{A} \wedge \overline{B}$ et $\overline{A} \vee \overline{B}$ s'ils sont en parallèle (figure 4.17).

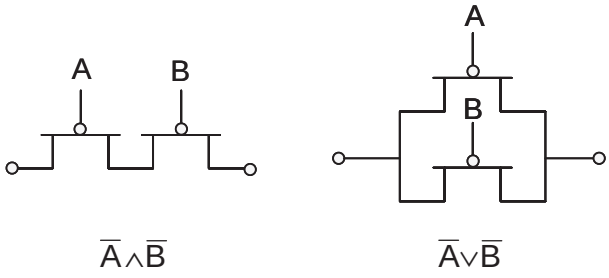


Figure 4.17

Il est possible de réaliser des réseaux de conduction complexes (figure 4.18).

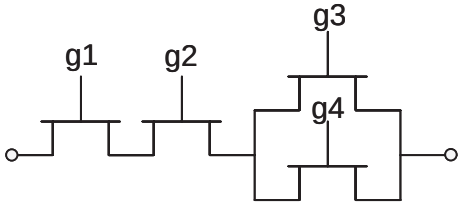


Figure 4.18 Exemple d'un réseau N réalisant la fonction de conduction : $g1 \wedge g2 \wedge (g3 \vee g4)$

Comme les transistors qui les composent, les réseaux de conduction réalisés à l'aide de transistors N transmettent mal les 1, tandis que ceux réalisés à l'aide de transistors P transmettent mal les 0.

c) Réseaux de conduction CMOS

L'assemblage d'un transistor N et d'un transistor P permet de réaliser un interrupteur, dit CMOS (figure 4.19), qui laisse aussi bien passer les 1 que les 0. En effet, chaque transistor compense les défauts de l'autre. Toutefois, la grille du transistor P doit être excitée par un signal complémenté pour que les deux transistors offrent la même fonction de conduction.

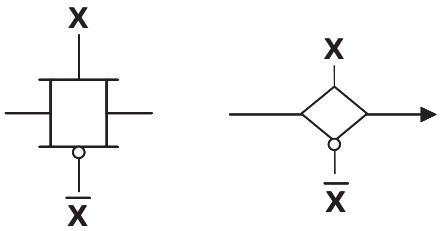


Figure 4.19 Interrupteur CMOS

La permutation des commandes X et $\overline{X}$ entre les transistors N et P permet d'inverser la fonction de conduction de l'interrupteur CMOS.

4.3.2 Utilisation des réseaux de conduction

Les réseaux de conduction sont utilisés pour amener, conditionnellement, des valeurs logiques sur certains nœuds du circuit. Ces valeurs logiques peuvent provenir d'autres nœuds ou des alimentations (Vdd et Vss). Plusieurs réseaux peuvent être utilisés conjointement (figure 4.20). Chacun d'eux ne transmet qu'une partie du tableau de vérité de la fonction à réaliser qui peut être complète ou partiellement définie. Les contributions des différents réseaux peuvent être disjointes ou avoir des superpositions. Dans ce cas, les valeurs amenées simultanément au nœud destination ne doivent pas être contradictoires. Si aucun réseau ne fournit de valeur à ce nœud, celui-ci est isolé. Ceci peut être intentionnel (voir portes 3 états paragraphe 4.4.4).

Exemple : réseau « tally » [CAL58] (figure 4.21) : Un tel réseau est destiné à compter le nombre de bits à « 1 » dans un mot. Il est organisé comme un réseau d'aiguillages qui oriente un bit initial à « 1 » sur une série de lignes parallèles correspondant aux différentes valeurs. Le circuit se complique car les autres lignes de sortie doivent être portées à « 0 ».

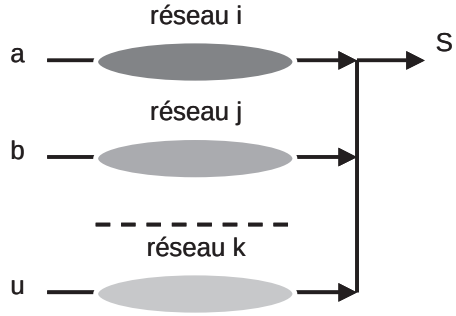


Figure 4.20 Signal généré par plusieurs réseaux de conduction

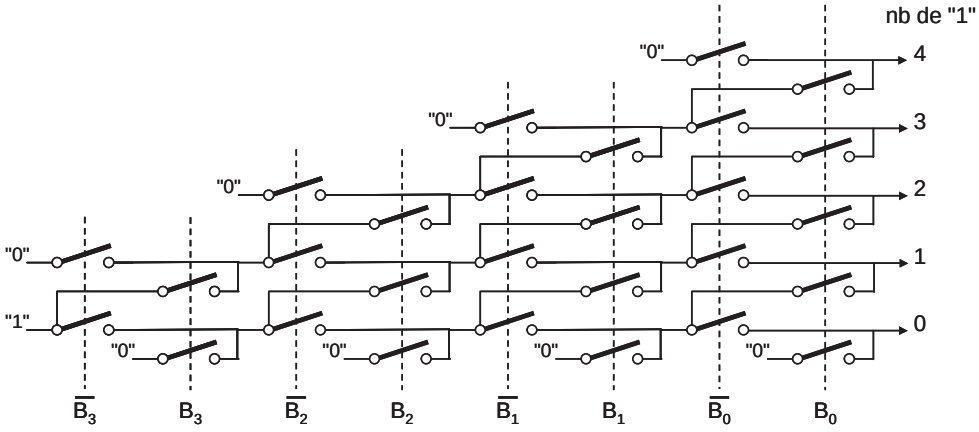


Figure 4.21 Circuit Tally vu comme un réseau d'interrupteurs

4.4 PORTES LOGIQUES

Dans un circuit CMOS, les *portes* sont constituées par l'assemblage de réseaux de conduction branchés entre Vdd, Vss, les entrées et leurs sorties. Les différents schémas possibles permettent d'obtenir toute une gamme de rapports performance/complexité qui offrent différent choix pour optimiser la conception d'un circuit.

4.4.1 Consommation des portes logiques

Dans la grande majorité des cas, les portes d'un circuit CMOS sont chargées par de pures capacités. En effet, les portes sont très souvent connectées sur les grilles des transistors qui constituent les portes suivantes. Ceci entraîne qu'elles n'ont pas de consommation continue. L'énergie qu'elles consomment n'est dissipée que lors des charges

et décharges des capacités constituées par les grilles de transistors qui constituent les entrées des portes suivantes.

Pour une entrée et pour un couple de transitions ($0 \rightarrow 1 \rightarrow 0$), la charge transférée de l'alimentation vers la masse sera donc exprimée par :

$$Q = CV$$

Si f représente la fréquence d'excitation de la porte, c'est-à-dire le nombre moyen de couples de transitions par seconde, le courant consommé par cette entrée est :

$$i = CVf$$

La puissance moyenne consommée par un circuit CMOS est donc :

$$W = C_{\text{tot}} V^2 f$$

dans laquelle C_{tot} représente l'ensemble des capacités du circuit et f la fréquence moyenne de commutation des portes.

La fréquence particulièrement élevée des microprocesseurs modernes explique leur consommation importante.

4.4.2 Portes CMOS « classiques »

Les portes CMOS dites classiques sont constituées par l'assemblage de deux réseaux de conduction branchés entre Vdd, Vss et la sortie et qui reçoivent les mêmes entrées.

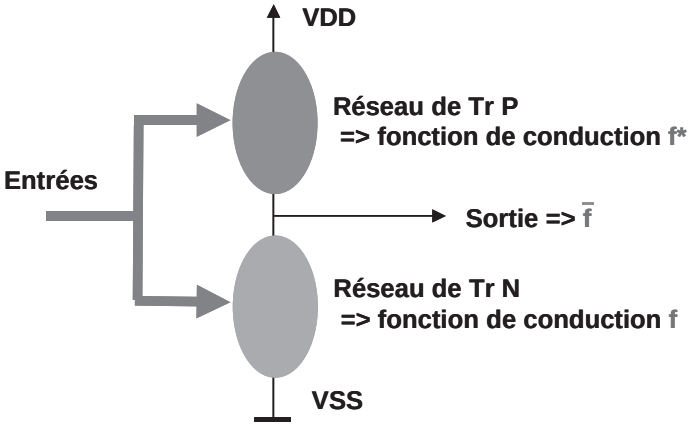


Figure 4.22 Une porte classique vue comme l'assemblage de deux réseaux de conduction duaux

Un premier réseau de conduction, dit N, constitué de transistors N, est chargé de tirer la sortie vers Vss. Il est donc branché entre la sortie et Vss. Si la porte doit réaliser la logique f, ce réseau doit donc avoir $gN = \overline{f}$ comme fonction de conduction. Comme il n'est pas possible d'avoir des termes complémentés dans un réseau constitué de transistors N, la fonction réalisée par la porte doit donc toujours comporter

une négation globale (par exemple, $f = \overline{(A \vee B)}$). On parle alors de portes *inverseuses*.

Le second réseau de conduction, dit P, est constitué de transistors P. Il est chargé de tirer la sortie vers Vdd. Il est donc branché entre Vdd et la sortie.

Si la porte doit réaliser la fonction *f*, ce réseau doit donc avoir $gP = f(\overline{\text{entrées}})$ comme fonction de conduction.

Par rapport au réseau N, sa fonction logique est donc : $gP = \overline{gN(\overline{\text{entrées}})}$, soit *gN**.

Comme la dualisation d'une fonction s'obtient en remplaçant dans son écriture les `ET` par des `OU` et réciproquement (voir annexe sur les fonctions booléennes). De même, le réseau de transistors P est soit obtenu à partir du réseau de transistors N par le remplacement des mises en série par des mises en parallèle et réciproquement, soit par la synthèse directe de *gP* à partir de $f(\overline{\text{entrées}})$.

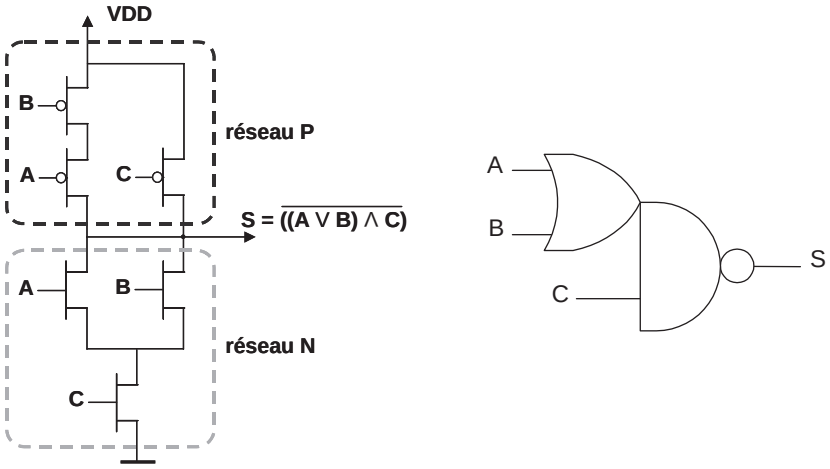


Figure 4.23 Exemple de porte $\overline{((A \vee B) \wedge C)}$

Les portes classiques sont celles qui permettent d'atteindre les meilleures performances en vitesse. En effet, les réseaux de transistors N et P sont utilisés de manière optimale. Toutefois, ces portes comportent beaucoup de transistors qui occupent de la surface et chargent les entrées de manière importante.

a) Dimensionnement des portes CMOS

➤ Dimensionnement des transistors

Pour obtenir un fonctionnement équilibré, les réseaux N et P doivent présenter des résistances comparables lorsqu'ils sont conducteurs, de manière à ce que les temps de montée et de descente de la porte soient proches. Ces durées correspondent à la charge et à la décharge des capacités attaquées par la porte. Pour cela, il est nécessaire de tenir

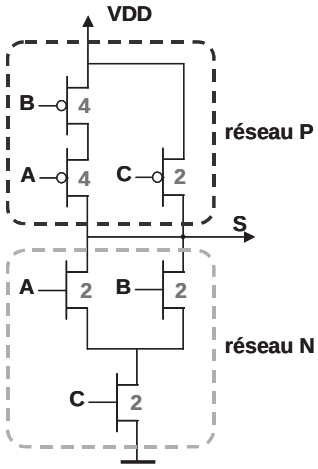


Figure 4.24 Exemple de dimensionnement des transistors d'une porte classique

compte de la différence de mobilité entre les transistors N et P ainsi que de la différence entre la mise en parallèle ou en série des transistors dans les réseaux de conduction. Comme la longueur de ces transistors est minimale (donnée par la largeur du ruban de polysilicium qui constitue leur grille), il convient de moduler leur largeur en conséquence. Les portes classiques seront dimensionnées par rapport à un inverseur choisi comme référence. Elles auront donc la même sortance. La largeur du transistor N de cet inverseur donnera l'unité de dimensionnement des transistors de la porte. Celle-ci devrait donc avoir les mêmes temps de montée et de descente que l'inverseur, à la différence près de l'influence de son impédance de sortie, différente de celle de l'inverseur.

L'obtention d'une sortance différente peut être obtenue par la multiplication de la largeur de tous les transistors par le coefficient désiré.

➤ Calcul de l'entrance des portes

L'*entrance* d'une porte correspond à la charge que chaque entrée de cette porte ramène sur la sortie de la porte qui l'attaque. Elle est surtout constituée de la capacité des grilles des transistors connectées à cette entrée. Elle peut se déterminer par rapport à celle de l'inverseur de référence *via* la capacité grille de son transistor N. Comme la capacité grille unitaire des transistors P est identique à celle des transistors N, la capacité de l'entrée d'un inverseur sera donc trois fois celle de la grille de son transistor N.

L'entrance de la porte de l'exemple précédent est :

Entrées	∑ larg. Tr. N	∑ larg. Tr. P	∑ larg. Tr. P + N.	Équiv. inv
A	2	4	6	**2**
B	2	4	6	**2**
C	2	2	4	**1,3**

Chacune des entrées A, B charge donc la porte qui l'attaque comme deux inverseurs. L'entrée C la charge comme 1,3 inverseurs.

Cette technique de calcul relatif peut s'étendre aux interconnexions, en considérant la capacité d'une unité de longueur de fil (de largeur standard) de chaque matériau par rapport à la capacité d'entrée de l'inverseur de référence. Ceci permet d'assimiler chaque connexion longue à l'entrée d'un certain nombre d'inverseurs de référence.

b) Optimisation des portes CMOS

Les réseaux de conduction N et P qui constituent une porte CMOS peuvent être simplifiés en cherchant à partager, entre plusieurs portes, des sous-réseaux identiques. Ces sous réseaux doivent être en série et branchés à Vdd ou Vss pour ne partager que les effets de leurs variations de conduction. Il est également possible de permuter des sous-réseaux en série ou en parallèle pour pouvoir les partager.

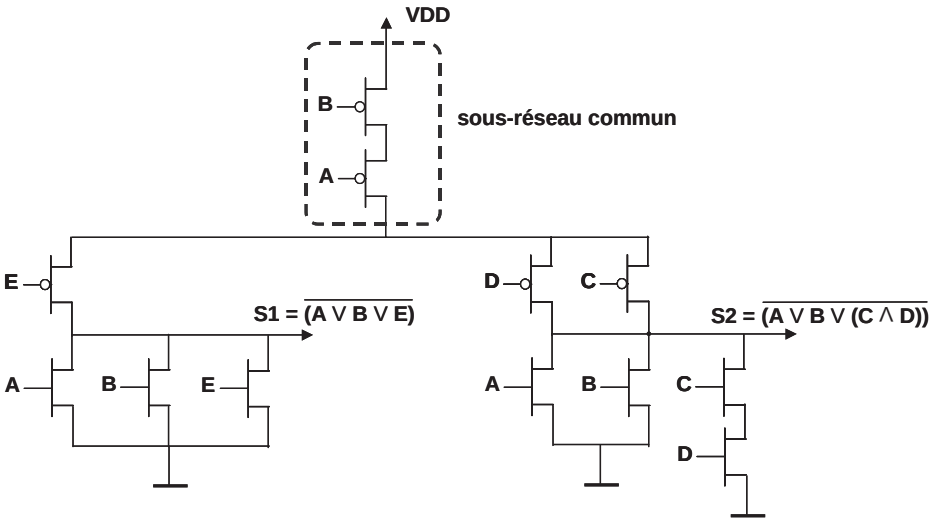


Figure 4.25 Exemple de deux portes partageant un sous-réseau P

4.4.3 Portes CMOS « non classiques »

Il est possible de réaliser des portes CMOS qui utilisent beaucoup moins de transistors que les portes « classiques » au prix d'une sortance plus faible et d'un dessin des masques moins régulier. La réalisation de ces portes s'appuie sur plusieurs techniques :

a) Portes obtenues par l'assemblage de réseaux de conduction non duaux

Dans ces portes, plusieurs réseaux de conduction participent à la réalisation des valeurs de sortie à partir des alimentations (Vdd et Vss) mais aussi à partir des entrées. L'utilisation de ces portes nécessite souvent de disposer des valeurs directes et complémentaires des signaux d'entrée.

➤ Portes ET (OU)

Celles-ci utilisent les propriétés « dissymétriques » des fonctions $\wedge$ et $\vee$.

$$A \wedge B = A \text{ si } B = 1, \quad 0 \text{ si } B = 0$$
$$A \vee B = A \text{ si } B = 0, \quad 1 \text{ si } B = 1$$

Elles sont constituées de 3 transistors constituant deux réseaux de conduction :

– Un interrupteur CMOS transmet la valeur d'entrée A en sortie lorsque l'autre entrée B est vraie (fausse dans le cas d'une porte OU).
– Un transistor N (P dans le cas d'une porte OU) qui force la sortie à 0 (1) lorsque l'entrée B est fausse (vraie)

L'utilisation de ces portes nécessite de disposer des valeurs directe et complémentaire de l'une des entrées.

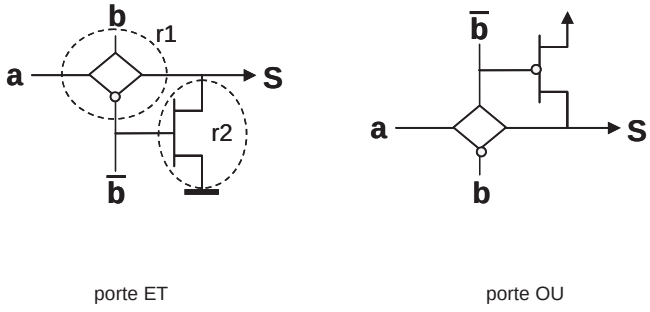


Figure 4.26 Portes ET et OU

Il existe d'autres façons de réaliser des portes ET et OU. Nous pouvons mentionner le schéma suivant de porte ET qui consiste à alimenter un inverseur par l'un des opérandes :

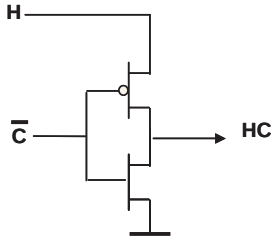


Figure 4.27 Porte ET pour le conditionnement d'une horloge

Ce type de porte est utilisé pour générer des sous-horloges en conditionnant une horloge primaire. La conception de la porte assure une grande propreté du signal de sortie

lorsque la condition est fausse. Il faut aussi remarquer que toute la puissance utilisée par l'horloge secondaire provient de l'horloge primaire.

➤ Portes OUEX (NON-OUEX)

La réalisation d'une fonction OUEX, ou NON-OUEX, avec des portes classiques demande beaucoup de transistors. Les approches présentées ci-dessous proposent des solutions plus économiques. Il existe une grande variété de portes OUEX (NON-OUEX) non classiques. Nous allons examiner les 3 plus utilisés. Elles sont toutes réalisées à l'aide de 4 transistors, toutefois leur mise en œuvre demande de disposer des valeurs directes et complémentées de tout ou partie des entrées. La réalisation des portes NON-OUEX repose sur la remarque suivante :

$$NON\text{-}OUEX(A, B) = \overline{(A \oplus B)} = (\bar{A} \oplus B) = (A \oplus \bar{B})$$

– Réalisation à l'aide de deux interrupteurs CMOS :
Il est possible de réaliser une porte OUEX en remarquant que :

$$\text{OUEX}(A, B) = A \text{ si } B = 0, \quad \bar{A} \text{ si } B = 1$$

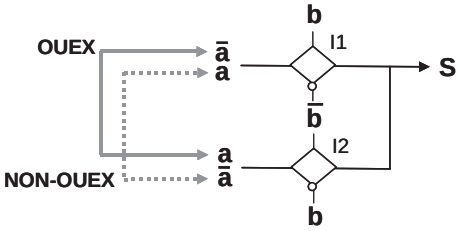


Figure 4.28 Porte OUEX constituée de deux interrupteurs CMOS

Cette porte OUEX nécessite de disposer des valeurs directes et complémentées des entrées. La permutation des entrées A et $\bar{A}$ (ou B et $\bar{B}$) fournit la porte NON-OUEX.

– Réalisation à l'aide d'un interrupteur CMOS et d'un inverseur alimenté par l'un des opérandes :
Cette porte OUEX utilise la même écriture que la précédente. Toutefois, le terme : $\bar{A}$ *si* $B = 1$ est ici réalisé par un inverseur alimenté par B et $\bar{B}$ (figure 4.29).
Cette porte OUEX ne nécessite que de disposer des valeurs directes et complémentées que d'une entrée. La permutation des entrées B et $\bar{B}$ fournit la porte NON-OUEX.

– Réalisation ne nécessitant que de disposer des deux entrées directes :
Cette porte, symétrique, repose sur l'écriture suivante de la fonction OUEX :

$$\text{OUEX}(A, B) = A \text{ si } B = 0, B \text{ si } A = 0, 0 \text{ si } A = 1 \text{ et } B = 1$$

ce qui peut être réalisé par la figure 4.30 :
Cette porte possède l'inconvénient de générer une sortie 0 altérée lorsque $A = B = 0$.
La réalisation de la fonction NON-OUEX repose sur l'écriture suivante :

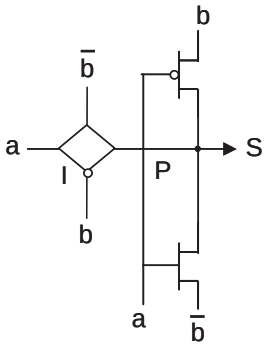


Figure 4.29 Porte OUEX
constituée d'un interrupteur et d'un inverseur à alimentation commandée

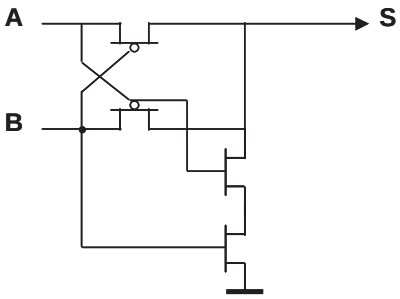


Figure 4.30 Porte OUEX à quatre transistors

NON-OUEX(A, B) = A si B = 1, B si A = 1, 1 si A = 0 et B = 0
ce qui peut être réalisé par la figure 4.31 :

Cette porte possède l'inconvénient de générer une sortie 1 altérée lorsque A = B = 1.

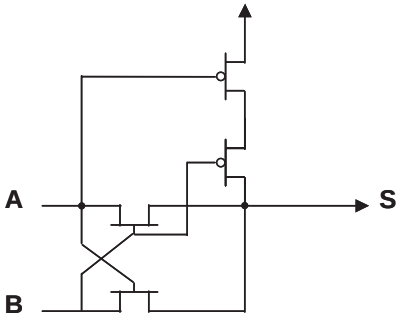


Figure 4.31 Porte NON-OUEX à quatre transistors

➤ Porte MINORITÉ

Une porte Minorité (non-Majorité) à trois entrées peut être conçue de manière symétrique, avec des réseaux N et P apparemment non duaux qui réalisent les fonctions de connection gN et gP symétriques.

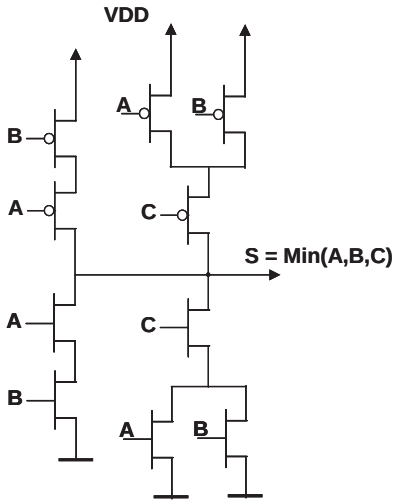


Figure 4.32 Porte MINORITÉ symétrique (et duale)

En fait, les réseaux N et P sont bien duaux. Pour le voir, il suffit d'appliquer la relation $(A \wedge B) \wedge (A \vee B) \equiv A \wedge B$.

*b) Logique CVSL (*Cascade Voltage Switch Logic*)*

La logique CVSL [HEL84] est basée sur la génération simultanée d'une fonction logique et de son complément. Les portes CVSL sont dissymétriques car les réseaux P sont remplacés par de simples transistors P commandés par l'autre polarité. Cette approche présente des avantages mais aussi des inconvénients :

– Le bénéfice en surface apporté par le remplacement des réseaux P par de simples transistors n'est sensible que pour les portes complexes.
– Il est quelquefois possible de partager des transistors entre les deux réseaux N.
– La disponibilité des compléments permet de réaliser directement des fonctions complexes.
– Les portes CVSL sont environ deux fois plus lentes que les portes classiques car leur commutation demande l'établissement successif des deux polarités.
– Nous verrons que le dessin des masques des portes CVSL n'est pas aussi systématique que celui des portes classiques.

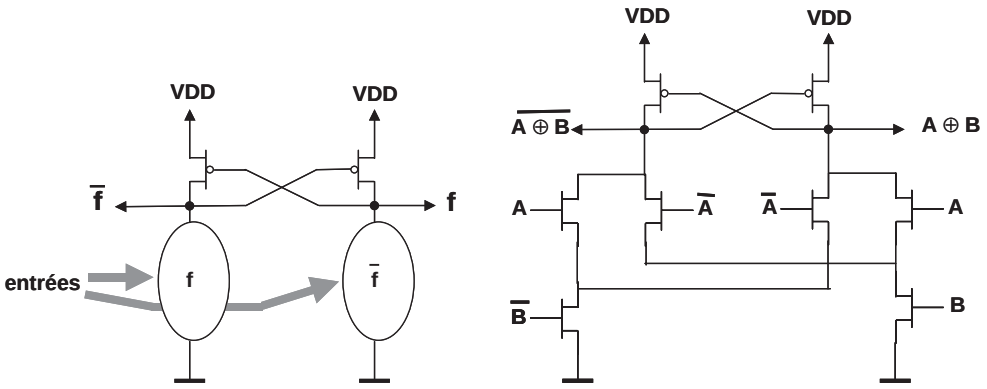


Figure 4.33 Principe et exemple d'une porte CVSL

4.4.4 Portes « 3 états »

La sortie de certaines portes obtenues par l'assemblage de réseaux de conduction peut ne pas être totalement définie. Pour certaines configurations des entrées, aucun réseau ne définit la valeur de la sortie qui reste isolée. Nous dirons que cette sortie travaille sur une logique dite à *3 états* (deux états logiques 1 et 0 et un état *isolé*).

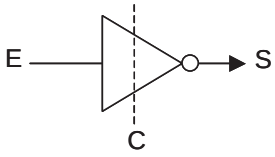


Figure 4.34 Symbole général d'un inverseur 3 états

Les portes 3 états sont utilisées lorsque plusieurs sources doivent être connectées à un même nœud, par exemple dans le cas d'un bus. À un instant donné, une seule source définit la valeur logique de ce nœud, tandis que toutes les autres sont isolées.

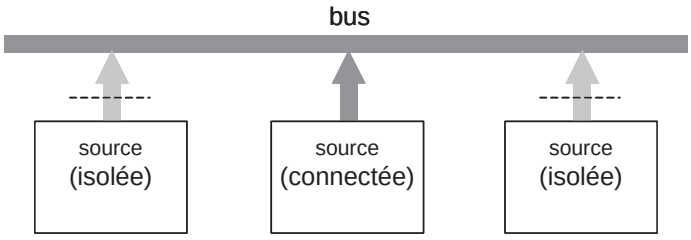


Figure 4.35 Utilisation de portes 3 états pour la connexion à un bus

a) Réalisation des portes 3 états

Il existe deux grandes techniques pour réaliser des portes 3 états :

➤ Utilisation d'un interrupteur CMOS

Il est aisé de transformer une porte classique en porte 3 états en la faisant suivre d'un interrupteur CMOS.

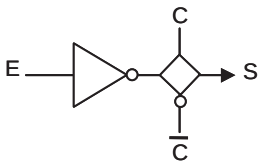


Figure 4.36 Inverseur 3 états obtenu avec un interrupteur CMOS

➤ Modification des réseaux de conduction N et P d'une porte pour obtenir son isolation

Il est possible d'ajouter des transistors à ces réseaux pour obtenir leur isolation conjointe.

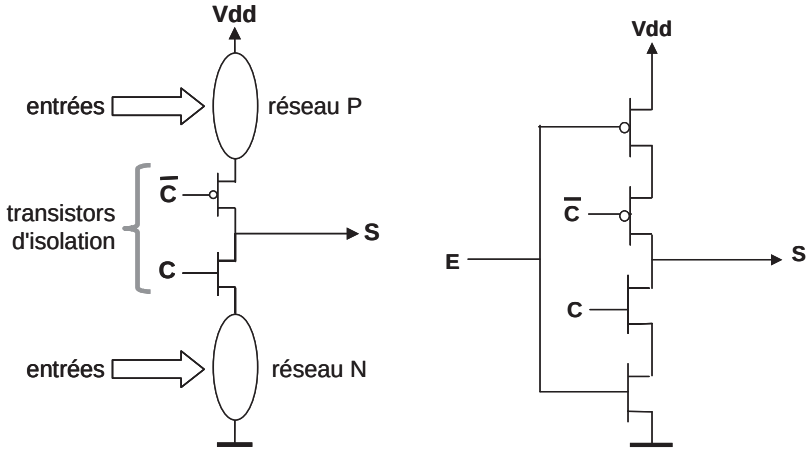


Figure 4.37 a) Principe des portes 3 états ; b) schéma d'un inverseur 3 états

b) Dimensionnement des portes de transfert

Le dimensionnement des transistors des portes de transfert se fait de la même manière que pour ceux des portes standard. Il faut identifier les chemins électriques de la capacité de charge à VDD et à la masse puis dimensionner les transistors qui les composent de manière à ce que leur assemblage donne les mêmes transistors équivalents que ceux d'un inverseur pris comme référence.

Les longues chaînes d'interrupteurs dégradent l'énergie des signaux qui les parcourent. Il faut qu'il en reste suffisamment à l'extrémité pour exciter la charge. Ce qui limite leur longueur. Toutefois, les fonctions logiques réalisées à l'aide d'interrupteurs ne consomment aucune énergie, ce qui peut être intéressant pour réaliser des montages à basse consommation.

Exemple : Dimensionnement des transistors d'un interrupteur CMOS suivant un inverseur.

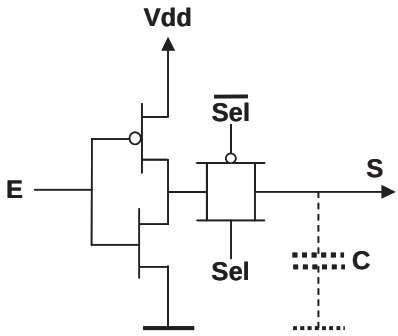


Figure 4.38 Interrupteur suivant un inverseur

➤ Chemin de charge de la capacité de sortie

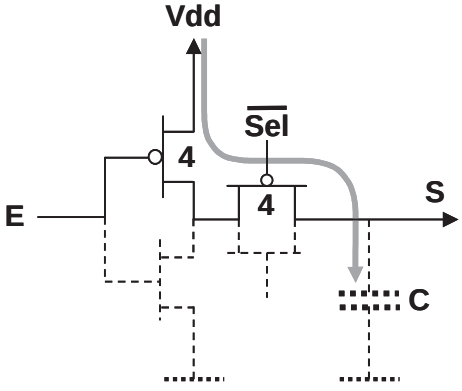


Figure 4.39 Chemin de charge de la capacité de sortie et dimensionnement des tr. P

➤ Chemin de décharge de la capacité de sortie

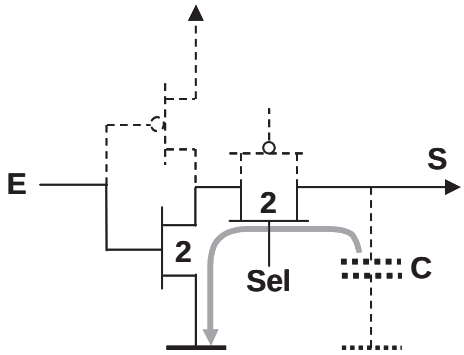


Figure 4.40 Chemin de décharge de la capacité de sortie et dimensionnement des tr. N

4.5 LOGIQUE DYNAMIQUE

La logique dynamique consiste à utiliser les propriétés de mémorisation naturelle de l'entrée d'une porte lorsqu'elle est isolée. En effet, l'impédance des entrées d'une porte est généralement capacitive. Celle-ci retiendra donc la dernière valeur qui lui a été appliquée avant son isolation par une porte amont 3 états. Comme la capacité en jeu est généralement très faible (de l'ordre de la dizaine de fentofarads) et que son isolation n'est pas parfaite, la durée de rétention de l'information n'est que de quelques microsecondes. Elle est toutefois suffisante pour les applications rapides et surtout lorsque l'on est sûr que cette rétention reste dans une durée acceptable.

Cette technique est utilisée pour réaliser des portes logiques particulièrement économiques dans lesquelles la valeur de la sortie est élaborée sur plusieurs instants. Dans

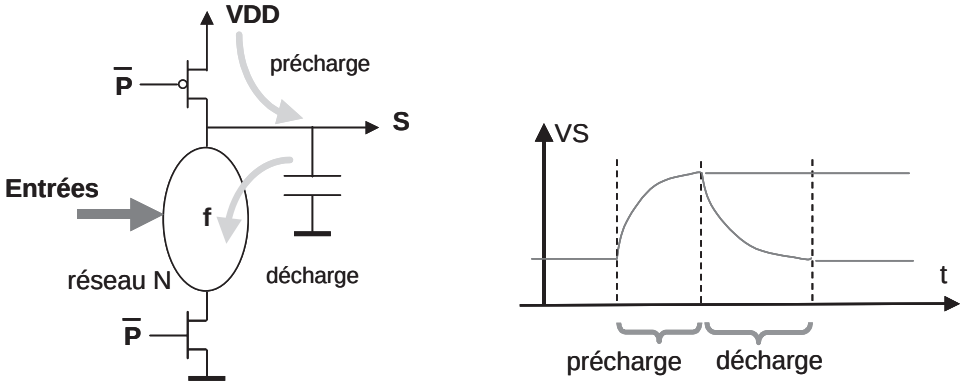


Figure 4.41 Principe de la logique dynamique

un premier temps, dit de *précharge*, la sortie est portée à un potentiel déterminé, généralement Vdd, par un transistor P dit de *précharge*. Dans un second temps, dit d'évaluation, la sortie est déchargée conditionnellement par un réseau de conduction N pour obtenir la valeur de sortie désirée.

Cette technique permet de remplacer un réseau P complexe et coûteux par un simple transistor P. Il est ainsi possible de réaliser économiquement des portes (non) OU à un très grand nombre d'entrées.

L'inconvénient majeur de cette technique est que ces montages ne peuvent être testés à vitesse réduite car le fonctionnement de la logique dynamique nécessite de respecter une vitesse minimale d'excitation.

4.5.1 Logique Domino

L'évaluation d'un ensemble de fonctions logiques peut être réalisé à l'aide d'une suite de portes dynamiques fonctionnant à la suite les unes des autres dans le même cycle de précharge-décharge. Pour cela, les sorties de chaque porte devront être inversées pour ne pas décharger la porte suivante pendant la phase de précharge. La décharge des sorties du premier niveau (pour leur donner leur valeurs) positionne les entrées des portes du second niveau et ainsi de suite jusqu'au portes terminales (effet « domino »).

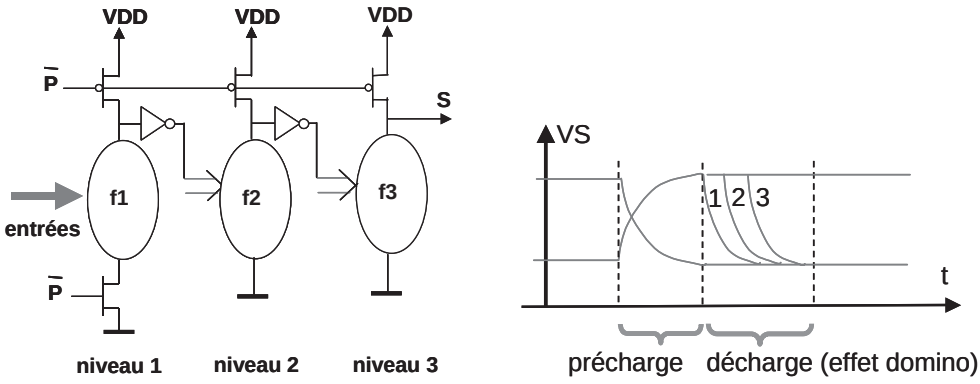


Figure 4.42 Principe de la logique Domino

4.5.2 Partage de charges

L'utilisation d'une succession de mémorisations dynamiques sans régénération des signaux amène une dégradation de la qualité des signaux par le phénomène du *partage de charges*. L'information, stockée dans des capacités, est reprise par des interrupteurs qui la connectent à d'autres capacités. Il se produit alors une égalisation des potentiels par transfert des charges au travers des interrupteurs (bidirectionnels). La tension finale peut être dégradée, voire imposée par la sortie si sa capacité est plus élevée que celle d'où provient l'information.

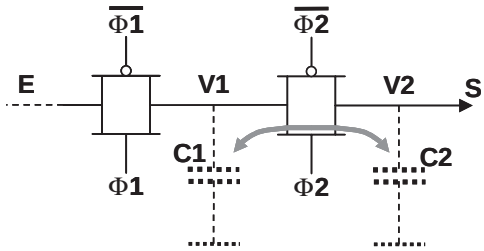


Figure 4.43 Effet de transfert de charges

La capacité C1 est chargée à V1 (VDD) pendant Φ1. La capacité C2 est alors chargée à V2. Pendant Φ2 se produit un transfert de charges et une égalisation des potentiels. En négligeant les pertes résistives, la valeur du potentiel final est :

$$Vf = \frac{C1V1 + C2V2}{C1 + C2}$$

Le phénomène de partage de charges peut aussi se produire à l'intérieur d'une porte 3 états et amener une perturbation de la sortie isolée.

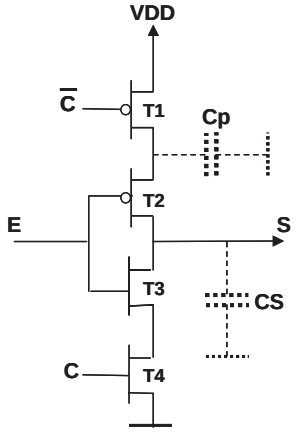


Figure 4.44 Exemple de partage de charges dans une porte 3 états

Pour montrer l'effet du partage de charges dans cette porte 3 états, supposons un état initial avec E = 1 et C = 1. Le transistor T2 est bloqué et tous les autres saturés. La capacité parasite Cp est portée à VDD. Si C < = 0, cette capacité devient isolée et reste chargée. Si maintenant l'entrée E devient 0, le transistor T2 devient conducteur et il se produit un transfert de charges entre Cp et CS, ce qui peut perturber le potentiel s'il correspond à une mémorisation dynamique. Les schémas ci-dessous ne présentent pas cet inconvénient (C^2MOS [SUZ73]).

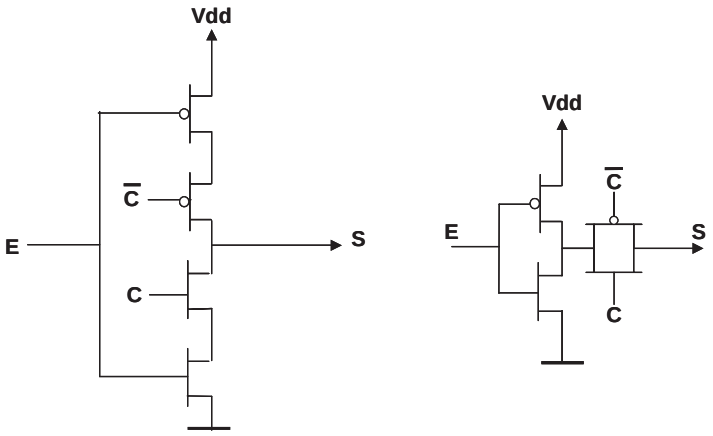


Figure 4.45 Portes 3 états sans effet de partage de charges

4.6 LOGIQUE MATRICIELLE

La réalisation matricielle de fonctions logiques ou de mémorisation permanente conduit à des réalisations très denses mais souvent peu rapides.

4.6.1 Matrice de ROM

La structure de base de la logique matricielle est la matrice de ROM (*Read Only Memory*/mémoire morte).

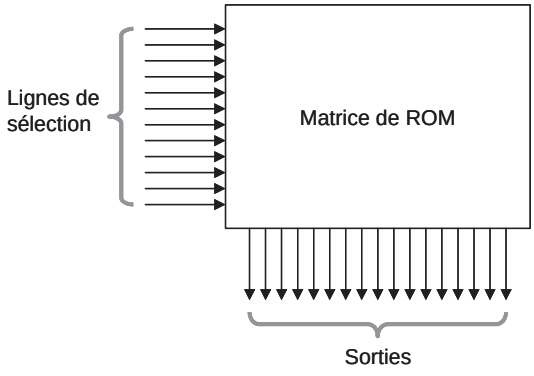


Figure 4.46 Matrice de ROM

Une matrice de ROM est excitée par plusieurs lignes de sélection. Elle fournit des valeurs sur des lignes de sortie. L'excitation d'une ligne de sélection (code 1 parmi n) provoque l'apparition d'un profil binaire particulier, lié à cette ligne de sélection, sur

les lignes de sortie. L'excitation simultanée de plusieurs lignes de sélection provoque la sortie d'une combinaison des profils liés aux lignes de sélection excitées. La nature de cette combinaison dépend de l'organisation de la matrice de ROM.

Deux organisations de matrice de ROM sont possibles qui diffèrent par leur encombrement et leur performance.

a) Matrice de ROM NOR

Elle se compose d'une série de portes NOR qui reçoivent un flux d'entrée commun sur ses lignes de sélection. Chaque transistor de la structure est branché entre la ligne de sortie du NOR auquel il appartient et Vss. Sa grille est connectée sur une ligne de sélection. Les lignes de sortie des portes NOR doivent être rappelées à 1 :

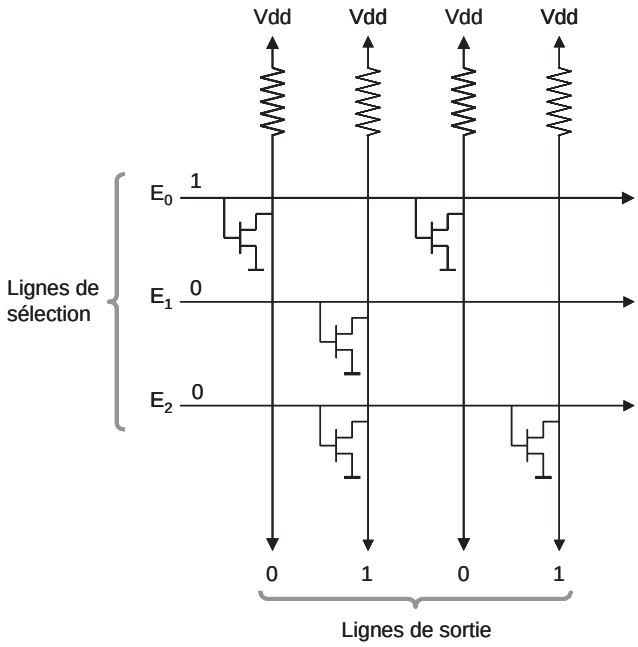


Figure 4.47 Matrice de NOR

L'excitation d'une ligne de sélection provoque la mise à 0 des lignes de sortie pour lesquelles un transistor est devenu conducteur. Cela revient à dire qu'un profil de sortie est associé à chaque ligne de sélection. La définition de ce profil, appelée sa *programmation*, se fait en insérant des transistors aux jonctions entre cette ligne de sélection et les lignes de sortie que l'on souhaite porter à 0. Les autres lignes resteront à 1, préchargées ou tirées à Vdd par leurs transistors de charge.

Les informations stockées dans la matrice constituent des mots dont la valeur apparaît en sortie lorsque la ligne de sélection correspondante est excitée.

b) Matrice de ROM NAND

Elle se compose d'une série de portes NAND qui reçoivent un flux d'entrée commun sur ses lignes de sélection. Tous les transistors qui constituent chaque porte NAND sont montés en série entre la ligne de sortie du NAND auquel ils appartiennent et Vss. La grille de chaque transistor de la série est connectée sur une ligne de sélection. Les lignes de sortie des portes NAND doivent être rappelées à 1. La ligne de sélection excitée est portée à 0, ce qui rend isolés les transistors auxquels elle est connectée.

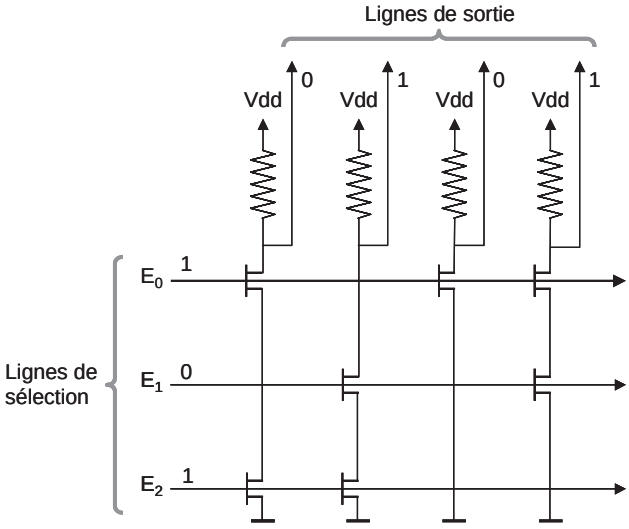


Figure 4.48 Matrice de NAND

c) Rappel des lignes de sortie

Qu'une matrice de ROM soit constituée de portes NOR ou NAND, ses lignes de sortie doivent être rappelées à Vdd :

– soit par des transistors P de précharge, dans le cas d'un fonctionnement dynamique de la matrice ;
– soit par des transistors P, dit « de charge », montés en résistance et connectés à Vdd. Ce type de porte (NOR ou NAND) est appelé NMOS et possède l'inconvénient de consommer lorsque la sortie est portée à 0. Les transistors de charge peuvent être commandés par un signal limitant la conduction à une phase d'évaluation (alimentation dite « pulsée »).

d) Programmation des matrices de ROM

La programmation des matrices de ROM consiste à introduire ou non des transistors dans les structures NOR ou NAND qui les constituent. Les techniques de programmation de ces matrices différent suivant la facilité avec laquelle on veut pouvoir modifier leur contenu.

La programmation des matrices NOR consiste à créer ou non des transistors aux croisements des lignes de sélection et de sortie de la matrice. Cette présence ou absence de transistors peut se faire de différentes manières.

– par la connexion, ou non, de transistors pré-existants ;
– par la modification, par implantation ionique, de la tension de seuil de transistors pré-existants pour qu'ils ne soient jamais conducteurs ;
– par la création même de transistors en mettant ou non une zone active sous la ligne de poly qui constitue leur grille.

La programmation des matrices NAND consiste à créer des suites de transistors en série. L'absence d'un transistor signifie qu'il remplacé par une connexion pour maintenir les autres transistors en série. La présence de transistors ou de connexions peut se faire de différentes manières :

– par le fait de court-circuiter des transistors pré-existants.
– par la modification, par implantation ionique, de la tension de seuil de transistors pré-existants pour qu'ils soient toujours conducteurs.

4.6.2 Utilisation des matrices de ROM comme reconnaisseurs/décodeurs

Les matrices de ROM peuvent être utilisées en reconnaisseur pour reconnaître les profils d'entrée. Pour cela, les lignes de sélection sont groupées par paire et attaquées par une entrée et son complément.

Un reconnaisseur programmé pour reconnaître tous les profils binaires qui peuvent apparaître sur ses lignes d'entrée est appelé un *décodeur.* Son entrée est alors appelée une *adresse* dont la valeur correspond à l'indice de la ligne de sortie excitée.

a) Reconnaisseurs NOR

Dans cet usage, une matrice NOR est utilisée pour mesurer les éventuels « désaccords » entre un profil en entrée et un autre enregistré dans la matrice sous la forme de transistors. Il suffit d'un seul désaccord sur une ligne d'entrée pour que la ligne de sortie correspondante soit portée à 0. Cette ligne ne reste à 1 que s'il n'y a aucun désaccord (figure 4.49).

La mesure des désaccords se fait en disposant des transistors entre la ligne de sortie et Vss. Les grilles de ces transistors sont branchées, soit sur une entrée directe, soit sur une entrée complémentée. Si la grille d'un transistor est branchée sur l'entrée directe, alors celui-ci devient conducteur si cette entrée est portée à 1, ce qui portera la ligne de sortie à 0. Une telle configuration est donc utilisée pour reconnaître un 0. Symétriquement, un transistor excité par la ligne d'entrée complémentée reconnaît un 1. L'absence de transistors sur ces deux lignes d'entrée ne produit jamais de désaccord. Cette configuration est appelée Φ et utilisée pour ignorer cette entrée dans le processus de reconnaissance. Par contre, la présence de deux transistors (sur les entrées directes et complémentées) produit toujours un désaccord qui porte toujours la ligne de sortie à 0. Cette configuration est donc inutilisée.

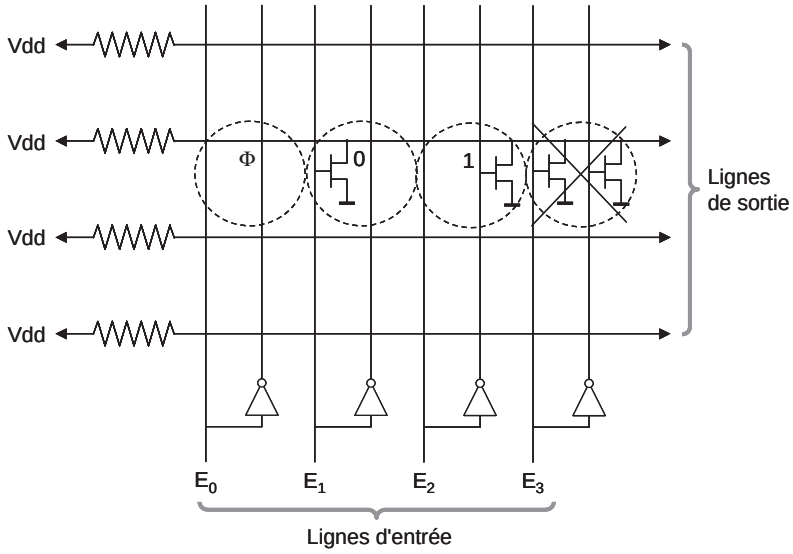


Figure 4.49 Programmation d'un reconnaisseur NOR

b) Reconnaisseurs NAND

Dans cet usage, chaque NAND d'une matrice reconnaît un profil particulier entre les lignes d'entrées directes et complémentées. Si la grille d'un transistor d'un NAND est branchée sur une entrée directe, alors ce transistor devient conducteur lorsque cette entrée est portée à 1. Si sa grille est branchée sur une entrée complémentée, alors ce transistor devient conducteur lorsque cette entrée est portée à 0. Lorsque tous les transistors d'un NAND sont conducteurs alors sa sortie passe à 0 (valeur active). Un NAND peut ignorer une entrée s'il ne comporte aucun transistor sur une entrée ou sur sa valeur complémentée. La configuration à deux transistors n'est jamais conductrice donc à éliminer (figure 4.50).

4.6.3 PLA-ROM

Le couplage d'un reconnaisseur et d'une matrice de ROM permet de câbler une fonction booléenne vectorielle (éventuellement incomplète) en tabulant la suite de ses arguments dans le reconnaisseur et la suite des valeurs correspondantes dans la matrice de ROM (figure 4.51).

Lorsque les deux matrices sont de même type NOR ou NAND, les lignes de sortie du reconnaisseur sont directement connectées sur les entrées de la matrice de ROM. Si elles sont de type différent il est nécessaire de prévoir une ligne d'inverseurs entre les deux matrices.

Le reconnaisseur est souvent appelé *matrice ET* et la matrice de ROM *matrice OU*. Nous verrons bientôt la raison de cette dénomination.

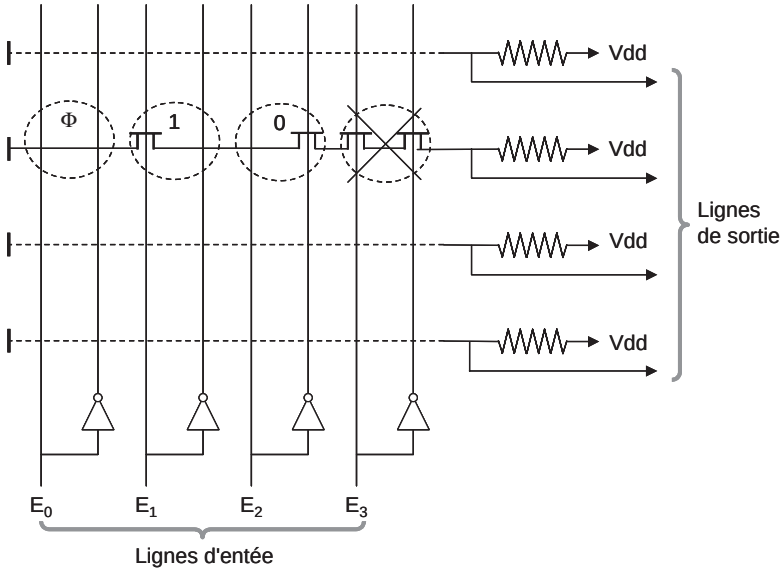


Figure 4.50 Programmation d'un reconnaisseur NAND

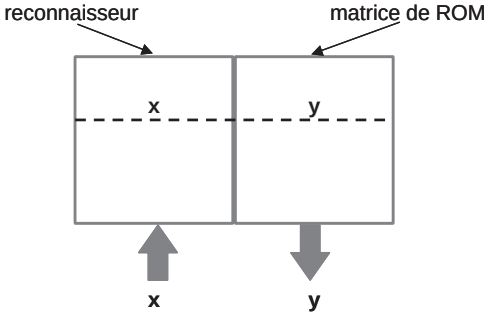


Figure 4.51 Tabulation de y = f(x)

Dans le cas d'un PLA NOR-NOR, la reconnaissance d'un profil d'entrée provoque le fait que seule sa ligne reste à 1 provoquant la sortie de la valeur stockée dans la matrice de ROM pour cette entrée (figure 4.52).

Dans le cas d'un PLA NAND-NAND, la reconnaissance d'un profil d'entrée provoque le fait que seule sa ligne est portée à 0. Ce qui provoque la sortie de la valeur de la fonction stockée dans la matrice de ROM (figure 4.53).

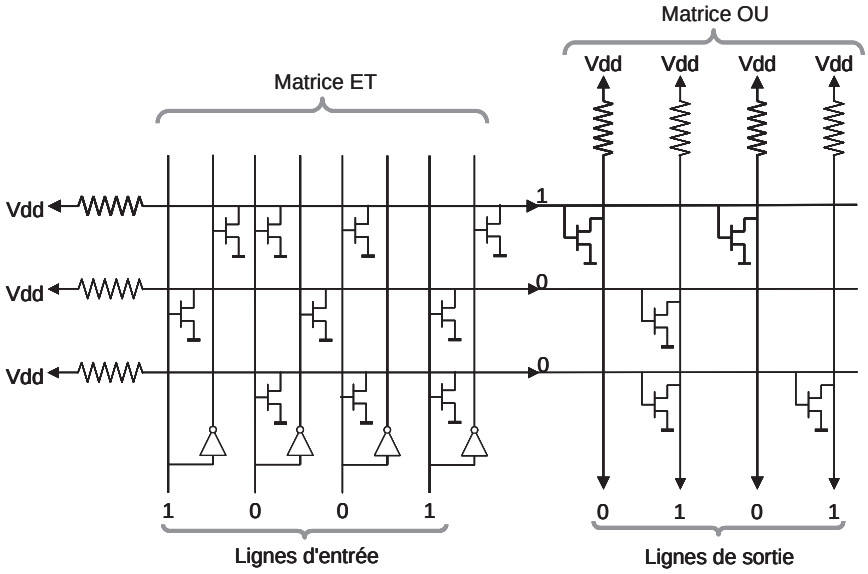


Figure 4.52 PLA NOR-NOR

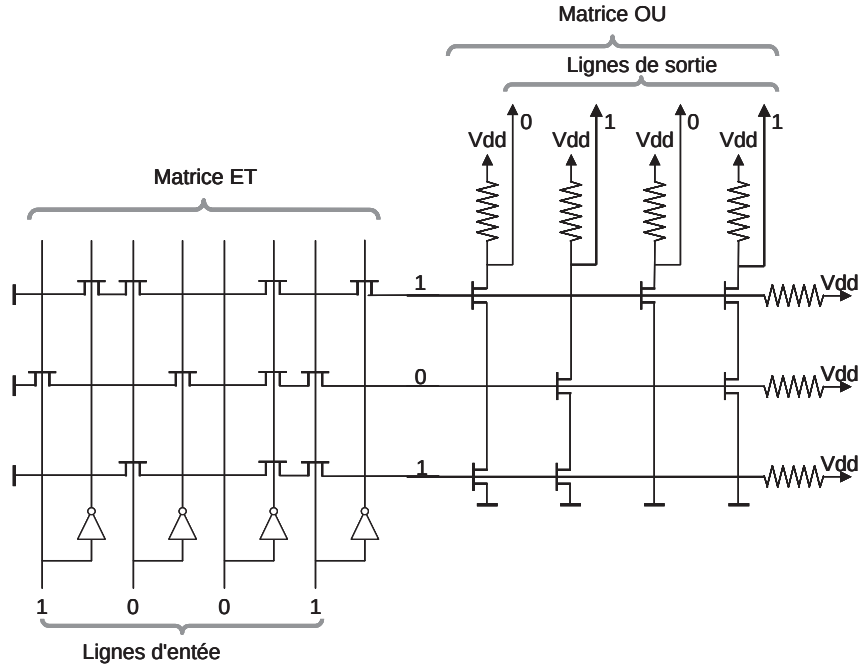


Figure 4.53 PLA NAND-NAND

4.6.4 ROM

Le couplage d'un décodeur et d'une matrice de ROM donne une ROM (*Read Only Memory*/mémoire morte). Ce dispositif permet de stocker des informations dans les différents mots de la matrice de ROM sélectionnés par le décodeur. L'inscription de ces information constitue la programmation de la ROM. Pour une matrice NOR, elle se fait en disposant des transistors entre les lignes de sortie et Vss. Les grilles de ces transistors sont connectées aux sorties du décodeur. Il existe plusieurs façons de réaliser et de connecter ces transistors suivant la facilité avec laquelle on souhaite pouvoir modifier le contenu de la ROM ainsi que la densité du dessin que l'on souhaite obtenir.

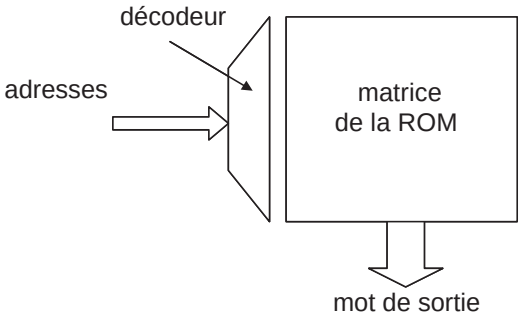


Figure 4.54 Représentation schématique d'une Rom

Les décodeurs de ROM ne sont pas toujours réalisés à l'aide de reconnaisseurs. Ils peuvent être réalisés à l'aide de portes ou de structures hybrides portes et matrices (figure 4.55).

L'utilisation de portes NAND se justifie par le fait q'elles nécessitent des transistors P plus petits, malgré la nécessité d'un inverseur supplémentaire qui peut servir d'amplificateur pour exciter des matrices importantes.

4.6.5 PLA booléen

Le couplage d'un reconnaisseur (matrice ET) et d'une matrice de ROM (matrice OU) peut aussi être vu comme un moyen de réaliser autant de fonctions booléennes qu'il y a de sorties de la matrice OU.

a) PLA NOR-NOR

Ce point de vue s'appuie sur le fait que l'expression des sorties de la matrice ET est donné par l'expression :

$$mi = \overline{\sum(E_j, \overline{E_k})}$$

qui peut se réécrire :

$$mi = \prod(\overline{E_j}, E_k)$$

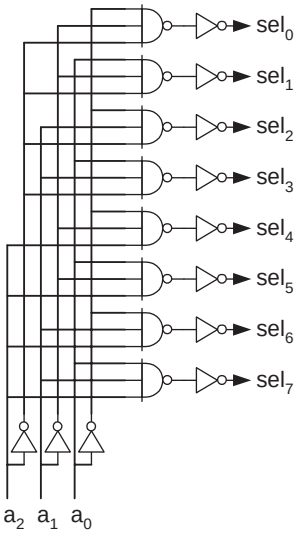


Figure 4.55 Décodeur réalisé avec des portes « classiques »

Ce qui justifie le nom de matrice ET donné au reconnaisseur.

Le fonctionnement de la matrice OU est donné par l'expression :

$$sp = \overline{\sum_i (mi)}$$

d'où, pour l'ensemble du PLA :

$$sp = \overline{\sum_i (\prod (\overline{E_j}, E_k))}$$

qui montre que, à une négation près, les sorties du PLA sont des fonctions booléennes exprimées en somme de produits des variables d'entrée directes et complémentées. La négation terminale peut être facilement incluse dans la circuiterie qui exploite les sorties du PLA. Dans ce cas, plusieurs lignes intermédiaires, appelées *monômes*, peuvent être simultanément activées.

b) PLA NAND-NAND

Ce point de vue s'appuie sur le fait que l'expression des sorties de la matrice ET est donné par l'expression :

$$mi = \overline{\prod (E_j, \overline{E_k})}$$

ce qui justifie, à une négation près, le nom de matrice ET donné au reconnaisseur.

Le fonctionnement de la matrice OU est donné par l'expression :

$$sp = \overline{\prod_i (mi)}$$

qui peut se réécrire :

$$sp = \sum_i (\overline{mi})$$

d'où, pour l'ensemble du PLA :

$$sp = \sum_i (\prod (E_j, \overline{E_k}))$$

qui montre que les sorties du PLA sont des fonctions booléennes exprimées en somme de produits des variables d'entrée directes et complémentées. Dans ce cas, plusieurs lignes intermédiaires, appelées *monômes*, peuvent être simultanément activées.

4.6.6 Alimentation pulsée

La réduction de la consommation des PLA peut être obtenue en ne les alimentant que lorsqu'ils sont utilisés.

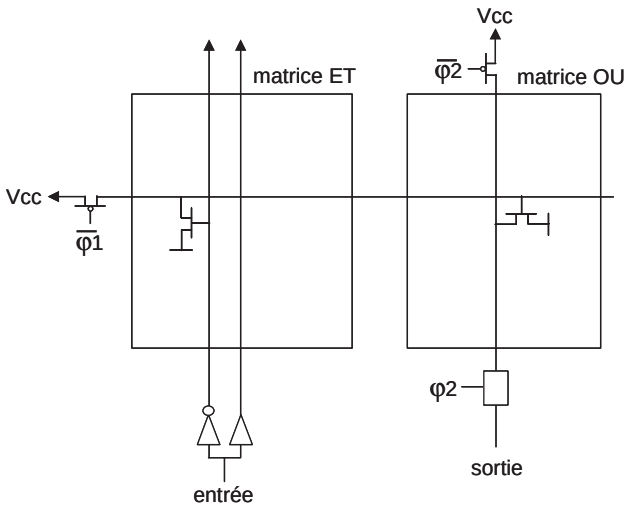


Figure 4.56 PLA à alimentation pulsée

Cette technique peut être raffinée en alimentant successivement les matrices ET et OU, mais il faut alors mémoriser la valeur des monômes pour permettre le fonctionnement de la matrice OU. Les valeurs maintenues pendant tout le cycle de fonctionnement du PLA, le sont de manière dynamique.

4.6.7 PLA dynamique

Il est possible de réaliser un PLA ayant un fonctionnement dynamique. Pour cela, il faut réaliser des matrices ET et OU ayant elles-mêmes un comportement dynamique. Les matrices doivent pouvoir être isolées car la matrice OU fonctionne pendant que les lignes de sortie de la matrice ET sont préchargées.

– Phase de précharge : PLA NOR-NOR : Les lignes de sortie de la matrice sont préchargées, tandis que toutes les lignes d'entrée sont maintenues à 0 pour éviter toute décharge. PLA NAND-NAND : Les séries de transistors constituant les portes NAND peuvent être complétées par un transistor, commandé par la phase d'évaluation, évitant la décharge
– Phase d'évaluation : Le mécanisme de précharge est coupé et les lignes d'entrée prennent leur valeur. Les lignes de sortie se déchargent conditionnellement.

Il faut aussi noter que, comme toute structure dynamique, le fonctionnement d'un PLA dynamique impose une vitesse minimale de fonctionnement, peu compatible avec des procédures de test.

a) PLA NOR-NOR dynamique

La réalisation d'un PLA NOR-NOR dynamique pose plusieurs problèmes :

– Les lignes d'entrées de la matrice OU étant excitées à partir des lignes de sortie de la matrice ET, il est nécessaire d'insérer un mécanisme pour les maintenir à 0 pendant la précharge de la matrice OU.
– Le fonctionnement biphasé du PLA nécessite de croiser celui de ses matrices : La précharge de la matrice ET se produit pendant l'évaluation de la matrice OU. Cette dernière doit donc travailler avec les valeurs des monômes élaborés dans la phase précédente. De même, l'évaluation de la matrice ET se produit pendant la précharge de la matrice OU dont les entrées doivent être maintenues à 0.

La réalisation d'un PLA biphasé nécessite donc l'introduction d'une circuiterie relativement importante entre les deux matrices, précisément là où les lignes de monôme sont très serrées.

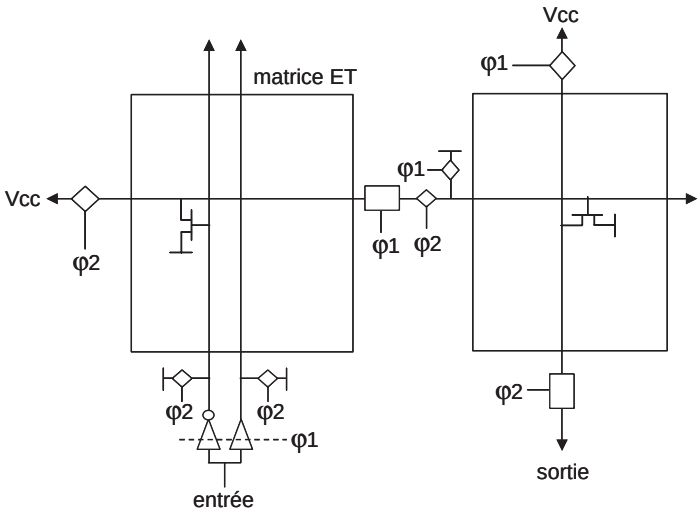


Figure 4.57 PLA NOR-NOR biphasé

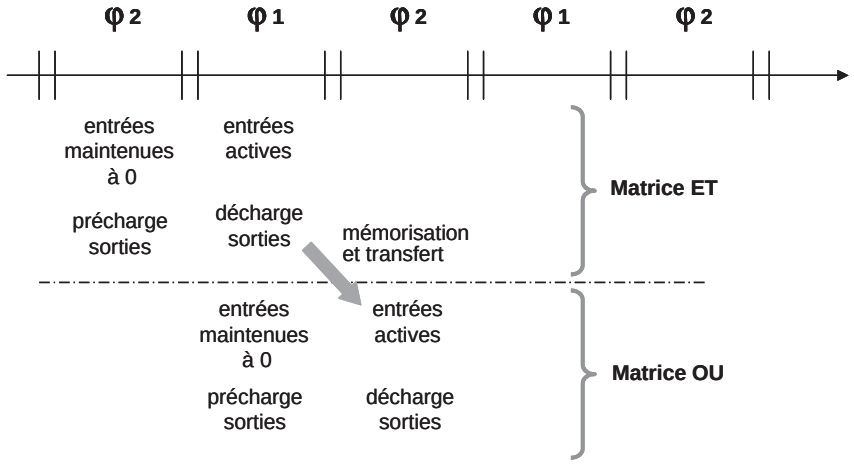


Figure 4.58 Fonctionnement du PLA NOR-NOR biphasé

b) PLA NAND-NAND dynamique

La réalisation d'un PLA NAND-NAND dynamique est plus simple puisqu'il suffit d'ajouter un transistor aux séries qui constituent les portes NAND pour éviter leur décharge. Il faut néanmoins séparer les matrices pendant la phase Φ2 puisque les sorties de la matrice ET doivent être préchargées pendant que la matrice OU évalue ses sorties.

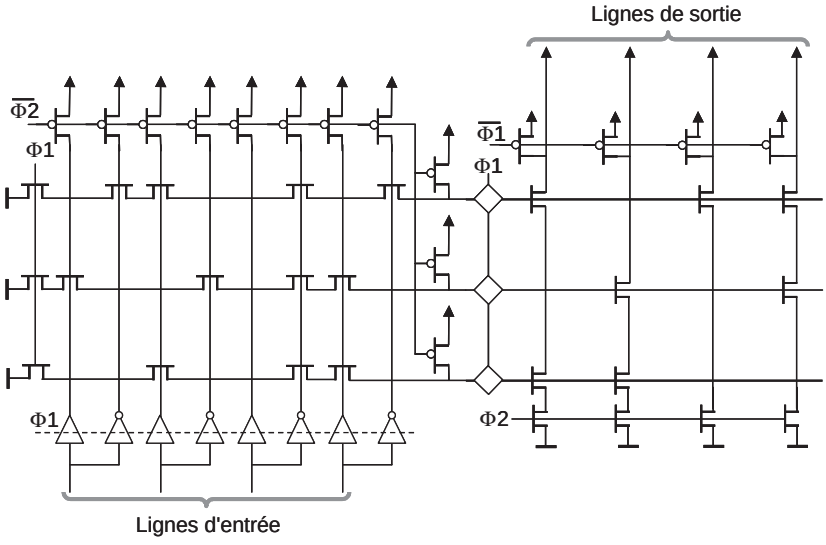


Figure 4.59 PLA NAND-NAND biphasé

4.6.8 Optimisation des PLA

Les PLA peuvent faire l'objet de différentes techniques d'optimisation. Il faut noter que les matrices NAND sont plus compactes que les matrices NOR qui sont elles-mêmes plus rapides.

a) Optimisation booléenne

Tout PLA peut être considéré comme un PLA booléen et être optimisé avec des techniques de minimisation de l'ensemble des fonctions booléennes qu'il réalise. Toutefois, cette optimisation, en réduisant la taille des matrices va accroître la difficulté de les connecter aux organes qui leur fournissent des valeurs et à ceux qui utilisent leurs résultats. Les techniques d'optimisation topologiques qui permettent d'améliorer la connectivité du PLA s'avèrent plus efficaces que celles qui portent sur son contenu.

b) Coins coupés (PLA NOR-NOR)

La présence de Φ dans la matrice ET et de 1 dans la matrice OU correspond à l'absence de transistors. La permutation des monômes, des entrées et des sorties permettent de repousser ces absences dans des coins qui peuvent être enlevés, libérant de la place pour y installer d'autres fonctions.

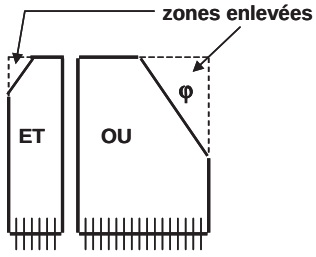


Figure 4.60 Optimisation des PLA par création de zones vides

Il faut remarquer que cette optimisation n'améliore pas la « connectabilité » du PLA.

c) Sorties latérales (PLA NOR-NOR)

Les PLA complexes sont souvent assez longs. Les sorties de la matrice OU peuvent alors être réalisées sur son côté à des positions qui permettent des connexions directes avec les blocs qui les utilisent [SEG85]. Ceci nécessite une réorganisation des monômes qui peuvent être disposés près du barycentre des positions des sorties auxquelles ils contribuent. Les monômes et les lignes de sortie peuvent être réduites à des segments les plus courts possibles. La place libérée dans la matrice OU par les monômes raccourcis permet de tirer des connexions entre les segments de sortie et leur utilisation, à des emplacements qui permettent leur connexion directe.

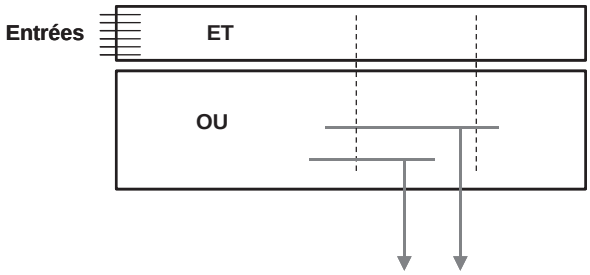


Figure 4.61 PLA à sorties latérales

BIBLIOGRAPHIE

[CAL58] S.H. Caldwell, *Switching Circuit and Logical Design*, Willey, New York 1958.

[SUZ73] Y. Suzuki, Clocked CMOS Calculator Circuitery, *IEEE journal of Solid State Circuits*, Dec. 1973.

[HEL84] L. Heller *et al.*, *Cascade Voltage Switch Logic : A Differential CMOS Logic Family*, Proc. IEEE ISSCC Conference, pp. 16-17, Feb. 1984.

[SEG85] T. Perez Segovia, *PAOLA : Un système d'optimisation topologique de P.L.A.*, thèse INPG, 25 octobre 1985, Grenoble.

Chapitre **5**

Dessin des masques d'un circuit intégré

5.1 DÉFINITION DU PROBLÈME

Le dessin des masques d'un circuit intégré peut se faire à l'aide de différents outils suivant le degré d'optimisation choisi (nous verrons les différents outils qui peuvent être utilisés dans le pragraphe 11.4). Très souvent, les cellules sont dessinées manuellement. Dans ce cas, il s'agit de représenter, dans les différentes couches d'un dessin, les motifs qui devront être réalisés dans les différents niveaux de photogravure. Ces dessins doivent obéir à un ensemble de règles géométriques très précises fournies par le fondeur (taille, espacement…). Ces règles représentent les contraintes de la technologie utilisée.

Le dessin des masques d'un circuit complexe peut devenir un travail gigantesque si l'on ne s'organise pas de manière efficace. Par rapport à la conception des cartes électroniques, la conception des circuits VLSI offre des degrés de liberté supplémentaires qui sont liés au fait que l'on doit tout dessiner, depuis les transistors jusqu'aux interconnexions. Cela permet d'optimiser fortement la densité des blocs et d'innover sur la circuiterie. Il est ainsi possible de créer des fonctions adaptées à l'application (comme par exemple des hybrides de ROM et de PLA).

L'organisation spatiale d'un circuit complexe s'apparente à de l'urbanisme (ou de l'organisation territoriale !). L'utilisation d'organisations particulières permet de réaliser des circuits plus optimaux. L'enjeu de la conception d'un circuit VLSI consiste souvent à transposer dans un domaine topologique le problème initial qui est souvent posé de manière fonctionnelle (figure 5.1).

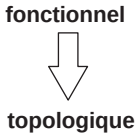


Figure 5.1

5.2 CONCEPTION TOPOLOGIQUE

Bien que très grande, la surface d'un circuit intégré s'avère souvent trop petite pour y loger tout ce que l'on souhaiterait y mettre. Il ne faut jamais oublier que le coût d'un circuit et son rendement de fabrication dépendent fortement de sa surface. Plus celle-ci est importante, moins il y aura de circuits réalisés sur une tranche et plus la probabilité de défauts sur un circuit sera importante.

Un circuit intégré est constitué d'une hiérarchie de blocs imbriqués dont les feuilles représentent les cellules. Le dessin d'un bloc de niveau i est obtenu par l'assemblage et l'interconnexion de blocs du niveau i + 1 (figure 5.2). Cet assemblage doit se faire le plus efficacement possible. Pour cela, on ajustera la forme des blocs de niveau i + 1 pour faciliter leur imbrication. Jusqu'à un certain niveau, on recherchera à ce que les interconnexions s'effectuent par la simple juxtaposition des blocs plutôt que de devoir tirer des fils qui occupent de la surface de silicium ou des couches d'interconnexion.

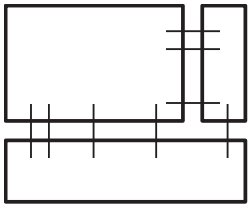


Figure 5.2 Assemblage et interconnexion directe des principaux blocs d'un circuit VLSI

Pour améliorer l'imbrication des blocs, ceux-ci doivent pouvoir se déformer. Comme ils contiennent des éléments « incompressibles » tels que les transistors et les contacts, leur loi de *déformabilité* a l'allure présentée figure 5.3.

Plutôt que de contourner un bloc par une connexion, il est préférable de chercher à le traverser. Nous appellerons *transparence* la propriété d'un bloc de se laisser traverser par des connexions qui ne le concernent pas (figure 5.4).

La conception des circuits VLSI est ainsi un bon exemple du fait qu'une optimisation globale ne résulte pas forcément de l'assemblage de sous-ensembles optimaux (optimisations locales).

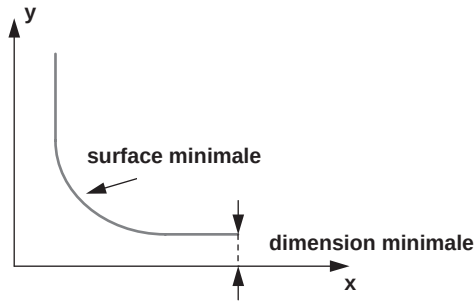


Figure 5.3 Loi de déformation d'un bloc dessiné manuellement

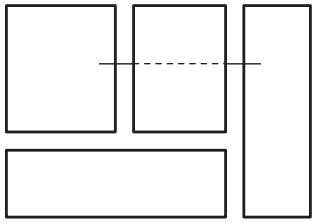


Figure 5.4 Transparence d'un bloc

5.3 RÈGLES SYMBOLIQUES

L'utilisation des règles technologiques se heurte à leur spécificité et à leur précarité. En effet, les dessins réalisés avec un jeu de règles ne peuvent pas être utilisés directement par la technologie suivante et encore moins par un autre fondeur. Pour contrer ces effets, plusieurs types de règles standardisées ont été développées. Ces règles dites « symboliques » ont pour objectif d'assurer la portabilité des dessins d'une technologie à la suivante et d'un fondeur à l'autre. Cette généralité se paye par un degré d'optimisation moindre qui se traduit par une perte de surface. Différentes approches ont été utilisées :

– dessin sur grille ;
– dessins symboliques à pas fixe ou variable, « engraissés » ou non.

Une autre approche, utilisée par les fondeurs pour la migration de leurs cellules d'une technologie à la suivante, consiste à utiliser des outils de migration qui réalisent une homothétie « intelligente » en isolant les motifs dont l'évolution n'est pas homothétique. Ceux-ci seront alors remplacés automatiquement ou à la main.

5.3.1 Règles dites « au Lambda »

Cette approche à été introduite par C. Mead et L. Conway [MEA80]. Elle consiste à dessiner sur une grille « unitaire » avec un jeu de règles définitivement fixé pour toute

une famille technologique (par exemple le CMOS). L'adaptation à une technologie réelle se fait en choisissant un pas de grille suffisamment grand pour permettre la transposition directe des dessins vers cette technologie (figure 5.5).

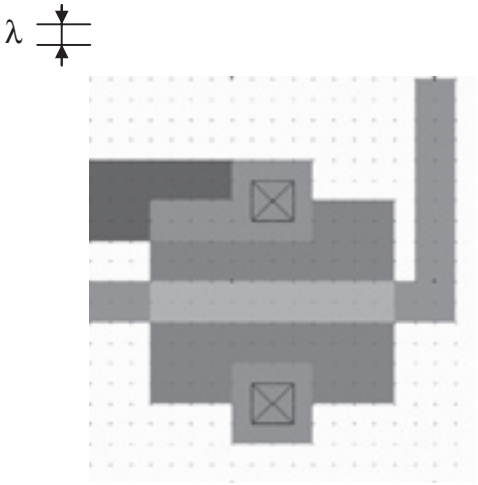


Figure 5.5 Exemple de dessin « au Lambda »

Le principal inconvénient de cette approche réside dans le fait que ces règles, définies pour la technologie nMOS de la fin des années 1970, ne correspondent plus aux technologies modernes qui n'ont pas évoluées homothétiquement depuis cette date et qui en plus, comportent maintenant des motifs de taille fixe (par exemple les contacts). L'utilisation de ces règles peut amener des pertes de surface et de performance importantes (du simple au double). Elles ne sont plus utilisées par l'industrie mais seulement dans l'enseignement.

5.3.2 Dessin symbolique sur grille

Dans cette approche, le dessin est toujours effectué sur une grille à pas unitaire (figure 5.6). Toutefois, le dessin est réalisé à l'aide de symboles (souvent paramétrables qui représentent des éléments (transistors, contacts, fils…). La transposition (appelée *expansion*) de ces dessins vers une technologie donnée est réalisée par un processus complexe :

– Le pas de la grille est calculé à partir des propriétés de la technologie visée.
– Chaque symbole est transformé en une pièce de dessin en utilisant les règles fines de la technologie visée.

Le résultat de cette expansion est assez optimisé. La perte n'est que de l'ordre de 20 %. Pour faciliter le confort du dessin, les symboles sont immédiatement transformés :

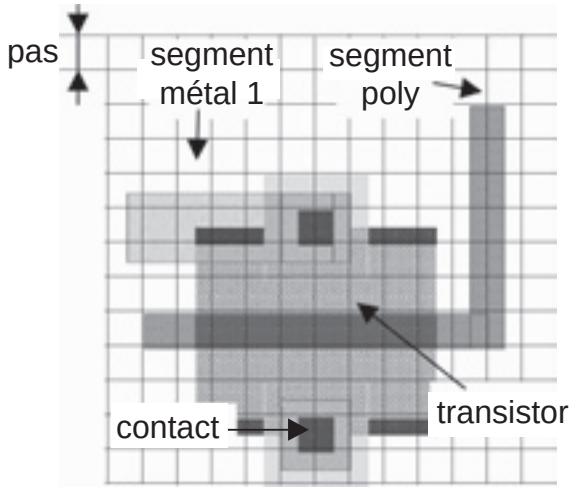


Figure 5.6 Dessin symbolique sur grille

– soit en dessins squelettiques (voir § 5.6) ;
– soit en utilisant des motifs correspondants à une technologie « moyenne » (dite *symbolique*).

Un jeu de règles définis sur le dessin symbolique permet d'assurer que son expansion sera toujours correcte.

Cette approche à été utilisée par l'industrie (Bull S.A.) pour réaliser des circuits très complexes qui ont pu être fabriqués chez plusieurs fondeurs. Elle est aussi à la base du système Alliance.

5.4 COUCHES TECHNOLOGIQUES ET FLUX D'INFORMATION

Un circuit intégré est d'abord une surface plate qui offre à son concepteur un certain nombre de couches technologiques correspondants aux principaux masques à dessiner :

– *zones actives* de type N et P (implantées ou diffusées) ;
– *polysilicium* (grilles et connexions à très courte distance) ;
– métaux (connexions et alimentations).

Outre leurs propriétés spécifiques, comme celles de permettre la réalisation des transistors, les qualités de conductions de ces couches vont en s'améliorant des zones actives aux métaux.

5.4.1 Organisation matricielle du dessin des blocs

Pour réduire le travail de conception d'un circuit intégré, on cherchera à dessiner ses blocs par la répétition de cellules identiques. Nous verrons que cette approche facilite

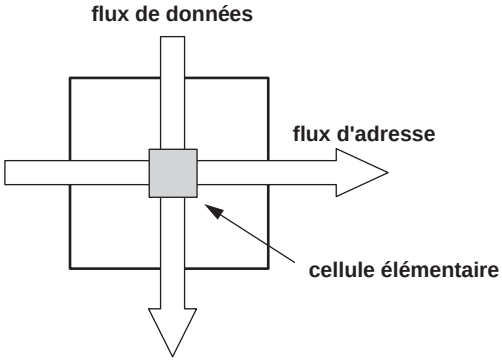


Figure 5.7 Le croisement des flux dans un bloc matriciel

aussi les interconnexions. Pour cela, nous chercherons à voir chaque bloc comme le croisement de deux, ou plus, flux d'information, d'horlogerie ou d'alimentation. Par exemple, une mémoire peut être vue comme le croisement d'un flux de données (à écrire ou à lire) et d'un flux de sélection des mots (issu de l'adresse). Chaque point de croisement entre les informations élémentaires de ces flux donnera une cellule (figure 5.7).

Un bloc de ce type se présente alors comme une matrice rectangulaire constituée de cellules identiques qui s'interconnectent par simple juxtaposition dans les deux directions (figure 5.8).

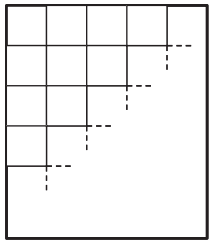


Figure 5.8 Constitution d'un bloc par la juxtaposition de cellules

De telles organisations matricielles sont utilisées pour le dessin de nombreux blocs comme des mémoires vives ou mortes, des chemins de données, etc. La régularité des dessins obtenus fait qu'ils peuvent être automatiquement réalisés par des programmes informatiques appelés *générateurs* ou *assembleurs de silicium*.

On peut caractériser l'efficacité d'une stratégie de dessin par le *taux de réutilisation* des cellules. Un bon critère de mesure de la qualité du dessin d'un circuit consiste à évaluer l'inverse de la proportion de sa surface qui ne comporte pas de transistors (zones vides ou d'interconnexion).

5.4.2 Affectation des flux aux couches technologiques

Nous affectons chaque flux, ainsi que les alimentations, à une couche technologique (figure 5.9).

TABLEAU 5.1

Niveau	Couches techno.	Réalise
0	caissons	
1	zones actives (dopées)	Transistors Connexions locales
2	polysilicium	Connexions locales
3	métal 1	flux 1 + alim locales
4	métal 2	flux 2 + alim globales
.....	.....	.....

Cela signifie :

– que les couches sont croisées ;
– que les matériaux sont utilisés de manière « linéaire » ;
– qu'il faut éviter les motifs en « L ».

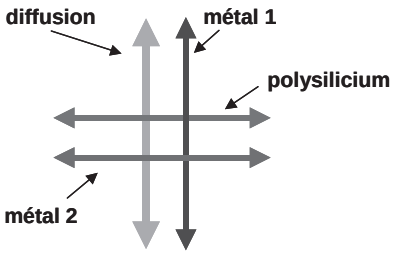


Figure 5.9 Croisement des flux dans le dessin des cellules

5.5 DESSIN DES PORTES CMOS « CLASSIQUES »

Les principes généraux précédents peuvent être appliqués au dessin des portes CMOS en considérant le flux des signaux logiques et celui des alimentations.

Le dessin d'une porte de ce type nécessite de réaliser deux réseaux de conduction (un N et un P). Comme chaque entrée attaque le même nombre de transistors dans chacun de ces deux réseaux, il faut donc essayer de simplifier leurs interconnections en alignant les transistors N et P ayant les mêmes grilles.

Il existe deux stratégies pour dessiner les portes classiques :

– Une approche que nous pouvons qualifier de « verticale ». Ces cellules sont généralement destinées à être assemblées par juxtaposition, donc reliées par des connections très courtes, qui peuvent être en polysilicium et constituer directement les grilles des portes suivantes. La faible longueur de ces interconnections permet d'utiliser des transistors de taille minimale.
– Une approche que nous pouvons qualifier d'« horizontale ». Ces cellules sont généralement destinées à être assemblées par des outils de câblage automatiques (approche dite à base de cellules *précaractérisées*). Ces outils tirent, en général, de longs fils métalliques dans des canaux de câblage ou au-dessus des cellules. L'excitation de ces longs fils nécessite de dessiner des transistors plus larges.

5.5.1 Dessin d'un réseau de conduction

L'idéal consiste à dessiner un réseau de conduction comme une bande d'active barrée par des lignes de polysilicium qui définissent des transistors. Des connections locales en métal sont utilisées pour mettre en parallèle ou en série les transistors du réseau (figure 5.10).

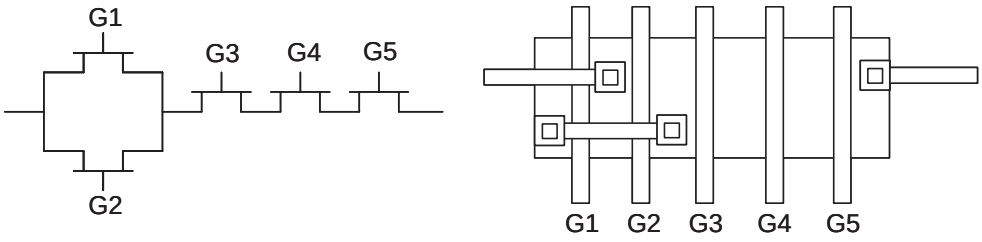


Figure 5.10 Exemple de réseau de connexion et de son dessin

Le dessin d'un réseau de conduction d'un seul tenant n'est pas toujours possible. Il doit quelquefois être découpé en plusieurs sous-réseaux. Il est important de remarquer que, dans chaque paquet de transistors en série ou en parallèle, les transistors peuvent être permutés, et que les paquets en série (ou en parallèle) peuvent être permutés entre eux.

Théorie

Un réseau de conduction peut être représenté par un graphe dans lequel les sommets sont les équipotentielles du réseau et les arcs les transistors (figure 5.11). On montre que ce réseau peut être réalisé d'un seul tenant si le graphe correspondant admet un *parcours d'Euler*[1]. L'ordre des transistors sur la bande d'active correspond à l'ordre des arcs dans ce parcours.

1. On appelle parcours d'Euler d'un graphe, un chemin qui parcourt tous ses arcs une seule et unique fois.

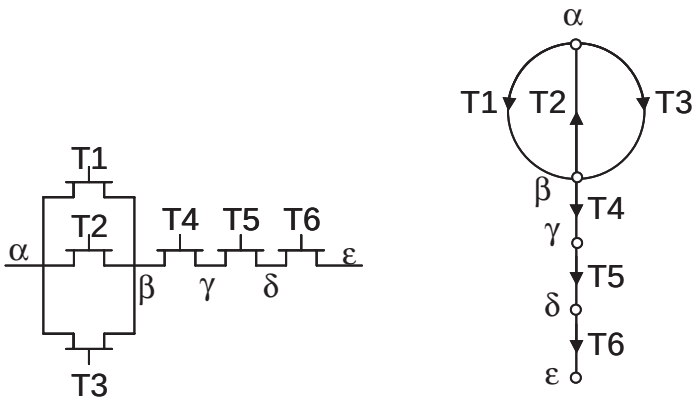


Figure 5.11 Exemple de réseau de connexion et de son graphe

5.5.2 Dessin des portes classiques

Les réseaux de conduction N et P qui constituent ces portes sont duaux et attaqués par les mêmes entrées. Pour des raisons d'optimisation évidentes, les transistors N et P reliés à une même entrée doivent être alignés, ce qui signifie que les parcours d'Euler des graphes correspondants doivent parcourir leurs transistors dans le même ordre (figure 5.12).

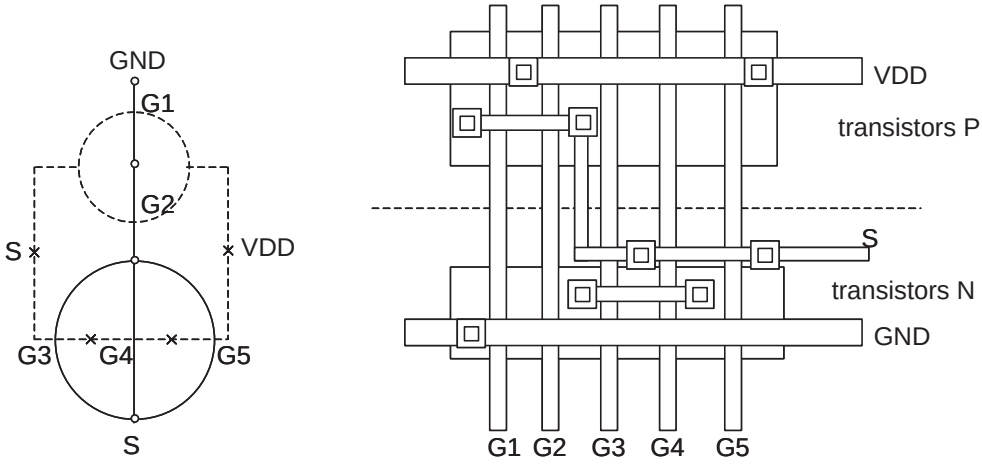


Figure 5.12 Exemple de graphes de réseaux duaux et de leur dessin

Du fait de la dualité des réseaux de connexion, les paquets de transistors en série dans le réseau N correspondent à des transistors en parallèle dans le réseau P et inversement.

Il est facile de voir que la réalisation d'un seul tenant d'un nombre impair de transistors en parallèle ne pose aucun problème. La mise en série des transistors complémentaires correspondants est triviale.

Une solution pour réaliser les paquets comportant un nombre pair de transistors en parallèle consiste à leur ajouter un transistor dit « virtuel », toujours bloqué. Ces transistors virtuels pourront ensuite être enlevés en laissant des trous (discontinuités dans les bandes d'active). La permutation des transistors dans un paquet et la permutation des paquets en série (ou en parallèle) permet de repousser une partie des trous aux extrémités d'où ils peuvent être éliminés en raccourcissant la porte.

a) Approche verticale

Les bandes d'active de faible largeur sont disposées verticalement, alternativement N et P. Les portes successives se développent horizontalement. Leurs réseaux de conduction sont permutés à chaque étage ce qui permet de regrouper les bandes d'active de même type dans des caissons verticaux. Les grilles sont constituées de lignes de polysilicium horizontales qui partent de la sortie des portes amont, c'est-à-dire de la liaison de leurs réseaux de conduction N et P. Des lignes métalliques verticales servent à effectuer les connections de mise en parallèle dans les réseaux de conduction N et P, ainsi que les alimentations primaires. Une seconde couche de métal, horizontale, peut éventuellement servir à la distribution globale des alimentations et aux connections à grande distance (figure 5.13). Le strict respect de l'utilisation « linéaire » des couches facilite la transparence des cellules.

Cette approche convient particulièrement pour le dessin de cellules communiquant directement entre voisines immédiates, ce qui correspond à l'utilisation de « petits » transistors. C'est l'approche utilisée pour le dessin manuel.

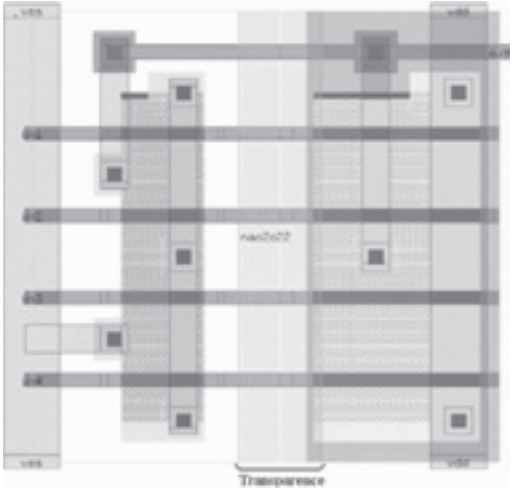


Figure 5.13 Cellule nao2o22 dessinée « verticalement »

b) Approche horizontale

Les bandes d'active de grande largeur (variable) sont disposées horizontalement constituant des transistors de grande largeur. Les portes successives se développent sous la forme de bandes horizontales. Les caissons N et P sont aussi des bandes horizontales qui parcourent chaque bande de cellules. Les grilles sont des bandes de polysilicium verticales qui ne servent pas à l'interconnexion des cellules (figure 5.14).

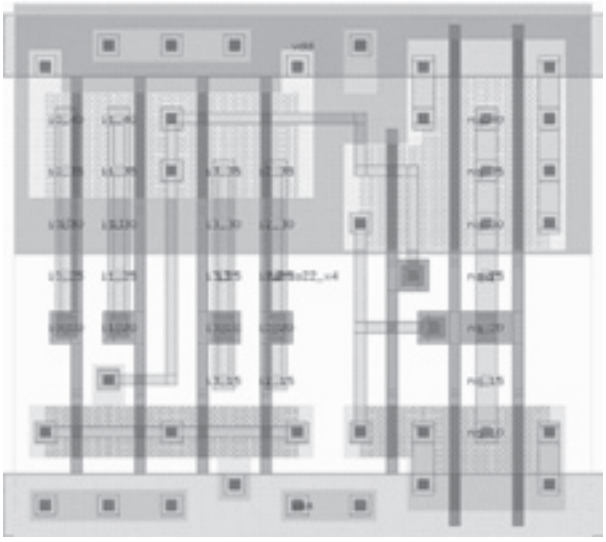


Figure 5.14 Dessin « horizontal » de la cellule nao2o22_X4 de la librairie SXLIB du système Alliance

La cellule nao2o22_X4 se compose d'une porte CMOS complexe réalisant la fonction $(A \vee B) \wedge (C \vee D)$ suivie de deux étages d'amplification pour obtenir une très forte sortance. On remarquera l'écartement des transistors pour permettre les connexions. Cette approche convient particulièrement au dessin de cellules communiquant à grande distance. Elle est systématiquement utilisée par les outils de conception automatique.

➤ Interconnexion des cellules

La disposition et l'interconnexion des cellules peut se faire de différentes manières suivant que l'on considère un bloc dessiné manuellement ou automatiquement (figure 5.15) :

– Dans un dessin manuel les cellules sont placées côte à côte au plus près et reliées avec des bandes de polysilicium (voisinage direct) et avec des bandes métalliques. Les cellules, dessinées verticalement, sont alternativement retournées sur un axe vertical pour partager les lignes d'alimentation et les caissons.
– Dans le cas de dessin automatique, les cellules dessinées horizontalement sur un gabarit de hauteur fixe, sont alignées, par un outil de *placement*, sous la forme de

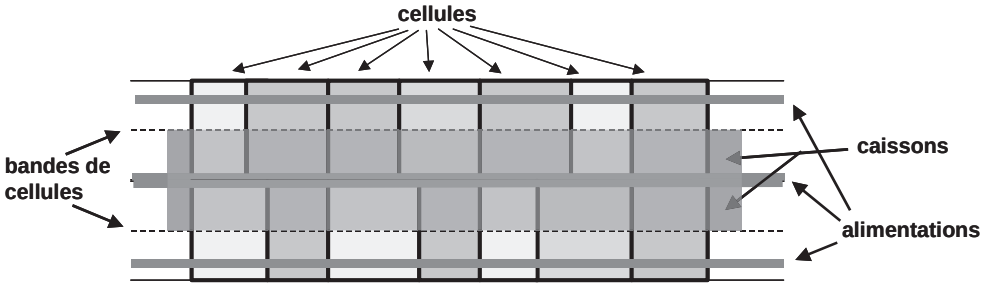


Figure 5.15 Bandes de cellules partageant leurs caissons

bandes horizontales. Les lignes d'alimentation se prolongent ainsi sur toute la longueur des bandes. Alternativement, les bandes de cellules sont retournées suivant un axe horizontal de manière à partager les caissons et les barres d'alimentation avec la bande suivante.

Tant que les technologies n'ont offert que peu de niveaux de métal, l'interconnexion des cellules s'est faite *via* des canaux de câblage dessinés automatiquement. Le dessin d'un canal suit les règles générales et n'utilise que deux couches de métal (une horizontale et une verticale) (figure 5.16).

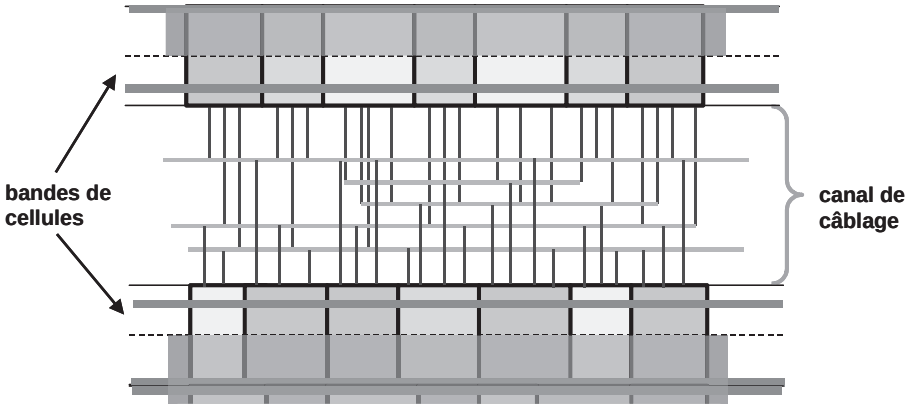


Figure 5.16 Interconnexion des bandes de cellules par des canaux de câblage

Les technologies modernes offrent de nombreux niveaux de métal. Ceux-ci permettent de supprimer les canaux de câblage. Celui-ci se fait maintenant au-dessus des cellules en utilisant les couches supérieures de métal (figure 5.17).

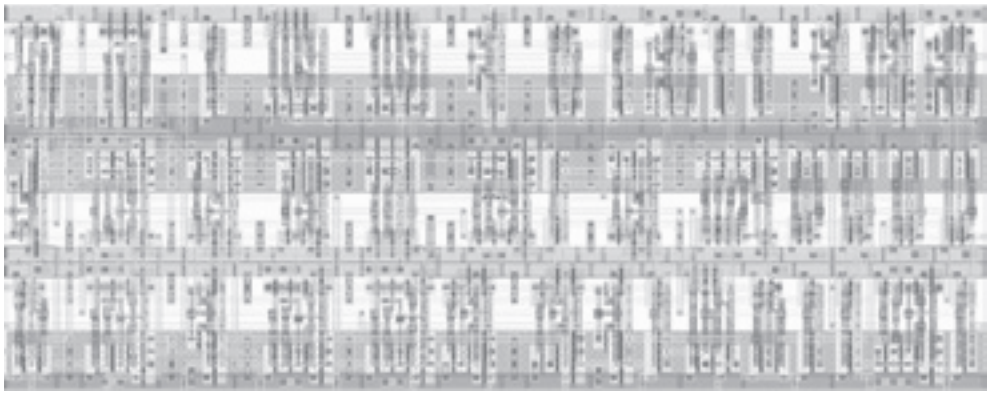


Figure 5.17 Dessin d'un séquenceur câblé (*cf.* chapitre 9) synthétisé avec le système Alliance

5.6 DESSINS SQUELETTIQUES

Cette approche, adaptée au dessin « vertical », permet de projeter le dessin d'une cellule d'une manière schématique en représentant chaque connexion par une ligne d'une couleur représentative de la couche utilisée (figure 5.18).

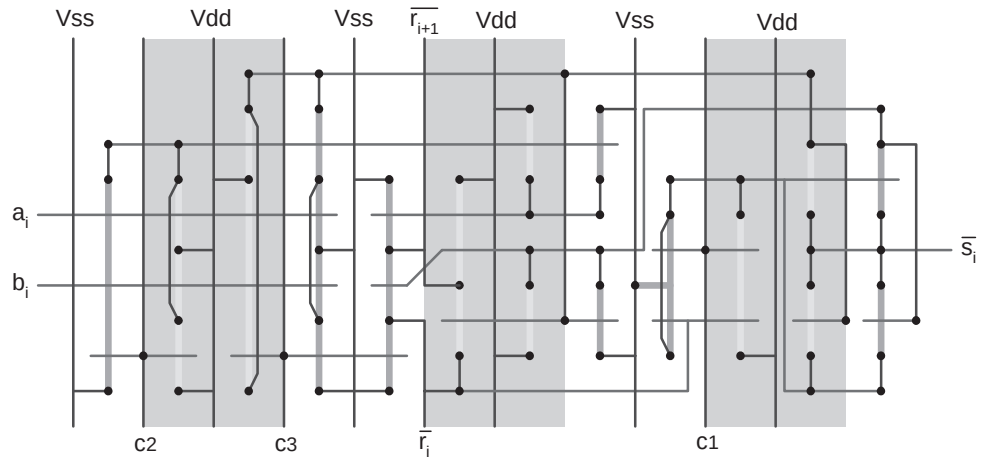


Figure 5.18 Dessin squelettique d'une cellule

Le dessin squelettique présente l'avantage d'être relativement indépendant des caractéristiques dimensionnelles des technologies. Il peut ensuite être transformé en dessin des masques pour une technologie particulière, d'une manière manuelle ou automatique, en donnant une largeur correcte aux lignes. Il est même possible de

TABLEAU 5.2

Couche	Couleur
Caisson	Gris
zones implantées N	Vert
zones implantées P	Marron
Polysilicium	Rouge
métal 1	Bleu
métal 2	bleu ciel

définir des règles génériques pour le dessin squelettique, assurant son expansion correcte pour toute une gamme de technologies (figure 5.19).

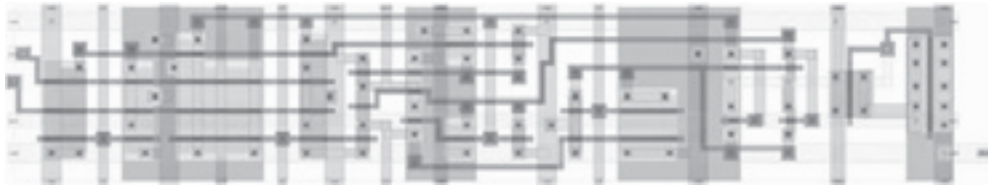


Figure 5.19 Expansion de la même cellule

5.7 DESSIN DES ROM ET DES PLA

Le dessin des matrices de ROM/PLA suit exactement les mêmes règles de croisement des couches. La technique de dessin dépend du nombre de maques à altérer pour modifier le contenu du PLA.

5.7.1 Matrices NOR

Suivant le contenu du PLA, une cellule élémentaire consiste soit en un transistor, soit en une simple connexion. Les lignes sont dessinées tête-bêche pour partager les contacts et les lignes de masse. Il faut également prévoir des lignes métalliques périodiques de rappel de masse pour éviter les effets parasites dus à la forte résistivité des lignes d'active qui véhiculent la masse (figure 5.20).

Pour permettre la modification aisée du contenu du PLA par l'altération d'un seul masque, il est possible de dessiner une matrice uniforme de transistors puis de venir ajouter un contact lorsque l'on désire connecter un transistor sur une ligne de bit.

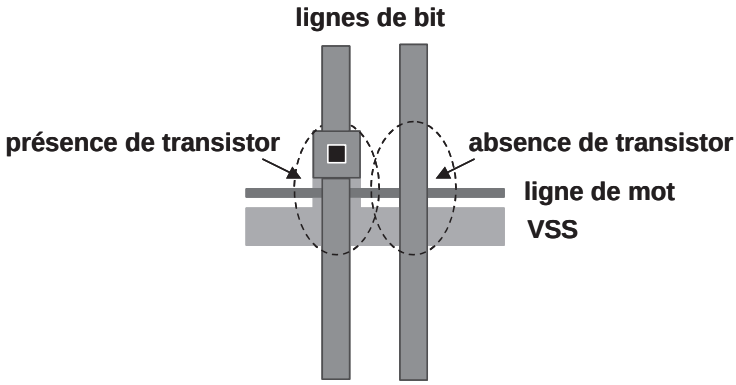


Figure 5.20 Dessin des cellules d'un PLA NOR

5.7.2 Matrices NAND

Les transistors qui agissent sur une ligne de bit sont tous en série. Il faut toutefois permettre l'absence de certains transistors en les remplaçant par des court-circuits, ce qui nécessite l'usage de deux contacts et d'un pont métallique (figure 5.21).

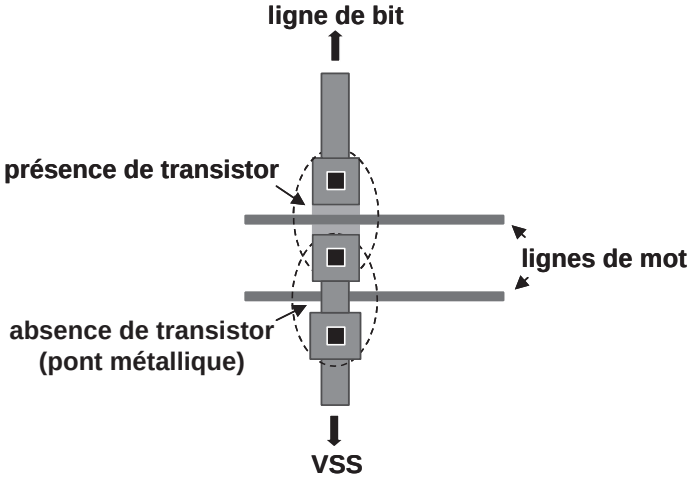


Figure 5.21 Dessin des cellules d'un PLA NAND

Pour permettre la modification aisée du contenu du PLA par l'altération d'un seul masque, il est possible de dessiner une matrice uniforme de transistors puis de venir court-circuiter ceux que l'on désire éliminer.

5.8 ASSEMBLAGE DES MACRO-BLOCS D'UN CIRCUIT

Les macro-blocs d'un circuit possèdent généralement une transparence faible ou nulle. Ils sont conçus pour s'imbriquer au mieux. Leur câblage se fait à l'aide de canaux de câblage dessinés à l'aide d'outils spécifiques au dernier niveau d'interconnexion du circuit (figure 5.22).

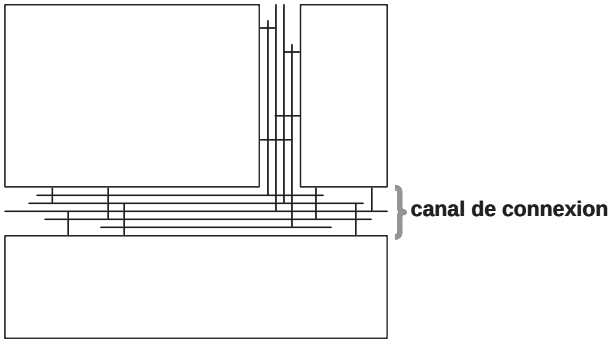


Figure 5.22 Interconnexion des macro-blocs constituant un circuit

BIBLIOGRAPHIE

[MEA80] C. Mead et L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980. Traduction : *Introduction aux systèmes VLSI*, InterEditions, 1983.

[TAK81] Takao Uehara and William M. vanCleemput, Optimal Layout of CMOS Functionnal Arrays, *IEEE Transactions Computer C30*, n° 5, May 1981, p. 305-312.

[MAZ92] R.L. Mazias and J.P. Hayes, *Layout Minimization of CMOS Cells*, Kluwer Academic Publishers, Boston, 1992.

[RIE03] M.A Riepe and K.A Sakallah, Transistors Placement for Non complementary Digital Vlsi Cell Synthesis, *ACM Transaction on Design Automation on Electronic Systems*, Vol. 8 n° 1, January 2003, pp. 81-107.

Chapitre **6**

Opérateurs arithmétiques

6.1 INTRODUCTION

Les opérateurs arithmétiques sont des composants importants des circuits intégrés logiques. Ils permettent l'exécution des opérations arithmétiques. Ils sont indispensables aux microprocesseurs et aux circuits de traitement du signal.

6.1.1 Opérations réalisées

Bien que le nombre d'opérations arithmétiques intéressantes soit assez important, les opérateurs arithmétiques câblés n'en réalisent qu'une faible partie à laquelle vient s'ajouter la réalisation des opérations logiques et des décalages.

TABLEAU 6.1 OPÉRATIONS RÉALISÉES PAR UN OPÉRATEUR ARITHMÉTIQUE

a + b	addition
a – b	soustraction
a + b + r, r ∈ {0, 1}	addition avec report entrant
a – b – r, r ∈ {0, 1}	soustraction avec report entrant
A ⊕ b	ou-exclusif
A ∨ b	ou
A ∧ b	et

En forçant le premier opérande à 0 et r à 1, on obtient les opérations suivantes :

$$b + 1$$

$$a - 1$$

De même, on remarque que $a + a = 2a$ produit le décalage gauche de a d'une position.

D'autres opérateurs, plus spécialisés, réalisent des multiplications très rapides. Toutes les autres opérations complexes (division, extraction de racine, opérations trigonométriques…) se font par des enchaînements d'opérations plus simples, de tests et de décalages.

6.1.2 Représentation des nombres

Après plusieurs essais, dont on voit les traces dans l'histoire de l'informatique, les nombres sont maintenant toujours représentés par des vecteurs de bits en numérotation binaire.

Exemple :

$$241 \Rightarrow 11110001$$

$$a = \sum_{i = 0, n-1} 2^i a_i = a_0 + 2a_1 + 4a_2 + \ldots$$

Comme ils ont toujours un nombre de bits limité, les vecteurs binaires représentent des ensembles quotients $N/2^n$, c'est-à-dire les nombres de $[0, 2^n - 1]$. Toute sortie de ces limites sera appelée un *débordement*. Les nombres sont généralement représentés par des vecteurs de 8, 16, 32 ou maintenant de 64 bits.

Une autre convention consiste à utiliser le même intervalle de nombres binaires pour représenter les nombres positifs et négatifs (*relatifs*) de $[-2^{n-1}, 2^{n-1} - 1]$.

Dans ce cas :

$$11110001 \Rightarrow -15$$

Le même profil binaire peut donc représenter soit un nombre positif, soit un nombre négatif, soit encore un code, par exemple un caractère. Cette ambiguïté n'est pas gênante dans la mesure où l'« on sait » que le vecteur binaire représente un entier « normal » ou relatif ou autre chose. Certaines machines ont utilisé des représentations *typées* dans lesquelles quelques bits additionnels (le *type*) décrivaient la nature du contenu des mots binaires.

6.2 ADDITIONNEUR

L'opérateur arithmétique de base est l'additionneur. Celui-ci peut être utilisé :

– soit pour réaliser de simples additions, par exemple pour effectuer des calculs d'adressage ;

– soit complété de la logique nécessaire pour lui faire exécuter les opérations classiques d'un opérateur arithmétique (appelé UAL pour Unité Arithmétique et Logique) ;
– soit dupliqué pour réaliser des multiplications.

Les autres opérations complexes (division, extraction de racine, logarithmes, opérations trigonométriques…) se font par des enchaînements d'opérations plus simples, de tests et de décalages. Le calcul des élévations à des puissances élevées ou fractionnaires, des logarithmes, des fonctions trigonométriques… se fait par le calcul optimisé de leur développement en série.

6.2.1 Réutilisation de l'addition

Plusieurs opérations élémentaires se font par la réutilisation de l'addition. Par exemple, la soustraction nécessite l'utilisation d'une circuiterie amont, sur l'entrée « b » de l'opérateur, qui réalise l'inversion logique du second opérande de la soustraction. En effet :

$$\bar{b} = -b - 1$$

d'où :

$$a - b = a + \bar{b} + 1$$

et :

$$a - b - 1 = a + \bar{b} \quad \text{(soustraction avec retenue)}$$

Le troisième opérande (+1) représente la retenue entrante (à droite) de l'additionneur qui est normalement à 0, mais qui peut être portée à 1.

6.2.2 Addition binaire

L'addition binaire s'effectue de la même manière qu'une addition décimale, mais en se situant dans l'arithmétique des nombres binaires.

Exemple :

```
    1111   retenues
 1001101   a
+0101011   b
 1111000   s résultat
```

Si nous isolons le processus d'addition de deux bits a_i et b_i, nous voyons que le calcul du bit de résultat s_i et de la retenue suivante r_{i+1} correspond aux fonctions booléennes suivantes :

$$\boxed{s_i = a_i \oplus b_i \oplus r_i}$$

et :

$$\boxed{r_{i+1} = \text{maj}(a_i, b_i, r_i)}$$

Nous voyons que les trois opérandes a_i, b_i, r_i jouent des rôles strictement identiques.

Le calcul de la retenue peut s'exprimer par :

$$r_{i+1} = (a_i \wedge b_i) \vee (a_i \oplus b_i \wedge r_i)$$

Posons :

$g_i = a_i \wedge b_i$ qui correspond à la *génération* d'une retenue ;

$p_i = a_i \oplus b_i$ qui correspond à la *transmission* (propagation) de la retenue entrante :

$$\boxed{r_{i+1} = g_i \vee (p_i \wedge r_i)}$$

qui signifie que la retenue sortante est soit générée, soit transmise à partir de celle des poids plus faibles.

De même :

$$\boxed{s_i = p_i \oplus r_i}$$

6.2.3 Synthèse d'une cellule d'additionneur

Nous allons nous intéresser à la conception d'une cellule d'additionneur capable de générer s_i et r_{i+1} à partir de a_i, b_i et r_i. La réalisation d'un additionneur capable d'additionner deux vecteurs binaires a et b consistera à assembler autant de cellules que de bits dans les vecteurs.

a) Cellule d'additionneur réalisée avec des portes de passage

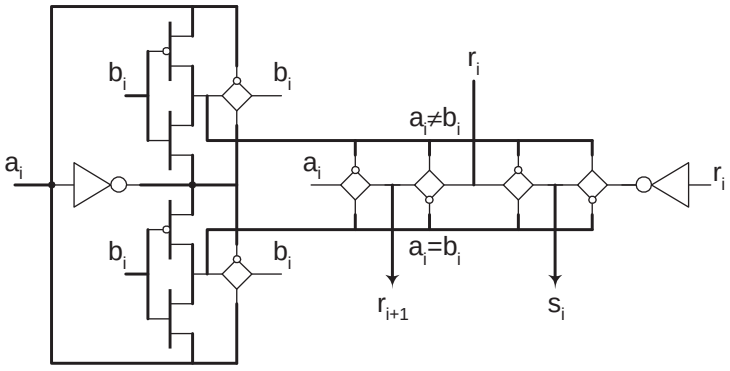


Figure 6.1 Cellule d'additionneur à 20 transistors

si $a_i \neq b_i$ alors $r_{i+1} \Leftarrow r_i$, $s_i \Leftarrow \overline{r_i}$

si $a_i = b_i$ alors $r_{i+1} \Leftarrow a_i$, $s_i \Leftarrow r_i$

Cette cellule est réalisée à l'aide de 20 transistors.

b) Cellule d'additionneur symétrique

La symétrie des expressions booléennes qui définissent la cellule se traduit par le fait que les trois entrées sont permutables (figure 6.2).

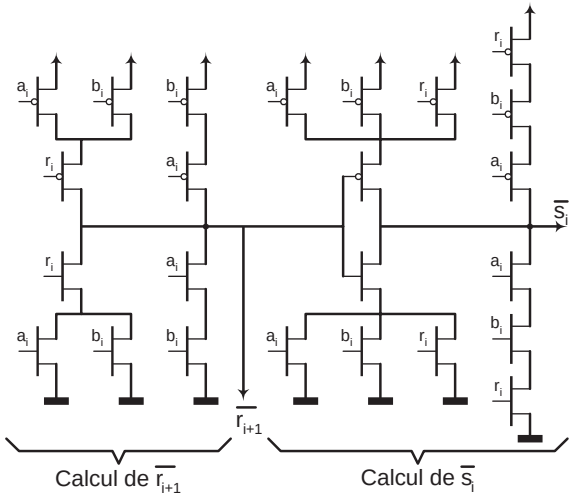


Figure 6.2 Cellule d'additionneur symétrique

Cette cellule est réalisée à l'aide de 24 transistors.

c) Cellule d'additionneur avec génération de p_i *et* g_i

La synthèse de ce type de cellule d'additionneur correspond à la synthèse simultanée de g_i , p_i , r_{i+1} et s_i .

$p_i = a_i \oplus b_i$ peut se réécrire $p_i = \overline{(a_i \wedge b_i)} \wedge (a_i \vee b_i)$, d'où une synthèse en portes :

$$\overline{p_i} = \overline{\overline{(a_i \wedge b_i)} \wedge (a_i \vee b_i)}$$

Nous remarquons que le terme $\overline{(a_i \wedge b_i)}$ correspond à $\overline{g_i}$.

La synthèse simultanée de $\overline{g_i}$ et $\overline{p_i}$ est représentée figure 6.3.

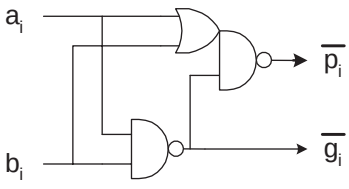


Figure 6.3 Circuit de calcul de $\overline{p_i}$ $\overline{g_i}$

La synthèse d'une cellule d'additionneur complète nécessite de choisir le mode de propagation de la retenue. En effet, la propagation de la retenue de cellule en cellule s'avère être le chemin le plus long d'un additionneur (chaîne longue).

Nous changerons aussi la polarité de la sortie s_i en $\overline{s_i}$ pour permettre l'utilisation d'un inverseur de puissance pour amplifier et distribuer ce signal.

➤ Propagation « statique » de la retenue

de :

$$r_{i+1} = g_i \vee (p_i \wedge r_i)$$

$$\overline{r_{i+1}} = \overline{(g_i \vee (p_i \wedge r_i))} = \overline{(g_i \vee (\overline{p_i} \wedge \overline{r_i}))}$$

Ce qui correspond au schéma de la figure 6.4.

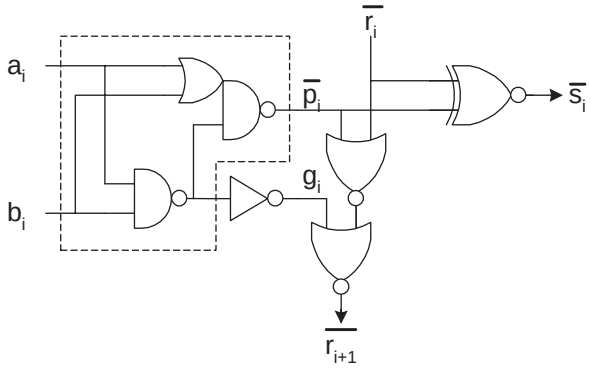


Figure 6.4 Cellule d'additionneur à propagation statique

Le changement de polarité des signaux de retenue permet de simplifier la réalisation du circuit de retenue. Cette cellule est réalisée à l'aide d'au moins 24 transistors, suivant la technique utilisée pour réaliser le non-ouex de sortie.

➤ Propagation « dynamique » de la retenue

Avec précharge (figure 6.5)

La retenue est transmise par un simple interrupteur CMOS, tandis que la génération d'une retenue sortante est réalisée par un simple transistor N, après une précharge. Le fait de disposer de $\overline{p_i}$ et de p_i permet de simplifier la réalisation du non-ouex de sortie.

Cette technique de réalisation du circuit de retenue nécessite peu de transistors, mais est moins rapide. Cette cellule est réalisée à l'aide de 22 transistors.

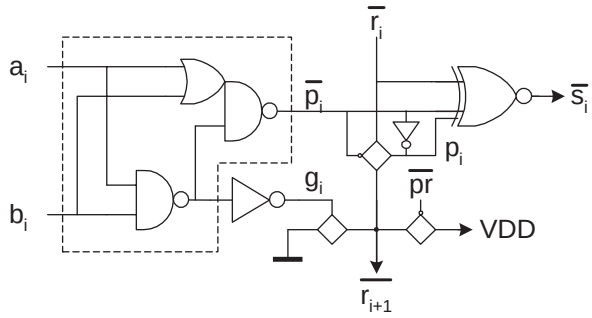


Figure 6.5 Cellule d'additionneur à propagation dynamique

Précharge « automatique » (figure 6.6)

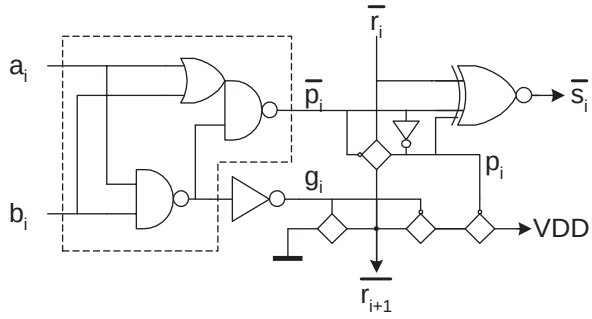


Figure 6.6 Cellule d'additionneur à recharge automatique

La précharge de la sortie de retenue est déclenchée lorsqu'il n'y a ni génération, ni propagation d'une retenue. Cette cellule est réalisée à l'aide de 23 transistors.

➤ Non-ouex de sortie

Nous verrons que ce circuit (figure 6.7) présente les plus faibles contraintes de vitesse de toute la cellule d'additionneur.

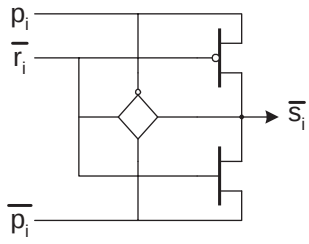


Figure 6.7 Ouex de sortie

Pour les cellules qui utilisent la propagation de la retenue à l'aide de transistors de passage, ce non-ouex peut être (économiquement) réalisé avec une cellule non standard en profitant de la disponibilité de $\overline{p_i}$ et de p_i.

La simple permutation des entrées $\overline{p_i}$ et p_i permet de transformer ce non-ouex en ouex.

➤ Réalisation d'un additionneur complet

La cellule d'additionneur que l'on vient d'étudier ne nécessite que 23 transistors. Elle est donc beaucoup plus optimisée que si elle était réalisée avec des portes classiques. Il existe plusieurs façons de réaliser un additionneur capable d'additionner deux vecteurs de bits à partir d'une telle cellule. Soit les différents bits du vecteur sont traités séquentiellement, soit ils sont traités en parallèle (addition parallèle).

À part quelques applications particulières, la quasi-totalité des additionneurs modernes travaillent en parallèle.

6.2.4 Additionneur parallèle

Un tel additionneur est constitué d'une suite de cellules d'additionneur qui traitent les différents bits (figure 6.8). Ces cellules sont interconnectées par les retenues.

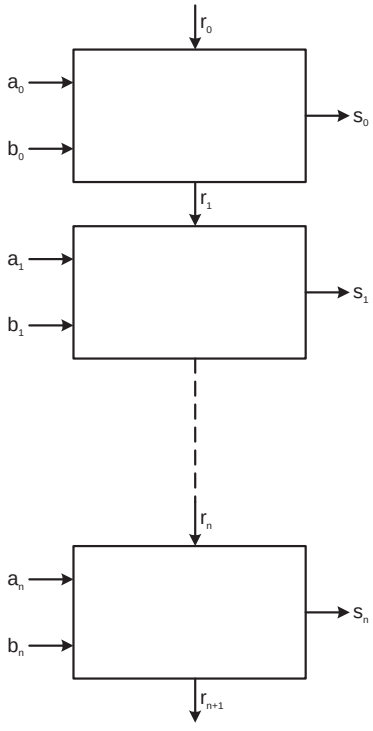


Figure 6.8 Addition parallèle de n bits

La retenue initiale r_0 est normalement mise à 0. Elle peut toutefois être mise à 1 si l'on souhaite effectuer une addition avec retenue $a + b + 1$.

La retenue sortante est analysée pour détecter les débordements.

Un tel additionneur peut être utilisé pour additionner des entiers naturels ou relatifs. Dans ce cas, la valeur de s_n indique le signe de la somme.

Le chemin critique de ce circuit combinatoire va de (a_0, b_0) à s_n *via* la chaîne de retenue. Il intervient dans une opération telle que :

$$\begin{array}{r} 111111 \ldots 111 \\ +\ 000000 \ldots 001 \end{array}$$

dans laquelle la retenue, générée par les poids faibles, se propage jusqu'au dernier étage. Ce trajet devient prohibitif, même au prix d'une circuiterie performante pour le calcul du report.

Propagation anticipée de la retenue

La disponibilité des signaux p_i et g_i au niveau de chaque cellule d'additionneur permet d'accélérer la propagation de la retenue (figure 6.9). Pour cela les cellules d'additionneur seront groupées par paquets de 4 à 8 bits. Lorsque toutes les cellules d'un paquet propageront la retenue, la retenue entrante dans le paquet sera directement transmise en sortie, évitant sa propagation au travers des cellules. Dans les autres cas, une retenue générée dans un étage sera propagée normalement jusqu'à la sortie du paquet. Avec

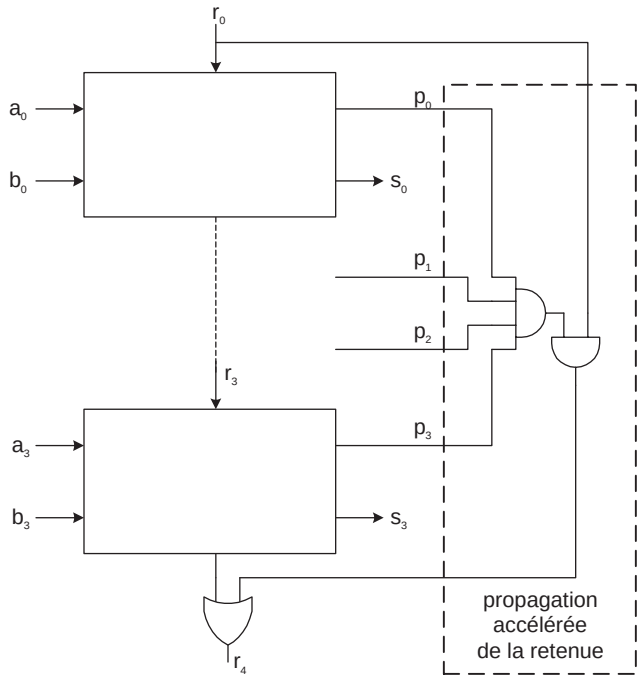


Figure 6.9 Accélération de la retenue pour un groupe de 4 bits

un tel dispositif, le temps maximal de propagation de la retenue sera le temps de propagation d'un paquet plus le temps de traversée des mécanismes d'accélération des autres paquets.

D'autres organisations plus complexes peuvent être imaginées comme :

– la génération directe des valeurs générées en sortie d'un paquet ;
– un système de propagation hiérarchique portant sur plusieurs niveaux de regroupement des additionneurs élémentaires.

6.3 UNITÉ ARITHMÉTIQUE ET LOGIQUE (UAL)

Un additionneur peut être modifié pour lui permettre d'exécuter les principales opérations logiques : $\oplus$, $\vee$, $\wedge$ en plus de l'addition.

6.3.1 Calcul du OU-exclusif

La cellule d'additionneur, présentée en § 6.2.3c peut être facilement modifiée pour que $\overline{S_i}$ devienne $\overline{a_i \oplus b_i}$.

De :

$$\overline{p_i} = \overline{a_i \oplus b_i}$$

et :

$$\overline{s_i} = \overline{p_i} \oplus r_i = \overline{p_i \oplus r_i}$$

Le forçage de r_i à 0, produit le résultat escompté (figure 6.10).

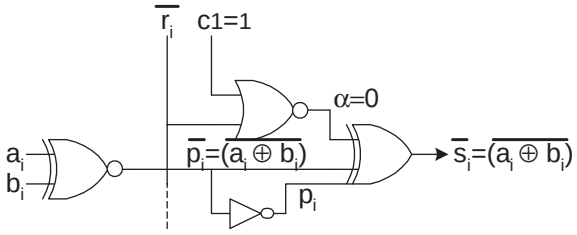


Figure 6.10

Un signal de commande $c1$ est utilisé pour forcer à 0 la retenue entrant dans le ouex de sortie. Celui-ci transmet alors en sortie ce qu'il reçoit sur son autre entrée.

$$\text{si } c1 = 1 \Rightarrow \alpha = 0 \Rightarrow \overline{s_i} = \overline{a_i \oplus b_i}$$

$$\text{si } c1 = 0 \Rightarrow \alpha = r_i \Rightarrow \overline{s_i} = \overline{p_i} \oplus r_i = \overline{p_i \oplus r_i}$$

6.3.2 Calcul du OU

Comme précédemment, on utilise la commande $c1$ pour rendre transparent le ouex de sortie.

$$\text{si } c1 = 1 \Rightarrow \overline{s_i} = \overline{p_i}$$

Le circuit de génération de $\overline{g_i}$ et $\overline{p_i}$ est modifié pour forcer sa sortie à 1 par une commande $c2$. $\overline{p_i}$ devient alors (figure 6.11) :

$$\overline{p_i} = \overline{(\overline{g_i} \wedge (a_i \vee b_i))} = \overline{(a_i \vee b_i)}$$

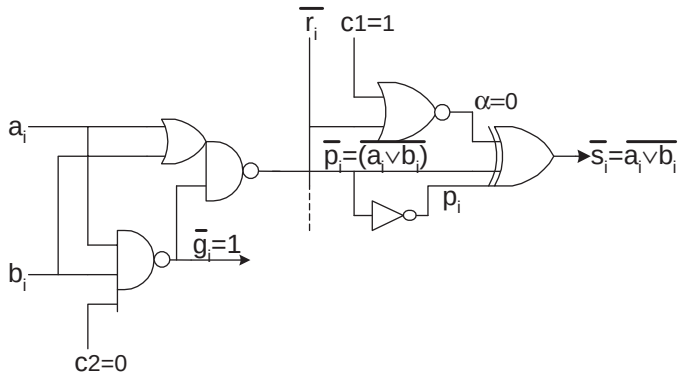


Figure 6.11 Calcul du OU

6.3.3 Calcul du ET

Cette fois-ci, on va chercher à transmettre en sortie la valeur de $\overline{g_i} = \overline{a_i \wedge b_i}$. Toutefois, dans ce cas, le ouex de sortie doit transmettre la valeur de p_i et non celle de $\overline{p_i}$. Grâce aux propriétés du ouex, ceci pourra être obtenu en forçant la valeur de α à 1 par le forçage à 0 de toute la chaîne de report, par une commande $c3$. Cette même commande est utilisée pour forcer à 1 la valeur de $a_i \vee b_i$ entrant dans la composition de $\overline{p_i}$ (figure 6.12).

6.3.4 Schéma et dessin de la cellule d'UAL complète

La réalisation de cette UAL (figures 6.13 à 6.15) ne nécessite que 32 transistors, soit beaucoup moins que sa réalisation en portes classiques. Compte tenu de la faible sortance du ouex utilisé en sortie (qui génère $\overline{s_i}$), il faut le faire suivre par un inverseur de puissance, capable de piloter un bus.

Le dessin de la cellule est conçu pour être assemblé par simple juxtaposition. Cette cellule d'UAL comporte un inverseur de puissance en sortie destiné à attaquer un bus (figures 6.14 et 6.15).

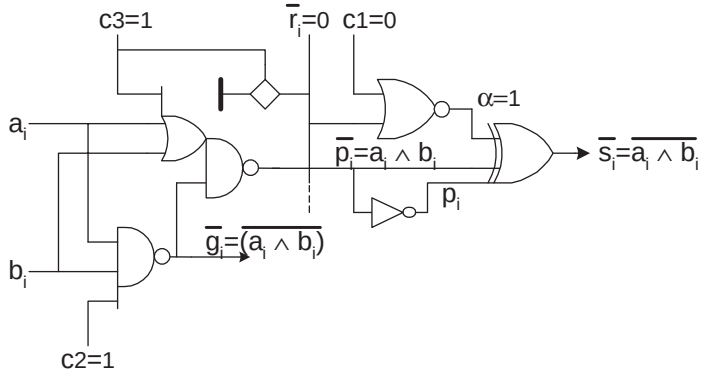


Figure 6.12 Calcul du ET

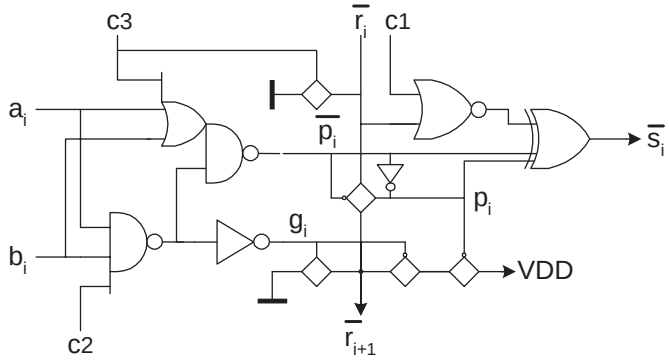


Figure 6.13 Cellule d'UAL

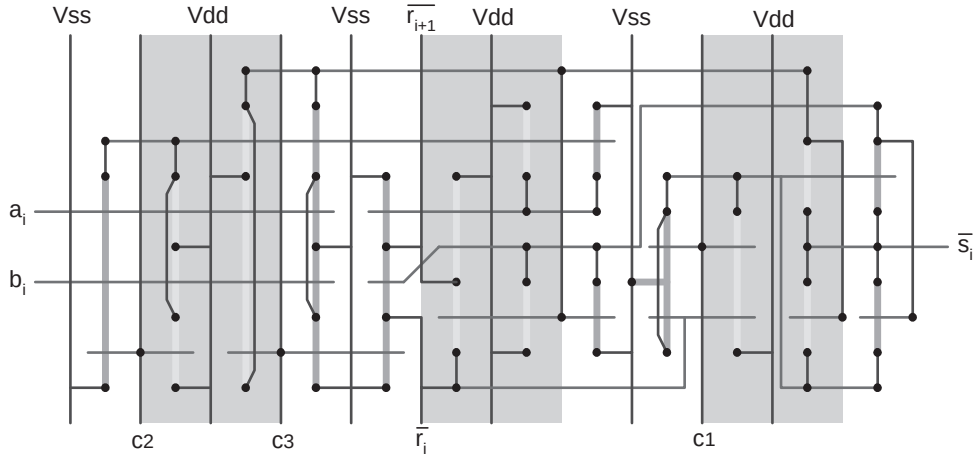


Figure 6.14 Schéma squelettique de l'UAL

TABLEAU 6.2 CODES DE FONCTION DE L'UAL

	c1	c2	c3
addition	0	1	0
ouex	1	1	0
ou	1	0	0
et	0	1	1

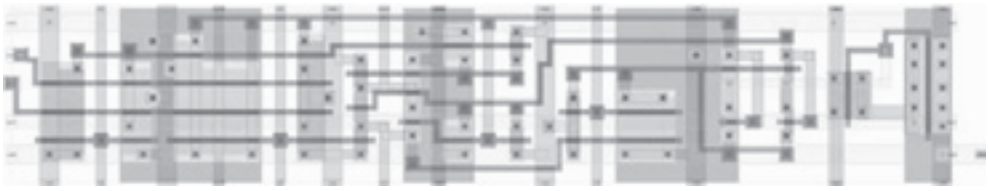


Figure 6.15 Dessin des masques de la cellule d'UAL (technologie CMOS standardisée (Alliance))

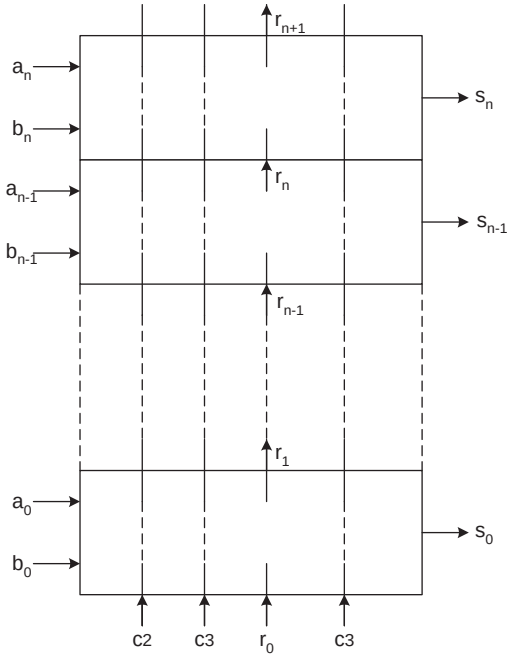


Figure 6.16 Assemblage des cellules par juxtaposition pour réaliser un additionneur de 4 bits

La valeur de r_0 est fixée à 0 pour une addition « normale » et pour une soustraction avec retenue. Elle est fixée à 1 pour une soustraction « normale » et pour une addition avec retenue.

6.4 MULTIPLIEUR CÂBLÉ

Le multiplieur est l'un des rares opérateurs complexes à être conçu de manière combinatoire. Comme une multiplication peut être décomposée en une suite d'additions et de décalages, un multiplieur est donc réalisé par l'assemblage d'une batterie de cellules d'additionneurs.

6.4.1 Multiplieur simple

Soient deux nombres binaires *a* et *b* de *n* bits :

$$a = \sum_{i=0,\,n-1} a_i 2^i \quad a_i = 0 \text{ ou } 1$$

$$b = \sum_{i=0,\,n-1} b_i 2^i \quad b_i = 0 \text{ ou } 1$$

Leur produit P s'écrit :

$$P = a \times b = \sum_{i=0,\,n-1} a_i 2^i \sum_{j=0,\,n-1} b_j 2^j$$

qui peut aussi s'écrire :

$$P = a \times b = \sum_{i=0,\,n-1} a_i \sum_{j=0,\,n-1} b_j 2^{i+j}$$

Ce produit peut s'écrire sur 2n bits en regroupant les termes affectés de la même puissance de 2.

$$P = a \times b = \sum_{i=0,\,n-1} \left(\sum_{j+k=i} a_j b_k \right) 2^i$$

Chacun de ses termes $p_i = \sum_{j+k=i} a_j b_k$ est la somme des produits partiels affectés de la même puissance i de 2.

On remarque que le produit arithmétique de deux bits est équivalent à leur ET booléen.

a_i	b_j	$a_i \times b_j$	$a_i \wedge b_j$
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Un multiplieur trivial consiste à utiliser des cellules d'additionneur pour additionner les produits partiels (pré-calculés par des ET) ligne à ligne puis colonne par colonne (figure 6.17).

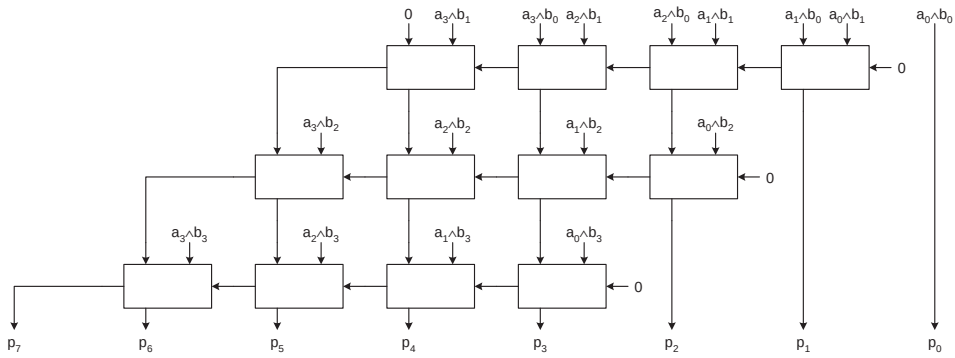


Figure 6.17 Exemple de multiplieur trivial pour des nombres de 4 bits

Le chemin critique de ce multiplieur est de 8 cellules. Il peut être ramené à 6 en remarquant que dans une colonne les produits partiels peuvent être additionnés dans n'importe quel ordre (figure 6.18).

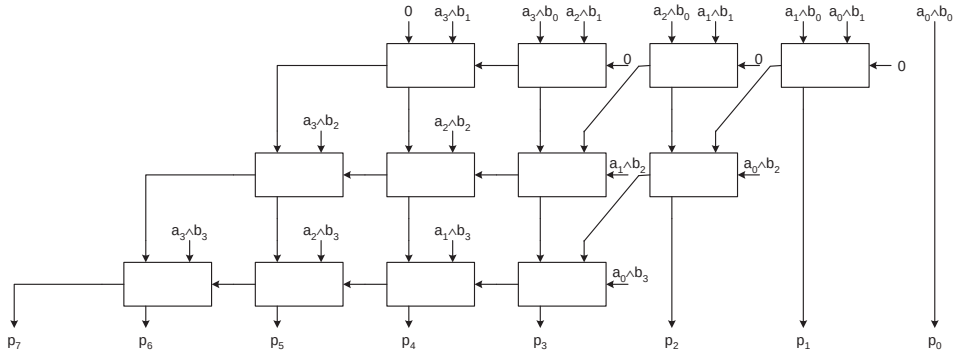


Figure 6.18 Multiplieur 4 bits optimisé en vitesse

Des techniques plus complexes permettent de réduire encore la longueur du chemin critique, donc d'accroître la vitesse du multiplieur au prix d'une augmentation de sa complexité.

BIBLIOGRAPHIE

[1] A. Guyot, notes informatisées du cours : *Opérateurs arithmétiques*, INP Grenoble, département Télécom, 2003.

Chapitre **7**

Systèmes séquentiels

7.1 DÉFINITIONS

Les *systèmes séquentiels*, appelés aussi *machines d'états finis* ou *automates*, suivant le domaine d'application, représentent une étape supplémentaire de complexification dans la conception des systèmes logiques. Par rapport aux circuits combinatoires qui simplement associent leurs sorties aux différents profils d'entrée qu'ils reçoivent, les systèmes séquentiels possèdent un état interne qui influence leur comportement.

Celui-ci peut être représenté par les deux formules suivantes :

$$S_t = f(E_t, Q_t)$$

$$Q_{t+1} \Leftarrow g(E_t, Q_t)$$

dans lesquelles :

S_t représente les sorties du système à l'instant t ;

Q_t représente l'état interne du système à l'instant t ;

Q_{t+1} représente l'état interne du système à l'instant suivant ;

$E_t \in \mathbf{E}$, ensemble fini des entrées possibles ;

$S_t \in \mathbf{S}$, ensemble fini de ses sorties possibles ;

$Q_t, Q_{t+1} \in \mathbf{Q}$, ensemble fini de ses états possibles.

Nous parlerons de *systèmes de Moore* dans le cas où la fonction *f* ne dépend que de l'état courant. Dans le cas général, nous parlerons de *systèmes de Mealey.* On montre que ces deux types de systèmes sont fonctionnellement équivalents : à tout système de Mealey correspond un système de Moore et réciproquement. On remarque toutefois

qu'à fonctionnalité équivalente un système de Mealey nécessite moins d'états internes qu'un système de Moore.

D'un point de vue électronique, la structure générale des systèmes séquentiels (figure 7.1) peut être représentée comme l'assemblage d'un réseau combinatoire calculant les deux fonctions *f* et *g* et d'un organe de mémorisation contenant l'état courant Q_t.

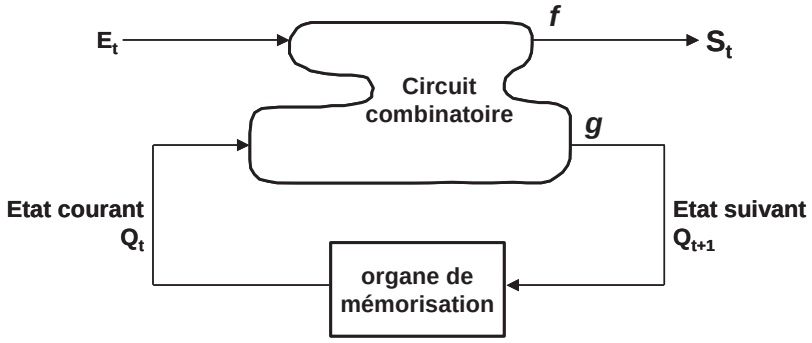


Figure 7.1 Système séquentiel

Comme l'ensemble **Q** peut comporter plus de deux états, ceux-ci doivent être *codés* par des vecteurs de plusieurs bits. Plusieurs codages sont possibles et le choix d'un bon codage permet de simplifier le réseau combinatoire.

Les systèmes séquentiels possèdent des propriétés supplémentaires (émergentes) par rapport aux portes logiques qui les constituent.

La notion de systèmes séquentiels est utilisée soit pour représenter des dispositifs qui doivent fournir une suite de sorties dépendant de l'évolution de leur état interne et de leurs entrées (par exemple des séquenceurs), soit des systèmes devant réaliser des opérations sur des données dont le format est prédéfini (par exemple des unités arithmétiques). Évidement, le codage des états ne concerne que les premiers.

La notion de systèmes séquentiels est une abstraction mathématique qui peut être approchée physiquement de différentes manières. Les différentes familles de systèmes séquentiels correspondent à différentes réalisations des organes qui mémorisent leur état interne.

Nous distinguerons, dans un premier niveau de classification :

– Les systèmes dit *asynchrones* dans lesquels l'organe de mémorisation n'est qu'un (ou plusieurs) simples retards (de durée plus ou moins précise). Dans ces systèmes, l'état suivant est automatiquement pris en compte « un certain temps » après sa génération par le réseau combinatoire.
– Les systèmes dites *synchrones* dans lesquels l'organe de mémorisation ne prend en compte l'état suivant qu'à des instants biens précis indiqués par le milieu extérieur à l'aide de signaux événementiels particuliers appelés *horloges*. Nous pouvons

remarquer que dans le cas des systèmes synchrones, les systèmes de Mealey ne sont pas strictement équivalents aux systèmes de Moore. En effet, dans un système de Mealey, la variation d'une entrée, entre les instants de variation de l'état interne, peut se reporter en sortie, alors que les sorties d'un système de Moore ne peuvent varier qu'avec son état.

Les systèmes séquentiels, tant asynchrones que synchrones, ont fait l'objet de nombreuses études ayant pour but l'analyse de leur comportement et leur synthèse.

7.1.1 Représentation du comportement des systèmes séquentiels

La description du comportement des systèmes séquentiels se ramène à celle des deux fonctions *f* et *g*. Celles-ci peuvent être représentées par des fonctions booléennes, toutefois les représentations qui permettent de visualiser l'enchaînement des états successifs sont préférées. On utilise traditionnellement :

– La représentation de la fonction *g* sous la forme d'un graphe d'enchaînement des états. La fonction *f* est alors représentée comme un étiquetage supplémentaire des arcs de ce graphe.
– La représentation des fonctions *f* et *g* sous la forme d'un tableau à double entrées (état courant, entrées).

La conception d'un système séquentiel consiste à :

– déterminer un codage des états de **Q** qui optimise la réalisation des fonctions *f* et *g* (le cas des systèmes asynchrones ajoute des contraintes spécifiques sur ce codage) ;
– déterminer les projections bit à bit des fonctions *f* et *g* ;
– réaliser leur synthèse dans la technologie choisie.

7.2 SYSTÈMES SÉQUENTIELS ASYNCHRONES

Une grande partie des systèmes séquentiels asynchrones correspond à de simples circuits combinatoires rebouclés (figure 7.2). Dans ce cas, les retards qui mémorisent l'état interne sont ceux des portes logiques qui apparaissent dans cette boucle. Comme il s'agit de retards « technologiques », ceux-ci ne sont pas définis avec une grande précision. Ils peuvent varier avec la technologie, la température, la tension d'alimentation,

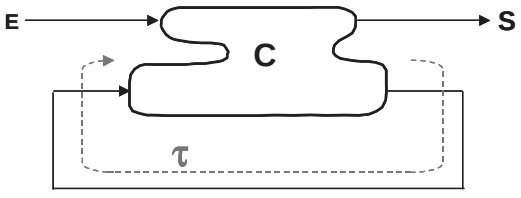


Figure 7.2 Système séquentiel asynchrone

le vieillissement du circuit, etc. Il convient donc que le comportement de ce type de systèmes soit, dans une large mesure, indépendant de la valeur de ces retards.

Une transition d'état est déclenchée par la variation d'une entrée (figure 7.3). Toutefois, le nouvel état atteint peut être :

– *transitoire* si la même entrée (qui n'a pas eu le temps de varier) enclenche une nouvelle transition vers un autre état ;
– *stable* si cette entrée provoque la regénération du même état (état rebouclé sur lui-même).

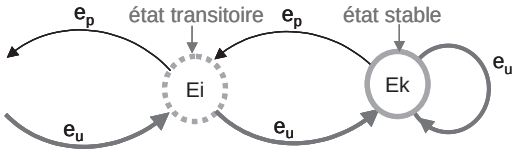


Figure 7.3 États transitoires et stables dans un système séquentiel asynchrone

On appelle *course* le passage par une succession d'états transitoires. Une telle course peut être soit :

– *finie*, c'est-à-dire qu'elle se termine sur un état stable ;
– *infinie*, c'est-à-dire qu'elle forme une boucle contenant plusieurs états, qui sera parcourue tant que l'entrée ne sera pas modifiée.

On supposera que les entrées n'ont pas le temps de varier pendant le parcours des courses finies.

Les courses infinies correspondent à une oscillation entre ces deux ou plusieurs, états. La période de cette oscillation correspond à $n\tau$, dans laquelle τ est le temps de traversée des portes dans la boucle et n le nombre d'états rebouclés. La fréquence de cette oscillation peut être très élevée et laisser le circuit dans un état indéterminé. Les courses infinies correspondent à un état pathologique du circuit et doivent être soigneusement évitées. Lorsque τ est très précis, l'oscillation se produit à une fréquence bien déterminée et le montage peut alors être utilisé comme un *oscillateur.*

L'état d'un système asynchrone est codé de telle manière que seul un bit varie à chaque transition (code de Gray). Si plusieurs bits variaient, on ne pourrait assurer la simultanéité rigoureuse de leurs transitions et le système pourrait se retrouver dans des états imprévus. Une contrainte similaire est utilisée pour le codage des entrées (et éventuellement aussi pour les sorties).

7.3 SYSTÈMES SÉQUENTIELS SYNCHRONES

Comme nous l'avons vu, l'état de ces systèmes ne peut varier qu'à des instants bien précis fournis par le milieu extérieur sous la forme d'un ou de plusieurs, signaux par-

ticuliers appelés *horloges*. On distinguera deux classes très différentes de systèmes synchrones :

– les systèmes *monophasés* dans lesquels une simple horloge rythme l'évolution de l'état interne ;
– les systèmes *polyphasés* dans lesquels plusieurs horloges rythment cette évolution.

D'une manière générale, tous les systèmes synchrones doivent satisfaire la relation suivante appelée *condition de synchronisme* :

$$\Delta T > \tau$$

dans laquelle ΔT correspond à la période (minimale) du ou des, signaux d'horloge et τ le temps (maximal) d'établissement du réseau combinatoire. Cette relation signifie simplement que la période de l'horloge doit être suffisamment longue pour permettre la génération des sorties du système, et de son état suivant, avant l'occurrence du prochain événement d'horloge.

Les domaines d'application des systèmes monophasés et polyphasés sont différents :

– Les systèmes polyphasés sont uniquement utilisés dans la réalisation des circuits intégrés complexes tels que les microprocesseurs. Cette technique permet la réalisation de circuits très optimisés. La technique polyphasée de conception des circuits séquentiels résulte de l'exploitation de propriétés particulières liées à la circuiterie VLSI (portes de transfert et circuiterie dynamique). Cette technique est née avec les microprocesseurs dans les années 1970. Elle présente l'avantage de bien se marier avec la logique dynamique et donc de nécessiter moins de transistors pour réaliser une fonction donnée. Toutefois, la difficulté de tester les structures dynamiques et de distribuer les différentes phases en respectant leurs relations temporelles amène les concepteurs à lui préférer une approche monophasée statique à base de registres maîtres-esclaves.
– Les systèmes monophasés sont beaucoup plus généralement utilisés. Ils correspondent à l'approche utilisée pour la réalisation des cartes électroniques utilisant des circuits de moyenne intégration (par exemple, la série 74xx). Leur conception correspond à la culture « classique » des électroniciens. Pour cette raison, elle est aussi utilisée dans les approches intégrées qui correspondent à la miniaturisation des cartes (par exemple les circuits prédiffusés, les FPGA, etc.). Historiquement, cette approche est apparue dans les années 1960, en même temps que les circuits de moyenne intégration. Son utilisation « classique » nécessite plus de transistors et une meilleure maîtrise technologique que l'approche polyphasée. L'utilisation de bascules maîtres-exclaves, qui revient à faire du polyphasé local, permet d'atteindre la même optimisation que dans le cas du polyphasé statique. L'avènement des technologies sub-microniques et le besoin de concevoir rapidement des circuits très complexes ont provoqué une généralisation des techniques de conception automatique dans les années 1990, ainsi que l'abandon de la circuiterie dynamique jugée difficilement testable.

La notion de système synchrone est aussi une abstraction mathématique. En toute rigueur, tous les systèmes sont asynchrones si l'on banalise leurs entrées d'horloge.

De manière à simplifier l'étude de ces systèmes, on isole les problèmes d'asynchronisme dans les éléments qui mémorisent leurs états. Ceux-ci sont appelés des *bascules*. Ce sont généralement de petits systèmes asynchrones qui « simulent » un certain degré de synchronisme. Les bascules utilisées par les systèmes polyphasés sont très simples et appelées des *latches*, tandis que celles utilisées dans les systèmes monophasés sont plus complexes.

Les enchaînements d'état d'un système synchrone peuvent être complexes. Ils résultent directement de l'application à réaliser. Évidemment, ils n'ont pas à satisfaire les contraintes propres aux systèmes asynchrones. Cela fait que le rebouclage accidentel d'un système synchrone (par la mise en mode transparent des bascules qui mémorisent son état) peut avoir des conséquences catastrophiques sur son comportement (oscillations, transitions aléatoires, états non prévus, etc.).

Nous allons commencer l'étude des systèmes synchrones par celle des systèmes polyphasés dont le fonctionnement est plus proche de la circuiterie.

7.3.1 Réalisation des systèmes synchrones

Il existe plusieurs techniques pour réaliser les systèmes synchrones polyphasés et monophasés. Nous distinguerons :

– les réalisation matérielles, dans lesquelles le réseau combinatoire est réalisé :
  • soit par un réseau de portes logiques,
  • soit par un PLA ;
– les réalisations logicielles, dans lesquelles le système séquentiel peut être :
  • soit microprogrammé sur une structure matérielle *ad hoc*,
  • soit programmé, par exemple sur un microcontrôleur.

7.4 SYSTÈMES POLYPHASÉS

Les systèmes polyphasés sont des systèmes synchrones, qui manipulent généralement des données et dont l'état est mémorisé dans des latches dont on exploite la possibilité de transparence.

7.4.1 Notion de latches

Un *latch* (portillon, loquet) est un organe de mémorisation minimal. Il possède deux états :

– un état *transparent* dans lequel le latch recopie en sortie ce qui lui est présenté en entrée ;
– un état *mémorisant* dans lequel la sortie du latch fournit la dernière information qui l'a traversé lors de son précédent état transparent.

Un latch est souvent symbolisé par un interrupteur suivi d'une capacité (figure 7.4). En effet, ce montage possède les propriétés d'un latch (si l'on ne maintient pas l'état de mémorisation plus longtemps que les fuites de la capacité ne le permettent !).

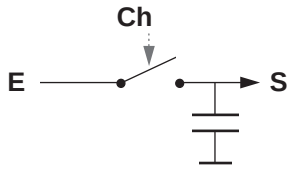


Figure 7.4 Schéma conceptuel d'un latch

Les latches peuvent être réalisés de différentes manières :

– *latches statiques*. Ceux-ci peuvent retenir l'information aussi longtemps qu'on le souhaite. Plusieurs montages sont possibles. Leur complexité se situe autour d'une dizaine de transistors. Les latches statiques sont constitués par deux inverseurs qui sont rebouclés :
  • soit par un interrupteur fermé pendant la période de mémorisation.

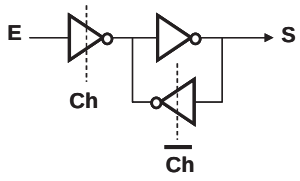


Figure 7.5 Schéma logique d'un latch à portes 3 états

Ce qui donne le schéma suivant :

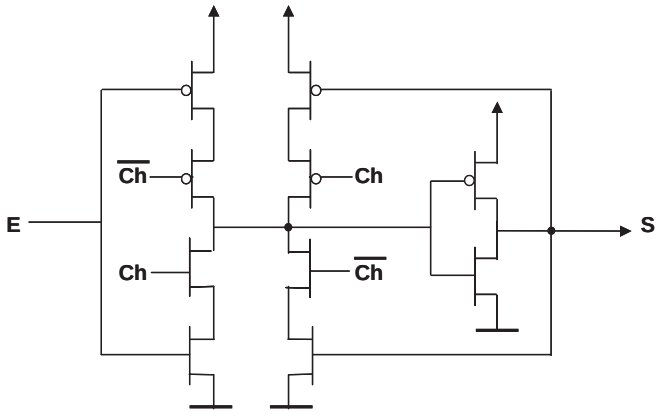


Figure 7.6 Schéma électrique d'un latch à portes 3 états

  • soit par une résistance qui affaiblit le signal de retour, ce qui permet au signal d'entrer de forcer le latch dans une nouvelle position.

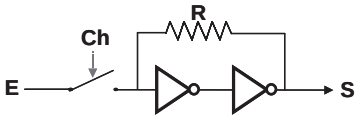


Figure 7.7 Schéma de principe d'un latch à résistance

Ce qui donne le schéma suivant :

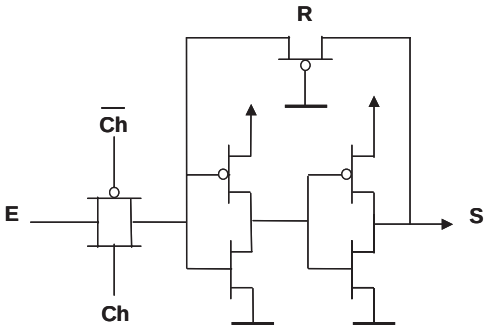


Figure 7.8 Schéma électrique d'un latch à résistance

– *latches dynamiques*. Ceux-ci correspondent au modèle symbolique du latch. La capacité est souvent celle (parasite) d'un élément (bus ou porte). Leur réalisation se résume à celle d'une simple porte de transfert, c'est-à-dire à deux transistors.

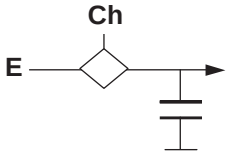


Figure 7.9 Schéma de principe d'un latch dynamique

Le fonctionnement d'un latch est caractérisé par ses périodes de transparence (figure 7.10). Dans cet état, l'information qui traverse le latch subit un retard τ_t appelé *temps de traversée*. Pour que l'information qui traverse le latch soit mémorisée il est nécessaire qu'elle arrive un temps τ_p avant que la période de transparence ne s'achève. Ce temps τ_p (voisin de τ_t) est appelé le *temps de pré-positionnement*. Comme le fonctionnement des systèmes polyphasés est basé sur des phases et non des événements, l'information à mémoriser reste généralement présente à l'entrée du latch pour la totalité de la phase suivante, ce qui évite de devoir se soucier de son éventuelle disparition prématurée.

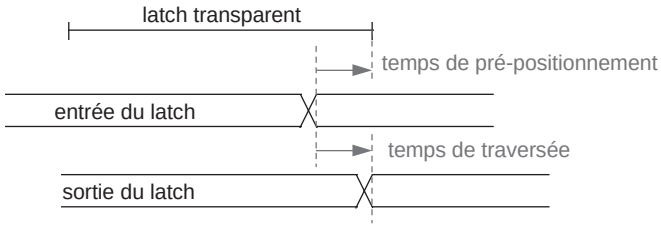


Figure 7.10 Contraintes temporelles d'un latch

Un latch est souvent utilisé pour maintenir un signal au-delà de sa durée de validité. On parlera alors de la *capture* de ce signal. Il est important de remarquer que le signal est présent à la sortie du latch avant que sa période de transparence ne s'achève, ce qui permet d'accélérer fortement les systèmes polyphasés en anticipant l'excitation des réseaux combinatoires suivants.

Un signal multi-bits véhiculé sur une nappe de fils peut être capturé dans une batterie de latches constituant un *registre* (figure 5.11).

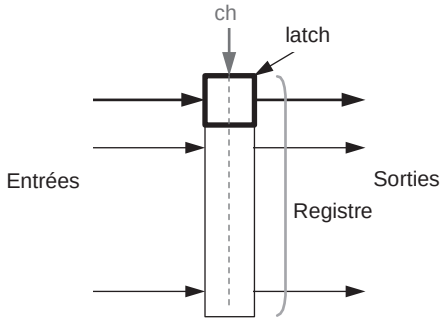


Figure 7.11 Registre constitué de latches

7.4.2 Systèmes polyphasés

Comme les latches ont un état de transparence, il faut éviter que le système ne puisse se retrouver rebouclé directement sur lui-même et devienne asynchrone. Pour cela, plusieurs niveaux de latches en série sont nécessaires. Ceux-ci doivent fonctionner sur le principe des écluses, c'est-à-dire qu'ils ne doivent jamais être simultanément transparents.

Les différentes horloges qui pilotent le système définissent les *périodes* de transparence et de mémorisation des différents latches et non des *événements* ponctuels.

Puisque la conception des latches utilise des portes de transfert et quelquefois des informations mémorisées dans des capacités parasites, la conception des systèmes polyphasés exploite bien les possibilités spécifiques aux circuits CMOS VLSI. Elle est très proche du niveau des transistors et demande de bien prendre en compte des

phénomènes « fins » concernant le séquencement, tels que la propreté des signaux. Les outils de conception automatiques actuels ne savent pas prendre en compte ce type de conception. Cette approche est actuellement réservée au « cœur » des microprocesseurs. Elle permet toutefois la conception de circuits très optimisés, denses, rapides et comportant peu de transistors.

a) Systèmes maître-esclaves

Un système polyphasé « minimal » consiste à réaliser une machine synchrone dans laquelle les bascules qui mémorisent l'état courant sont constituées de deux latches en série. De telles bascules sont appelées *maître-esclave* (figure 7.12). Nous verrons qu'elles sont également utilisées par les systèmes monophasés.

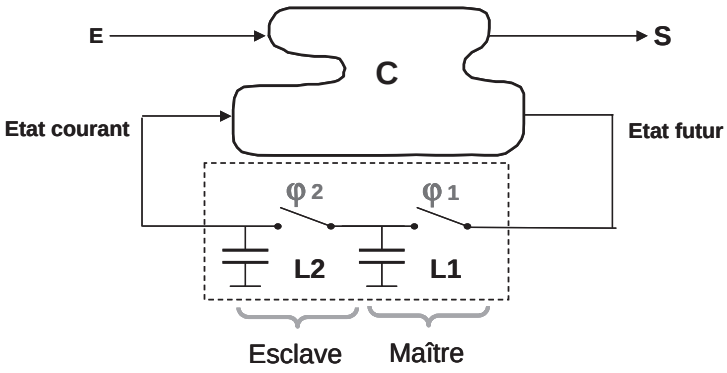


Figure 7.12 Système biphasé maître-esclave

Un tel système utilise une horlogerie *bi-phasée* dans laquelle chaque phase est utilisée pour commander la transparence d'un niveau de latches. Les deux phases ne doivent jamais se recouvrir pour éviter que deux latches en série ne soient simultanément transparents. Les phases sont séparées par des temps de *non-recouvrement* (figure 7.13).

Le fonctionnement temporel d'un tel système est un peu contre-intuitif. En effet, pendant la phase φ2, le réseau combinatoire commence à être excité par le latch

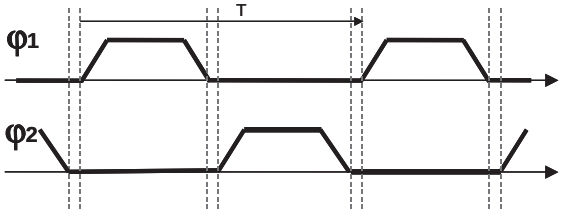


Figure 7.13 Système d'horloges biphasées

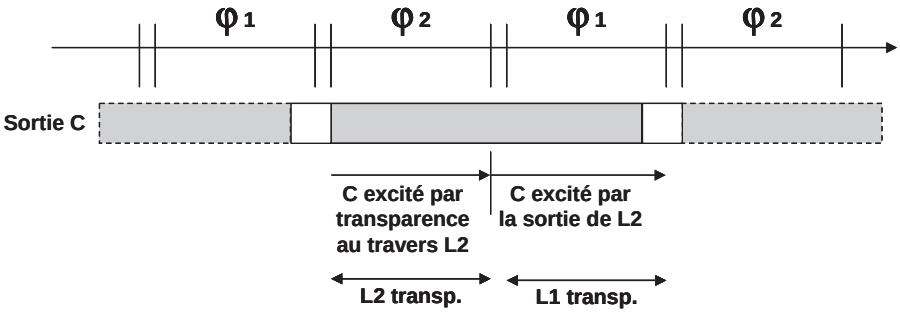


Figure 7.14 Séquencement d'un système biphasé maître-esclave

maître, au travers du latch esclave, dès que ce dernier passe en mode transparent. Cette excitation perdure lorsque le latch esclave passe en mode mémorisant (phase φ1). Le circuit combinatoire est donc excité pendant la totalité du cycle. Il doit fournir ses valeurs de sortie avant la fin de la phase correspondant à la transparence du latch maître (figure 7.14). Cette analyse montre que le fonctionnement d'un système bi-phasé qui utilise des bascules maître-esclave est beaucoup plus optimisé qu'il n'y parait à première vue.

b) Systèmes bouclés polyphasés

La première idée qui vient à l'esprit lorsque l'on analyse un système maître-esclave consiste à séparer les deux latches et à diviser le réseau combinatoire en deux parties que l'on peut répartir entre les latches. Ainsi transformé, le système prend la forme d'une boucle dans laquelle l'information tourne et se transforme au rythme des phases. Un tel système peut être vu comme un système fonctionnel naturellement rebouclé (par exemple un chemin de données de processeur) dans lequel des latches ont été ajoutés pour éviter l'occurrence d'un mode asynchrone et pour répartir l'activité sur plusieurs phases. Ces latches sont appelés *barrières temporelles* [NOG75] (figure 7.15).

Une telle approche peut être utilisée avec deux, (trois), quatre, etc., phases. Nous verrons que l'étude des systèmes polyphasés concerne surtout leur séquencement

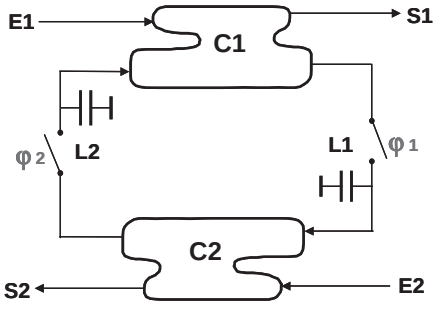


Figure 7.15 Système séquentiel biphasé à réseaux combinatoires distribués

que nous appellerons *architecture temporelle*. L'approche polyphasée permet aussi la prise en compte de problèmes topologiques, ce qui fait que les structures qui sont conçues avec cette approche sont souvent les plus optimisées d'un circuit VLSI.

Pour étudier le fonctionnement d'un tel système, faisons d'abord l'hypothèse qu'il est symétrique, c'est-à-dire que ses deux circuits combinatoires ont le même temps de traversée. Pour que les latches puissent capturer l'information, celle-ci doit être valide avant la fin de leur phase de transparence. Nous pouvons donc débuter notre raisonnement en supposant que les circuits combinatoires commencent à recevoir de l'information valable pendant la phase de transparence des latches qui les précèdent. À la fin de cette phase les circuits combinatoires continuent à être excités par les latches lorsqu'ils passent en mode mémorisant. Ces circuits combinatoires doivent fournir leur résultat avant la fin de la phase suivante (pendant la transparence du latch qui les suit) [ANC86].

– Si la somme des durées d'établissement des réseaux combinatoires (plus les temps de traversée des deux latches) est supérieure à la période, alors les instants ou leur sortie est établie et où le suivant commence à être excité, vont se déplacer vers le futur et vont finir par sortir de la phase de transparence des latches (figure 7.16). Le système va cesser de fonctionner, ce qui est normal puisqu'il n'obéit pas à la relation de synchronisme !

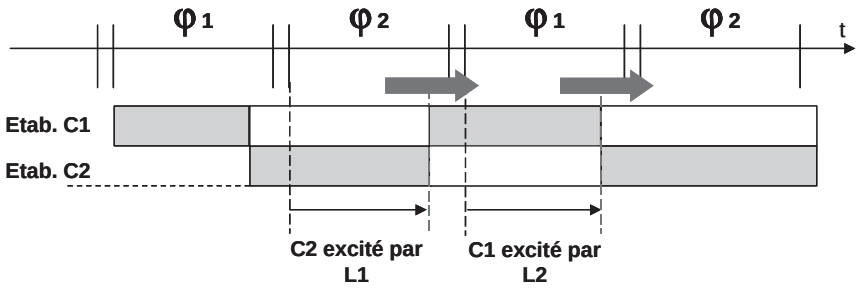


Figure 7.16 Séquencement d'un réseau combinatoire biphasé (cas de la somme des retards supérieure à la période)

– Si la somme des durées d'établissement des réseaux combinatoires (plus les temps de traversée des deux latches) est inférieure à la période, alors les instants ou leur sortie est établie, et où le réseau combinatoire suivant commence à être excité, vont se déplacer vers le passé (figure 7.17).
– L'instant au plus tôt où les circuits combinatoires peuvent commencer à être excités est le début de la phase de transparence des latches qui les précèdent. Cela signifie que les réseaux combinatoires s'établissent pendant cette phase de transparence. Lorsque la somme des durées d'établissement des réseaux combinatoires diminue encore, des zones d'attente apparaissent à la fin des phases de transparence. Ce sont ces périodes d'attente qui assurent la stabilité du système en absorbant les éventuelles fluctuations des durées d'établissement.

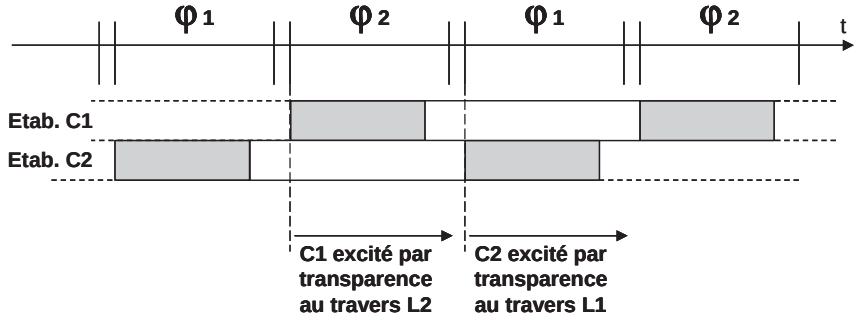


Figure 7.17 Séquencement d'un réseau combinatoire biphasé (cas de la somme des retards inférieure à la période)

– Si maintenant on diminue la durée d'établissement de l'un des réseaux combinatoires en augmentant celle de l'autre, (tout en maintenant la somme en dessous de la période de l'horlogerie), le système évolue vers le comportement d'un maître-esclave (figure 7.18).

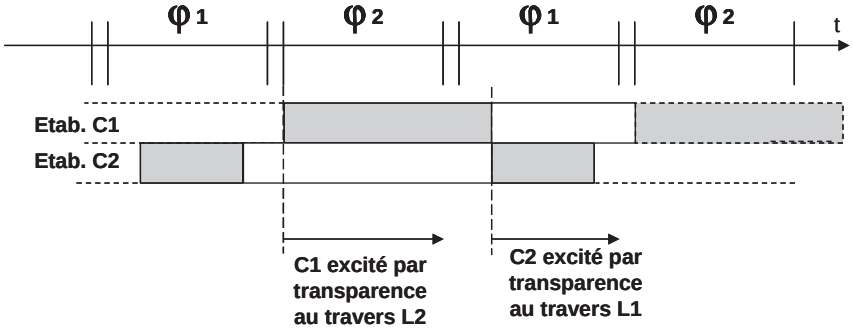


Figure 7.18 Séquencement d'un réseau combinatoire biphasé (cas de retards disymétriques)

c) Systèmes polyphasés réels

Comme nous l'avons déjà mentionné, l'utilisation des techniques polyphasées est réservée aux cœurs des microprocesseurs et plus particulièrement à la conception de leur chemin de données. Ces structures, généralement biphasées, sont caractérisées par le fait qu'elles sont rebouclées et que leur état interne est constitué par des données dont le codage est imposé.

Dans les systèmes polyphasés réels les informations pertinentes sont contenues dans des registres *pertinents* constitués de latches statiques. Les autres registres ne sont introduits que pour éviter que les registres pertinents ne soient rebouclés sur eux-mêmes. Ils seront appelés des *barrières temporelles* [NOG75]. Celles-ci ne contien-

dront que des valeurs transitoires. Ces registres pourront être dynamiques si la vitesse de fonctionnement du système le permet. Il sera commode de considérer que l'information circule dans ces boucles en partant et en revenant dans les registres pertinents.

Dans les structures de ce type tous les registres pertinents ne sont pas mis en jeux à chaque cycle. Certains appelés *sources*, sont sélectionnés en lecture au début du cycle (généralement φ1), tandis que d'autres, appelés *destinations*, sont sélectionnés en écriture en fin de cycle (φ2 dans un système biphasé). La sélection des registres sources se fait en les munissant d'une sortie 3 états qui peut être activée pendant le phase de sélection. Celle des registres de destination consiste à les mettre en mode transparent pendant la dernière phase du cycle. Certains registres peuvent être à la fois source et destination.

Un schéma de séquencement pour une telle structure peut être :

– Pendant la première phase (φ1), la sortie des registres sélectionnés est connectée sur les bus source. La barrière temporelle intermédiaire est mise en mode transparent. L'unité arithmétique et logique commence à être excitée par transparence, au travers cette barrière temporelle.
– Pendant la seconde phase (φ2), la barrière temporelle intermédiaire est mise en mode mémorisant et continue à exciter l'unité arithmétique et logique dont la sortie excite le bus destination. Le registre destination est mis en mode transparent. Il reçoit une information stabilisée vers la fin de cette phase.

Exemple : Chemin de données de processeur (figures 7.19 et 7.20).

Nous constatons que, grâce aux transparences, les temps d'établissement des différents organes mis en jeu (bus sources, unité arithmétique et logique, bus destination) s'enchaînent sans attente dans la période de l'horloge. Cela montre une utilisation quasi optimale du temps, synonyme d'une performance maximale.

Une autre approche aurait consisté à insérer une barrière temporelle après l'unité arithmétique et logique. Le schéma de séquencement aurait toutefois été plus complexe.

d) Relations temporelles

L'analyse du fonctionnement temporel des systèmes polyphasés conduit à définir des équations temporelles qui régissent leur fonctionnement.

➤ Décalage des phases

Nous devons d'abord prendre en compte les éventuels décalages dans la distribution de l'horloge. Cet écart, appelé *gigue* (ou *skew* en anglais) est généralement limité à la valeur des temps de non recouvrement. En effet, tout recouvrement d'une phase par la suivante risque de permettre un rebouclage asynchrone du système. Un tel recouvrement pourrait être légèrement toléré si l'information, même erronée, n'a pas le temps de venir modifier le contenu d'un registre pertinent pendant le recouvrement (figure 7.21).

Nous voyons que pour éviter un rebouclage asynchrone, la valeur de $\Delta\tau$ doit être inférieure au temps de transfert sans perturbation de la boucle.

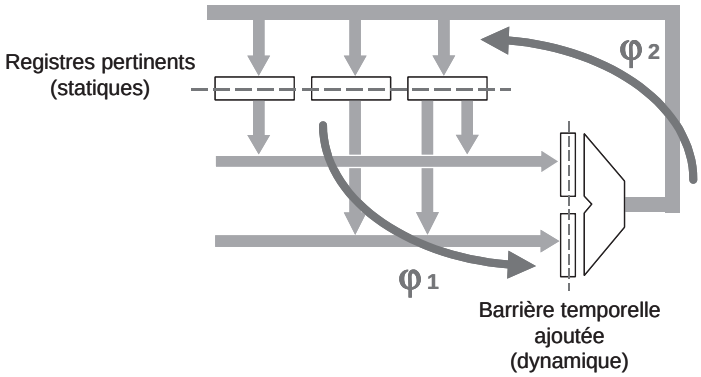


Figure 7.19 Chemin de données vu comme un système biphasé

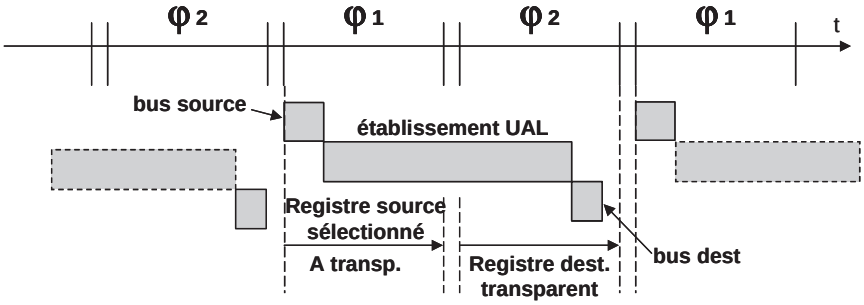


Figure 7.20 Chronogramme typique d'un chemin de données biphasé

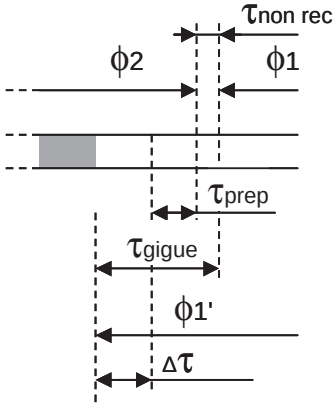


Figure 7.21 Cas du recouvrement de ∅2 par le ∅1 suivant

C'est-à-dire :

$$\Delta\tau < \tau \text{ sel.} + \Sigma\tau \text{ latches} + \Sigma\tau \text{ réseaux avant perturb.}$$

D'où :

$$\boxed{\tau \text{ gigue} < \tau \text{ sel.} + \Sigma\tau \text{ reg} + \Sigma\tau \text{ réseaux avant perturb.} + \tau \text{ prep.} + \tau \text{ non rec.}}$$

avec :

τ gigue : Temps maximum de décalage entre les horloges des registres sources et destinataires.

τ sel. : Temps de sélection de la sortie du (ou des) registre(s) source(s).

$\sum\tau$ reg. : Somme des temps de traversée des registres pertinents et intermédaires.

$\sum\tau$ réseaux avant perturb. : Temps min avant que la sortie des réseaux combinatoires ne soit perturbée.

τ prep : Temps de prépositionnement du latch de destination.

τ non rec. : Temps de non-recouvrement entre les phases.

Dans le cas (pire) d'une connexion directe entre les latches (τ sel. = 0 et $\Sigma\tau$ réseaux avant perturb. = 0), la relation devient :

$$\boxed{\tau \text{ gigue} < \Sigma\tau \text{ reg} + \tau \text{ prep.} + \tau \text{ non rec.}}$$

qui peut être vue comme une bonne valeur maximale de la gigue.

➤ Relation de synchronisme

Nous voyons que la relation de synchronisme peut aussi être affinée et s'écrire :

$$\boxed{\tau \text{ cycle} > \tau \text{ sel} + \Sigma\tau \text{ réseaux comb.} + \tau \text{ reg-int.} + \tau \text{ prep} + \tau \text{ non-rec.} + \tau \text{ gigue}}$$

avec :

$\sum\tau$ réseaux comb. : Temps max d'établissement des réseaux combinatoires concernés.

τreg-int. : Temps de traversée du registre intermédiaire.

Nous pouvons aussi établir que :

$$\boxed{\tau\varphi 1 > \tau \text{ sel}}$$

qui signifie que le temps de stabilisation des réseaux de sélection et celui d'éventuels organes situés avant la barrière temporelle intermédiaire doit être inférieur à la durée de la phase $\varphi 1$ pendant laquelle cette première barrière est transparente. Par contre, le temps d'établissement des organes situés après cette barrière temporelle peut être supérieur à la durée de la phase $\varphi 2$.

e) Utilisation de PLA

L'utilisation des techniques polyphasées pour réaliser des séquenceurs complexes conduit à utiliser des PLA.

La réalisation d'un système polyphasé avec un PLA revient à réaliser un système maître-esclave dans lequel le réseau combinatoire est un PLA (figure 7.22). Sa matrice

« ET » est programmée pour reconnaître les combinaisons utiles de l'état courant et des profils d'entrée. La matrice « OU » est programmée pour générer l'état suivant et les sorties correspondant à la configuration reconnue par la matrice « ET ». Le PLA dispose de la totalité de la période pour calculer ses sorties. Il peut s'agir soit d'un PLA statique pulsé ou non (qui consomme) ou d'un PLA dynamique (plus complexe) qui utilise les deux phases.

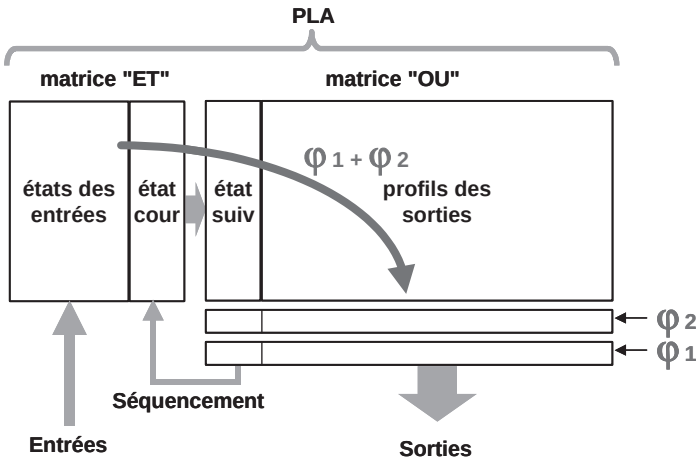


Figure 7.22 Séquenceur biphasé utilisant un PLA

7.5 SYSTÈMES MONOPHASÉS

L'étude des systèmes monophasés relève d'une toute autre approche. Ces systèmes sont principalement utilisés pour réaliser des séquenceurs. De très nombreux travaux ont été réalisés pour optimiser les systèmes monophasés en codant judicieusement leurs états et en effectuant une synthèse optimale de leurs réseaux combinatoires. De nombreux outils informatiques effectuent tout ou partie de ces travaux de manière automatique ou assistée.

Comme leur nom l'indique, les systèmes monophasés n'utilisent qu'un seul signal d'horloge, éventuellement complété par des *sous-horloges*, c'est-à-dire des copies de ce signal validées par des conditions. Le fonctionnement de ces systèmes est défini par les événements véhiculés par l'horloge (figure 7.23). La prise en compte de l'état futur, pour en faire l'état courant, se produit sur un front de l'horloge (souvent le front descendant).

Les systèmes monophasés utilisent des bascules pour mémoriser leur état. Ces bascules sont des petits systèmes asynchrones qui « simulent » un comportement synchrone. Leur étude est assez complexe, mais heureusement celles-ci sont souvent fournies comme des composants. La réalisation des bascules fait souvent appel à des astuces logiques ou électroniques. Celles-ci utilisent souvent un nombre important

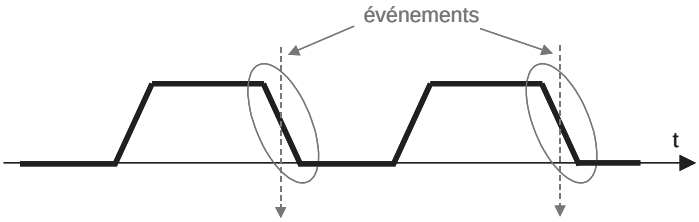


Figure 7.23 Événements d'une horloge monophasée

de transistors (voisin de la trentaine). Nous distinguerons les bascules asynchrones et celles synchronisées, c'est-à-dire munies d'une entrée d'horloge qui conditionne la prise en compte de leurs entrées. Il existe aussi des bascules mixtes munies des deux types d'entrées.

7.5.1 Bascules

L'étude des bascules est une partie importante de celle des systèmes synchrones monophasés. Toutes les basculent possèdent deux états internes « visibles » (1 et 0) et deux sorties (Q et QB). Nous allons commencer par l'étude de la bascule RS qui, bien qu'asynchrone, constitue la base de nombreuses autres bascules.

a) Bascule RS

Cette bascule est la plus simple du catalogue (figure 7.24). Elle est constituée par le rebouclage de deux portes NOR. Comme c'est le cas pour la majorité des systèmes asynchrones, son état est mémorisé dans les retards de ses portes. La bascule RS possède deux entrées (R pour Reset et S pour Set). L'excitation de l'entrée R a pour effet de remettre la bascule dans l'état 0, tandis que l'excitation de l'entrée S la met dans l'état 1. L'excitation simultanée des deux entrées est interdite.

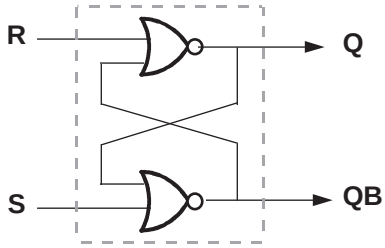


Figure 7.24 Bascule RS

Le schéma de cette bascule peut être redessiné pour faire apparaître le réseau combinatoire et le rebouclage (figure 7.25).

Le graphe d'état de cette bascule est représenté figure 7.26.

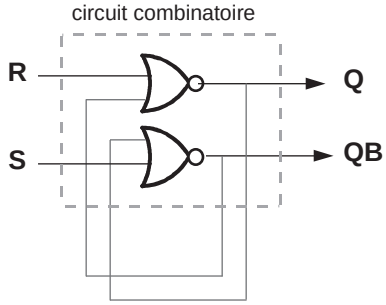


Figure 7.25 Une bascule RS vue comme un système séquentiel asynchrone

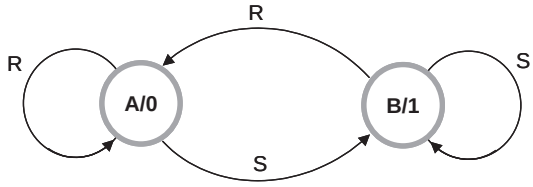


Figure 7.26 Graphe d'état de la bascule RS

Son tableau de transitions est (les états entourés sont stables) le suivant (figure 7.27).

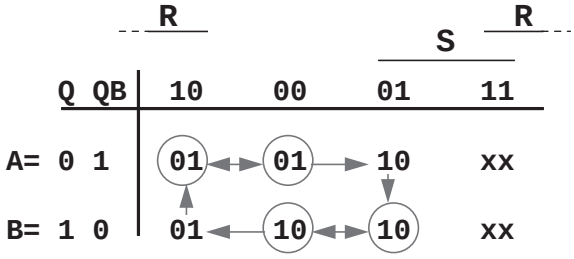


Figure 7.27 Tableau d'états du RS

Du tableau nous pouvons tirer l'explicitation des fonctions booléennes qui déterminent les sorties Q et QB (fonction g).

$$Q = \overline{R} \wedge (Q \vee S)$$

$$QB = \overline{S} \wedge (QB \vee R)$$

La synthèse séparée de chacune de ces fonctions donne le schéma suivant (figure 7.28).

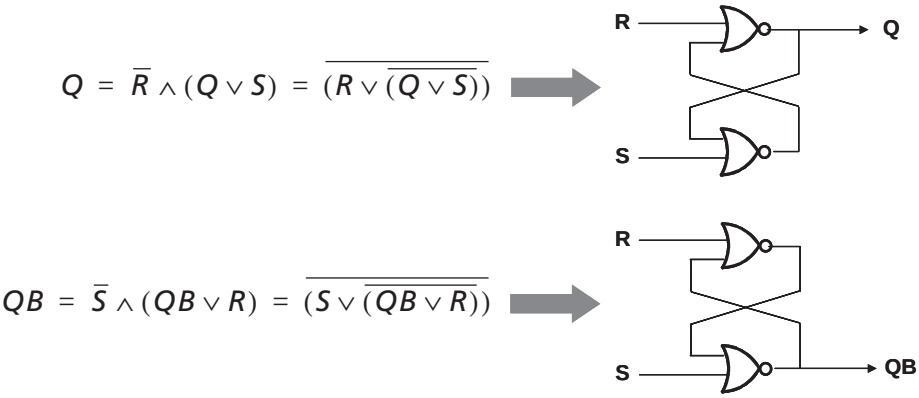


Figure 7.28 Synthèse séparée des deux fonctions booléennes du RS

Les deux dessins sont superposables, ce qui donne le schéma classique de la bascule RS, ainsi que ses équations.

$$Q = \overline{(R \vee QB)}$$

$$QB = \overline{(S \vee Q)}$$

b) Bascules synchronisées

Les bascules synchronisées possèdent une entrée d'horloge et changent d'état sur la transition (généralement descendante) de cette horloge. Nous pouvons dès à présent distinguer trois familles de bascules synchronisées :

– Les bascules dites *à niveau* sont des systèmes asynchrones qui possèdent des états transitoires en plus des états visibles. Ces bascules fonctionnent en détectant l'une des transitions de l'horloge.
– Les bascules dites *sur transitions* qui possèdent un circuit *différenciateur* qui transforme l'une des transitions de l'horloge en une brève impulsion.
– Les bascules *maître-esclave* qui sont des bascules polyphasées constituées de deux latches et d'un générateur de phases qui transforme l'horloge d'entrée en deux phases.

Les bascules sont souvent réalisées à partir d'un RS qui stocke leur état visible.

Les bascules synchronisées sont caractérisées par plusieurs paramètres temporels et aussi par les instants où elles sont sensibles à leurs entrées. Pour que l'information qui excite la bascule soit mémorisée il est nécessaire qu'elle arrive un temps τ_p avant la transition de l'horloge. Ce temps τ_p est appelé le *temps de pré-positionnement*. Le fonctionnement de certaines bascules nécessite que l'information soit maintenue un temps τ_m après la transition de l'horloge. Ce temps τ_m est appelé le *temps de maintien* de l'entrée. L'information apparaît en sortie avec retard τ_b appelé le *temps de basculement* de la bascule. Toutefois, elle peut être perturbée avant. Par sécurité, nous supposerons qu'elle peut être perturbée dès la transition de l'horloge (figure 7.29).

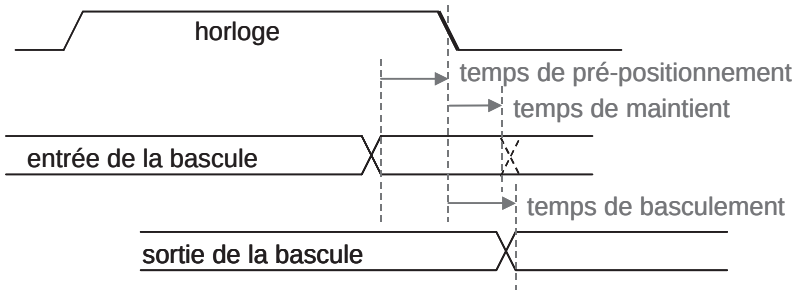


Figure 7.29 Contraintes temporelles d'une bascule

➤ Bascule D

La bascule D est très utilisée (figure 7.30). Elle dispose d'une seule entrée appelée D. Sa fonction consiste à mémoriser la valeur de son entrée au moment de la transition du signal d'horloge. Son nom vient du mot anglais *delay* (retard) qui exprime le fait que sa sortie recopie son entrée avec une période d'horloge de retard.

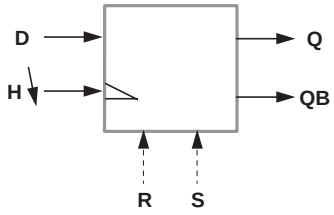


Figure 7.30 Symbole d'une bascule D

Les bascules D sont souvent munies d'entrées R et S asynchrones supplémentaires qui agissent directement sur le RS qui stocke l'état visible de la bascule.

Son tableau d'états est représenté figure 7.31.

Les bascules D sont, de loin, les plus nombreuses. Leurs différentes techniques de réalisation illustrent les différentes façons d'aborder le paradigme de la logique synchrone. Nous pouvons donc distinguer trois familles de bascules D.

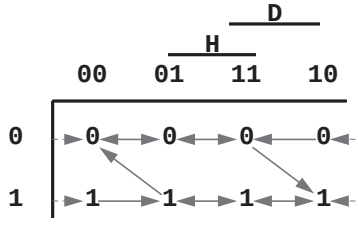


Figure 7.31 Tableau d'état d'une bascule D

Bascule D à niveaux

La réalisation classique d'une bascule D à niveaux (par exemple, la 7474 des catalogues MSI ex TTL) comporte 6 portes NOR soit 28 transistors (figures 7.32 et 7.33). Cette machine asynchrone possède 4 états dont deux stables et deux transitoires.

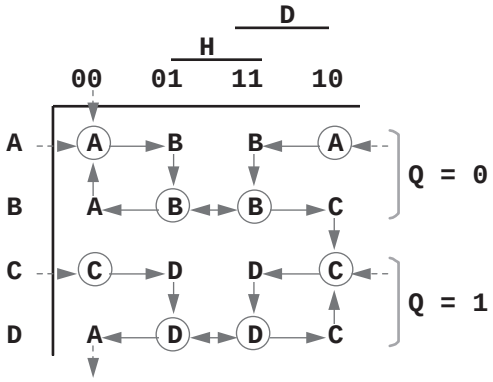


Figure 7.32 Tableau d'états d'une bascule D à niveaux

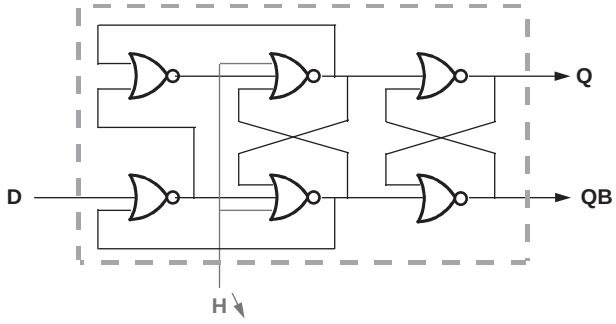


Figure 7.33 Schéma logique d'une bascule D à niveaux

Bascule D sur transition

L'une des réalisations possibles d'une bascule D sur transition est donnée par la figure 7.34 qui montre une bascule HLFF utilisée par AMD pour son microprocesseur K6 [PAR96]. Cette bascule nécessite 20 transistors. On distingue le différentiateur qui produit une impulsion brève à chaque transition positive de l'horloge.

Bascule D maître-esclave

Cette bascule D est constituée de l'assemblage de deux latches utilisant des portes 3 états C^2MOS [SUZ73] (figure 7.35). Cette bascule est réalisée avec 22 transistors. Un générateur de phases est simplement réalisé avec un inverseur. Les relations établies

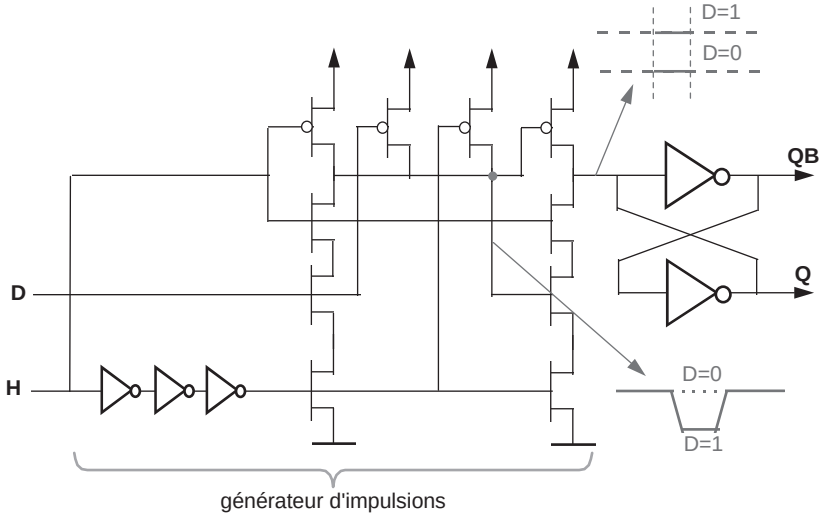


Figure 7.34 Bascule D de Partovi®

au § 7.4.2.e montrent qu'il est possible de tolérer une légère superposition dans ses phases sans risquer un rebouclage asynchrone de tout le système. Cette bascule est très utilisée soit comme composant logique individualisé (7474 CMOS), soit comme élément des bibliothèques utilisées pour réaliser la synthèse automatique des circuits intégrés.

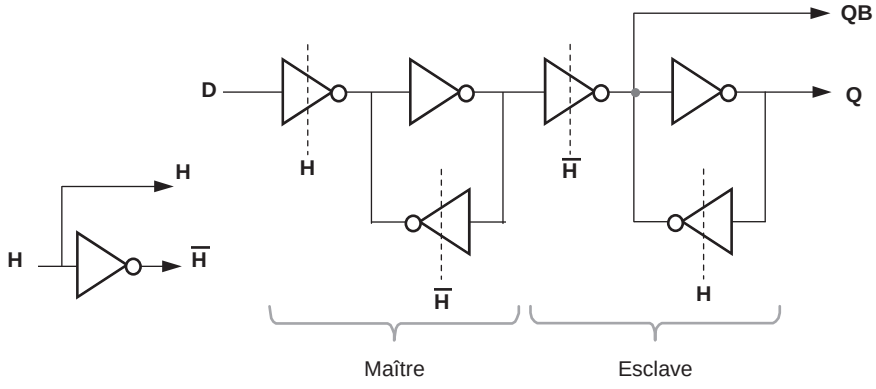


Figure 7.35 Bascule D C²MOS maître-esclave

➤ Bascule JK

La bascule JK est une extension synchronisée de la bascule RS (figures 7.36 et 7.37). Ses entrées sont alors appelées J et K. Cette bascule accepte que ses entrées soient simultanément portées à 1. Dans ce cas, la bascule change d'état à chaque transition

significative de l'horloge. La bascule JK peut également disposer d'entrées supplémentaires RS asynchrones.

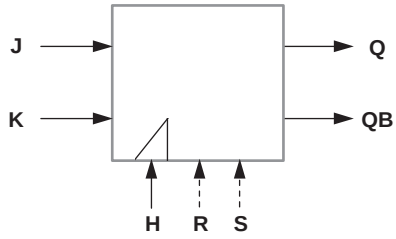


Figure 7.36 Symbole d'une bascule JK

K	J			K
	10	00	01	11
0	0	0	1	1
1	0	1	1	0

Figure 7.37 Tableau d'états d'une bascule JK

➤ Bascule T (figure 7.38)

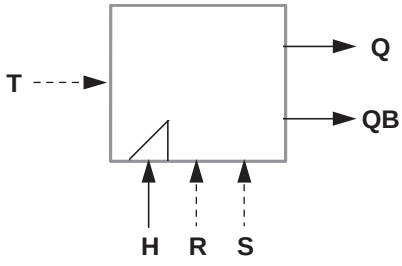


Figure 7.38 Symbole d'une bascule T

Les bascules T se présentent sous deux formes :

– soit avec une entrée T, elle constitue alors un diviseur par deux de l'horloge, commandé par l'entrée T ;
– soit sans entrée T, elle n'est alors qu'un simple diviseur par deux de l'horloge.

La bascule T peut également disposer d'entrées supplémentaires RS asynchrones.

c) Problèmes avec les bascules

L'approximation du modèle synchrone par les bascules n'est pas parfaite. Dans certaines conditions, des effets parasites peuvent apparaître.

➤ Risque de transparence

Les bascules à niveau possèdent un mode de transparence lorsque le niveau de l'horloge atteint une valeur intermédiaire (les portes se comportent alors comme des amplificateurs). Généralement, la vitesse de transition de l'horloge est telle que cette zone de transparence est traversée trop rapidement pour que des rebouclages asynchrones puissent apparaître. De même, le différenciateur des bascules à transition ne fonctionne correctement que si le front de l'horloge est suffisamment bref. Cela montre que les bascules synchronisées ne fonctionnent correctement que si le signal d'horloge répond à certaines contraintes de rapidité sur ses transitions. Les systèmes monophasés doivent donc obéir à deux conditions qui encadrent le comportement de l'horloge. Celle-ci doit donc :

– être suffisamment lente pour permettre au système de fonctionner (condition générale de synchronisme) ;
– avoir des transitions suffisamment rapides.

➤ Métastabilité

Le fonctionnement « normal » d'une bascule est obtenu lorsque ses entrées sont bien établies au moment de la transition du signal d'horloge. Il peut toutefois arriver que dans certaines applications une entrée de la bascule varie au moment précis de la transition du signal d'horloge. Si la relation temporelle entre ces deux transitions est suffisamment précise, la bascule peut basculer incomplètement car elle ne dispose pas de suffisamment d'énergie pour basculer complètement. Elle n'atteint alors qu'une position intermédiaire que nous appellerons 1/2. Cela ne serait pas très gênant si le retour à une position logique (1 ou 0) ne générait pas une transition asynchrone de sa sortie qui risque de mettre aussi la bascule suivante en état métastable (figure 7.39).

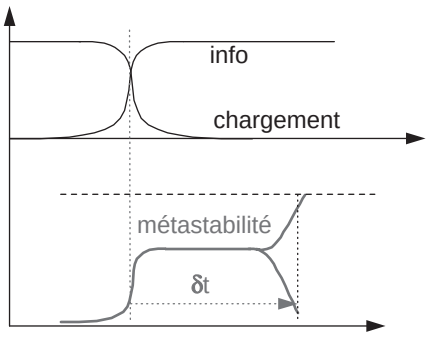


Figure 7.39 Mise en état métastable d'une bascule

La largeur de la relation temporelle qui peut mettre une bascule en état métastable est très faible (de l'ordre d'une fraction de ps). La probabilité d'occurrence de la métastabilité pour une bascule qui reçoit des signaux complètement désynchronisés, est donc très faible (10^{-12} à 10^{-14}). Toutefois, si la fréquence de fonctionnement est assez élevée, ce phénomène peut effectivement apparaître sur des systèmes en fonctionnement continu (satellites, centraux téléphoniques) et provoquer des pannes sans cause matérielle. Il semble, toutefois, qu'il puisse y avoir d'autres causes possibles pour de telles pannes.

Puisque le fonctionnement interne des bascules est asynchrone, il importe que leur état interne soit codé à l'aide du code de Gray pour que seul un bit varie à chaque transition. Si cela n'était pas le cas, la métastabilité pourrait provoquer une indécision supplémentaire entre les différents états dus aux changements non simultanés des bits d'état.

Il semble qu'il n'y ait aucune parade « logique » à la métastabilité. Les seules défenses possibles semblent être soit la conception de systèmes asynchrones qui peuvent attendre la fin d'un état métastable, soit la conception de bascules spéciales dans lesquelles l'énergie interne de l'état intermédiaire est particulièrement élevée.

7.5.2 Systèmes monophasés

Les systèmes monophasés sont des systèmes synchrones qui utilisent des bascules pour stocker leur état. Toutes leurs bascules reçoivent la même horloge ou des horloges dérivées. Le fait d'avoir pré-résolu tous les problèmes asynchrones dans la conception des bascules fait que la conception des systèmes monophasés peut se focaliser sur les problèmes d'optimisation logique. Toutefois, la mise en phase des horloges qui excitent les bascules introduit de nouvelles contraintes (§ 7.5.2.b).

a) Registres

L'état des systèmes monophasés est stocké dans des *registres*. Ceux-ci sont réalisés par l'assemblage de plusieurs bascules D en parallèle (figure 7.40). D'une manière générale, un *registre* peut être utilisé pour stocker une information vectorielle.

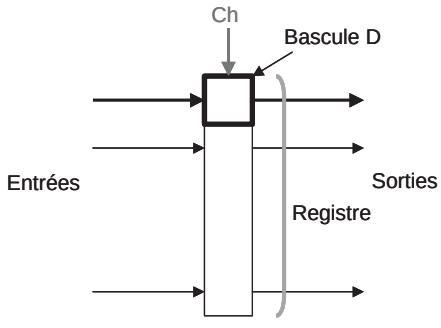


Figure 7.40 Registre constitué de bascules D

Le chargement des bascules D qui constituent un registre est, soit systématique à chaque événement d'horloge, soit conditionnel, c'est-à-dire sous l'effet d'une commande particulière :

– Chargement *systématique* des bascules à chaque cycle d'horloge. Dans ce cas, un multiplexeur permet de déterminer si la future valeur provient du reste du système où est le contenu de la bascule elle-même (figure 7.41).

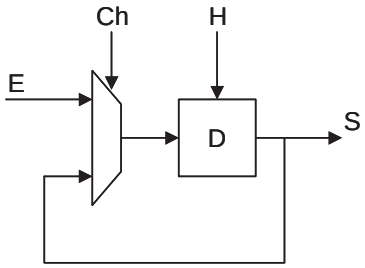


Figure 7.41 Élément de registre à chargement systématique de sa bascule D

Cette approche présente l'avantage que toutes les bascules reçoivent strictement la même horloge, ce qui réduit sa gigue et ses conséquences. Par contre, elle nécessite du matériel supplémentaire (qui constitue une sorte de seconde boucle de mémorisation) et surtout elle accroît la dissipation d'énergie par le fonctionnement continu de ses bascules.

– Le chargement *conditionnel* des bascules D est obtenu par la validation des impulsions d'horloge nécessaires à leur chargement (figure 7.42). Des portes sont alors utilisées pour conditionner les signaux d'horloge par les commandes de chargement. Le chargement se produit à la fin du cycle, ce qui est convenable pour capturer les résultats.

  Cette approche présente l'avantage de provoquer une consommation minimale d'énergie pour le chargement des bascules et de nécessiter peu de matériel. Son

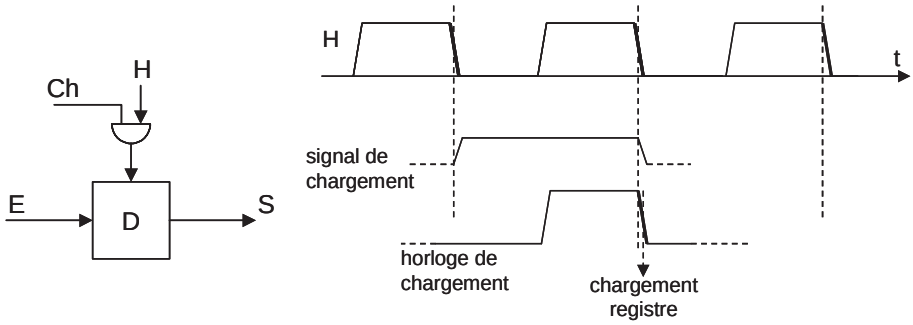


Figure 7.42 Chargement d'une bascule par la validation de son horloge

inconvénient est de produire un léger décalage des horloges validées. Toutefois, ce décalage peut être compensé en décalant systématiquement l'horloge non validée.

b) Relations temporelles

L'analyse du fonctionnement temporel des systèmes monophasés conduit à définir des équations temporelles qui régissent leur fonctionnement.

➤ Gigue de l'horloge

Les déphasages dans la distribution de l'horloge peuvent avoir des effets perturbateurs. Au moment de l'occurrence de l'événement d'horloge, Toutes les bascules devraient afficher leur nouvelles sorties qu'elles devraient avoir élaboré à partir de leurs valeurs d'entrée avant l'événement (moins le temps de pré-positionnement). Ces valeurs d'entrées sont elles-mêmes élaborées, par le réseau combinatoire qui calcule la fonction *g*, à partir des valeurs de sortie des bascules avant l'événement. Or l'horloge arrive sur les différentes bascules avec une gigue variable dépendant de la façon dont elle est distribuée. Si, pour une bascule particulière, ce déphasage est suffisamment important, une bascule peut capturer une valeur d'entrée erronée calculée à partir de la sortie d'une autre bascule prise *après* son événement d'horloge (figure 7.43). Ce risque est important lorsque les bascules sont connectées directement ou au travers de seulement quelques couches logiques.

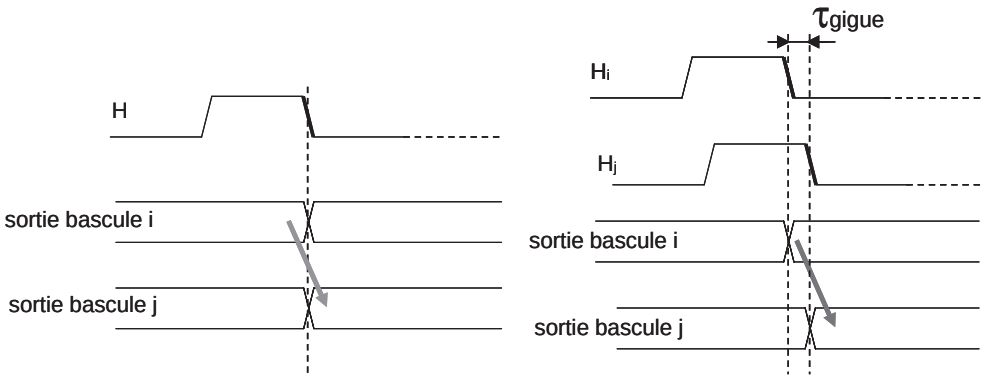


Figure 7.43 Sorties correctes et erronées d'une bascule à la suite d'un déphasage d'horloge (cas d'une connexion directe entre les bascules i → j)

Ce phénomène introduit une nouvelle relation temporelle, en plus de celle de synchronisme. Cette relation indique que le décalage maximum τ gigue de l'horloge d'un système monophasé doit être inférieur au temps, avant perturbation, du réseau combinatoire entre les sorties des bascules et leurs entrées (figure 7.44).

D'où la relation :

$$\tau \text{ gigue} < \Sigma\tau \text{ réseaux avant perturb.} + \tau \text{ prep.}$$

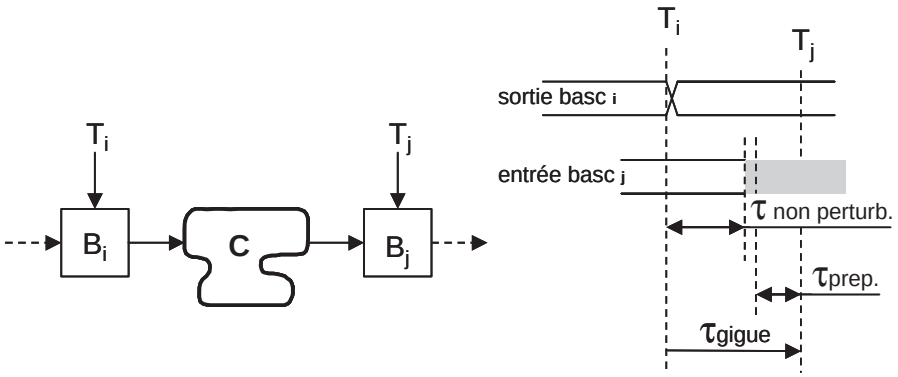


Figure 7.44 Cas d'une gigue perturbant la valeur lue par la bascule j

avec :

τ gigue : Temps maximum de décalage entre les horloges des registres sources et destinataires.

$\sum\tau$ réseaux avant perturb. : Temps min avant que la sortie des réseaux combinatoires ne soit perturbée.

τ prep. : Temps de prépositionnement du latch de destination.

Dans le cas d'un couplage direct entre les bascules, la relation devient :

$$\tau \text{ gigue} < \tau \text{ prep.}$$

Cela correspond au cas des registres à décalage. Ces organes sont généralement directement réalisés sous la forme de composants.

Cette relation montre que plus le réseau combinatoire possède un temps avant perturbation important, plus l'horloge accepte des déphasages importants. Elle peut être affinée en analysant le cas particulier de chaque bascule (et en prenant en compte la marge de déphasage de chacune de ses bascules sources).

Dans de nombreux systèmes monophasés, seules certaines bascules sont chargées à un instant déterminé. Cela est souvent obtenu en créant des sous-horloges par la validation de l'horloge de référence par des conditions. Les portes qui réalisent cette validation introduisent des retards qui doivent être compatibles avec la relation précédente.

➤ Relation de synchronisme

Nous voyons que la relation de synchronisme peut aussi être affinée et s'écrire :

$$\tau \text{ cycle} > \tau \text{ basc.} + \Sigma\tau \text{ réseaux comb.} + \tau \text{ prep.} + \tau \text{ gigue}$$

avec :

τ basc. : Temps de basculement des registres sources.

$\sum\tau$ réseaux comb. : Temps max d'établissement des réseaux combinatoires concernés.

c) Méthode de Huffman

La méthode générale de conception des systèmes séquentiels peut se décliner pour les systèmes monophasés sous le nom de *méthode de Huffman* :

- Les états symboliques de la spécification et les états supplémentaires qu'il est nécessaire d'ajouter sont explicités dans un graphe ou un tableau d'état cohérent.
- Le graphe, ou le tableau, est optimisé pour réduire les états superflus.
- Les états sont codés pour viser un certain type de réalisation (réseau combinatoire, PLA, logiciel…).
- On calcule les différentes projections en bits des fonctions *f* et *g*.
- On effectue la synthèse des réseaux combinatoires, le remplissage des PLA, la synthèse des réseaux combinatoires ou la génération du code.

d) Codage des états et synthèse des fonctions combinatoires

Comme nous l'avons déjà mentionné, un codage astucieux des états peut réduire la complexité du réseau combinatoire d'un système séquentiel. Suivant la complexité de ce système plusieurs solutions sont possibles :

- Systèmes simples (quelques dizaines d'états) : les états peuvent être codés dans un code 1 parmi n, ce qui revient à utiliser une bascule par état. Cette approche permet la réalisation de réseaux combinatoires particulièrement simples.
- Systèmes de complexité moyenne (jusqu'à une centaine d'états) : le système peut être réalisé en utilisant une structure matricielle (PLA). Les états sont alors codés en binaire. La réalisation du système sous la forme de Mealey permet de minimiser son PLA.
- Systèmes lents de forte complexité (plusieurs milliers d'états) : l'utilisation d'une structure de type microprocesseur (par exemple un micro-contrôleur) permet de réaliser le système sous la forme d'un programme.

e) Exemple de synthèse d'un système séquentiel câblé

Soit un système séquentiel (de Moore) qui comporte 4 états. Son diagramme d'état est représenté figure 7.45.

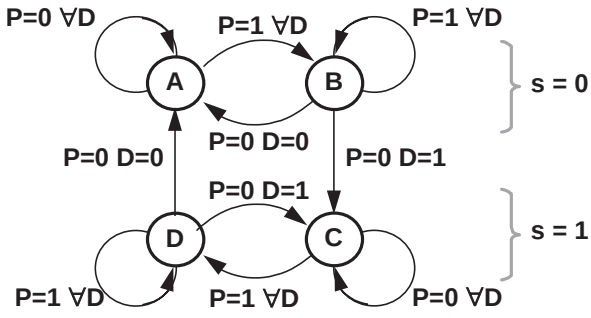


Figure 7.45 Exemple de graphe d'états d'un système synchrone

Sa synthèse est réalisée simplement en « câblant » le diagramme d'état à l'aide de quatre bascules D représentant les quatre états. Des multiplexeurs permettent d'aiguiller la propagation du bit d'état en fonction des entrées. Une seule porte est utilisée pour générer la sortie. Cette technique est utilisée pour réaliser de *énérateurs de temps* destinés à fournir des signaux permettant de rythmer une succession d'actions (figure 7.46).

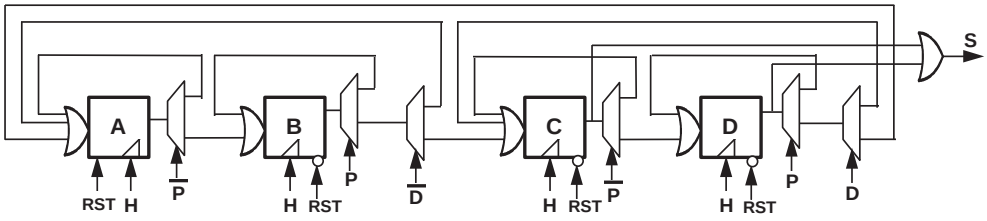


Figure 7.46 Machine séquentielle obtenue par le câblage du graphe d'états précédent

Dans ces montages, les retards apportés par les couches logiques entre les bascules sont relativement faibles. Cela signifie que la distribution de l'horloge doit être particulièrement soignée pour que les écarts de phase entre les arrivées d'horloge au niveau des bascules restent inférieurs à leur valeur maximum admissible. Pour cette raison, toutes les bascules reçoivent directement l'horloge de base.

f) Utilisation de PLA

La réalisation d'un système monophasé avec un PLA revient à utiliser cet organe comme un réseau combinatoire (figure 7.47). Sa matrice « ET » est programmée

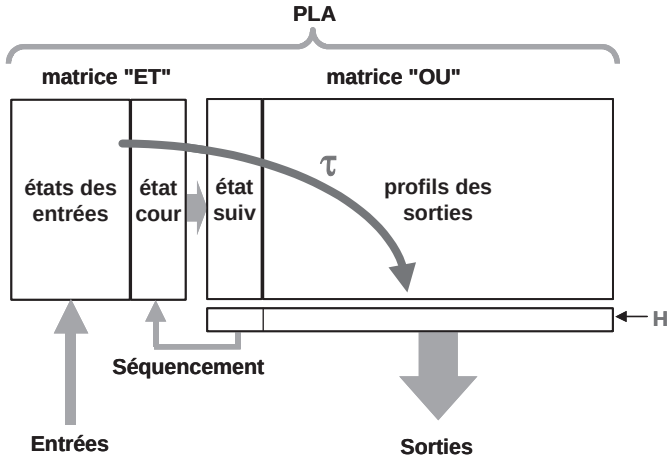


Figure 7.47 Réalisation d'un système monophasé avec un PLA

pour reconnaître les combinaisons utiles de l'état courant et des profils d'entrée. La matrice « OU » est programmée pour générer l'état suivant et les sorties correspondant à la configuration reconnue par la matrice « ET ». Le PLA dispose de la totalité de période pour calculer ses sorties. Il peut s'agir soit d'un PLA statique (qui consomme) ou d'un PLA dynamique pour lequel deux phases doivent alors être générées.

g) Utilisation de bascules maître-esclaves

La réalisation d'un système séquentiel monophasé à l'aide de bascules maître-esclaves permet de retrouver le degré d'optimisation des systèmes polyphasés statiques sans l'inconvénient d'avoir à distribuer les phases d'horloge. En effet, de par son fonctionnement, chaque bascule maître-esclave utilise un signal complémentaire de l'horloge comme une seconde phase. La tolérance de ces bascules aux légères superpositions de ces phases permet de s'abstraire des problèmes de non-recouvrement locaux. Les transparences successives des deux latches qui constituent chaque bascule maître-esclave permettent la même optimisation temporelle que celle obtenue avec les systèmes polyphasés. L'établissement du réseau combinatoire des systèmes monophasés peut donc occuper pratiquement presque toute la période.

7.6 SYSTÈMES MIXTES MONOPHASÉS/POLYPHASÉS

Certains systèmes comportent à la fois des blocs de circuiterie monophasée (plus ou moins conçus automatiquement) et des blocs polyphasés (conçus manuellement en tant que briques élémentaires). Le fonctionnement de ces systèmes est rythmé par une horloge unique à partir de laquelle on génère localement les phases nécessaires aux blocs polyphasés (figure 7.48).

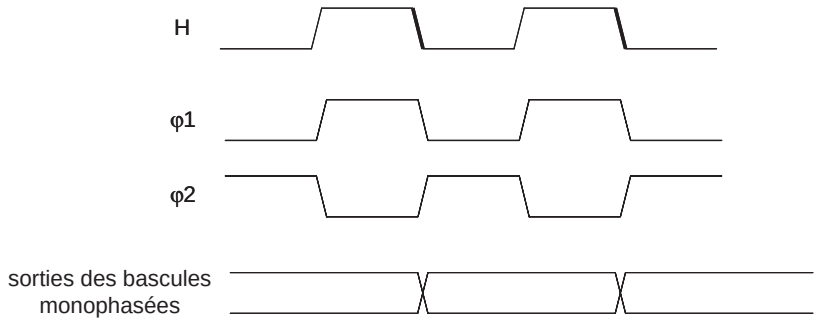


Figure 7.48 Relations temporelles entre les sous-systèmes monophasés et biphaséss

Les relations temporelles entre ces deux types de blocs doivent être soigneusement étudiées. Les latches des blocs biphasés, transparents sur φ2, seront vus comme des sorties de maître-esclave par les blocs monophasés. Les sorties de bascules mono-

phasées pourront être capturées par des latches transparents sur φ2 ou sur le φ1 suivant en fonction des retards des circuits combinatoires traversés.

De nombreux systèmes monophasés fabriquent localement des phases φ1 et φ2 en considérant l'horloge et son complément sans se soucier de leur non-recouvrement (cas des bascules maître-esclave). Il leur arrive aussi d'utiliser plusieurs horloges décalées pour décomposer la période en instants élémentaires et ainsi optimiser leur fonctionnement temporel. Toutefois, ils n'utilisent généralement pas de bascules transparentes (figure 7.49).

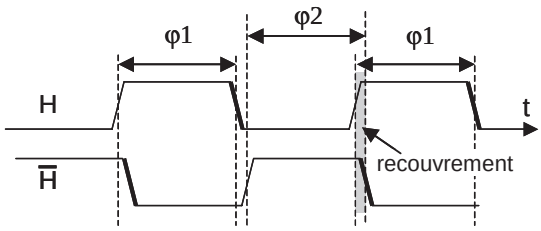


Figure 7.49 Mauvais recouvrement des phases générées par un simple inverseur

BIBLIOGRAPHIE

[CAT66] I. Catt, Time loss through gating of asynchronous logic signal pulses, *IEEE Trans. Electronic. Comput.*, Vol. EC-12 Feb. 1966, pp. 108-111.

[CHA73] T.J. Chaney, C.E. Molnar, *Anomalous behaviour of synchronizer and arbiter circuits*, *IEEE Trans. Comput.*, Vol. C-22 April 1973, pp. 421-422.

[SUZ73] Y. Suzuki, Clocked CMOS Calculator Circuitery, *IEEE journal of Solid State Circuits*, Dec. 1973.

[NOG75] G. Noguez, *Étude d'un modèle temporel des systèmes séquentiels*, Thèse d'État, Institut de programmation, Paris, septembre 1975.

[LAG76] J. Lagasse, M. Courvoisier, J.-P. Richard, *Logique séquentielle*, Dunod Université, 1976.

[ANC86] F. Anceau, *The Architecture of Micro-Processors*, Addison-Wesley, 1986.

[PAR96] H. Partovi *et al*., Flow-through latch and edge-triggered flip-flop hybrid elements, *ISSCC Dig. Tech. Papers*, Feb. 1996.

Chapitre **8**

Éléments de VHDL

8.1 BREF HISTORIQUE DES LANGAGES DE DESCRIPTION DU MATÉRIEL

Les approches modernes de la conception des circuits (logiques) électroniques nécessitent que l'on puisse décrire, d'une manière la plus abstraite possible, la fonctionnalité souhaitée pour ces montages (c'est-à-dire leur spécification). Un long effort, a permis de dégager les notions de base de telles descriptions. D'un point de vue externe, la *syntaxe* de ces formalismes ressemble à celle des langages de programmation (VHDL est un dérivé du langage ADA), mais leur *sémantique* est liée au comportement des montages électroniques.

Après une vingtaine d'années de recherches, de tels formalismes ont été normalisés dans les années 1990 (langages VHDL et Verilog). Ceux-ci sont loin d'être parfaits, ce qui montre que la « sémantique de l'électronique » n'a pas encore été complètement assimilée. L'aspect négatif de cette normalisation a été d'arrêter toutes les recherches dans ce domaine pour se concentrer sur l'assimilation et la mise en œuvre des nouveaux venus.

D'un point de vue industriel, ces langages de description du matériel électronique sont une nécessité. Ils sont très utilisés. Ils permettent de simuler les circuits avant leur réalisation, d'échanger des descriptions de circuits, de constituer des bibliothèques de modules, de préciser la spécification d'un circuit et d'alimenter des outils automatiques de conception. Il existe un marché (dit d'*Intellectual Property*) pour des descriptions « synthétisables » de blocs internes complexes de circuits intégrés. La description préalable du comportement d'un futur circuit devient une étape obligée dans son processus de conception. Cette description constitue souvent un élément

contractuel. Elle est souvent exigée dans les marchés militaires, aérospatiaux, télécom, sécuritaires…

Le passage du fer à souder à la description para-informatique des futurs circuits constitue un véritable bouleversement dans les habitudes des électroniciens qui voient leur métier évoluer profondément et adopter des méthodes de raisonnement inspirées de celles des informaticiens.

L'existence d'un formalisme précis permet aussi de réaliser des traitements formels sur les descriptions des futurs circuits tels que des transformations, des optimisations et surtout des vérifications. Par exemple, les années 1980 et 1990 ont vu apparaître des outils capables de vérifier que les comportements de deux descriptions sont formellement équivalents (outils V-Formal et Chrysalys).

Le langage VHDL résulte d'un effort conjoint des compagnies Intermetrics, IBM et Texas dans les années 1980 sous l'égide du DoD (Ministère de la défense des États-Unis). Le résultat de cet effort a été normalisé en 1987 (norme IEEE 1076). Curieusement, VHDL est surtout utilisé en Europe et c'est l'un de ses concurrents : Verilog qui est le plus utilisé aux États-Unis.

La démarche qui consiste à capturer la « sémantique de l'électronique » est en passe d'être abandonnée au profit de descriptions purement algorithmiques qui n'ont pour objectif que la description du *comportement* des futurs circuits. Ces formalismes sont basés sur des extensions de langages de programmation existant, comme SystemC qui est basé sur C++.

Le langage VHDL est assez complexe car il prétend à un vaste domaine d'applications (peut être qu'un ensemble de langages couplés aurait été préférable). VHDL a apporté des idées très intéressantes mais ses lacunes sont assez gênantes et dénotent une mauvaise prise en compte des spécificités du matériel. Cela a amené les réalisateurs de compilateurs à donner des interprétations diverses à certains points mal définis. Très souvent, les descriptions VHDL sont de bas niveau et n'utilisent qu'un petit sous-ensemble du langage.

L'objectif de ce chapitre n'est pas de donner une description exhaustive du langage VHDL, mais d'en présenter les grandes lignes et de mettre en évidence ses profondes différences avec un langage informatique. Même si la syntaxe d'une description VHDL possède une forme informatique, sa sémantique est électronique et il est très important de conserver le point de vue d'un électronicien lors de l'écriture des descriptions de circuits.

Il est très important de bien distinguer entre le temps d'exécution des algorithmes de simulation et le temps des phénomènes simulés. Ces deux temps n'ont rien à voir. Le temps d'exécution des algorithmes de simulation dépend de la puissance de l'ordinateur utilisé, de la taille de la description et de sa complexité. Le temps simulé dépend des caractéristiques des circuits décrits. L'évolution du temps simulé se fait très rarement en temps réel. La complexité du processus de simulation fait qu'il se déroule souvent (beaucoup) plus lentement que le temps du circuit simulé. La simulation VHDL d'un dispositif matériel ne peut donc pas être utilisée comme une substitution de cet organe dans un système physique. Dans la suite de ce document, nous ne parlerons que du temps simulé.

La description d'un circuit en VHDL ne concerne que le circuit lui-même. Celle-ci ne comporte rien de ce qui constitue l'environnement du circuit :

– les dispositifs qui assurent son excitation (générateurs, autres circuits…) ;
– ceux qui analysent son fonctionnement (sondes, analyseur digital, oscilloscope…).

La fonction de ces organes est généralement assurée, de manière spécifique, par les systèmes informatiques qui permettent la mise en œuvre des descriptions VHDL.

Les systèmes VHDL sont souvent orientés vers la synthèse ou la paramétrisation de circuits (par exemple des FPGA). Il existe aussi quelques systèmes assez complets qui permettent de simuler des descriptions qui utilisent toutes les ressources du langage. Les outils de la première famille sont généralement beaucoup moins onéreux que ceux de la seconde, toutefois leurs possibilités de simulation sont souvent réduites à l'interface externe des circuits ainsi crées ou paramétrés.

8.2 STRUCTURE D'UNE DESCRIPTION VHDL

Une description VHDL se compose de deux parties :

– La description de l'*interface* du circuit (appelée `entity`) avec le monde qui l'utilise (connecteur, brochage, interface…). Celle-ci est constituée par la liste des signaux de cette interface, leur sens, leur nature, etc.
– La description de la réalisation du circuit (appelée `architecture`) qui peut contenir trois formes de descriptions :
  • La description de l'interconnexion de sous-circuits dont l'assemblage constitue le circuit étudié (cette forme de description peut être appelée *structurelle*).
  • La description des fonctions booléennes mises en œuvre (cette forme de description peut être appelée *fonctionnelle*).
  • Des algorithmes dont l'exécution simule le comportement du circuit ou de certains de ses sous-ensembles (cette forme de description peut être appelée *procédurale*).

Ces trois formes peuvent être utilisées conjointement dans la description de l'architecture d'un circuit.

Plusieurs architectures peuvent être associées à une même entité pour montrer l'évolution du processus de conception. Toutefois, cette facilité est souvent illusoire car le « raffinement » des descriptions ne s'effectue pas que dans le détail des fonctions logiques mais aussi par :

– le passage de types énumérés (avant codage) à des types `bit_vector` (après codage) ;
– la transformation du séquencement qui devient plus fin en détaillant les opérations complexes, ainsi que pour la prise en compte de contraintes temps réel.

Ces transformations peuvent remettre en cause profondément l'interface des entités et ainsi limiter leur substitution à des formes plus abstraites.

8.2.1 L'entité

La description d'une entité correspond à celle d'un composant, d'un sous-circuit, d'un module ou d'une carte. Elle comporte nécessairement une *interface* constituée d'un brochage, d'un connecteur… Une entité est constituée d'une entête contenant son nom suivi de la liste ordonnée des différents signaux qui constituent son interface. Chaque connexion est constituée :

– du nom du, ou des, signaux considérés ;
– de leur sens qui peut être :
  • `in` pour des signaux entrants,
  • `out` pour des signaux sortants (non visibles à l'intérieur de l'entité),
  • `buffer` pour les signaux sortants qui peuvent être aussi utilisés à l'intérieur de l'entité,
  • `inout` pour les signaux bidirectionnels ;
– de leur type qui peut être :
  • `bit` pour de simples fils,
  • `bit_vector(<dimension>)` dans le cas d'une nappe de fils. Les dimensions seront données par les deux indices extrémaux. Les fils d'une nappe peuvent être indicés dans le sens croissant (par exemple de 0 à 15 `(0 to 15)`) ou dans le sens décroissant (par exemple de 32 à 1 `(32 downto 1)`),
  • un type spécifique (qui doit être pré-déclaré) ;
– de leur valeur de rappel s'ils ne sont pas connectés (optionnelle) :
  • `:= '1'`, pour un rappel à 1 d'un `bit (pull-up)`,
  • `:= '0'`, pour un rappel à 0 d'un `bit (pull-down)`.

Nous allons utiliser comme exemple la description d'une cellule d'additionneur.

```
Entity Full_Adder is
  port(
       X, Y : in bit;         -- entrées (fils)
       Cin : in bit := '0';   -- entrée avec rappel à 0
       Sum, Cout : out bit); -- sorties (fils)
end Full_Adder;
```

Il faut noter que, dans sa forme actuelle, VHDL est insensible à la case des caractères.

8.2.2 L'architecture

Une description d'architecture correspond à un niveau particulier de description du montage électronique qui réalise l'entité. Elle débute par une entête contenant le nom de cette architecture et celle de l'entité à laquelle elle se réfère.

Format :

```
Architecture <nom> of <nom d'entité> is
  <déclarations>
  begin
```

```
    <description de l'architecture>
end <nom>
```

Les descriptions d'architectures contiennent pratiquement toujours la déclaration de signaux internes. Celles-ci comportent le nom de ces signaux, leur type, éventuellement leurs dimensions et éventuellement leurs valeurs par défaut.

Exemples :

```
signal S1, S2, S3 : bit;
```

Ces signaux, ainsi que les entrées, les bus `inout`, et les sorties `buffer`, sont utilisables dans la description interne de l'architecture du circuit.

8.3 LES DIFFÉRENTS TYPES DE DESCRIPTION

8.3.1 Descriptions structurelles

C'est la forme la plus « naturelle » de description d'un circuit électronique. Elle consiste, tout simplement, à décrire la mise en œuvre et l'interconnexion de ses composants. Elle est directement inspirée des techniques de description des cartes électroniques (liste de câblage). Elle consiste en :

– la liste des signaux internes qui sont de simples fils, ou des nappes de fils, utilisés pour interconnecter les composants ;

Exemples :

```
signal AU : bit;                        -- un simple fil
signal VZ : bit_vector (31 downto 0);   -- une nappe de 32 fils
```

– la liste des *types de composants* utilisés dans la description structurelle avec la liste de leurs connexions telle qu'elle apparaît dans la description de ces entités ;
– la liste des entités utilisées pour réaliser ces types de composants ainsi que des architectures utilisées ;
– la liste des composants interconnectés constituée :
  • du nom de chaque composant,
  • de son type,
  • de la liste de ses connections dans le même ordre que celle de la déclaration de son type (d'autres formes existent).

Ce type de description permet une conception hiérarchique en séparant la conception des composants et celle de l'entité globale

Exemple : Description structurelle de la cellule d'additionneur (figure 8.1).

```
Architecture Structural of Full_Adder is
  signal S1, S2, S3 : bit;
  -- declaration des types de composants
  component XOR_G  -- portes XOR
    port(
     X1, X2 :in bit;
```

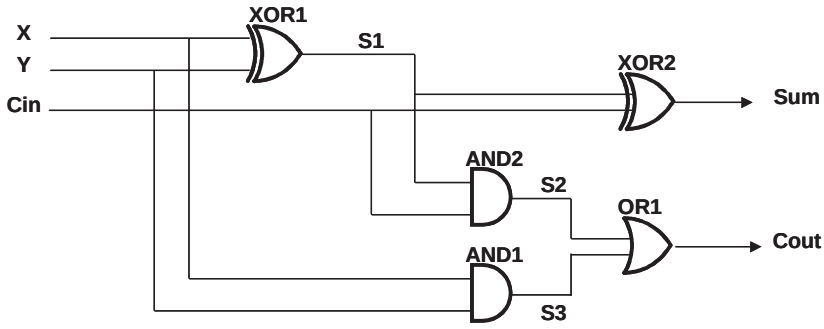


Figure 8.1 Schéma logique de la cellule d'additionneur

```
     XO1 :out bit);
  end component;
component AND_G  -- portes AND
  port(
       A1, A2 :in bit;
       AO1 :out bit);
  end component;
component OR_G   -- portes OR
  port(
       O1, O2 :in bit;
       OO1 :out bit);
  end component;

-- liaison avec les descriptions des types des composants
for all :XOR_G use entity XOR_G port map (x1, x2, xo1);
for all :AND_G use entity AND_G port map (x1, x2, xo1);
for all :OR_G use entity OR_G port map (x1, x2, xo1);

begin

-- description de l'interconnexion des composants
  XOR1 : XOR_G port map (X, Y, S1);      -- une porte XOR
  XOR2 : XOR_G port map (S1, Cin, Sum);  -- une autre porte XOR
  AND1 : AND_G port map (X, Y, S3);      -- une porte AND
  AND2 : AND_G port map (S1, Cin, S2);   -- une autre porte AND
  OR1 : OR_G port map (S2, S3, Cout);    -- une porte OR

end Structural;
```

Cet exemple met en évidence le caractère « verbeux » de VHDL qui allonge les descriptions. Les commentaires sont simplement précédés de `--` (il est déconseillé d'y faire figurer des lettres accentuées).

La spécification des entités composantes peut préciser leur architecture.

Évidement, si l'on souhaite simuler le comportement de cette description de l'additionneur, il faut lui adjoindre celles des différents types de portes qu'elle utilise.

8.3.2 Descriptions fonctionnelles

C'est la forme la plus courante de description VHDL. Elle est souvent appelée *behavioral* dans la littérature. Elle consiste à décrire les équations booléennes qui définissent les différents signaux. Elle contient :

– la liste des signaux internes que nous assimilerons, dans un premier temps, à de simples fils qui véhiculent les valeurs qui leur sont données ;
– la liste des fonctions booléennes qui définissent les valeurs des différents signaux.

Ces fonctions booléennes sont décrites comme des instructions d'un langage informatique évolué. La partie gauche correspond au signal qui doit être défini et connecté. Elle est séparée de la définition de la fonction (l'*expression*) par un signe <= dit de *connexion*. La fonction booléenne est écrite comme une expression utilisant des opérateurs booléens. Les connexions servent à définir la *valeur future* d'un fil ou celle, totale ou partielle, d'une nappe de fils à partir de la *valeur précédente* de ses arguments.

Exemples :

```
BT01 <= VZ;                -- connexion de 32 bits
HC16 <= VZ(17 downto 2);   -- connexion de 16 bits
VZ <= x"A15F";             -- valeur immédiate hexadécimale
```

Bien que cette forme d'écriture ressemble à celle des programmes informatiques, son « comportement » (sa sémantique) est très différent. Elle décrit un réseau de fonctions booléennes interconnectées. Ce qui signifie que :

– Ces *instructions de connexion* sont **toujours valides** (comme l'est le matériel électronique !). Il n'y a pas de notion de point d'exécution comme dans un programme informatique.
– Pour respecter la causalité physique, la valeur des signaux définis par les instructions de connexion apparaît un certain temps après celle des arguments des expressions qui les calculent. Ce temps peut être explicitement spécifié par une clause `after` située à la fin de l'instruction ou être implicite. Dans ce cas, il est égal à une valeur δt considérée comme incommensurable vis-à-vis des temps de fonctionnement du système décrit.
– Il n'y a aucun ordre de préséance entre les instructions de connexion. L'ordre d'écriture n'a aucune influence sur leur comportement. Il est tout à fait possible d'utiliser un signal comme argument d'une instruction écrite avant celle qui définit la valeur de ce signal.
– Les signaux ne doivent être définis qu'une seule fois (sinon il pourrait y avoir des conflits). Toutefois, nous verrons ultérieurement comment résoudre de tels conflits.
– Les instructions de connexion peuvent être arbitrairement complexes ce qui peut amener à la suppression de signaux internes.

La notion d'instructions toujours valides n'est pas directement réalisable lors de la simulation du circuit sur un ordinateur. Elle peut être simulée de différentes manières :

– Soit par une exécution « événementielle ». Dans ce cas, une instruction est exécutée chaque fois que l'un de ses arguments est modifié. Par abus de langage, ce type de description VHDL est souvent appelé *data-flow*.

– Soit par une exécution répétitive de l'ensemble des instructions jusqu'à ce que la valeur des signaux se stabilise.

Les retards explicites ou δt assurent le respect de la causalité dans le circuit décrit en faisant en sorte que les effets se produisent toujours après les causes.

Ce fonctionnement *asynchrone* est la base du mécanisme de simulation des descriptions VHDL. Le caractère incommensurable des retards δt fait que ceux-ci peuvent s'accumuler sans être visibles. Cela permet de dire que VHDL utilise deux axes de temps (le temps simulé et les δt). L'accumulation des δt ne correspond qu'à une approximation des retards technologiques cumulés. Elle peut être différente des retards réels et ne doit donc pas être fonctionnellement exploitée. Par exemple :

```
C <= A and B;
```

Si les valeurs des signaux `A` et `B` sont respectivement retardées de δt et de 2δt par rapport à une référence temporelle alors, la valeur de `C` peut subir une altération (aléa) entre 2δt et 3δt. Dans un montage réel, les retards des portes seront différents et l'aléa pourra ne pas exister ou apparaître à un autre instant.

Exemple : Description fonctionnelle de la cellule d'additionneur :

```
architecture Dataflow of Full_Adder is
  signal S1, S2 : bit;
  begin

    Cout <= S2 or (X and Y) after 2ns;
    S2 <= S1 and Cin after 2ns;
    Sum <= S1 xor Cin after 3ns;
    S1 <= X xor Y after 3ns;

end Dataflow;
```

8.3.3 Descriptions procédurales

C'est une forme très courante de description VHDL. C'est la forme dans laquelle les compilateurs transforment toutes les autres formes de description pour pouvoir les simuler. Une telle description consiste à décrire un ou des programmes dont l'exécution simule le comportement du circuit. Elle consiste en :

– la liste des signaux internes ;
– la liste des programmes appelés `process` qui simulent tout ou partie du circuit. Ces programmes sont constitués de :
  • la déclaration des *variables* qu'ils utilisent, qui ne sont que des intermédiaires de calcul,
  • les instructions de simulation qui peuvent avoir des signaux et des variables comme arguments. Celles ci peuvent :
    . soit définir la valeur de signaux (instructions de connexions),
    . soit définir la valeur de variables (instructions procédurales).

La structure d'un processus n'a généralement rien à voir avec l'organisation du matériel dont il simule le fonctionnement.

Ces programmes s'exécutent normalement, c'est-à-dire de manière séquentielle. Leur temps d'exécution est supposé nul vis-à-vis du temps de simulation. Les conditions de démarrage d'un programme permettent de l'assimiler globalement à une super-instruction de connexion d'une description fonctionnelle. Ce démarrage s'effectue :

– soit parce qu'un signal, spécifié dans une *liste de sensibilité*, varie ;
– soit parce que le programme était mis en attente par une instruction `wait` et que cette attente est échue.

Pendant l'exécution du `process`, la valeur des signaux qui ont provoqué son déclenchement est celle **postérieure** à la transition.

La valeur des signaux modifiés par un `process` ne sera réellement actualisée que δt après l'instant de son exécution.

La réutilisation d'un signal qui vient d'être positionné dans un `process` portera sur l'ancienne valeur de ce signal et non pas sur celle qui résulte de l'instruction qui l'a positionné (erreur fréquente !).

Exemple :

```
signal A : bit := '0';
process (A)
  begin
    A<= '1';
    if A = '1' then .....
end process;
```

Au premier tour du `process`, le test trouve une valeur de `A` égale à `0`, car son affectation à `1` ne sera effective que δt après la fin de ce `process`.

La sémantique des instructions de connexion invoquées dans un `process` est différente de celle des mêmes instructions situées dans les parties fonctionnelles de la description. En effet, dans un `process` les connexions ne durent qu'un moment très bref, à l'instant d'exécution du `process`. Les signaux de destination de ces instructions sont donc isolés pendant tout le reste du temps et ils se comportent comme des bascules.

L'évolution d'un `process` peut être suspendue par des instructions `wait`. Celles-ci peuvent attendre :

– l'occurrence d'une condition (`wait until <condition>;`) ;
– la variation d'un signal d'une liste de sensibilité (`wait on <liste de sensibilité>;`) ;
– une durée spécifiée (`wait for <durée>;`).

La durée (en temps simulé) de l'exécution du `process` entre deux instructions `wait` est nulle. Toutefois les nouvelles valeurs des signaux ne sont effectives que δt, ou un temps spécifié par une clause `after`, après cette exécution.

L'utilisation d'une liste de sensibilité est exclusive de celle de l'instruction `wait`. L'utilisation d'une liste de sensibilité est équivalent à l'utilisation d'une instruction `wait` sur cette même liste au début de la description.

Les détails de l'exécution d'un `process`, ainsi que la valeur de ses variables ne sont généralement pas visibles *via* l'interface d'un simulateur car ils n'ont aucune correspondance physique.

Exemple : Description procédurale de la cellule d'additionneur

```
architecture Computationnal of Full_Adder is
  begin
  process (X, Y, Cin)      -- liste de sensibilite'
    variables s1, s2, s3 : bit;
    begin
      s1 := X xor Y;       -- calcul de la variable s1
      s2 := s1 and Cin;
      s3 := X and Y;
      Sum <= s1 xor Cin;   -- de'finition des signaux de sortie
      Cout <= s2 or s3;
  end process;
end Computationnal;
```

8.3.4 Descriptions mixtes

Les trois types de descriptions (structurelles, fonctionnelles, procédurales) peuvent coexister dans une même architecture. La définition sémantique de leurs actions sur les signaux les rend compatibles entre elles.

8.4 TYPES DES SIGNAUX ET DES VARIABLES

8.4.1 Types standard et dérivés

Les types des signaux et des variables peuvent être :

– des types physiques simples (fils ou nappes de fils).
Exemples :

```
signal Z : bit;                          -- un simple fil
signal A : bit_vector (0 to 15);         -- une nappe de 16 fils
variable B : bit_vector (32 downto 1);   -- une nappe de 32 fils
signal C : bit := '0';                -- une valeur de repos égale à 0
```

– des types abstraits pour des signaux dont on ne souhaite pas encore préciser le codage :
  • Type entier.
  Exemple :

```
variable J2 : integer;
```

  • Type booléen (à ne pas confondre avec le type `bit` !). C'est le type des conditions. Il est sans correspondance matérielle. Ses valeurs sont `true` et `false`.
  • Type énuméré (celui-ci nécessite une déclaration préalable de type).
  Exemple :

```
type Com is (Marche, Arret, Attente);
signal Commande : Com;
```

L'utilisation de types abstrait est très importante. Elle permet de retarder au maximum les choix techniques dans le processus de conception (manuel ou automatique).

– des types structurés qui représentent des nappes de fils composées de sous-nappes (ceux-ci nécessitent une déclaration préalable de type)

Exemple :

```
type Couleurs is (Rouge, Vert, Bleu);
type Pixel is record
  lumin : bit_vector (7 downto 0);
  af : integer;
  cd : Couleur;
end record;

variable I4, I5 : Pixel;

A := I4.af;
I4.cd := Vert;
I5 := I4;
```

– des structures bi-dimensionnelles (par exemple des mémoires) (qui nécessitent une déclaration préalable de type)

Exemple :

```
type mem is array (0 to 512) of bit;
signal memoire :mem;
```

Il faut remarquer que ces structures ne peuvent être adressées que par des variables entières, ce qui réduit leur correspondance physique au seul cas de mémoires munies de décodeurs (figure 8.2).

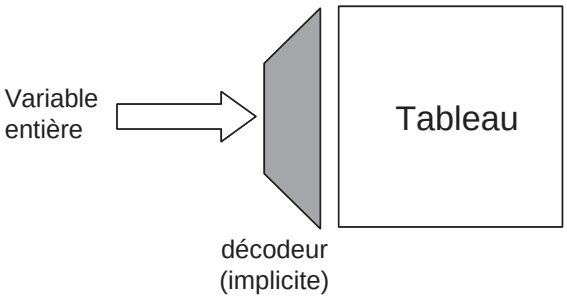


Figure 8.2 Adressage implicite d'un tableau

En particulier, la description de la sélection directe des lignes d'une matrice par une nappe de fils suppose l'introduction d'un matériel d'encodage-décodage parfaitement inutile.

Il est également possible de déclarer des sous-types :

Exemple :

```
subtype word is bit_vector (15 downto 0);
type mem is array (0 to 255) of word;

type octet is range 0 to 255;
```

et de donner un nom particulier à un sous-ensemble (contigu) de fils d'une nappe.

Exemple (figure 8.3) :

```
alias code_op : bit_vector (5 downto 0) is IR (15 downto 10);
```

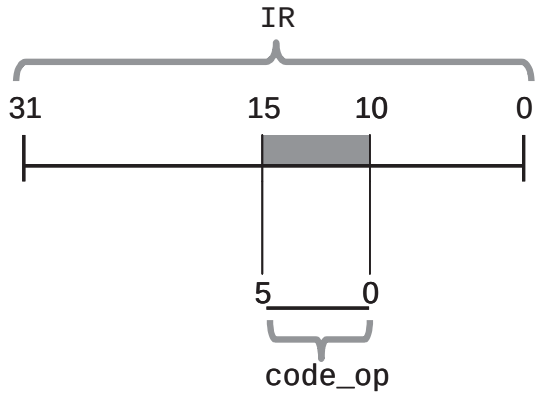


Figure 8.3 Redéfinition d'un sous-champ d'un bit_vector

8.4.2 Types IEEE

La société savante IEEE à développé de nouveaux types pour les variables et de nouvelles fonctions pour les manipuler. Ces nouveaux types permettent de mieux gérer et détecter les conflits électriques. Ce module (appelé *library*) contient :

– Un `package` appelé `std_logic_1164` qui définit un nouveau type ainsi que des fonctions de conversion et de détection des fronts. Les opérateurs standard sont automatiquement étendus à la manipulation de ces nouveaux types.

– Le type `std_ulogic` comporte 9 valeurs :

0 et **1** issus d'une sortie « forte »

L et **H** issus d'une sortie « faible » (positionnement à 0 et à 1 *via* une résistance)

Z l'état de haute impédance (sortie isolée)

X l'état de conflit entre deux sorties « fortes »

W l'état de conflit entre deux sorties « faibles »

- un état indifférent (pour optimiser la logique)

– Des packages appelés `numeric_bit` et `numeric_std` qui définissent des opérateurs arithmétiques qui opèrent sur les `bit_vector` et sur les `std_logic_vector` interprétés comme des nombres entiers.

8.5 EXPRESSIONS

Les expressions interviennent dans les instructions des descriptions fonctionnelles et dans l'écriture des `process` pour définir les nouvelles valeurs des signaux et des variables.

Il est possible d'utiliser des constantes dans l'écriture de ces expressions. Celles-ci correspondent à des connexions directes aux niveaux logiques 1 ou 0.

– pour les `bit` : '0' et '1'
– pour les `bit_vector` :

Exemples :

`"001100"` (constante binaire)
`X"AD01"` (constante hexadécimale)

Il est aussi possible de déclarer des constantes symboliques.

Exemple :

```
constant <nom> : <type> := <valeur>;
```

8.5.1 Attributs des signaux

Ils correspondent à l'extraction de propriétés des signaux à l'aide de suffixes. Leurs valeurs dépendent du type du signal sur lequel porte le suffixe. Ils sont relativement nombreux et ils peuvent être enrichis par l'utilisateur.

– résultat booléen sur des signaux :
`'event` (retourne `true` si la valeur du signal varie),
`'stable` (retourne `true` si la valeur du signal n'a pas varié),
`'quiet` (retourne `true` si le signal n'a aucune source (état isolé)) ;
– résultat entier sur les tableaux, les vecteurs :
`'left` `'right` (bornes gauches et droites d'un tableau, d'un vecteur),
`'length` (taille d'un tableau, d'un vecteur),
`'range` (couple des bornes d'un tableau, d'un vecteur).

Exemple d'utilisation :

```
A <= Z when CLK'event;
```

`Z` est (brièvement) connecté à `A` lors de chaque variation de `CLK` (pratiquement, cette connexion dure δt).

8.5.2 Opérateurs

Les opérateurs qui sont utilisables dans les expressions sont :

– opérateurs booléens (sur les `bit` et `bit_vector`)

  `not`, `and`, `or`, `xor`, `nor`, `nand`

– opérateurs arithmétiques (sur les `bit_vector` de même taille) (*via* une bibliothèque particulière). Les poids forts sont ceux déclarés à « gauche ».

  `+`, `-`, `-` (unaire), `*`, `/`, `mod` (modulo), `rem` (reste division)

– opérateurs de décalage

  `sll` (gauche), `srl` (droit), `sra` (arithmétique), `rol` (rotation gauche), `ror` (rotation droite)

– Concaténation de deux `bit_vector`

  `A&B`

8.5.3 Temps de transit

Le temps de transit d'une expression, sera indiqué par une clause `after`. Il représente celui du circuit combinatoire représenté par l'expression.

```
RES <= A xor B after 10ns;
```

Le temps de transit sera donné en s, ms, us, ns, ps. Il sera de δt si rien n'est mentionné.

L'existence d'un temps de transit « obligatoire » assure la *causalité* de la description. Il interdit la description de certains montages « irréalisables ».

La clause after est par défaut *inertielle*, c'est-à-dire qu'elle ne transmet que des variations de valeur dont la durée est supérieure au retard (figure 8.4).

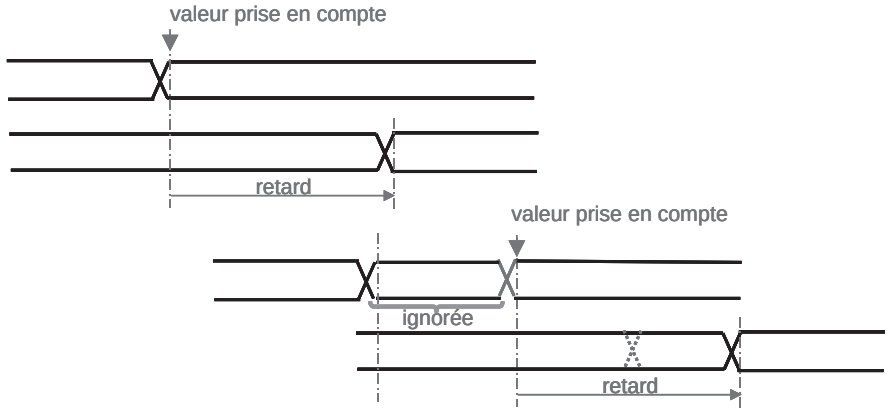


Figure 8.4 Fonctionnement de la clause after inertielle

Il faut mentionner l'existence d'une autre forme de la clause after appelée `transport` qui correspond à l'utilisation d'une ligne à retard.

```
S1 <= transport X xor Y after 3ns;
```

Dans ce cas, toutes les variation du signal initial seront reproduites après le délais indiqué.

8.6 INSTRUCTIONS DE CONNEXION CONDITIONNELLE

Il est possible de rendre conditionnelle une instruction de connexion, ce qui correspond à l'usage d'un multiplexeur ou de portes 3 états.

8.6.1 Multiplexeurs

Une condition permet de choisir entre deux expressions pour définir la valeur d'un signal (figures 8.5 et 8.6).

Exemple :

```
U <= K when A=B else
X when A>=B else
Z
```

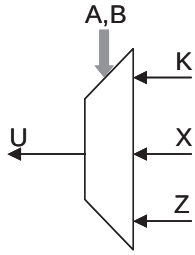


Figure 8.5 Le multiplexeur correspondant

Une seconde forme permet de décrire des multiplexeurs plus complexes.

Exemple :

```
with Z select
 D <= <expression1> when x"00",
      <expression2> when x"01" | x"02",
      <expression3> when others;
```

La variable z doit être énumérable et non structurée (`record`).

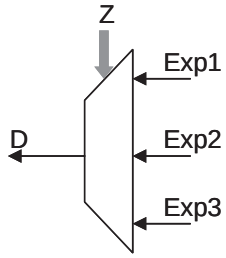


Figure 8.6 Le multiplexeur correspondant

8.6.2 Logique 3 états et latches

Une forme conditionnelle peut représenter une porte 3 états si toutes les alternatives ne sont pas utilisées. Dans ce dernier cas, Le comportement de VHDL fait que la connexion n'est pas réalisée et que le signal garde sa valeur précédente jusqu'a ce qu'il soit de nouveau connecté à une expression. Cela ne correspond pas complètement au comportement habituel du matériel, puisqu'un fil isolé ne garde sa valeur par rétention capacitive qu'un bref instant qui dépend de l'importance du courant de fuite.

Exemple :

```
Z <= A when (Clk='1');
Z <= A when (Clk='1' and Clk'event);
```

Les signaux A et Z ne sont connectés que si la condition est vraie.

Cette particularité de VHDL permet :

– de réaliser des signaux à sources multiples (bus). Ces sources doivent :
  • soit être mutuellement exclusives,
  • soit un éventuel conflit doit être géré par une fonction spéciale appelée fonction de *résolution des conflits ;*
– aussi de réaliser des latches dynamiques implicites. Cette confusion entre les latches et les signaux est regrettable. Elle n'est levée que pour les types de signaux dit *gardés* dont l'utilisation est plus complexe.

Remarque : Il est possible de réaliser des latches statiques sans utiliser la rétention dynamique :

```
D <= E when Ch else D;
```

Il faut toutefois faire très attention aux connexions conditionnées par des événements (`'event`). La connexion réalisée **n'est pas instantanée**. Elle peut permettre des **rebouclages asynchrones**.

Exemple :

```
A <= A(0)& A(15 downto 1) when Clk'event and Clk = '1';
```

Le signal `A` risque d'être décalé de **plusieurs positions** à l'occurrence de chaque transition montante de `Clk`.

8.6.3 Blocs

Les blocs VHDL correspondent à plusieurs usages :

– la déclaration de composants locaux constitués d'instructions fonctionnelles ou de processus. Il s'agit alors de structurer la description :
  • l'« interface » du bloc peut être explicitée par une liste de ports, comme celle d'une entité,
  • le bloc peut utiliser des signaux locaux ;
– la possibilité de conditionner collectivement un ensemble d'instructions fonctionnelles par la même condition. Les instructions concernées seront alors « gardées » : Par exemple :

```
Basc : block(Clk'event and Clk='1')
          begin
              BA <= guarded Tr15;
              BB <= guarded Mu37;
          end block;
```

– la déclaration de signaux `register` et `bus` qui doivent obligatoirement être gardés. Ces signaux doivent être munis de fonction de résolution des conflits, spécifiant leur valeur lorsque plusieurs sources sont simultanément actives, ou toutes déconnectées. Ce type de description est orienté vers la logique monophasée, alors que le langage permet de décrire des latch par le simple détournement des signaux « normaux ». Ces formes sont utilisées par certains outils de synthèse qui reconnaissent ainsi les registres et les bus.

8.7 COMPORTEMENT TEMPOREL DES DESCRIPTIONS

La sémantique de VHDL peut réserver quelques surprises qui peuvent poser des problèmes difficiles à localiser.

8.7.1 Intervalle temporel de définition des signaux

Un signal externe est défini sur une succession d'intervalles $[t_i, t_{i+1}[$ sur lesquels il porte ses valeurs. Le test d'un signal externe porte donc sur ses valeurs après ses transitions (figure 8.7).

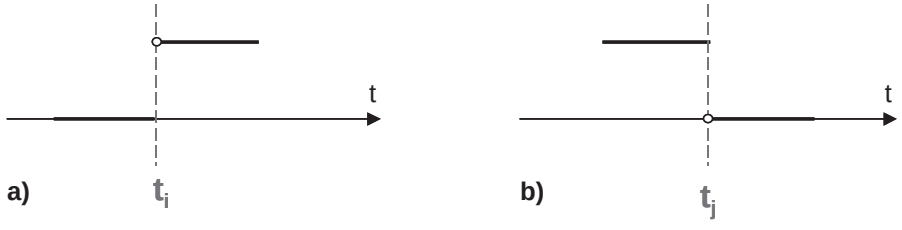


Figure 8.7 a) Le signal vaut 1 en t_i. b) Le signal vaut 0 en t_j

La nouvelle valeur d'un signal interne, défini à l'instant t, apparaîtra à l'instant $t+\delta t$. Ceci signifie que les signaux internes sont définis sur des intervalles $]t_i, t_{i+1}]$. Il existe donc un décalage entre la définition des signaux externes et internes. Ceci a vraisemblablement été fait pour éviter la gymnastique intellectuelle qui consisterait à devoir considérer les valeurs d'un signal externe avant ses transitions.

Il faut toutefois remarquer que le paradigme data-flow, utilisé pour le déclenchement des `process`, fait que ceux-ci démarrent sur la transition d'un signal. Pendant leur exécution, la valeur de ce signal sera donc celle après la transition, ce qui est cohérent avec le traitement des signaux externes (figure 8.8).

Il est utile de rappeler que dans toute instruction de connexion VHDL, toutes les valeurs de ses arguments sont prises en t_i, tandis que le résultat est fourni en $t_i+\delta t$.

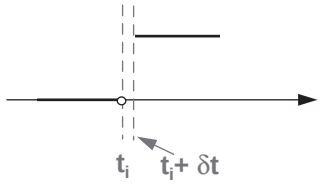


Figure 8.8 Décalage entre les arguments et le résultat d'une instruction de connexion

8.7.2 Cas des dispositifs à temps de réponse très long

La description de dispositifs à temps de réponse très long (PLA, ROM, circuits combinatoires complexes) nécessite l'utilisation de la clause `after`. Toutefois, son utilisation se heurte à plusieurs difficultés :

– Ce genre de dispositif fait appel à des valeurs de retard explicites qui ne sont pas toujours connus lors des premières étapes de la conception d'un système. Il serait souhaitable de pouvoir seulement mentionner qu'un retard est « très long » (figure 8.9).

Figure 8.9 Différence entre les instants de production des signaux simulés et ceux issus d'un circuit combinatoire à long temps de réponse

La meilleure solution pour contourner ce problème consiste à utiliser une valeur symbolique comme unité de temps de fonctionnement du système :

```
constant PERIODE : time := 1us;
```
(prenons arbitrairement 1 microseconde)
```
......
A <= f(Z) after 1.5*PERIODE;
```
(le résultat de f n'apparaît qu'au bout de 1,5 périodes)

Cette valeur de la période devra être impérativement réutilisée lors de la simulation.

– Il faut toutefois remarquer que VDHL calculera la valeur de sortie au début du cycle et transportera le résultat au moment où il doit être utilisé pour simuler le temps d'établissement. Cela ne donne aucune garantie sur le maintient des opérandes pendant toute cette période. Une forme de retard stipulant que les opérandes doivent avoir été continûment présents depuis le déclenchement du calcul jusqu'à la sortie du résultat aurait été utile.
– La valeur de sortie du dispositif est souvent considérée comme indéfinie pendant la durée d'établissement. La simulation de ces dispositifs devrait aussi tenir compte de ce phénomène.

8.8 INSTRUCTIONS SPÉCIFIQUES AUX PROCESSUS

Les processus sont programmés à l'aide d'un langage algorithmique qui correspond à un sous-ensemble d'ADA. La programmation des processus n'a rien à voir avec la structure matérielle qu'ils représentent.

8.8.1 Instructions conditionnelles

Première forme :

```
if A = B then <instruction>; end if;
```

Deuxième forme :

```
if not (C = '1') then
     <instruction1>;
  else
     <instruction2>;
end if;
```

Troisième forme :

```
if Z >= W then
     <instruction1>;
  elsif A /= E then
     <instruction2>;
  else
     <instruction3>;
end if;
```

Les opérateurs de relation sont : =, /=, >=, <=.

8.8.2 Instruction de choix

Exemple :

```
case CX is
   when "001100" => <instruction1>;
   when "000000" | "110000" => NULL; --OU de deux conditions
   when others => <cas résiduels>;
end case;
```

La variable `CX` doit être énumérable et non structurée (`record`).

Remarques :

Comme un `process` s'exécute en temps nul, le test des signaux, dont la variation l'a fait démarrer, concerne les valeurs de ces signaux *après* cette transition.

Le langage d'écriture des `process` se veut puriste et n'admet pas de `goto` ce qui peut obliger à une écriture plus lourde des algorithmes.

L'absence d'instructions est interdite en VHDL. Il faut utiliser l'instruction `null`.

8.8.3 Instructions de bouclage

– Boucles `for` :

Exemple :

```
for i in 1 to K'length loop
  <instructions>;
end loop;
```

L'indice n'a pas à être déclaré :

– Boucles `while` :

Exemple :

```
while <condition> loop
  <instructions>;
end loop;
```

– Il est possible de programmer un bouclage infini :

```
loop
  <instructions>;
end loop;
```

d'où l'on sort par :

`exit [when <condition>];` (sortie de boucle)
`next [when <condition>];` (saut à l'itération suivante)

Les boucles situées dans les `process` ne servent pas à décrire des processus séquentiels mais sont utilisées pour parcourir des structures spatiales (`bit_vector`, tableaux.

8.8.4 Mise en attente d'un processus

Comme nous l'avons déjà mentionné, la mise en attente d'un `process` se fait à l'aide d'instructions `wait`. L'usage de celles-ci est exclusif de l'utilisation d'une liste de sensibilité. Les différentes formes de l'instruction `wait` sont :

```
wait [on <liste de sensibilité>]
     [until <condition>]
     [for <durée];
```

L'instruction `wait` suspend l'exécution du processus jusqu'à ce qu'un signal change de valeur, ou que la condition soit vraie, ou pour la durée indiquée.

Exemple de la génération d'une horloge polyphasée :

```
signal phase1, phase2 : bit;
constant Non_recouv : time := 3ps;
constant Duree_phase : time := 100ps;

process
begin
  phase1 <= '1';
  wait for Duree_phase;
  phase1 <= '0';
  wait for Non_recouv;
  phase2 <= '1';
  wait for Duree_phase;
  phase2 <= '0';
  wait for Non_recouv;
end process;
```

Exemple : La génération d'un signal d'horloge peut se faire de différentes manières.

Par une instruction fonctionnelle :

```
Clk <= not Clk after Demi_periode;
```

Par un `process` :

```
process
  begin
  Clk <= not Clk after Demi_periode;
  wait on Clk;
end process;
```

Ou encore :

```
process
  begin
  Clk <= not Clk;
  wait for Demi_periode;
end process;
```

8.8.5 « Filtrage » des événements lors de l'exécution d'un `process`

La sélection des événements pour lesquels on désire effectuer certaines opérations s'effectue à l'aide d'instructions `if` qui testent la valeur ou les attributs des signaux de déclenchement du `process`.

Exemple :

```
process (RST, CLK)
    -- de'marrage sur toutes les variations de RST et CLK
  begin
    ..............
    -- filtrage des fronts avants de CLK
    if (CLK'event and CLK='1') then
       A <= B;
    end if;
    ..............
  end process;
```

8.8.6 Choix du front de déclenchement d'un `process`

- Déclenchement sur un front montant

Exemple :

```
process(Clk)  -- De'clenchement a' chaque transition
begin
  if Clk='1' then  -- Filtrage des monte'es à 1
    <corps du process>
  end if
```

Cela peut aussi s'écrire :

```
process
begin
  wait until clk'event and Clk='1';  -- Attente monte'e à 1
  <corps du process>
```

- Déclenchement sur un front descendant

Exemple :

```
process(Clk)  -- De'clenchement a' chaque transition
begin
  if Clk='0' then  -- Filtrage descente à 0
    <corps du process>
```

Cela peut aussi s'écrire :

```
process
begin
  wait until clk'event and Clk='0';  -- Attente descente à 0
  <corps du process>
```

Comme l'exécution du `process` se déroule juste après la transition de Clk, la valeur de ce signal pendant l'exécution est celle après la transition.

8.9 DESCRIPTIONS « COMPORTEMENTALES »

Il s'agit d'un type de description dans lequel on souhaite décrire le comportement temporel d'un organe complexe sous la forme d'une suite d'actions rythmées par une horloge. Pour cela, on suppose un séquencement réel entre des instructions qui manipulent des signaux. Malheureusement, VHDL ne permet pas l'écriture de ce type de description. La solution habituellement utilisée consiste à utiliser un `process` dans lequel on précise le séquencement à l'aide d'instructions `wait` ce qui est très lourd ! Une autre possibilité consiste à détourner un simple `process` dont on utilisera le séquencement « descriptif ». Ce `process` sera déclenché par une horloge « macroscopique » sans réalité physique. Dans ce cas, la séquence se trouvera donc être exprimée en un temps simulé nul !

Pour écrire une telle description, il faudra bien prendre soin de recopier les signaux dans des variables au début de l'exécution pour pouvoir les manipuler librement, puis de les mettre à jour à la fin. Les variables complexes, telles que des mémoires, ne seront pas recopiées si leur valeur n'est modifiée qu'au plus une fois pendant chaque exécution. Nous verrons une telle description dans le chapitre consacré à la conception algorithmique.

8.10 FONCTIONS

Il est possible d'utiliser des fonctions (déclarées ou prises dans une bibliothèque) dans les expressions. Leur exécution « simule » le fonctionnement d'un réseau combinatoire. L'exécution d'une fonction se produit à chaque modification de l'un de ses arguments (paradigme data-flow) et l'affectation du résultat se produit donc δt après l'exécution de la fonction (ou plus tard si l'on utilise une clause `after`).

Exemple :

Transformation `bit_vector` -> entier (pour indicer les tableaux) :

```
A <= memoire(to_natural(Adresse));  (bibli spécifique)
```

Test de nullité d'un `bit_vector` :

```
signal Z : bit;
Z <= Is_zero(Adresse);  (fonction « locale »)
```

8.10.1 Programmation des fonctions

Les fonctions sont des programmes séquentiels. Elles utilisent les mêmes instructions que les processus. Par rapport à ceux-ci, elles doivent présenter des caractères de généralité. Par exemple, la taille de leurs arguments ne doit pas être spécifiée et la fonction doit donc être capable de s'adapter à toutes les tailles possibles.

Comme pour un `process`, la programmation d'une fonction n'a rien à voir avec la structure matérielle qu'elle représente.

Exemple d'une fonction qui retourne une valeur de type `bit` suivant qu'un `bit_vector` est nul ou non :

```
function Is_zero ( a : bit_vector ) return bit is
  begin
      for i in a'range loop
        if a(i)='1' then
          return '0';
        end if;
      end loop;
  return '1';
end Is_zero;
```

Cette fonction admet n'importe quel `bit_vector` comme argument. La programmation est indépendante de sa taille.

8.10.2 Fonctions de résolution de conflits

Celles-ci sont généralement utilisées pour résoudre les cas où plusieurs sources veulent définir simultanément la valeur d'un signal. Elles sont utilisées pour représenter des bus réalisant des OU ou des NI de connexion. Elles peuvent aussi être utilisées pour donner une valeur à ces signaux lorsque aucune source ne les excite (cas des registres).

Exemple (cas d'un bus réalisant un OU de connexion) :

```
function wired_or (input : bit_vector) return bit is
begin
    for i in input'range loop
       if input(i)='1' then return '1'; end if;
    end loop;
    return '0';
end wired_or;
```

L'argument de la fonction est un `bit_vector` auxiliaire, généré automatiquement pour représenter l'état de toutes les sources du bus.

Cette fonction est générale. Elle n'est pas spécifique à un signal donné. Son utilisation doit être précisée lors de la déclaration d'un bus.

Exemple :

```
signal B : wired_or bit;
```

8.11 PACKAGES

Ceux-ci sont utilisés pour déclarer des types et des fonctions globales. Ils permettent, entre autres, d'utiliser des types déclarés dans les connexions d'une `entity`.

```
package <nom> is
  <déclaration de types globaux>
```

```
  <entête de fonctions globales>
end <nom>;
package body <nom> is
  <corps des fonctions>
end <nom>;
```

Le « corps » d'un `package` est inutile si celui-ci ne contient que des déclarations de type.

8.11.1 Mise en œuvre des packages

L'utilisation des packages est propre à chaque `entity`. Les instructions `library` et `use` doivent être répétées avant chaque `entity`.

Les packages peuvent être :

– soit dans le même fichier que la description :

```
use work.<nom du package>.all;
```

(`.all` pour utiliser la totalité du contenu du package)

– soit dans une bibliothèque :

```
library <nom de bibliothèque>;
use <nom de bibliothèque>.<nom du package>.all;
```

Exemple : Déclaration de constante et de types dans un `package` :

```
package N_types is
  constant PERIODE : time := 1us;           -- période arbitraire
  subtype word is bit_vector(31 downto 0);  -- mots de 32 bits
  type COM is (Marche, Arret, Veille);      -- commande non codée
end N_types;
-- utilisation des types globaux dans l'entité suivante
use work.N_type.all;
entity My_box is
  port(
    IN1 : in word;      -- entre'e de 32 bits
    Fonction : in COM;  -- entre'e non encore codée
    .........
architecture AR102 of My_box is
  signal Z : word;
  ...
  U <= Z xor IN1 after 1.5*PERIODE;
```

8.12 DUPLICATION ET PARAMÉTRISATION DU MATÉRIEL

8.12.1 Structures vectorielles et matricielles

La description d'un dispositif matériel constitué par la répétition linéaire ou matricielle d'un même motif peut être décrite économiquement à l'aide d'un petit programme de génération utilisant une instruction particulière `generate`. Le programme de géné-

ration est constitué d'une ou de plusieurs boucles `for generate` imbriquées. Des instructions `if generate` servent à isoler les irrégularités de la duplication (par exemple les premiers et les derniers éléments). Chaque instruction `for generate` ou `if generate` doit être étiquetée et terminée par un `end generate` suivit du rappel de cette étiquette.

Une boucle `for generate` duplique les composants qu'elle invoque. Elle donne le même nom à toutes les copies.

Exemple :

```
signal lien : bit_vector(1 to taille-1);
......
module : for i in 1 to taille generate
cond : if i<taille generate
DecadeB : decade port map(, , ,lien(i));
          end generate cond;
fin : if i=taille generate
 DecadeF :decade port map(, , ,sortie);
                  end generate fin;
        end generate module;
```

Malheureusement, cette construction automatique est très limitée. Elle ne peut agir à l'intérieur même des instructions ni dans la définition des constantes, ni dans l'écriture des noms.

8.12.2 Paramétrisation du matériel

Il est possible de décrire en VHDL du matériel générique qui peut être adaptée à chaque cas d'utilisation (par exemple la dimension d'un registre). Cela peut être obtenu par la clause `generic` qui permet de décrire la liste des paramètres utilisés pour décrire une telle entité.

Exemple : FIFO paramétrable.

```
entity FIFO is
generic( Larg : integer := 8;    -- valeur par défaut
         Long : integer);
port( Entree : in bit_vector(Larg -1 downto 0);
....
```

Lors de l'instantiation de cette entité, La valeur des paramètres sera donnée par une clause `generic`.

Exemple : FIFO de 5 mots de 32 bits.

```
FIFO1 : FIFO
             generic( Larg => 32, Long =>5)
             port map (IN1,...
```

Généralement, la généricité et la génération algorithmique de matériel vont de pair. Toutefois, les limitations de ces mécanismes en réduisent beaucoup l'intérêt. Un véritable mécanisme de macro-génération syntaxique aurait été préférable.

8.13 MATÉRIEL COMPLÉMENTAIRE

Pour permettre la simulation de certaines descriptions, il est quelquefois nécessaire de leur adjoindre des signaux et des instructions qui ne correspondent pas au matériel souhaité. Cela peut avoir deux origines :

– Le contournement d'une particularité sémantique gênante de VHDL. Ce cas doit être, si possible, évité. Par exemple, le fait que les tableaux ne puissent être indicés que par des variables entières et non par des `bits_vectors`.
– L'amélioration de l'accès au matériel simulé. Ces extensions correspondent alors au matériel d'essai qui est souvent utilisé lors du test d'un montage électronique. Il ne faut pas oublier de l'enlever lorsque la description est au point.

8.13.1 Environnement de simulation

De même que le test d'un montage électronique fait appel à des appareils de laboratoire (générateurs de signaux, oscilloscopes, analyseurs logiques, la simulation d'une `entity` VDHL fait appel à des fonctions spécifiques du simulateur pour :

– définir la forme des signaux d'excitation ;
– visualiser les signaux de sortie et internes ;

Les variables des `process` n'ayant aucune réalité physique ne peuvent pas être visualisées par les simulateurs.

BIBLIOGRAPHIE

[1] R. Airiau, J.-M. Bergé, V. Olive, J. Rouillard, *VHDL*, Presses Polytechniques et Universitaires Romandes.

Chapitre **9**

Conception algorithmique des circuits VLSI complexes

9.1 INTRODUCTION

La technique de conception des circuits complexes qui va être présentée dans ce chapitre décrit une méthode, relativement performante, pour passer de la description du comportement espéré pour un futur circuit (description dite *comportementale*) à la description de sa structure (description dite *structurale*). Cette méthode est directement issue de celle utilisée pour la conception des microprocesseurs simples. Elle a été mise au point dans les années 1975-1985 et consiste en la conception de circuits *à la demande* sous la forme de microprocesseurs spécialisés. Les progrès et le niveau d'optimisation atteints lors de la conception des microprocesseurs a incité à réutiliser ces techniques pour la conception d'autres types de circuits. Nous présenterons cette démarche dans le contexte de la conception manuelle en utilisant une circuiterie polyphasée, mais la description structurelle résultante peut être facilement orientée vers la circuiterie monophasée, d'où vers des outils de synthèse automatiques pour des circuits intégrés ou vers des circuits programmables (FPGA).

La conception d'un microprocesseur démarre à partir de la description de son comportement (son algorithme d'interprétation) (figure 9.1). De même, la conception des autres types de circuits débute aussi par la description de leur comportement.

Plusieurs outils ont été réalisés pour automatiser ce processus de conception (*compilateurs de silicium*). Nous pouvons citer :

– l'outil AMICAL développé par le laboratoire TIMA (INPG), maintenant intégré dans les outils de TNI Valiosys ;

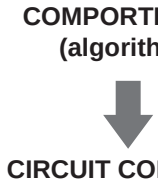


Figure 9.1

– le système ALLIANCE développé par le laboratoire ASIM (Université Paris 6) ;
– l'outil CATHEDRAL développé à l'IMEC (Université de Leuven (Belgique)), spécialisé pour la conception des circuits de traitement du signal, devenu N2C de Coware.

Ces outils produisent des circuits raisonnablement optimisés. Ils opèrent en adaptant un motif générique au problème particulier. Ils sont considérés comme plus puissants que les outils qui partent d'un schéma fonctionnel, puisqu'ils démarrent leur travail à partir de la description du comportement du circuit à réaliser et non d'une structure déjà imaginée par le concepteur.

9.2 DOMAINES D'APPLICATION DE CETTE TECHNIQUE DE CONCEPTION

Cette technique de conception trouve son intérêt pour la conception de circuits (c'est-à-dire non réalisables par la programmation d'un microcontrôleur pré-existant). Ceux-ci peuvent être :

– soit des circuits rapides devant être produits en grande série (c'est-à-dire devant être optimisés en terme de surface et de rapidité). L'augmentation de la performance des technologies récentes diminue le domaine d'emploi de cette technique par l'accroissement des performances des microcontrôleurs standard ;
– soit des circuits à réaliser à l'aide de circuits paramétrables (par exemple des FPGA).

On trouve de tels circuits :

– dans les télécoms :
  • encrypteurs/décrypteurs de messages,
  • filtres numériques,
  • …
– dans le matériel informatique :
  • contrôleurs de périphériques,
  • …
– dans l'horlogerie :
  • montres complexes,
  • …

– dans les appareils grand-public :
  • décodeurs pour lecteurs de CD,
  • …

Les circuits spécifiques gagneront à être réalisés en polyphasé avec des techniques d'assemblage et de compilation de silicium. La réalisation sur des circuits paramétrables se fera toujours en monophasé par compilation. Pour cela, nous aborderons ces deux techniques de réalisation.

Tout l'enjeu de cette méthode consiste à passer du comportement du futur circuit à sa topologie.

9.3 DESCRIPTION DU COMPORTEMENT

Le comportement du futur circuit (ou de l'un de ses modules) peut être décrit :

– soit par un organigramme ;
– soit sous la forme d'un `process` VHDL.

Le fait de décrire le comportement et non la structure d'un futur circuit devrait être considéré comme une simplification du processus de conception. En fait, cela s'avère très perturbant pour les électroniciens plus habitués à manipuler des notions concrètes qu'abstraites.

9.3.1 Description du séquencement

La description du comportement pourra, ou non, préciser le séquencement fin du circuit. Lorsqu'il s'agit d'une application purement algorithmique (par exemple, pour réaliser un circuit d'encryption), il n'est pas utile d'imposer un séquencement fin. Par contre, dans le cas d'une application temps-réel (par exemple, pour réaliser un circuit d'horlogerie) la précision du séquencement est importante.

Si l'on ne souhaite pas préciser le séquencement fin on partira de celui donné implicitement par l'organigramme ou par la succession des instructions du `process`. Si l'on souhaite le préciser, on utilisera des instructions `wait` dans la description du `process`. Il peut également arriver que l'on ne souhaite préciser qu'un niveau de séquencement macroscopique, nécessaire au fonctionnement de l'application, sans s'occuper des niveaux de séquencement plus fins qui résulteront du processus de conception lui-même. On utilisera alors des instructions `wait` pour préciser le séquencement souhaité, mais sans faire la supposition que les instructions entre les `wait` sont simultanées.

Le langage VHDL n'est pas conçu pour donner un « sens matériel » au séquencement des instructions dans un `process`.

Lorsque l'on utilise le séquencement implicite des instructions d'un `process`, les signaux ne sont pas directement utilisables puisqu'ils se situent dans un autre environnement temporel. Il n'est pas possible de les modifier et d'utiliser leurs nouvelles valeurs dans les instructions suivantes. Il faut dont travailler avec des variables. Comme la valeur de celles-ci n'est pas conservée entre deux exécutions du `process` et n'est pas visible lors d'une simulation, il faudra utiliser des signaux pour sauvegarder les

variables. Celles-ci seront chargées au début du `process` et sauvegardées à la fin. Pour alléger la simulation, on pourra souvent éviter de recopier les signaux complexes de grande taille (par exemple les mémoires) en travaillant directement dessus, sous la réserve de ne pas, à la fois, les écrire et les lire entre deux instructions `wait`.

a) Exemple de description comportementale d'une montre

Nous illustrerons cette démarche de conception en l'appliquant à la réalisation d'une montre numérique. Pour limiter la complexité de la présentation, les fonctions de cette montre seront réduites au maximum. Il est évidemment possible de compliquer cet exemple jusqu'à l'amener à la réalisation d'une montre numérique multifonctions.

La description comportementale de cette montre est :

```
entity montre is
port(clk : in bit;
     rst : in bit);
end montre;
architecture arch1 of montre is
signal s, m, h :integer;
begin
process(rst)
begin
  s <= 0;
  m <= 0;
  h <= 0;
 end process;
 process
variable sv, mv, hv :integer;
  begin
wait until clk'event and clk='1'; --front montant de clk
   sv :=s;  -- Copie des signaux dans des variables
   mv :=m;
   hv :=h;
   sv :=sv +1;
     if sv = 60 then
        sv :=0;
        mv :=mv +1;
          if mv = 60 then
             mv :=0;
             hv :=hv +1;
               if hv = 24 then
                  hv :=0;
               end if;
          end if;
     end if;
   s <=sv; -- Recopie des variables dans les signaux
   m <=mv;
   h <=hv;
 end process;
end arch1;
```

Sous cette forme, nous préciserons uniquement la partie temps-réel du séquencement de la montre. Celle-ci doit avancer d'une seconde à chaque période du signal `clk`. Le séquencement fin des différentes opérations d'incrémentation et de report n'est pas encore défini. Cela nous amène à supposer un certain séquencement fin implicite dans le `process` et donc à devoir utiliser des variables qui seront sauvegardées dans des signaux.

Pour simplifier, nous ne traiterons pas les parties concernant la mise à l'heure et l'affichage de la montre. Une simple remise à zéro commandée par un signal `rst` permettra de démarrer les simulations dans un état bien défini.

9.3.2 Choix du compromis coût/performance

Le choix d'un bon compromis coût/performance est critique pour la conception d'un circuit optimisé. Il est très fréquent de concevoir un circuit trop rapide (donc trop complexe et trop cher !) ou, au contraire, pas assez performant. Un paramètre important pour choisir un bon compromis coût/performance est donné par la séquentialité des traitements. Celle-ci se traduit d'abord par la taille des bus et des opérateurs. Plus ceux-ci sont étroits, plus le traitement sera sérialisé. La séquentialité des traitements découle aussi de la complexité des opérateurs matériels utilisés. Par exemple, une multiplication pourra se faire en quelques cycles par un multiplieur câblé ou séquentiellement, en beaucoup de cycles, par une suite d'opérations d'additions et de décalages.

a) Adaptation de l'algorithme aux caractéristiques du futur circuit

Les opérations décrites dans l'algorithme devront être réalisables par le circuit et correspondre au compromis coût/performance recherché :

– Les transferts entre les éléments matériels (registres, opérateurs…) devront se faire avec des bus dont la largeur correspondra au taux de sérialisation recherché.
– La nature des opérations qui seront réalisées de manière combinatoire par des opérateurs matériels correspondra à la mise en œuvre de plus ou moins de matériel et à une réalisation plus ou moins séquentielle des opérations complexes. Les opérations habituellement réalisées par les opérateurs matériels combinatoires sont :
  • les opérations logiques bit-à-bit : *et*, *ou*, *ouex*, *non*,
  • les décalages, les rotations, et éventuellement les extractions et les compositions de champs de bits,
  • les opérations arithmétiques « de base » : +, –, incrémentation, décrémentations et complément arithmétique,
  • les *multiplications* ne seront câblées comme des opérateurs spéciaux que si leur fréquence d'utilisation le justifie, compte tenu du coût important de tels opérateurs,
  • des opérations simples, spécifiques à l'application, comme par exemple la transformation d'un code BCD en allumage des segments pour un affichage 7 segments.

Des opérations comme des divisions, des extractions de racines, etc. seront pratiquement toujours réalisées de manière séquentielle en utilisant des opérateurs matériels plus simples, de manière répétitive.

Avec ces adaptations, la séquence d'opérations décrite par l'algorithme deviendra identique à celle que le circuit doit réaliser.

b) Contraintes temps-réel

Certaines applications peuvent imposer des contraintes temps-réel sur l'exécution de l'algorithme. Par exemple, les applications chronométriques peuvent souhaiter que toutes les branches de l'algorithme aient la même durée de manière à ce que leurs exécutions comporte toujours le même nombre d'étapes. Ainsi modifié, l'algorithme devient un diviseur de la fréquence du quartz utilisé pour rythmer le séquencement fin des instructions.

➤ Description « adaptée » du séquencement de la montre

Ce circuit n'a aucune contrainte de performance car il fonctionne à basse fréquence. Par contre, il doit être le plus petit possible. Cela nous conduira à utiliser des bus et des opérateurs de 4 bits, aptes à transférer et à traiter des chiffres BCD.

La description VHDL adaptée deviendra :

```
architecture arch2 of montre is
subtype quad is bit_vector(3 downto 0); -- definition format
signal s1, s2, m1, m2, h1, h2 :quad;
begin
process(rst)  -- mise dans un etat initial connu (23h 59m 40s)
begin
s1 <= x"0";
s2 <= x"4";
m1 <= x"9";
m2 <= x"5";
h1 <= x"3";
h2 <= x"2";
end process;

 process
variable sv1, sv2, mv1, mv2, hv1, hv2 :quad;
  begin
wait until clk'event and clk='1'; --front montant de clk
sv1 :=s1; -- transferts signaux => variables
sv2 :=s2;
mv1 :=m1;
mv2 :=m2;
hv1 :=h1;
hv2 :=h2;
-- description de l'algorithme
sv1 :=sv1 + x"1";
if sv1 = x"A" then  -- test a 10
sv1 :=x"0";
```

```
sv2 :=sv2 + x"1";
if sv2 = x"6" then
sv2 :=x"0";
mv1 :=mv1 + x"1";
if mv1 = x"A" then
mv1 :=x"0";
mv2 :=mv2 + x"1";
if mv2 = x"6" then
mv2 :=x"0";
hv1 :=hv1 + x"1";
if (hv2 = x"2") and (hv1 = x"4") then
hv1 :=x"0";
hv2 :=x"0";
elsif hv1 = x"A" then
hv1 :=x"0";
hv2 :=hv2 + x"1";
end if;
      end if;
     end if;
    end if;
   end if;
   s1 <=sv1; -- transferts variables => signaux
s2 <=sv2;
m1 <=mv1;
m2 <=mv2;
h1 <=hv1;
h2 <=hv2;
 end process;
end arch2;
```

9.4 DÉMARCHE GÉNÉRALE DE CONCEPTION

Bien que pris dans son ensemble, le système à réaliser soit une machine séquentielle, cette démarche de conception consiste à le diviser en deux machines séquentielles couplées. Celles-ci sont spécialisées, l'une pour la manipulation et la transformation des données (appelée *chemin de donnée*) et l'autre pour le séquencement (appelée *séquenceur*). Cette décomposition découle de celle, classique, des systèmes de conduite de processus industriels dans lesquels le processus industriel est appelé la « partie opérative » et l'organe de commande la « partie commande » (figure 9.2).

Le rôle du séquenceur est d'envoyer une succession de commandes élémentaires au chemin de données (figure 9.3). Celui-ci les exécute et retourne au séquenceur des informations sur les propriétés des résultats des opérations et sur la valeur de certaines variables. Le séquenceur utilise ces informations pour orienter son exécution dans une direction ou dans une autre.

Le chemin de données ne sait que transférer et transformer des données. Il est constitué de bus et d'opérateurs reliés par des interrupteurs.

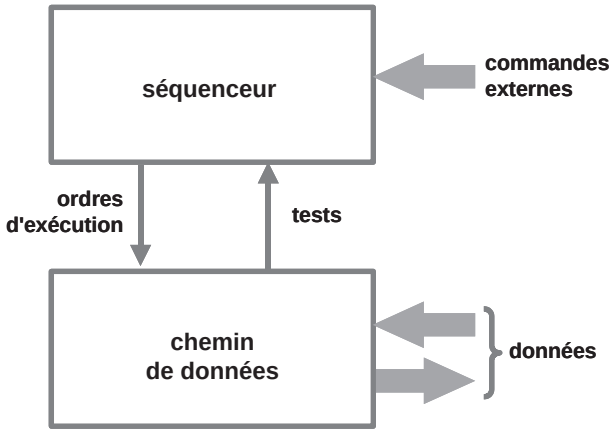


Figure 9.2 Décomposition d'un circuit complexe en un séquenceur et un chemin de données

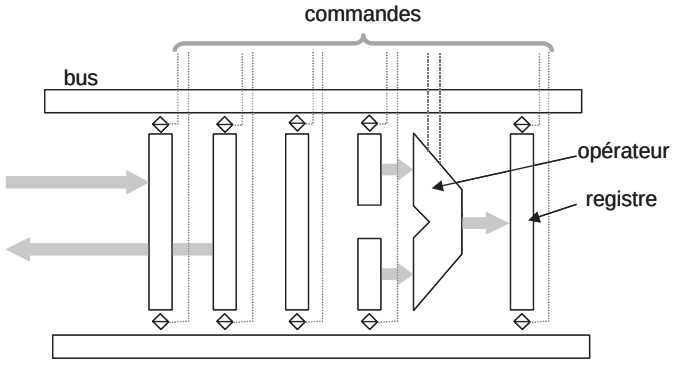


Figure 9.3 Chemin de données

Il peut être comparé au réseau de tuyauteries, de réservoirs et de réacteurs d'une raffinerie. Il ne travaille que s'il est commandé par le séquenceur (c'est-à-dire que si un organe de commande excite les vannes et les pompes de la raffinerie).

Le séquenceur est une machine séquentielle dont les entrées sont les tests issus du chemin de données et les sorties les commandes qu'elle lui envoie.

La conception de chacune de ces deux parties est très spécifique. Elle est réalisée par des méthodes et des outils très différents.

Les spécifications de ces deux parties découlent directement d'une décomposition simple de l'algorithme à réaliser (figure 9.4) :

– L'ensemble de toutes les instructions opératives constituera la spécification du chemin de données.

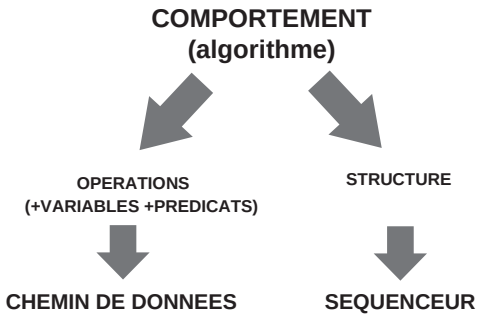


Figure 9.4

– La structure abstraite de l'algorithme (son schéma de contrôle) constituera la spécification du séquenceur.

Si le comportement souhaité comporte plusieurs processus fonctionnant en parallèle, ceux-ci peuvent être réalisés :

– soit de manière multiplexée au sein d'un algorithme unique qui conduira à la réalisation d'un module <séquenceur, chemin de données> unique ;
– soit par un ensemble de modules interconnectés réalisant les différents processus. Chaque module étant constitué d'un couple <séquenceur, chemin de données> ;
– Soit par une solution hybride comportant plusieurs modules, certains regroupant plusieurs processus.

9.5 CONCEPTION DU CHEMIN DE DONNÉES

9.5.1 Spécification du chemin de données de la montre

Cette spécification est constituée de la liste des variables utilisées (supposées être des signaux), des instructions opératives (c'est-à-dire modifiant les variables explicites) et des tests.

Variables utilisées :

```
signal s1,m1,h1 :quad; -- 4 bits
```

Instructions opératives :

```
s1<= s1 + x"1";
s1<= x"0";
s2<= s2 + x"1";
s2<= x"0";
m1<= m1 + x"1";
m1<= x"0";
m2<= m2+x"1";
m2<= x"0";
h1<= h1 + x"1";
```

```
h1<= x"0";
h2<= h2 + x"1";
h2<= x"0";
(null)
```

Tests :

```
s1 = x"A"
s2 = x"6"
m1 = x"A"
m2 = x"6"
(h2 = x"2")and(h1 = x"4")
h1 = x"A"
```

Comme la montre est un dispositif temps-réel, nous ajouterons une opération ineffective à la liste pour permettre le respect d'une contrainte d'égalisation des durées d'exécution de chaque branche de l'algorithme. Cette égalisation sera réalisée par des suites de longueur *ad hoc* d'instructions ineffectives.

Nous n'avons pas pris en compte l'instruction `wait` de démarrage du `process` car il ne s'agit que d'une opération destinée à contourner le fait que les `process` VHDL, muni d'une liste de sensibilité, démarrent sur les deux fronts.

9.5.2 Mise sous forme standard des instructions opératives

L'exécution de chaque instruction opérative va correspondre à la mise en œuvre d'une partie des ressources matérielles du chemin de données. Pour simplifier au maximum cette partie du circuit, il faut que les ressources matérielles du chemin de données soient utilisées par le maximum d'instructions opératives. Pour cela, nous allons les écrire d'une manière standardisée en leur donnant un *format commun*. De celui-ci découlera directement la structure matérielle du chemin de données.

a) Mise sous un format commun des instructions opératives de la montre

La majorité des instructions binaires sont des incrémentations. Les opérations unaires sont des chargement de variables à partir de constantes.

La forme standard pourra donc être :

```
<dest> <= incr <source>
```

Par exemple : `s1 <= s1 + x"1"` se réécrira : `s1 <= Incr(s1)`.

Les constantes à charger devront donc être réduites d'une unité.

Par exemple : `s1 <= x"0"` se réécrira : `s1 <= Incr(x"F")`

Nous verons, dans l'étude du séquenceur, qu'il est judicieux de coder l'instruction `null` comme `rien <= Incr(x"F")`.

Les champs <dest> et <source> de la forme standard pourront contenir :

<dest>	<source>
rien	X"F"
s1	s1
s2	s2
m1	m1
m2	m2
h1	h1
h2	h2

La destination `rien` correspond au fait de ne pas exploiter le résultat de l'instruction. Cela sera utile :

– pour réaliser les instructions ineffectives ;
– pour effectuer certains tests.

b) Cas des tests

Nous remarquerons que les tests portent presque toujours sur le résultat de l'opération d'incrémentation qui les précède. Cela nous permettra, dans le cas de la montre, de standardiser la forme des tests en :

```
<resultat> = {3,4,6,A}
```

Il existe une exception à cette règle pour les instructions suivantes :

```
.......
hv1 :=hv1 + x"1";
if (hv2 = x"2") and (hv1 = x"4") then
........
```

qui devront être réécrites en :

```
.......
hv1 <= Incr(hv1);
if (resultat = x"4") then
rien <= Incr(hv2);
if (resultat = x"3") then
........
```

L'utilisation de l'instruction `rien <= Incr(hv2);` produit un résultat « interne » qui peut être testé, toutefois la valeur à tester a dû, pour cela, être augmentée de 1.

Remarque :Les formes standard des instructions opératives et des tests permettent d'écrire d'autres algorithmes que celui dont on est parti. Cette généralisation se retrouve sur le chemin de données. Elle est d'autand plus importante que l'algorithme de départ est volumineux et complexe. Il est même souvent possible que le chemin de données obtenu soit suffisament général pour exécuter n'importe quel algorithme. Nous dirons alors que la machine correspondante est devenue *universelle*.

9.5.3 Conception physique du chemin de données

Le chemin de données peut être traité soit comme une machine séquentielle polyphasée ou monophasée (figure 9.5). Dans les deux cas sa conception découle directement des formes standard obtenues :

– les variables deviennent les *registres* de la machine ;
– les opérations sont réalisées par l'*unité arithmétique et logique* (appelée UAL), souvent unique ;
– les opérandes sont collectés par des *bus* qui aboutissent à l'UAL ;
– les résultats de l'UAL sont distribués par un autre bus.

Chaque instruction opérative sera exécutée en un tour dans le chemin de données :

– les opérandes sont extraits des registres sources et mis sur les bus sources ;
– l'UAL est excitée par les bus sources et fournit son résultat sur le bus destination ;
– le résultat sur le bus destination est chargé dans le registre destination.

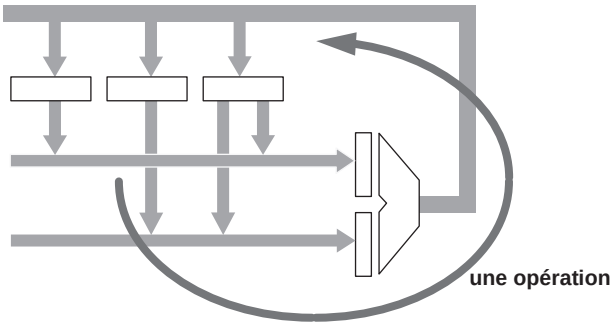


Figure 9.5 Exécution d'une instruction opérative sur un chemin de données

Il est quelquefois nécessaire d'effectuer une optimisation des sources de manière à équilibrer la charge des bus correspondants. Cette optimisation consiste à déplacer des registres sources d'un bus à l'autre. Elle se fait en jouant sur la commutativité de certaines opérations.

a) Prise en compte des problèmes électroniques

Le chemin de données pourra être réalisé comme un système séquentiel polyphasé ou monophasé. Il faudra faire très attention au couplage entre le chemin de données et le séquenceur.

Dans le cas d'un chemin de données polyphasé, l'exécution de chaque instruction sera décomposée en plusieurs phases (figure 9.6). Des barrières temporelles seront mises en place pour éviter les rebouclages intempestifs.

Le séquencement de ce type de chemin de données est tout à fait classique et correspond à une utilisation quasi-optimale du temps. Les chemins de données poly-

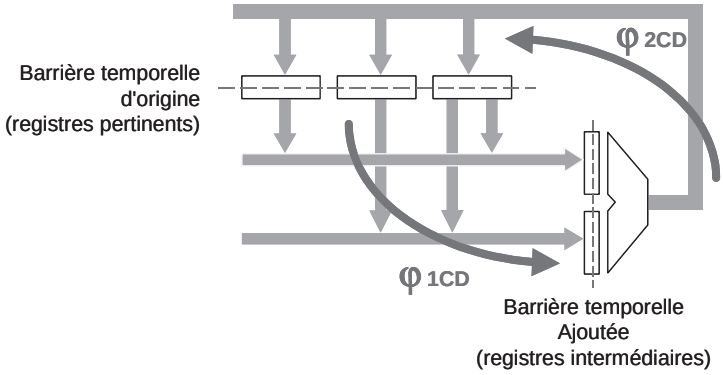


Figure 9.6 Séquencement bi-phasé d'un chemin de données

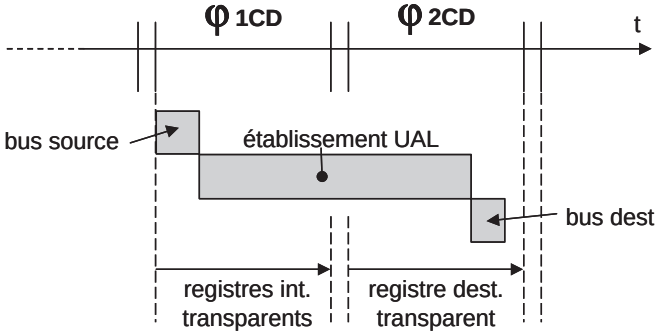


Figure 9.7 Chronogramme du fonctionnement d'un chemin de données biphasé

phasés sont généralement biphasés. Leurs deux phases de fonctionnement, que nous appellerons φ1CD et φ2CD seront incluses dans le séquencement global de la machine qui pourra comporter deux phases ou plus. La première phase (φ1CD) sera consacrée à la sélection des registres sources et à la mise en transparence de ceux de la barrière temporelle intermédiaire. La seconde phase (φ2CD) sera consacrée à la mise en transparence du (ou des) registre(s) destination (figure 9.7).

Dans le cas d'un chemin de données monophasé, l'exécution de chaque instruction correspondra à un tour complet de l'information en une période d'horloge. Ses registres seront réalisés par des bascules (figure 9.8). Aucun registre intermédiaire n'est nécessaire. L'établissement des bus et des opérateurs pourra occuper presque toute la période moins le temps de pré-positionnement des bascules. Le chargement conditionnel du registre destination se fera par le positionnement de la commande de chargement concernée pendant le cycle d'exécution.

Nous voyons que les différences entre ces deux approches sont assez réduites au niveau du chemin de données. Elles découlent donc de la même méthode de con-

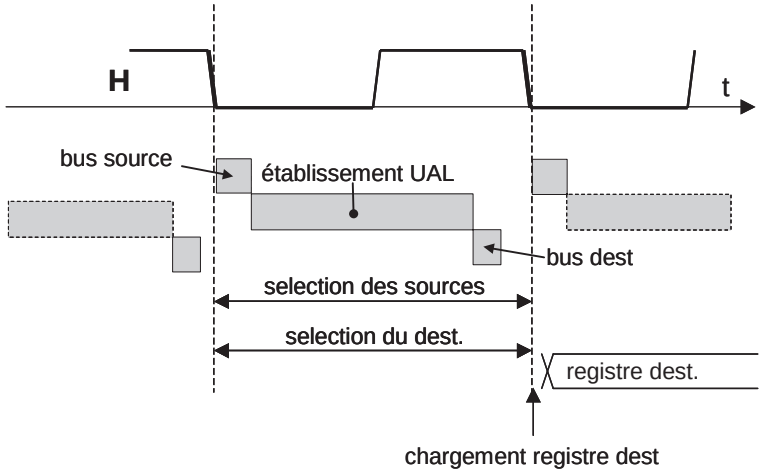


Figure 9.8 Chronogramme du fonctionnement d'un chemin de données monophasé

ception. L'exemple de la montre sera traité en polyphasé, tandis que la conception complète d'un circuit monophasé sera traité dans l'exercice n° xxx.

b) Structure du chemin de données de la montre

Nous choisissons de réaliser la montre en logique polyphasée. Son chemin de données ne comporte qu'un seul bus source (figure 9.9). Toutes les commandes de chargement des registres à partir du bus destination sont validées par la phase φ2CD. L'UAL ne dispose d'aucune commande puisqu'elle n'effectue, de manière combinatoire, qu'une seule opération. Le module de test compare le résultat de l'incrémentation aux valeurs 3, 4, 6 et A. Toutefois, pour laisser le maximum de liberté pour la conception, nous ne préciserons pas la forme sous laquelle il enverra ces informations au séquenceur.

c) Description VHDL du chemin de données de la montre

```
type val_cond is(VTA,VT6,VT4,VT3,VTX); -- dans un package
entity pop_montre is
port(
     LOS1,LOS2,LOM1,LOM2,LOH1,LOH2,
SELS1,SELS2,SELM1,SELM2,SELH1,SELH2,SELF : in bit;
     Tests : out val_cond;
     PH1CD,PH2CD,RST : in bit)
end pop_montre;
architecture rtl of pop_montre is
function Incr (a : bit_vector) return bit_vector is
alias av : bit_vector (1 to a'length) is a;
   variable inc_a : bit_vector (1 to a'length);
   begin
inc_a := av;
    for i in a'length downto 1 loop
```

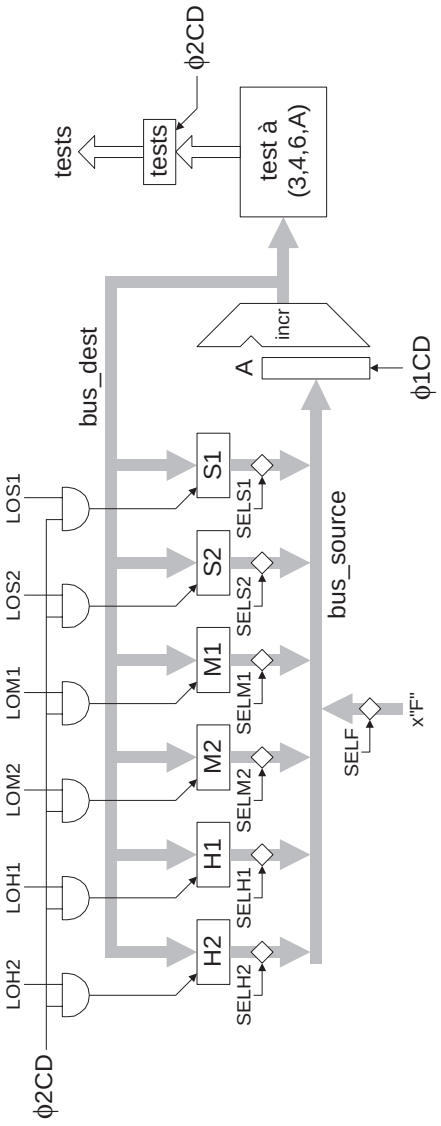


Figure 9.9 Le chemin de données de la montre

```
inc_a(i) :=not av(i);
        exit when av(i)='0';
    end loop;
    return inc_a;
  end Incr;

  signal S1, S2, M1, M2, H1, H2 :quad;
  signal A :quad;
```

```
  signal Bus_source, Bus_dest :quad;
  signal val-tests :val_cond;
begin
-- Bus_source
Bus_source <= S1 when SELS1 ='1'else
              S2 when SELS2 ='1'else
              M1 when SELM1 ='1'else
              M2 when SELM2 ='1'else
              H1 when SELH1 ='1'else
              H2 when SELH2 ='1'else
              x"F" when SELF ='1';
-- Chargement latch intermediaire (transparent sur PH1CD)
A <= Bus_source when PH1CD='1'else A;
-- Fonctionnement incrementeur
Bus_dest <= Incr(A);
-- calcul des tests
With Bus_dest select
         val_tests <= VT3 when x"3" else
         VT4 when x"4" else
         VT6 when x"6" else
         VTA when x"A" else
         VTX when others;
     Tests <= VTX when RST='1' else
       val_tests when PH2CD='1' else Tests;
-- Chargement registres destination (transparents sur PH2)
     S1 <= Bus_dest when LOS1 ='1' and PH2CD='1' else S1;
     S2 <= Bus_dest when LOS2 ='1' and PH2CD='1' else S2;
     M1 <= Bus_dest when LOM1 ='1' and PH2CD='1' else M1;
     M2 <= Bus_dest when LOM2 ='1' and PH2CD='1' else M2;
     H1 <= Bus_dest when LOH1 ='1' and PH2CD='1' else H1;
     H2 <= Bus_dest when LOH2 ='1' and PH2CD='1' else H2;
end rtl;
```

Cette description VHDL n'est qu'un modèle de simulation. Elle est écrite de manière à être la plus proche possible de la réalisation matérielle visée (par exemple, elle utilise la transparence des latches). Elle n'est pas prévue pour constituer l'entrée d'un outil de synthèse.

L'écriture de cette description contient une fonction `Incr()` dont la structure interne n'a rien à voir avec l'opérateur matériel combinatoire qui la réalise. La destination `rien` correspond simplement à aucun chargement de registre. La mise de la montre dans un état initial ne correspond à aucun dispositif matériel car il ne s'agit que d'un mécanisme d'essai pour la simulation.

d) Utilisation de décodeurs dans les chemins de données

Il arrive fréquemment que les opérations à réaliser par le chemin de données comportent des accès à des tableaux ou des indexations. Par exemple :

```
R(ASD) <= W17;
```

dans laquelle `ASD` et `W17` sont des registres et `R()` un tableau.

Le tableau est matérialisé par une mémoire constituée par un ensemble de registres. Sa commande de chargement signifie littéralement : *Chargez le mot de la mémoire* `R()` *indexé par* `ASD`. Cela peut être facilement obtenu en insérant un décodeur dans le chemin de données (figure 9.10). Celui-ci est branché sur le registre `ASD`, ses sorties commandent le chargement des mots de la mémoire `R()`. Le décodeur est validé par la commande de chargement issue du séquenceur.

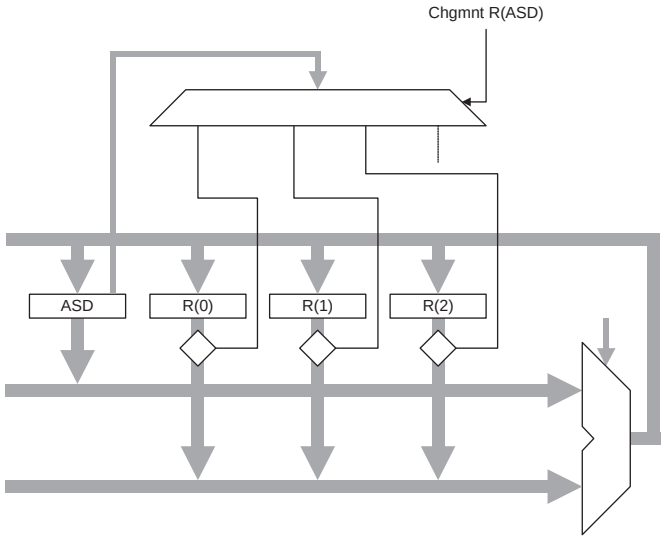


Figure 9.10 Décodeur pour adressage indexé

Un mécanisme similaire peut être utilisé pour accéder aux bits d'un registre. Le lecteur intéressé pourra voir l'utilisation d'un tel décodeur dans l'annexe 2 : *Étude d'une montre avec affichage*.

e) Dessin du chemin de données

Des techniques très efficaces sont utilisées pour dessiner les chemins de données (figures 9.11 et 9.12). La technique de base pour concevoir ce type de blocs est l'utilisation d'une approche dite par « tranches ». Le dessin du chemin de données est obtenu par la répétition et la juxtaposition de bandes de dessin (les tranches) qui représentent chacune un bit de la totalité du chemin de données. Ces bandes sont constituées par l'assemblage de cellules qui ont la même hauteur et qui s'interconnectent par simple juxtaposition dans les deux directions. Cette approche respecte le principe de croisement des flux d'information et des couches technologiques.

Le dessin du chemin de données est généralement obtenu à l'aide d'un outil, appelé *assembleur de silicium*, qui effectue l'assemblage des cellules élémentaires par simple juxtaposition.

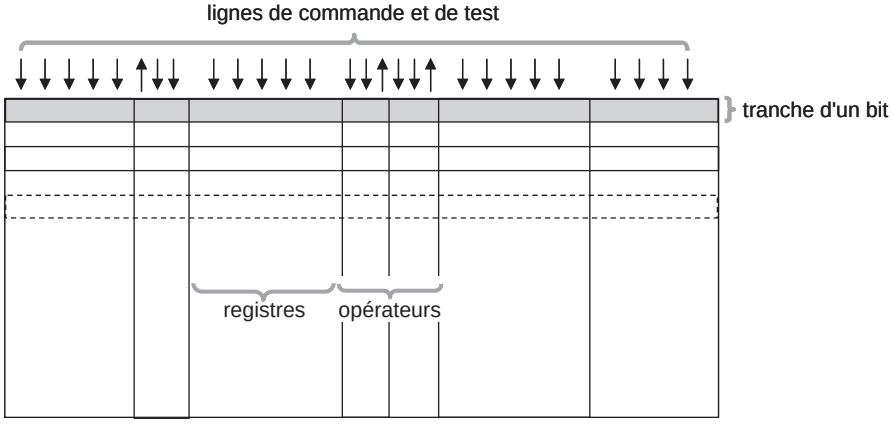


Figure 9.11 Organisation du dessin d'un chemin de données

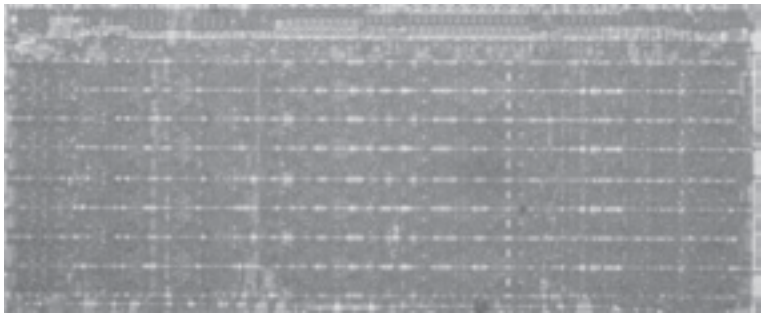


Figure 9.12 Chemin de données du microprocesseur Motorola MC 6800® (8 bits)

L'ensemble des tranches est surmonté d'une ligne d'amplificateurs et de bascules destinés à renforcer les commandes et éventuellement à les formater par les horloges, ainsi que pour stocker les résultats des tests.

Le dessin final est rectangulaire. Il est très dense et très optimisé. La place perdue est très faible. Les bus se superposent à la logique des cellules. Les alimentations sont (bien) distribuées *via* la grille qui est constituée par les lignes d'alimentation verticales et celles horizontales incluses dans les bus.

Le chemin de données constitue certainement la plus belle pièce de dessin de ce type de circuit. Généralement, sa surface est comprise entre le quart et la moitié de celle de tout le circuit.

➤ Dessin du chemin de données de la montre

Le chemin de données de la montre est conçu suivant ce principe. Il est constitué de quatre tranches (figure 9.13).

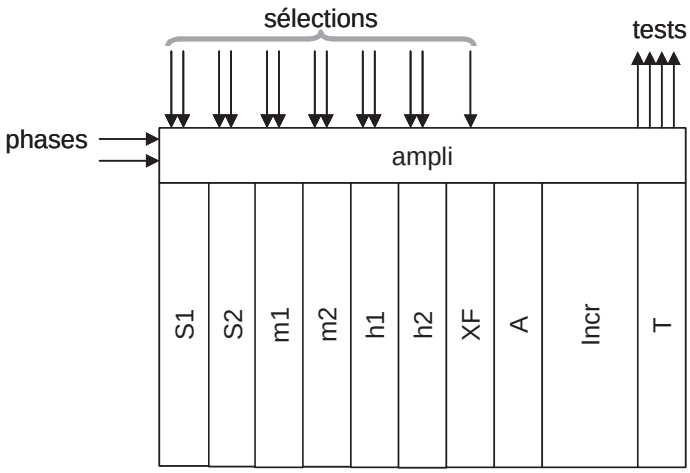


Figure 9.13 Chemin de données de la montre

Exemple de cellules :

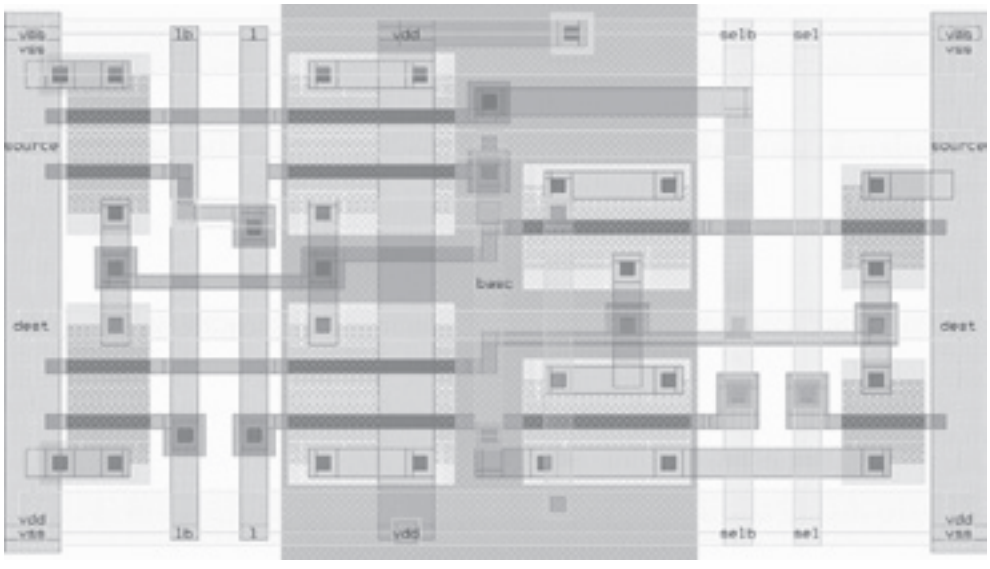


Figure 9.14 Élément de registre pour la montre (technologie symbolique Alliance)

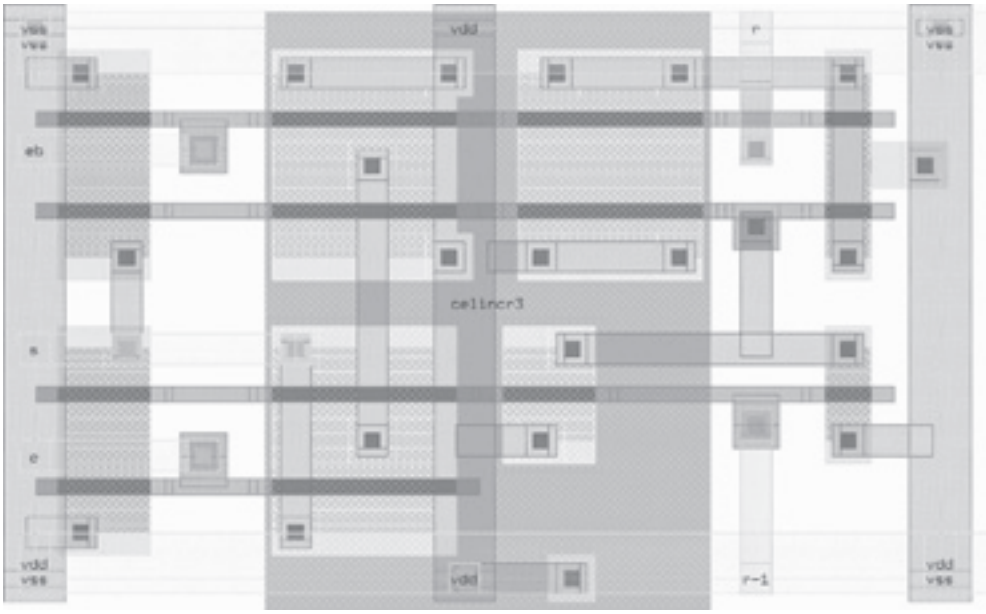


Figure 9.15 Tranche d'un bit d'incrémenteur pour la montre (technologie symbolique Alliance)

Ces dessins obéissent raisonnablement aux règles de croisement des couches technologiques. On reconnaît les bandes de métal de second niveau qui vont former la structure des bus de la tranche.

9.6 ARCHITECTURE TEMPORELLE

L'architecture temporelle d'un circuit complexe est très importante. Elle détermine directement ses performances, sa complexité mais aussi la fiabilité de son fonctionnement. En effet, beaucoup de problèmes des circuits VLSI proviennent d'une mauvaise architecture temporelle qui fait que dans certaines conditions limites certains signaux peuvent ne pas être stabilisés au moment où ils sont exploités.

L'objectif de l'architecture temporelle est de donner à chaque organe le temps dont il a besoin pour fonctionner (dans les pires conditions !) tout en minimisant le temps de cycle global. Un tel objectif est atteint en étudiant le *chemin critique*. Celui-ci est représenté par la chaîne des organes dont le fonctionnement est obligatoirement successif. La durée du chemin critique détermine directement le temps de cycle du circuit. L'art du concepteur est donc de chercher à diminuer sa longueur, soit en réduisant le temps de fonctionnement des organes les plus lents, soit en permettant un fonctionnement moins séquentiel de ces organes.

9.6.1 Fonctionnements relatifs du séquenceur et du chemin de données

a) Fonctionnements successifs

Cette technique est la plus simple (figure 9.16). Elle consiste à faire fonctionner successivement le séquenceur et le chemin de données. Elle est utilisée pour les machines lentes.

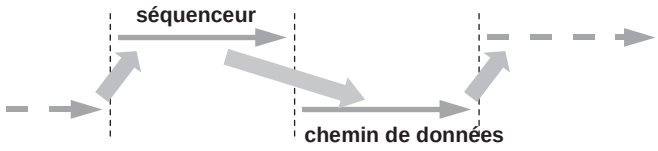


Figure 9.16 Fonctionnement successif du séquenceur et du chemin de données

Les commandes générées par le cycle courant du séquenceur servent immédiatement pour le chemin de données. Les résultats des tests générés par ce dernier servent immédiatement à sélectionner l'étape suivante dans le séquenceur.

Dans le cas de circuits biphasés, elle conduit à un fonctionnement global quadriphasé (figure 9.17).

– Deux phases pour le séquenceur (∅1S et ∅2S) ;
– deux phases pour le chemin de données (∅1CD et ∅2CD).

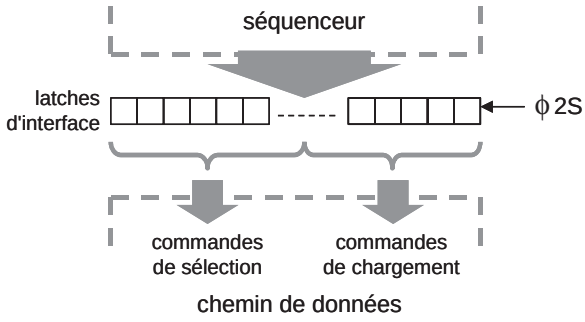


Figure 9.17 Latches d'interface entre le séquenceur et le chemin de données

Pour rappel, les commandes de chargement sont reformatées par ∅2CD à l'entrée du chemin de données (figure 9.18).

Les retours du chemin de données vers le séquenceur (tests, valeurs…) seront stockés, comme des destinations, dans des latches transparents sur ∅2CD. Le séquenceur pourra utiliser ces informations dès ∅1S du cycle suivant.

Dans le cas des circuits monophasés, il est possible de générer deux horloges successives, la première pour le séquenceur et la seconde pour le chemin de données.

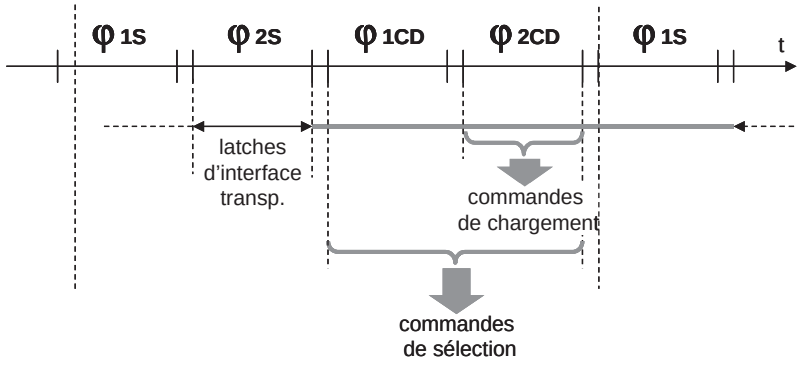


Figure 9.18 Chronogramme du fonctionnement de l'interface

Il sera même possible d'associer un séquenceur câblé monophasé compilé avec un chemin de données biphasé assemblé.

b) Fonctionnements superposés

Cette technique est utilisée pour les machines rapides. Elle consiste à faire fonctionner en même temps le séquenceur et le chemin de données. Dans le cas de la montre, elle conduirait à un fonctionnement global biphasé, soit deux fois plus rapide qu'un fonctionnement successif (figure 9.19).

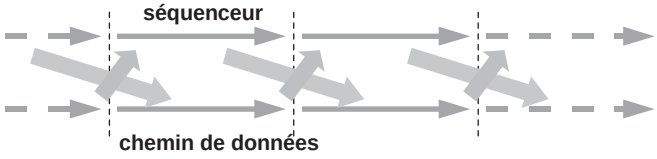


Figure 9.19 Fonctionnement superposé du séquenceur et du chemin de données

Le fonctionnement temporel d'un tel système est plus complexe. Les commandes pour le chemin de données proviennent du cycle précédent du séquenceur qui se trouve être en avance d'un cycle par rapport au chemin de données. Comme les résultats des tests générés par celui-ci ne peuvent influencer que le cycle suivant du séquenceur, il s'écoule deux cycles entre la sélection d'une étape dans le séquenceur et la possibilité de faire un choix conditionnel dans le séquencement. Du point de vue de l'écriture de l'algorithme, un test ne pourra porter que sur les résultats de l'étape qui précède la précédente ! L'algorithme doit être modifié en conséquence, en ajoutant des instructions ineffectives et en permutant l'ordre de certaines actions.

Cette technique de séquencement s'adapte directement aux circuits monophasés et biphasés.

➤ Cas des systèmes superposés biphasés (figure 9.20)

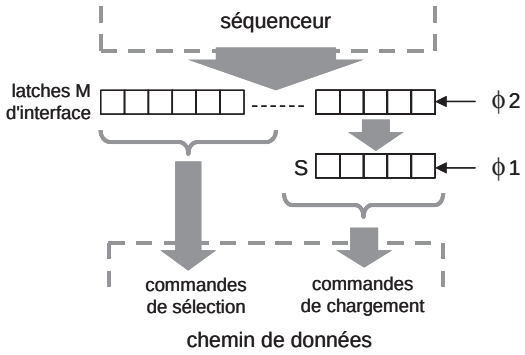


Figure 9.20 Latches maîtres-esclaves d'interface
entre le séquenceur et le chemin de données (système superposé biphasé)

Pour rappel, les commandes de chargement sont reformatées par ∅2 à l'entrée du chemin de données (figure 9.21).

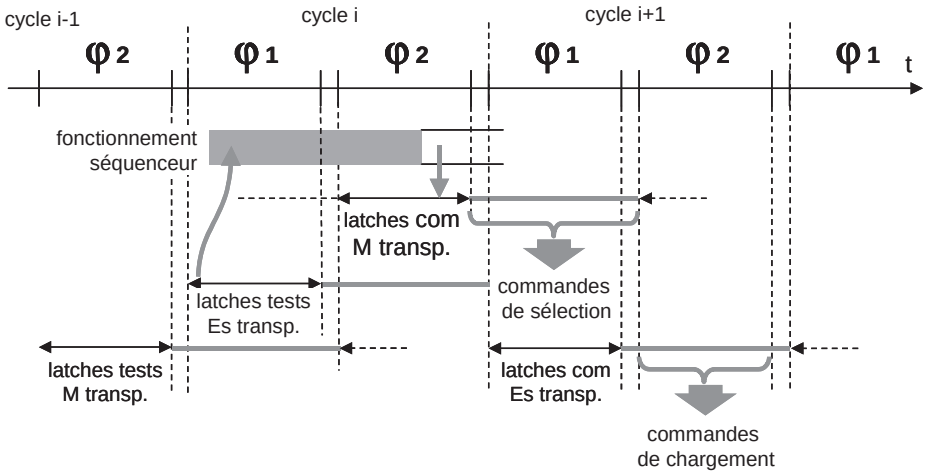


Figure 9.21 Chronogramme du fonctionnement de l'interface
(système superposé biphasé)

Les commandes de sélection nécessaires en ∅2 du cycle i + 1 doivent être mémorisées dans le registre esclave d'interface (par exemple la commande d'un opérateur).

Les retours du chemin de données vers le séquenceur (tests, valeurs…) seront stockés, comme des destinations, dans des latches maîtres transparents sur ∅2 du cycle i – 1, puis dans des esclaves transparents sur ∅1 du cycle i. Le séquenceur pourra

donc utiliser ces informations dans tout le cycle i pour générer des commandes pour le cycle i + 1.

➤ Cas des systèmes superposés monophasés (figure 9.22)

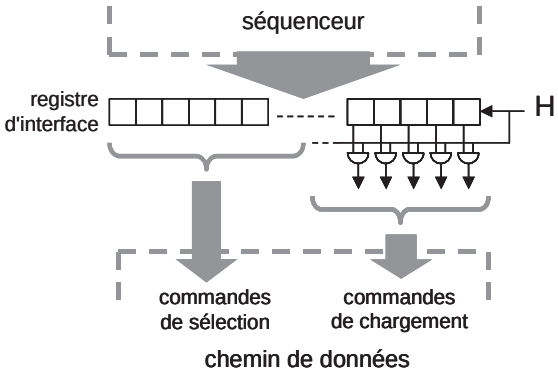


Figure 9.22 Registre d'interface et génération des sous-horloges (système monophasé superposé)

Le registre d'interface met les commandes à disposition du chemin de données pour le cycle qui suit celui du fonctionnement du séquenceur. Les commandes de chargement valident des sous-horloges qui attaquent les registres concernés du chemin de données (figure 9.23).

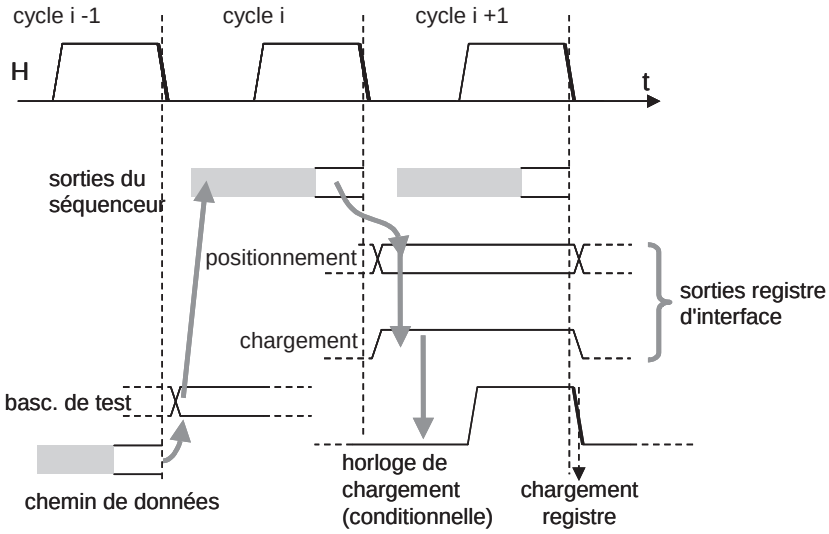


Figure 9.23 Chronogramme du fonctionnement de l'interface (système monophasé superposé)

Les retours du chemin de données vers le séquenceur (tests, valeurs…) seront stockés, comme des destinations, dans des registres par la transition d'horloge qui termine le cycle i – 1. Le séquenceur pourra utiliser ces informations dès le début du cycle i pour générer des commandes pour le cycle i + 1.

9.7 CONCEPTION DU SÉQUENCEUR DE LA MONTRE

9.7.1 Mise en forme de l'algorithme

L'algorithme de la montre est réécrit de manière à mettre en évidence les instructions opératives formatées ainsi que le séquencement qui doit être tel que l'algorithme se présente comme un diviseur précis de la fréquence de l'horloge. Pour cela, des instructions ineffectives (`null`) seront ajoutées de manière à ce que toutes les branches aient la même longueur (ici 16 instructions) (figure 9.24). L'exécution de cet algorithme se comportera alors comme un diviseur par 16, ce qui fait que, pour rester à l'heure, la montre devra être excitée par une horloge précise de 16 hz (issue de la division par 2^{11} d'un quartz horloger de 32,768 kHz).

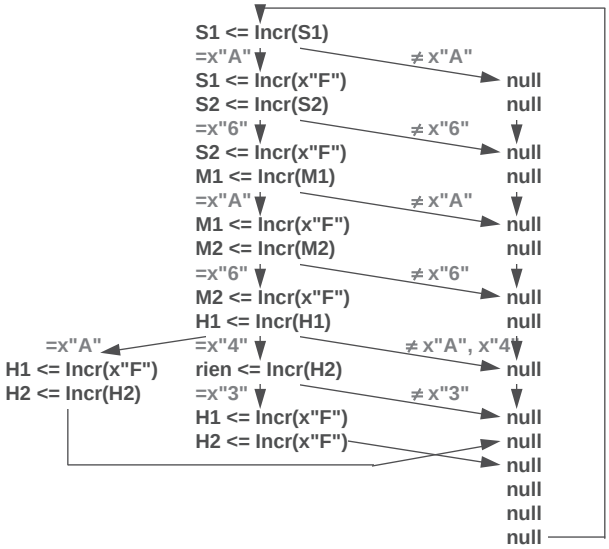


Figure 9.24

L'algorithme de la montre est écrit de manière à s'adapter aux contraintes du matériel. Les tests sont réalisés par le fait de choisir une instruction suivante parmi plusieurs possibles. Ce choix est effectué suivant les propriétés du résultat de l'instruction précédente.

9.7.2 Réalisation du séquenceur de la montre

Nous avons vu que le séquenceur est une machine séquentielle. Dans le chapitre 7, nous avons vu qu'il existe de nombreuses techniques pour réaliser un tel bloc.

Comme l'algorithme à réaliser est relativement simple, nous choisirons de réaliser le séquenceur de la montre sous la forme d'une machine de Mealey, à l'aide d'un simple PLA (figure 9.25). Nous présenterons aussi des réalisations microprogrammées et sous la forme d'un automate câblé.

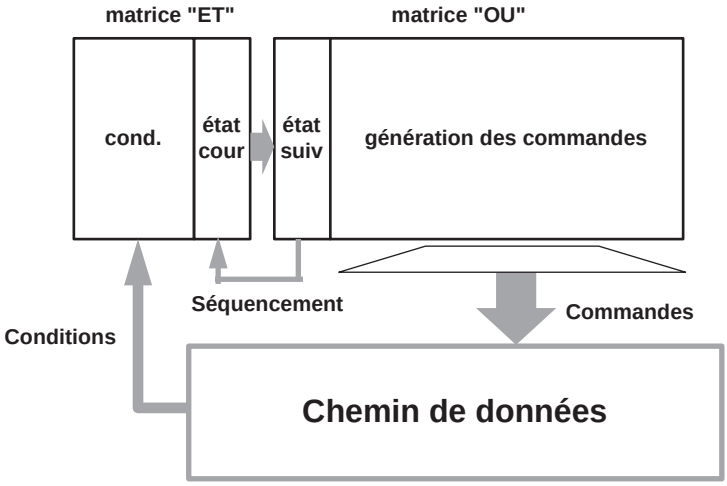


Figure 9.25 Organisation d'un circuit complexe avec un séquenceur à base de PLA

Cela entraîne que les tests soient effectués en sélectionnant, par la matrice ET, l'instruction suivante par le couple (état courant, valeur des tests). Une conséquence de cette approche est qu'une instruction appartenant à un « éclatement » ne peut pas être également le successeur d'une autre instruction non conditionnelle. Cette contrainte peut être contournée en codant les instructions `null` par :

```
rien <= Incr(x"F")
```

De cette manière, leur résultat est toujours 0, c'est-à-dire différent de 3, 4, 6, A., ce qui transforme les branchements conditionnels en des branchements inconditionnels vers l'instruction `null` suivante (figure 9.26).

9.7.3 Contenu du PLA

L'algorithme doit être écrit sous la forme d'instructions conditionnelles (dites *gardées*) :

<Etat courant, Conditions> => actions, Etat suivant

Toutes les valeurs possibles des conditions doivent être prévues (tableau 9.1).

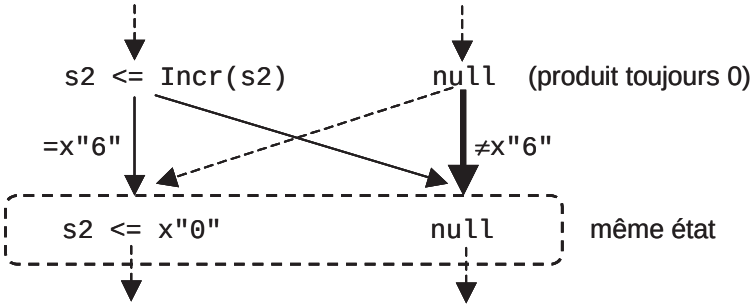


Figure 9.26

TABLEAU 9.1 VALEURS POSSIBLES DES CONDITIONS

matrice ET		matrice OU	
état	**conditions**	**actions**	**et. suiv.**
E1		S1<=Incr(S1)	E2
E2	=x"A"	S1<=Incr(x"F")	E3
E2	πx"A"	rien<=Incr(x"F")	E4
E3		S2<=Incr(S2)	E5
E4		rien<=Incr(x"F")	E5
E5	=x"6"	S2<=Incr(x"F")	E6
E5	πx"6"	rien<=Incr(x"F")	E7
E6		M1<=Incr(M1)	E8
E7		rien<=Incr(x"F")	E8
E8	=x"A"	M1<=Incr(x"F")	E9
E8	πx"A"	rien<=Incr(x"F")	E10
E9		M2<=Incr(M2)	E11
E10		rien<=Incr(x"F")	E11
E11	=x"6"	M2<=Incr(x"F")	E12
E11	πx"6"	rien<=Incr(x"F")	E13
E12		H1<=Incr(H1)	E14
E13		rien<=Incr(x"F")	E14
E14	=x"A"	H1<=Incr(X"F")	E15
E14	=x"4"	rien<=Incr(H2)	E16
E14	πx"4", πx"A"	rien<=Incr(x"F")	E16
E15		H2<=Incr(H2)	E18
E16	=x"3"	H1<=Incr(x"F")	E17
E16	πx"3"	rien<=Incr(x"F")	E18
E17		H2<=Incr(x"F")	E19
E18		rien<=Incr(x"F")	E19
E19		rien<=Incr(x"F")	E20
E20		rien<=Incr(x"F")	E21
E21		rien<=Incr(x"F")	E22
E22		rien<=Incr(x"F")	E1

9.7.4 Optimisation topologique du PLA

Les différentes lignes du PLA peuvent être déplacées indépendamment pour optimiser la matrice ET du PLA (figure 9.27).

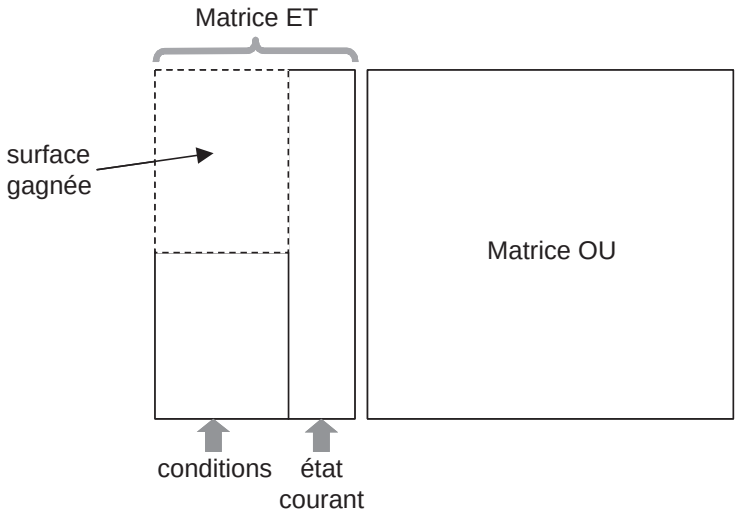


Figure 9.27 Surface gagnée par réarrangement du PLA

La surface ainsi gagnée peut servir à loger un autre bloc de dessin.

9.7.5 Séquencement global de la montre

Comme la montre est un dispositif lent, nous choisirons d'utiliser un séquencement successif à quatre phases dans lequel les deux premières phases ∅1S et ∅2S seront dédiées au fonctionnement du séquenceur et les deux suivantes ∅1CD et ∅2CD au chemin de données (figure 9.28). Naturellement, nous aurions pu renommer les phases ∅1, ∅2, ∅3 et ∅4.

Le registre de sortie du PLA joue le rôle de registre d'interface entre le séquenceur et le chemin de données. La validation des commandes de chargement par ∅2CD se fait dans le chemin de données (figure 9.29).

9.7.6 Description VHDL du séquenceur de la montre

```
type val_cond is
  (VTA,VT6,VT4,VT3,VTX,PHI,NVTA,NVT6,NVT3,NVT4A);
constant Duree_Phase : int :=xxx; -- duree symbolique de la phase
entity ctrl_montre is
 port(
     LOS1,LOS2,LOM1,LOM2,LOH1,LOH2,
     SELS1,SELS2,SELM1,SELM2,SELH1,SELH2,SELF : out bit;
```

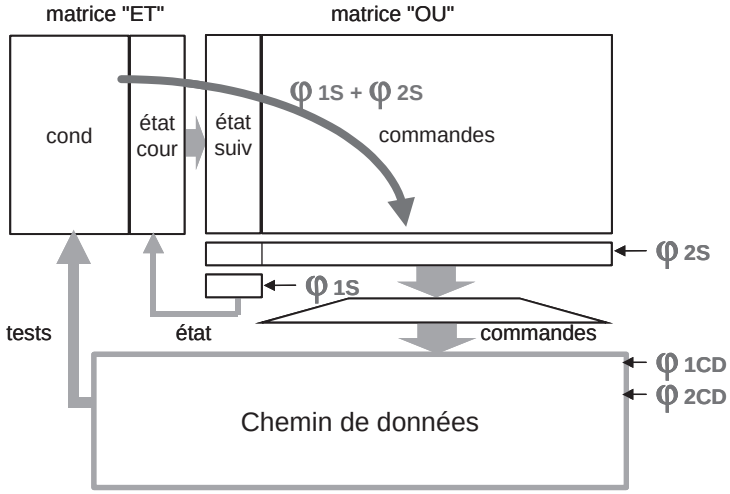


Figure 9.28 Architecture d'un circuit complexe utilisant un séquenceur à base de PLA

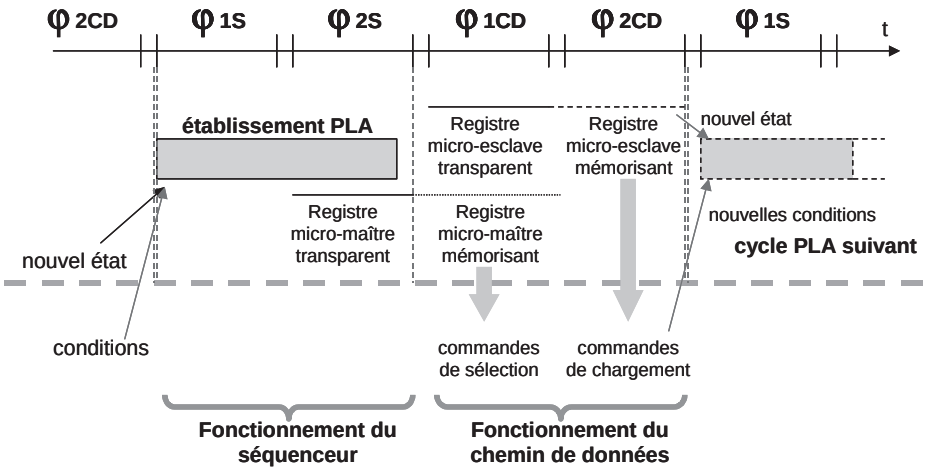


Figure 9.29 Chronogramme du fonctionnement successif d'un PLA et d'un chemin de données

```
      Tests : in val_cond;
      PH1S,PH2S,PH1CD,PH2CD,RST : in bit);
end ctrl_montre;
architecture rtl2 of ctrl_montre is
 type etiquettes is (E1,E2,E3,E4,E5,E6,E7,E8,E9,
           E10,E11,E12,E13,E14,E15,E16,E17,E18,
           E19,E20,E21,E22);
 type val_source is(F,S1,S2,M1,M2,H1,H2);
```

```
type val_dest is(rien,S1,S2,M1,M2,H1,H2);
type micro_sel is record
    Etat_courant :etiquettes;
    Condition :val_cond;
  end record;
type et_mat is array(1 to 29) of micro_sel;
constant PLA_ET :et_mat :=(
   (E1,PHI),(E2,VTA),(E2,NVTA),(E3,PHI),(E4,PHI),
   (E5,VT6),(E5,NVT6),(E6,PHI),(E7,PHI),
   (E8,VTA),(E8,NVTA),(E9,PHI),(E10,PHI),
   (E11,VT6),(E11,NVT6),(E12,PHI),(E13,PHI),
   (E14,VTA),(E14,VT4),(E14,NVT4A),(E15,PHI),
   (E16,VT3),(E16,NVT3),(E17,PHI),(E18,PHI),
   (E19,PHI),(E20,PHI),(E21,PHI),(E22,PHI));
type micro_mot is record
   dest :val_dest;
   source :val_source;
   Etat_suivant :etiquettes;
 end record;
type ou_mat is array(1 to 29) of micro_mot;
constant PLA_OU :ou_mat :=(
   (S1,S1,E2),(S1,F,E3),(rien,F,E4),
   (S2,S2,E5),(rien,F,E5),(S2,F,E6),(rien,F,E7),
   (M1,M1,E8),(rien,F,E8),(M1,F,E9),(rien,F,E10),
   (M2,M2,E11),(rien,F,E11),(M2,F,E12),
   (rien,F,E13),(H1,H1,E14),(rien,F,E14),
   (H1,F,E15),(rien,H2,E16),(rien,F,E16),
   (H2,H2,E18),(H1,F,E17),(rien,F,E18),(H2,F,E19),
   (rien,F,E19),(rien,F,E20),(rien,F,E21),
   (rien,F,E22),(rien,F,E1));
signal Reg_mic1 :micro_mot;
signal Reg_mic2 :etiquettes;
begin
       -- simulation lecture PLA
-- celui-ci travaille pendant PH1S et une partie de PH2S
process
variable i :integer;
begin
 wait until PH1S'event and PH1S='1'; -- de'but de PH1S
   i :=1;
 while not(   -- boucle de balayage du PLA
  ((PLA_ET(i).Etat_courant=Reg_mic2)
  and
  ((PLA_ET(i).Condition=PHI) or
  (PLA_ET(i).Condition=Tests) or
  (PLA_ET(i).Condition=NVT3 and Tests/=VT3) or
  (PLA_ET(i).Condition=NVT6 and Tests/=VT6) or
  (PLA_ET(i).Condition=NVTA and Tests/=VTA) or
  (PLA_ET(i).Condition=NVT4A and Tests/=VT4 and Tests/=VTA)))
  )loop
```

```
        i :=i +1;
        assert i >= 30
           report "erreur simulation PLA" severity failure;
     end loop;
     -- report sortie PLA fin de PH2S
        Reg_mic1<=PLA_OU(i) after Duree_Phase * 1.8;
   end process;
    -- chargement esclave et initialisation
       Reg_mic2 <=E1 when RST='1' else
                  Reg_mic1.Etat_suivant when PH1S='1'else
                  Reg_mic2;
    -- generation des commandes
    -- selections bus source
    SELS1<='1'when (Reg_mic1.source=S1) else '0';
    SELS2<='1'when (Reg_mic1.source=S2) else '0';
    SELM1<='1'when (Reg_mic1.source=M1) else '0';
    SELM2<='1'when (Reg_mic1.source=M2) else '0';
    SELH1<='1'when (Reg_mic1.source=H1) else '0';
    SELH2<='1'when (Reg_mic1.source=H2) else '0';
    SELF <='1'when (Reg_mic1.source=F)  else '0';
    -- commandes chargement bus destination
    LOS1<='1'when (Reg_mic1.dest=S1) else '0';
    LOS2<='1'when (Reg_mic1.dest=S2) else '0';
    LOM1<='1'when (Reg_mic1.dest=M1) else '0';
    LOM2<='1'when (Reg_mic1.dest=M2) else '0';
    LOH1<='1'when (Reg_mic1.dest=H1) else '0';
    LOH2<='1'when (Reg_mic1.dest=H2) else '0';
                   end rtl2;
```

Le nombre de symboles décrits dans le type énuméré `val_cond` a du être augmenté pour représenter le fonctionnement du PLA :

– `PHI` pour indiquer que l'on ne souhaite pas tester une condition ;
– `NVTy` pour indiquer que l'on souhaite tester spécifiquement l'absence du profil `VTy`.

Cette extension du type énuméré reste compatible avec l'usage qui en est fait dans le chemin de données.

Cette description VHDL doit simuler le fonctionnement associatif du PLA. La difficulté provient du fait que VHDL ne connaît que des tableaux adressés et non des matrices dans lesquelles les lignes seraient directement sélectionnables par un vecteur.

Le temps de fonctionnement du PLA est simulé à l'aide d'une clause `after` qui simule la disponibilité des résultats vers la fin de la phase `PH2S` (mais ne garantit pas le maintien des entrées !).

9.8 AUTRES ORGANISATIONS POSSIBLES DE SÉQUENCEUR

Il existe plusieurs autres techniques pour réaliser le séquenceur d'un circuit complexe. En particulier, il peut être câblé ou microprogrammé.

9.8.1 Séquenceurs microprogrammés

La microprogrammation est une technique ancienne qui fut largement utilisée dans la conception des processeurs complexes de 1965 à 1985. Elle fut inventée par M.V. Wilkes en 1953 [WIL53]. Dans ce cas, l'algorithme qui décrit le comportement de la machine est inscrit dans une mémoire ROM sous la forme d'une suite de *micro-instructions* appelé *microprogramme* (figure 9.30). Les microinstructions contiennent la description des commandes à envoyer, à chaque étape, au chemin de données ainsi que la commande d'un mécanisme d'enchaînement qui prend en compte le résultat des tests. Cette technique permet la prise en compte d'algorithmes complexes pouvant avoir plusieurs milliers d'étapes. Elle permet aussi une certaine indépendance entre la structure du séquenceur et l'algorithme, ce qui autorise certaines modifications après coup.

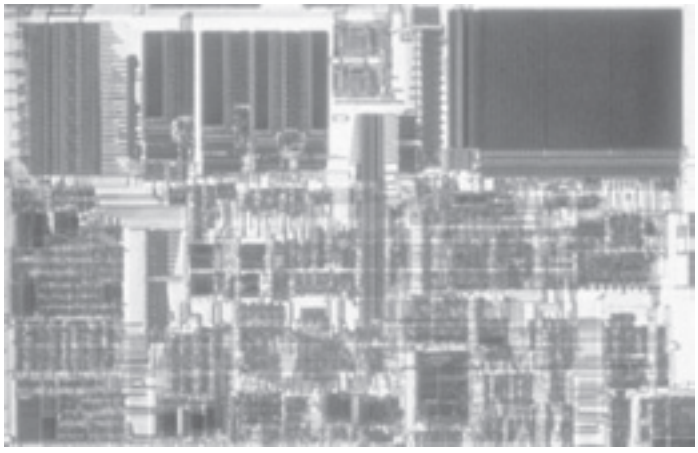


Figure 9.30 Séquenceur microprogrammé de l'Intel 38S®

a) Principe des séquenceurs microprogrammés

Les séquenceurs microprogrammés sont généralement des machines de Moore (figure 9.31). Chaque mot de la ROM constitue une microinstruction qui est constituée d'une suite de champs qui définissent les actions qu'elle déclenche lorsqu'elle est adressée. L'enchaînement séquentiel des microinstructions se fait de manière adressée en donnant l'adresse de la microinstruction suivante dans un champ particulier. Les enchaînements conditionnels se font en remplaçant certains bits de cette adresse par des bits de test. Ce remplacement est défini par un ou plusieurs champs particuliers

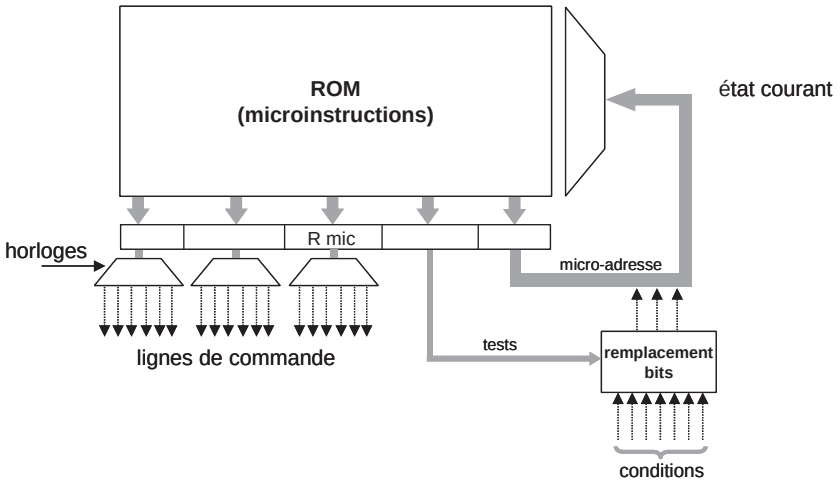


Figure 9.31 Principe d'un séquenceur microprogrammé

appelés *champs de test*. Cette technique permet des enchaînements rapides, mais impose des contraintes sur l'implémentation dans la ROM des microinstructions cibles de ces branchements. L'enchaînement adressé des microinstructions séquentielles permet de récupérer les trous laissés par l'implémentation, plus rigide, des branchements conditionnels.

Il existe des séquenceurs microprogrammés qui sont des machines de Mealy. Dans ce cas, certains champs ne commandent pas directement des actions mais des blocs de circuiterie qui génèrent des commandes en fonction du contenu de ces champs mais aussi d'entrées (conditions, codes, adresses…).

La technique de la microprogrammation permet aussi d'effectuer des décodages complexes, comme par exemple celui des codes instruction et utilisant directement ces codes pour adresser la ROM et sélectionner directement les séquences de microprogramme appropriées.

b) Séquenceur microprogrammé pour la montre

Le séquencement de la montre se prête bien à la réalisation d'un séquenceur microprogrammé, bien qu'il soit très simple (figure 9.32).

L'organigramme est le même que pour le séquenceur à base de PLA. Il contient 29 microinstructions dont l'enchaînement se fait soit de manière inconditionnelle, soit de manière conditionnelle par des branchements à 2 ou 3 voies. Ces branchements conditionnels sont les éléments les plus difficiles à disposer dans la ROM. Pour cela, ils sont traités au début de la phase d'implémentation du microprogramme dans la ROM. Ils sont implémentés par taille décroissante : celui de 3 voies, puis ceux de 2 voies et enfin l'enchaînement séquentiel inconditionnel. Il faut aussi tenir compte de l'implantation de la première microinstruction à l'adresse 0. Nous ferons l'hypothèse que le séquencement du séquenceur et du chemin de données est successif à quatre phases.

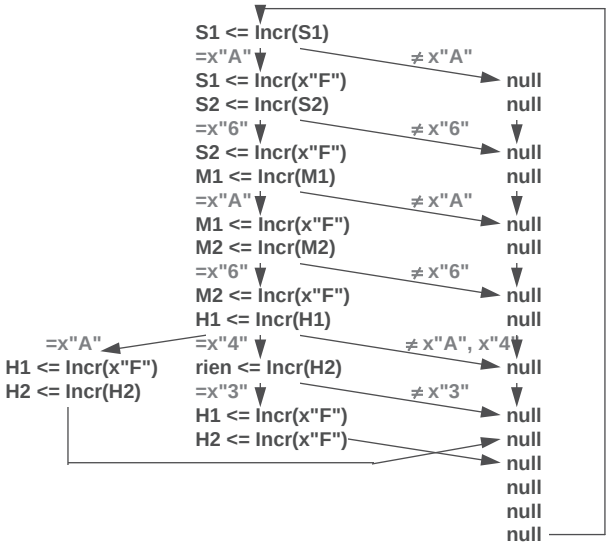


Figure 9.32

Un fonctionnement superposé aurait été possible, au prix d'une adaptation de l'algorithme.

Branchement à 3 voies : celui-ci se fera en remplaçant les deux bits d'adresse de poids faible par les deux conditions X4 et XA.

X4	XA	
0	0	=> Null
0	1	=> H1<= Incr(x"F")
1	0	=> Rien <= Incr(H2)
1	1	impossible

Ce branchement doit concerner un groupe de trois adresses consécutives se terminant par 00, 01 et 10. Comme l'adresse 0 est déjà occupée, le trio suivant est 4, 5 et 6.

Nous choisissons de mettre XA en remplacement du bit de poids faible car cette condition est aussi utilisée par un branchement à 2 voies.

Branchements à 2 voies : ces cinq branchements conditionnels se font en remplaçant le bit de poids faible d'adresse par XA, X6 ou X3. Les couples d'adresses doivent se terminer par 0 et 1 (figure 9.33). Les couples possibles sont :

– Pour XA : (2, 3) et (10, 11) ;
– Pour X6 : (8, 9) et (12, 13) ;
– Pour X3 : (14, 15).

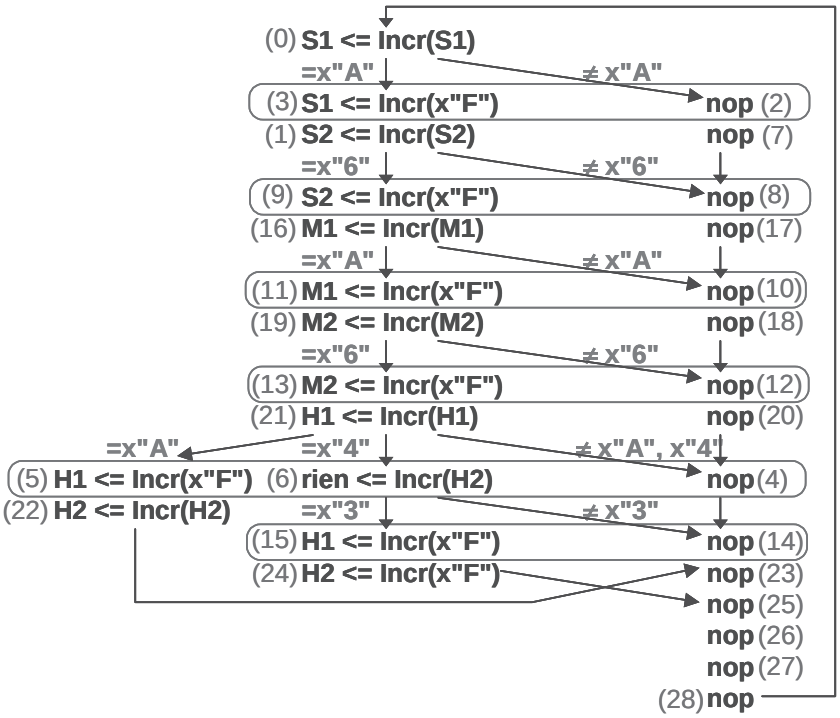


Figure 9.33 Micro-adresses affectées aux microinstructions

Format des microinstructions : celui-ci doit préciser la nature des actions commandées et l'enchaînement des microinstructions.

Plusieurs techniques peuvent être utilisées pour préciser cet enchaînement : la plus économique consiste à utiliser un champ pour donner les 3 bits de poids fort de l'adresse de la microinstruction désignée et deux autres champs pour spécifier la nature des 2 bits de poids faible. Le bit de poids le plus faible peut prendre les valeurs (0, 1, X3, X6, XA), ce qui est déterminé par un champ T1 de 3 bits. Le bit précédent peut prendre les valeurs (0, 1, X4), ce qui est déterminé par un champ T2 de 2 bits (figure 9.34).

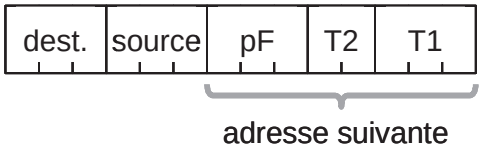


Figure 9.34 Format des microinstructions

Tableau 9.2 Implantation du microprogramme dans la ROM

add. impl.	dest.	source	add. suivante		
			poids forts	T2	T1
00000	S1	S1	000	1	XA
00001	S2	S2	010	0	X6
00010	rien	XF	001	1	1
00011	S1	XF	000	0	1
00100	rien	XF	011	1	0
00101	H1	XF	101	1	0
00110	rien	H2	011	1	X3
00111	rien	XF	010	0	0
01000	rien	XF	100	0	1
01001	S2	XF	100	0	0
01010	rien	XF	100	1	0
01011	M1	XF	100	1	1
01100	rien	XF	101	0	0
01101	M2	XF	101	0	1
01110	rien	XF	101	1	1
01111	H1	XF	110	0	0
10000	M1	M1	010	1	XA
10001	rien	XF	010	1	0
10010	rien	XF	011	0	0
10011	M2	M2	011	0	X6
10100	rien	XF	001	0	0
10101	H1	H1	001	X4	XA
10110	H2	H2	101	1	1
10111	rien	XF	110	0	1
11000	H2	XF	110	0	1
11001	rien	XF	110	1	0
11010	rien	XF	110	1	1
11011	rien	XF	111	0	0
11100	rien	XF	000	0	0

9.8.2 Séquenceurs câblés

Il s'agit de câbler avec des portes les fonctions *f* et *g* de l'automate qui constitue le séquenceur (figure 9.36). Cette technique produit des séquenceurs très rapides mais non modifiables. Elle n'est applicable qu'au cas des séquenceurs dont l'algorithme est assez simple. Ce type de séquenceur est beaucoup plus rapide de ceux constitués de PLA.

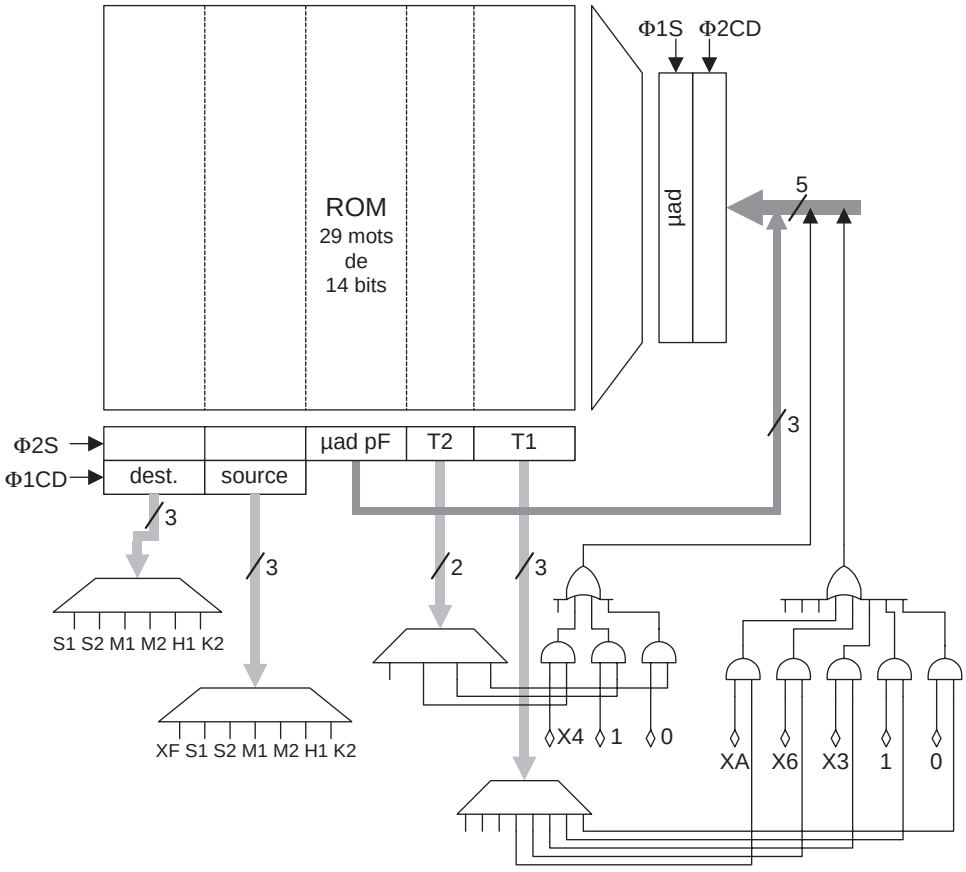


Figure 9.35 Schéma du séquenceur microprogrammé de la montre

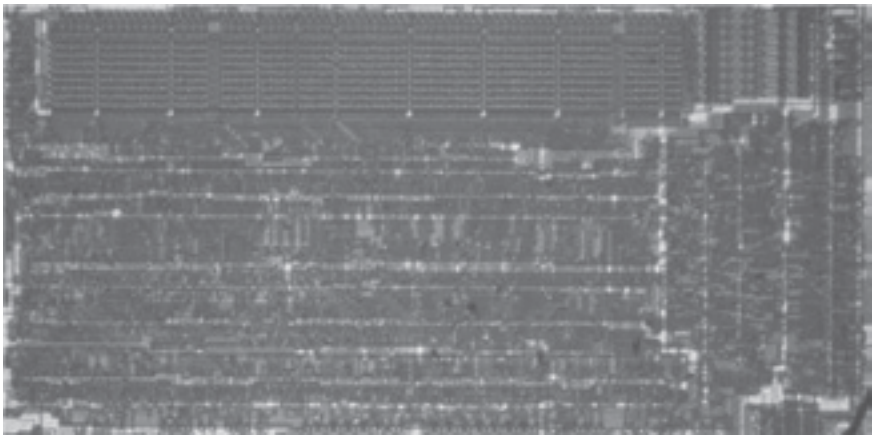


Figure 9.36 Séquenceur câblé du Motorola MC 6800®

Un façon efficace de réaliser des séquenceurs câblés comme celui de la montre est de câbler son algorithme sous la forme d'un « organigramme câblé » à l'aide de couples de latches constituant des bascules D maîtres-esclaves.

a) Séquenceur câblé pour la montre

Le séquencement de la montre se prête bien à la réalisation d'un séquenceur câblé optimisé. Pour cela, nous remarquerons que ce séquencement est constitué d'une boucle de 16 étapes (figure 9.37). Les différentes voies possibles de l'algorithme sont au nombre de 3 :

– La voie « normale » qui consiste à incrémenter les secondes, les minutes et les heures.
– La voie « ineffective » qui consiste en une chaîne d'instructions `null`.
– Une petite séquence « spéciale » pour traiter l'une des branches du cas particulier du passage des 24 heures.

Le séquenceur sera donc réalisé à partir :

– d'un *générateur d'instants* constitué d'un registre à décalage de 16 étapes. Celui-ci excite successivement des lignes qui représentent les 16 instants du cycle. Le signal RST a pour effet de positionner ce générateur sur l'étape T1 ;
– de trois *bascules de mode* (asynchrones) qui représentent le fait d'être dans l'une des 3 voies (*normale*, *spéciale*, *ineffective*) ;
– de portes qui génèrent les commandes et les changements de mode par la sélection des combinaisons entre les modes et les instants.

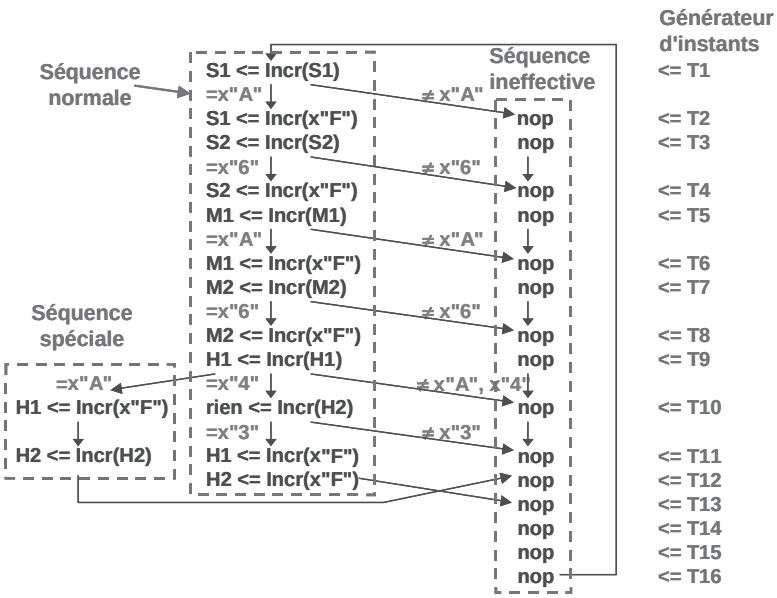


Figure 9.37

L'utilisation de bascules asynchrones est ici possible, compte tenu de la lenteur du circuit. Pour un circuit rapide, il faudrait utiliser des bascules synchrones (figure 9.38).

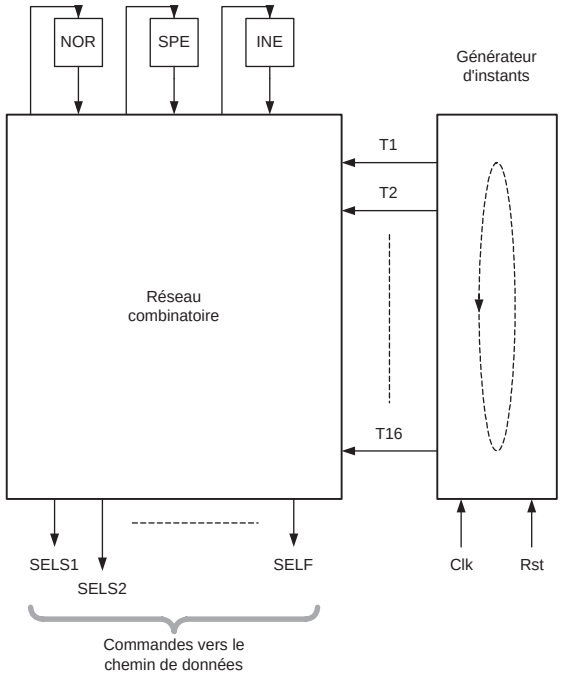


Figure 9.38 Organisation du séquenceur câblé de la montre

b) Description VHDL du séquenceur câblé de la montre

La description VHDL ci-dessous concerne un séquenceur câblé pour la montre, à l'exclusion du générateur d'instants. Elle décrit les conditions d'activation et de désactivation des bascules de mode, ainsi que la génération des signaux de commande du chemin de données. L'outil de synthèse utilisé ne permettait pas de décrire des bascules comme des fils isolés. Il a donc fallu les décrire comme des signaux explicitement rebouclés.

```
entity sequenceur is
port(
 T :in bit_vector(1 to 16); -- sorties du generateur d'instants
 C1, -- X"3"
 C2, -- X"4"
 C3, -- X"6"
 C4 :in bit; -- X"A"
 SELS1,SELS2,SELM1,SELM2,SELH1,SELH2,SELF :out bit; -- lignes de
selection
 LOS1,LOS2,LOM1,LOM2,LOH1,LOH2 :out bit );  -- lignes de chargement
end sequenceur;
```

```
architecture RTL of sequenceur is
signal Basc_Norm, Basc_Spe, Basc_Inef :bit;  -- bascules RS de mode
begin
-- Commande des bascules
Basc_Norm <= '1'when  T(1)='1' else
             '0'when  (C4='1' and T(10)='1') or
                      (C4='0' and (T(2)='1' or T(6)='1')) or
                      (C3='0' and (T(4)='1' or T(8)='1')) or
                      ((C4='0' and C2='0') and T(10)='1') or
                      (C1='0' and T(11)='1') or T(13)='1'
             else Basc_Norm;
Basc_Spe  <= '1'when  C4='1' and T(10)='1'else
             '0' when T(12)='1' or T(1)='1' -- T(1) pour le reset
             else Basc_Spe;
Basc_Inef <= '1'when (C4='0' and (T(2)='1' or T(6)='1')) or
                     (C3='0' and (T(4)='1' or T(8)='1')) or
                     ((C4='0' and C2='0') and T(10)='1') or
                     (C1='0' and T(11)='1') or T(13)='1' or
                     (T(12)='1' and Basc_Spe='1') else
             '0' when T(1)='1' else Basc_Inef;
-- Positionnement des lignes de sortie
SELS1 <= '1' when T(1)='1' else '0';
SELS2 <= '1' when T(3)='1' and Basc_Norm='1' else '0';
SELM1 <= '1' when T(5)='1' and Basc_Norm='1' else '0';
SELM2 <= '1' when T(7)='1' and Basc_Norm='1' else '0';
SELH1 <= '1' when T(9)='1' and Basc_Norm='1' else '0';
SELH2 <= '1' when (T(10)='1' and Basc_Norm='1') or
                  (T(11)='1' and Basc_Spe='1') else '0';
SELF  <= '1' when T(2)='1' or T(4)='1' or T(6)='1' or
                  T(8)='1' or Basc_Inef='1' or
                  (T(10)='1' and Basc_Spe='1') or
                  ((T(11)='1' or T(12)='1') and Basc_Norm='1')
        else '0';
LOS1  <= '1' when  T(1)='1' or (T(2)='1' and Basc_Norm='1') else '0';
LOS2  <= '1' when (T(3)='1' or T(4)='1') and Basc_Norm='1' else '0';
LOM1  <= '1' when (T(5)='1' or T(6)='1') and Basc_Norm='1' else '0';
LOM2  <= '1' when (T(7)='1' or T(8)='1') and Basc_Norm='1' else '0';
LOH1  <= '1' when (T(9)='1' and Basc_Norm='1') or
                  (T(10)='1' and Basc_Spe='1') or
                  (T(11)='1' and Basc_Norm='1') else '0';
LOH2  <= '1' when (T(11)='1' and Basc_Spe='1') or
                  (T(12)='1' and Basc_Norm='1') else '0';
end RTL;
```

c) Schéma du séquenceur câblé de la montre

Le schéma ci-contre du séquenceur a été obtenu à l'aide d'un outil de synthèse. Les rebouclages qui constituent les trois bascules sont en gras. Le générateur de temps, qui est un simple registre à décalage, n'est pas représenté (figure 9.39). Il convient de noter la simplicité de ce schéma qui le rend très compétitif vis-à-vis d'un PLA.

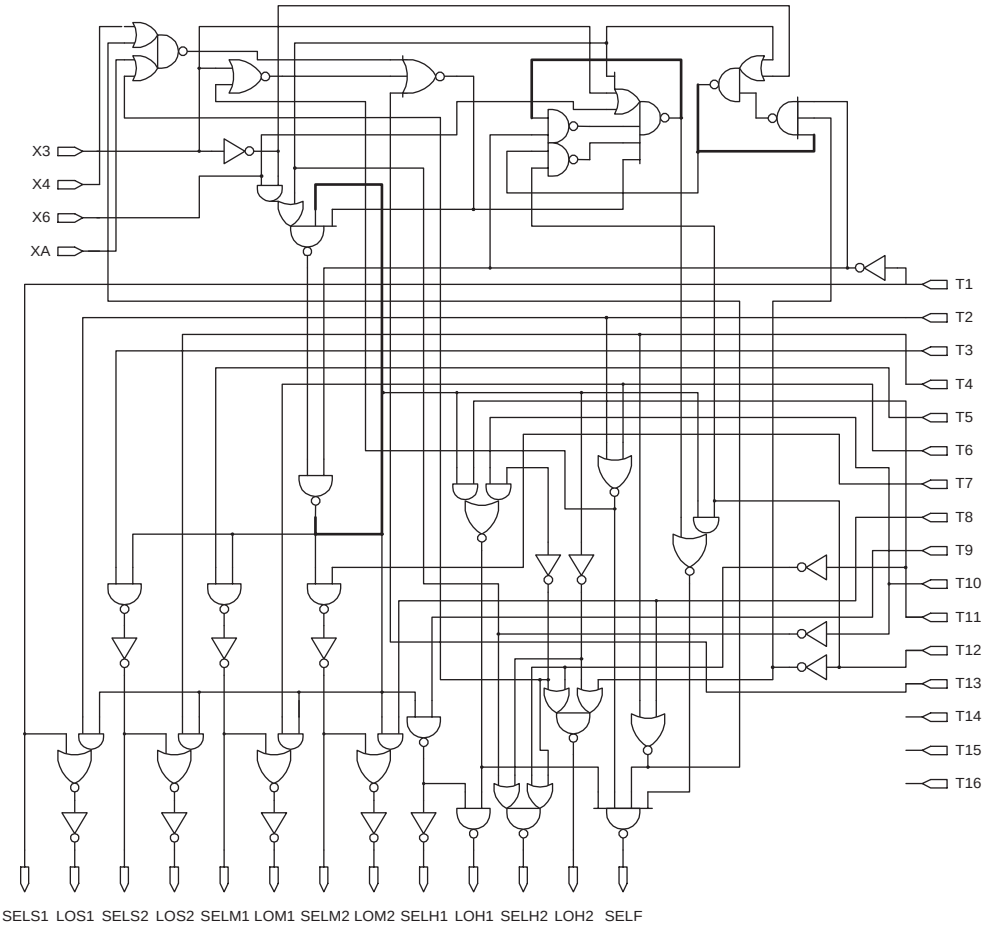


Figure 9.39 Schéma du séquenceur câblé de la montre (compilé)

Ce type de séquenceur permet une grande optimisation et un fonctionnement rapide puisque ses changements d'état résultent de l'évolution de quelques bascules. Toutefois, il peut apparaître des portes complexes pour regrouper les commandes générées par les différentes étapes à destination du chemin de donnée. Les portes ayant un très grand nombre d'entrées sont difficiles à réaliser en CMOS classique. Des solutions dynamiques peuvent être utilisées, mais elles compliquent le test et demandent une vitesse de fonctionnement minimale, certainement non compatible avec la montre. Des portes en « pseudo NMOS » peuvent aussi être utilisées, mais elles augmentent la consommation. Le séquencement doit être étudié dans le détail pour permettre les changements de mode sans générer de commandes parasites. Nous voyons qu'il permet un fonctionnement successif du chemin de données et du séquenceur. La grande rapidité du séquenceur permet toutefois de rester biphasé. L'influence des parasites en début de cycle sur les commandes de sélection devra être soigneusement étudiée et éventuellement éliminée par un formatage adéquat de ces signaux (figure 9.40).

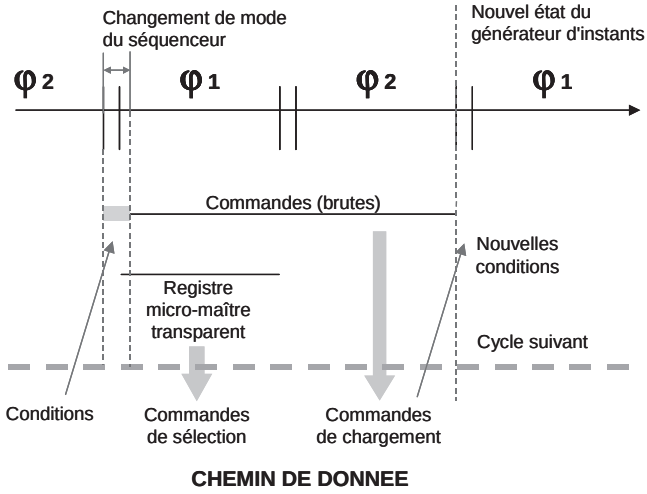


Figure 9.40 Fonctionnement du séquenceur câblé

Ce type de séquenceur peut être étendu à des séquences de plusieurs durées possibles en introduisant des aiguillages dans le générateur d'instants. Il convient pour des algorithmes de faible complexité. Il existe plusieurs outils qui permettent sa synthèse automatique à partir de la description du comportement.

9.9 DESSIN DES SÉQUENCEURS

Contrairement au chemin de données, il n'existe pas de technique spécifique pour réaliser le dessin d'un séquenceur. Celui-ci ne constitue pas vraiment un bloc de dessin mais plutôt l'assemblage de modules séparés tels que des ROM, des PLA, des décodeurs et de la logique non structurée. Le dessin d'un séquenceur est obtenu par l'utilisation des techniques classiques de placement et d'interconnexion des différents blocs qui le constituent.

La forme de certains blocs pourra être optimisée pour faciliter leur interconnexion. Par exemple, un PLA peut être dessiné de manière à répartir ses sorties sur son côté. Ceci permet d'adapter la position de ces sorties à la topologie des blocs auxquels elles doivent être connectés (figure 9.41).

Une optimisation supplémentaire consiste à enlever les parties de la matrice ET qui ne contiennent que des φ (c'est-à-dire aucun transistor).

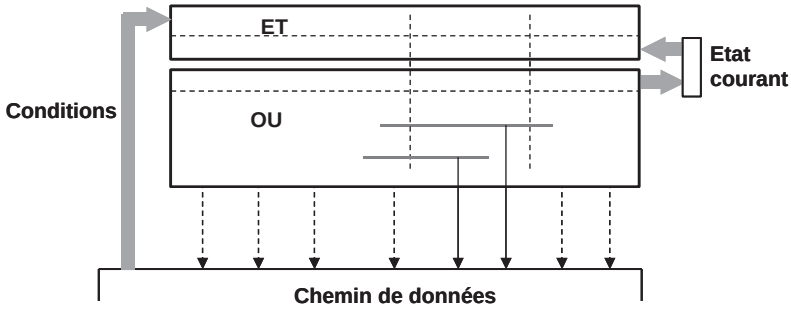


Figure 9.41 PLA à sorties latérales

BIBLIOGRAPHIE

[WIL53] M.V. Wilkes, J.B. Stringer, Microprogramming and the design of the Control Circuits in an Electronic Digital Computer, *Proc. Cambridge Phil. Soc. pt 2*, 49, pp. 30-38, Manchester, 1953.

Chapitre **10**

Mécanismes d'horlogerie

10.1 MÉCANISMES « CLASSIQUES » D'HORLOGERIE

La performance et le bon fonctionnement des circuits intégrés complexes sont souvent liés à la qualité de leurs mécanismes d'horlogerie.

Un mécanisme d'horlogerie classique comprend (figure 10.1) :

– Un oscillateur, souvent piloté par un quartz ou un résonateur piézo-électrique externe (figure 10.2). Les quartz et les résonateurs permettent de réaliser des oscillateurs jusque vers 150 MHz. L'oscillateur est souvent réalisé par un couple d'inverseurs polarisés dans leurs zones de conduction à Vcc/2 et formant un amplificateur. Le quartz, ou le résonateur, reboucle ces inverseurs, provoquant une contre-réaction positive pour sa fréquence de résonance. Le gain de l'amplificateur formé par le couple d'inverseur doit être suffisamment important pour que l'oscillation démarre facilement, mais pas trop pour maintenir un signal sinusoïdal et éviter l'apparition d'harmoniques. Certains circuits fonctionnent avec un oscillateur externe.
– Un circuit de formatage, généralement constitué par un inverseur qui se comporte comme un amplificateur de gain important. Celui-ci amplifie et écrête le signal sinusoïdal pour en faire un signal logique.

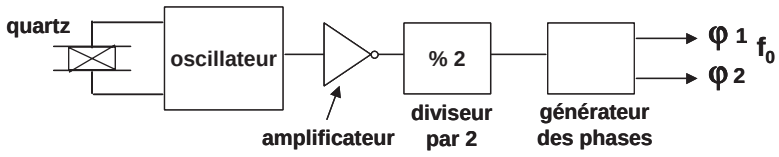


Figure 10.1 Mécanisme « classique » d'horlogerie

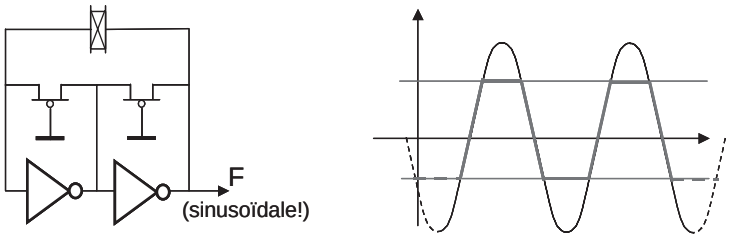


Figure 10.2 Oscillateur à quartz (à gauche) ;
sortie de l'oscillateur et seuillage par l'amplificateur (à droite)

– Un diviseur par 2 fournit un signal carré de rapport cyclique égal à 1. Ce qui signifie que la fréquence de l'horloge interne au circuit est généralement la moitié de celle du quartz.
– Un générateur de phases fabrique ces signaux en respectant les temps de non-recouvrement.

Les amplificateurs de sortie du générateur de phase sont rebouclés pour éviter tout recouvrement des phases (figure 10.3). L'installation de retards (couples d'inverseurs) dans ces rebouclages permet d'élargir les temps de non-recouvrement.

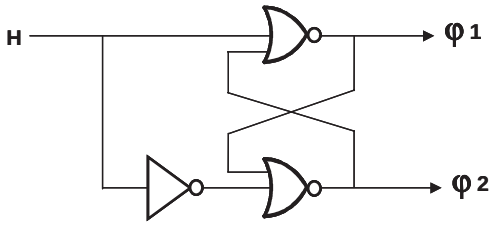


Figure 10.3 Générateur de phases biphasé

Si le nombre de phases est supérieur à 2, le générateur de phases est généralement bâti autour d'un compteur de Johnson très facile à décoder (figure 10.4).

Dans les circuits de petite taille, la distribution des signaux d'horloges se fait à partir d'un ensemble d'amplificateurs uniques et centralisés de manière à éviter d'introduire des retards entre les phases distribuées. Ces amplificateurs d'horloge sont de taille importante et contribuent de manière significative à la consommation globale du circuit.

10.2 HORLOGERIE DES CIRCUITS RAPIDES ET COMPLEXES

L'augmentation de la puissance de calcul des microprocesseurs suit depuis trente ans une loi exponentielle, appelée *loi de Moore*, sans aucune marque de fléchissement.

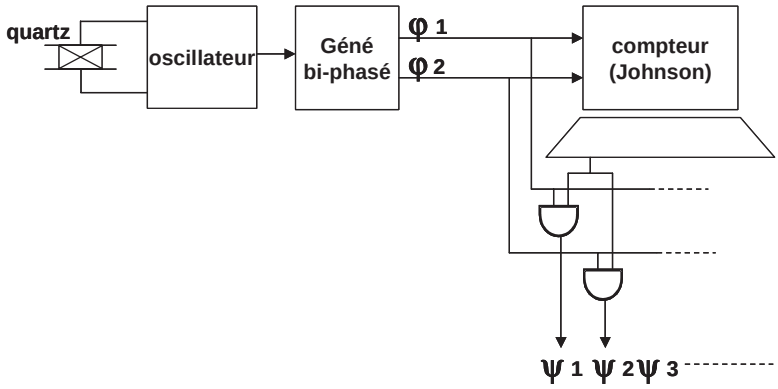


Figure 10.4 Générateur de phases

Plus des 2/3 de l'évolution de la puissance de calcul de ces machines provient de l'accroissement régulier de leur fréquence d'horloge [GES01]. La fréquence de ces circuits rapides atteint aujourd'hui des valeurs, proches des limites des technologies utilisées (figure 10.5). De telles fréquences, qui se situent maintenant dans le domaine des micro-ondes, étaient inimaginables il y a seulement une quinzaine d'années. Il faut toutefois remarquer que les microprocesseurs conçus pour les stations de travail représentent une certaine forme de « monstruosité » dans le monde de la conception des circuits intégrés. En effet, de telles performances ne sont obtenues qu'au prix de la dissipation d'une puissance énorme (tableau 10.1) qui nécessite la ventilation forcée de ces circuits et bientôt l'utilisation d'un refroidissement par liquide.

Tableau 10.1 Consommation de processeurs X86

Processeur	Horloge (MHz)	Techno (µm)	Conso (W)
I 386	16	1,5	3
I 486	33	1	6
Pentium	66	0.7	13
Pentium III	1 000	0,18	26
P4 Willamette	2 000	0.18	75
P4 Northwood	2 200	0.13	50

Le développement de versions basse-consommation de ces processeurs, pour les ordinateurs portables, montre qu'il est possible de réduire fortement leur consommation (par un facteur voisin de dix) au prix d'une division de leurs performances par un facteur voisin de deux. L'industrie a opté pour cette seconde classe de machines. Celles-ci règnent dans les applications embarquées. Il existe toutefois un marché important pour la première classe de processeurs malgré leurs inconvénients. Leur

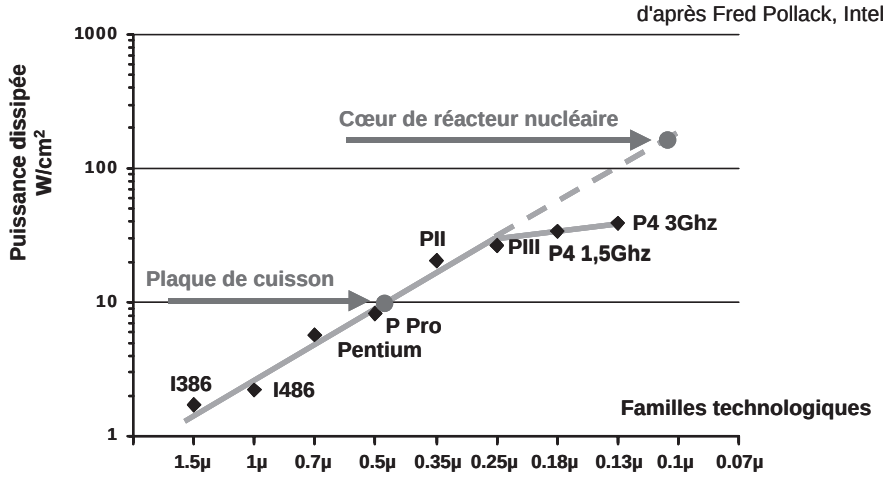


Figure 10.5 Évolution de la puissance dissipée par unité de surface des processeurs Intel X86 (d'après [POL99])

évolution ne peut se poursuivre qu'au prix de l'utilisation, dans un avenir assez proche, de techniques de refroidissement complexes. Toutefois, l'avenir de cette gamme de machines ne semble pas assuré à long terme [POL99].

Les techniques utilisées pour réduire la consommation d'un circuit se divisent en trois groupes complémentaires :

– Réduction de la fréquence d'horloge à une valeur « normale ». L'étude de la circuiterie CMOS suggère l'emploi d'une fréquence d'horloge raisonnable qui correspond à un dimensionnement normal des transistors. Cette fréquence est environ moitié de celle des circuits hyper-rapides.
– Élimination de la logique non complémentaire (pseudo NMOS). Des alimentations temporaires permettent de réduire la consommation des structures matricielles (ROM, PLA).
– Réduction importante de la fréquence d'horloge lorsque le processeur est en attente (ce qui est assez fréquent !). Cette technique peut aussi s'appliquer à des blocs fonctionnels dont on coupe l'horloge lorsqu'ils ne sont pas utilisés (délestage).

À titre d'exemple, le Mobile Intel Pentium III® faible consommation dissipe moins de 1 W à 500 MHz, au lieu des 26 W du Pentium III® « normal » à 1 Ghz.

Il est bon de rappeler que la consommation de la circuiterie CMOS est directement liée à sa fréquence de fonctionnement. Toutefois, dans les circuits géants la longueur relative des lignes de communication devient telle qu'elles doivent être attaquées par des amplificateurs puissants si l'on souhaite que la performance du circuit soit maximale. Dans ce cas, l'augmentation de puissance consommée n'est plus linéraire relativement à la fréquence d'horloge, mais exponentielle.

a) Utilisation de multiplicateurs de fréquence (PLL)

Lorsque la fréquence interne d'un circuit complexe dépasse celle qu'il est possible d'atteindre avec un oscillateur à quartz, on utilise un multiplicateur de fréquence. Il s'agit d'un système asservissant la fréquence d'un oscillateur interne sur celle, plus faible, d'un oscillateur à quartz. De tels systèmes, appelés PLL (pour *Phase Locked Loop*) ont été introduits au début des années 1990 pour l'horlogerie des microprocesseurs. Ils ont permis d'atteindre des fréquences d'horloge de plusieurs giga-hertz. Des fréquences d'horloge de l'ordre de 50 à 60 Ghz pourraient être atteintes en 2010 pour les microprocesseurs les plus rapides. Toutefois, l'augmentation de la puissance dissipée vers des limites intolérables et l'orientation du marché vers les machines portables a conduit les principaux fabricants de microprocesseurs à limiter la fréquence à 3 Ghz et à rechercher l'augmentation des performances *via* celle de la complexité (structures multiprocesseurs et opérateurs spécialisés) (figure 10.6).

Un PLL est constitué d'une boucle d'asservissement comprenant (figure 10.7) :

– Un oscillateur dont la fréquence peut être ajustée électriquement (appelé VCO pour *Voltage Controlled Oscillator*) et qui fournit la fréquence désirée.
– Un diviseur qui divise la fréquence du VCO par un coefficient N (fixe).

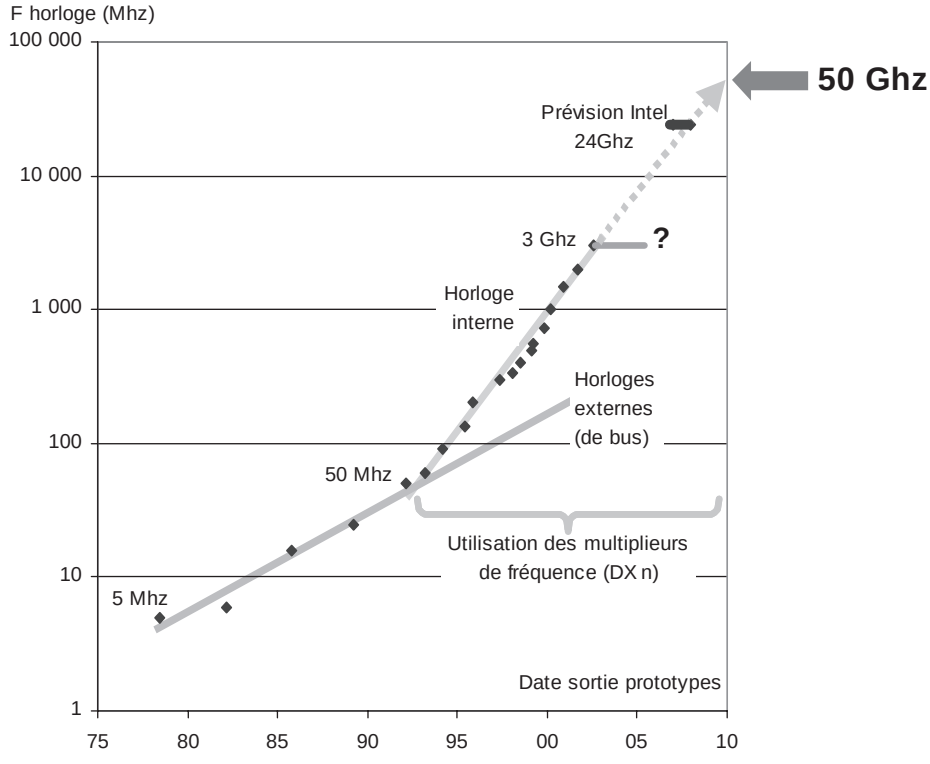


Figure 10.6 Évolution de la fréquence d'horloge des processeurs Intel X86

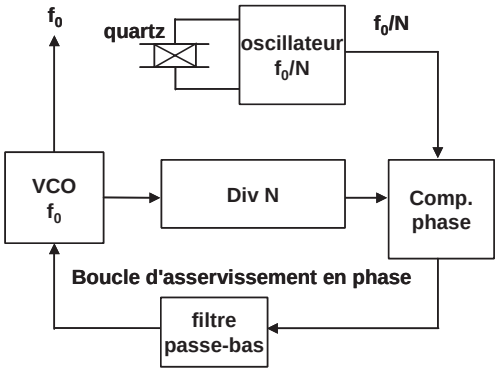


Figure 10.7 Multiplicateur de fréquence à PLL

– Un comparateur de phase (de fréquence) qui compare la fréquence divisée avec celle d'un oscillateur à quartz de référence.
– Un filtre passe-bas qui génère la tension de commande du VCO à partir de la sortie du comparateur de phase.

De cette manière, la fréquence de la sortie du VCO se trouve verrouillée à être N fois celle de l'oscillateur à quartz.

10.2.1 Notion de zone isochrone

On appelle *zone isochrone*, la zone maximale d'un circuit intégré que l'on peut synchroniser avec un seul générateur d'horloge et un seul jeu d'amplificateurs (figure 10.8). Au-delà de cette zone, les déphasages entre les horloges et entre les signaux ne permettent plus de synchroniser la circuiterie.

Un circuit complexe sera donc constitué de plusieurs zones isochrones (à titre d'exemple, le microprocesseur Intel Pentium 4® est constitué de 47 zones isochrones [KUR01]). Les zones isochrones ont souvent un sens fonctionnel, c'est-à-dire qu'elles correspondent à des fonctions bien identifiées. Cela présente deux avantages :

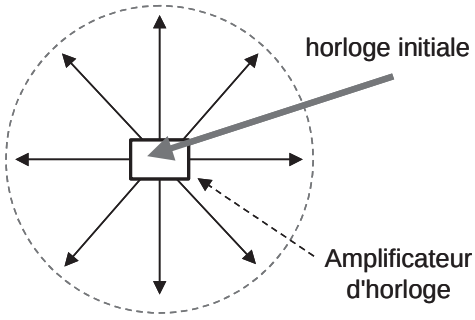


Figure 10.8 Zone isochrone

– Les zones fonctionnelles correspondent généralement à des maxima de connexions internes, ce qui minimise les connexions inter-zones isochrones.
– Elles constituent des blocs réutilisables pour la conception d'autres circuits.

Chacune des zones isochrones d'un circuit complexe sera excitée par son propre générateur d'horloge. Le dialogue entre les zones isochrones n'est possible que si ces générateurs respectent une relation de phase précise entre eux (figure 10.9).

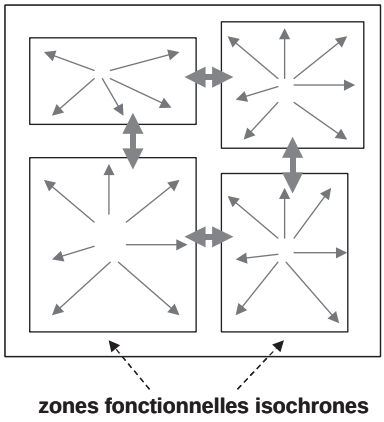


Figure 10.9 Communication entre les zones isochrones d'un circuit complexe

La communication entre deux zones isochrones voisine consiste à émettre un signal par une zone isochrone et à le charger dans un registre de l'autre (figure 10.10). On remarque que si les horloges de ces deux zones isochrones sont en phase, les conditions temporelles à cette communication sont exactement les mêmes que celles qui régissent le transfert d'un signal dans une zone isochrone, bien que la génération et la réception du signal se fassent avec des horloges différentes.

Globalement, ce genre de circuit continue à satisfaire les conditions de synchronisme pour des communications à faible distance. Seules, celles à grande distance doivent faire l'objet de mesures particulières, par exemple en utilisant plusieurs périodes d'horloge pour un transfert.

10.2.2 Distribution de l'horloge

Pour respecter la condition précédente, il importe que les générateurs d'horloge des zones isochrones soient exactement en phase. Pour cela, il faut que la distribution de l'horloge à partir de l'horloge principale vers les générateurs des zones isochrones respecte cette condition (figure 10.11). On parlera alors de distribution *équilibrée* de l'horloge.

La génération et la distribution de l'horloge occupent une place importante dans les circuits rapides (jusqu'à 30 % de la surface) et sont responsables d'une grande partie

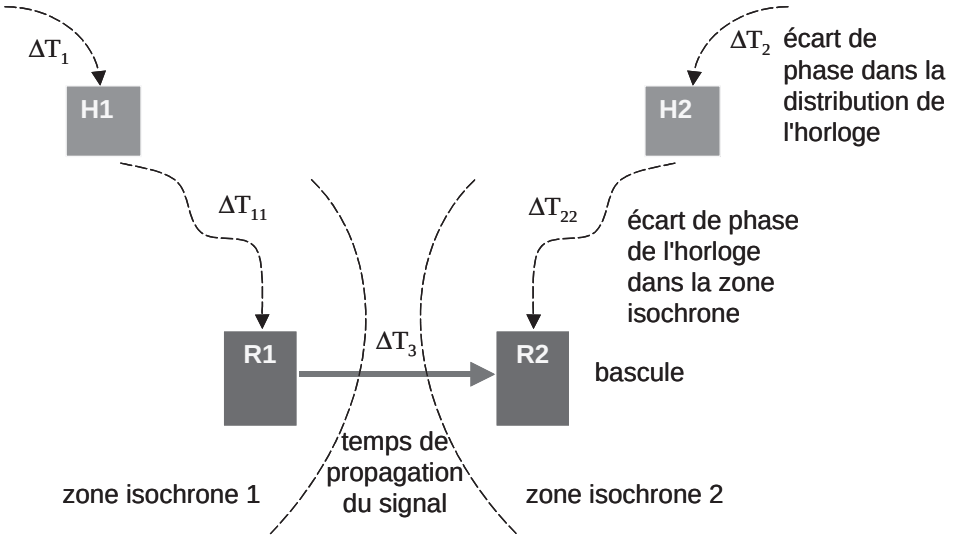


Figure 10.10 Transfert de données entre deux zones isochrones voisines

de leur consommation (40 à 70 % de la puissance dissipée). Plusieurs techniques permettent de réaliser une distribution équilibrée de l'horloge [FRI01] :

– Des plans ou des grilles de distribution. Cette technique, utilisée dans le microprocesseur Alpha 21 264® présente l'inconvénient de présenter une capacité importante, source de consommation importante [BAI98]. Elle a ensuite été utilisée pour distribuer l'horloge au niveau des zones isochrones [RUS00].
– Une arborescence d'amplificateurs.
– Une distribution dite en « H » (figure 10.12). Cette technique de distribution arborescente, maintenant universellement adoptée, consiste en une succession de distributions binaires assurant que les différents chemins de distribution de l'horloge sont équivalents (lorsque les zones isochrones sont des carrés identiques).

Comme la forme et la taille des zones isochrones ne sont pas identiques, l'équilibrage parfait de la distribution d'horloge est impossible. L'utilisation de fréquences d'horloge très élevées (plusieurs giga-hertz) rend ces écarts inacceptables. Pour compenser les dérives résiduelles, il est possible de munir chaque branche de l'arbre de distribution de son propre circuit de déphasage programmable (figure 10.13).

Le microprocesseur Pentium 4® semble utiliser une telle technique [KUR01]. Au moment du test du circuit, des comparateurs de phase semblent fournir la mesure du déphasage relatifs des horloges des différentes zones isochrones. Des déphaseurs programmables seraient alors positionnés pour ramener les déphasages à être inférieurs à 50 ps.

Une telle technique présente l'inconvénient que lorsque le circuit vieillit, ses caractéristiques technologiques évoluent et les réglages initiaux peuvent ne plus convenir. Une technique encore plus élaborée, appelée *multi-PLL*, consiste à utiliser les compa-

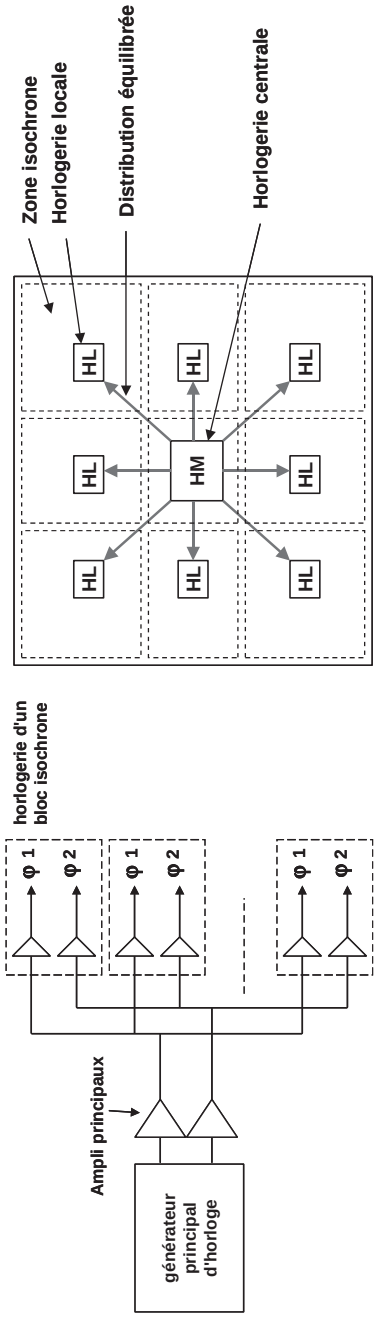


Figure 10.11 Distribution de l'horloge aux zones isochrones

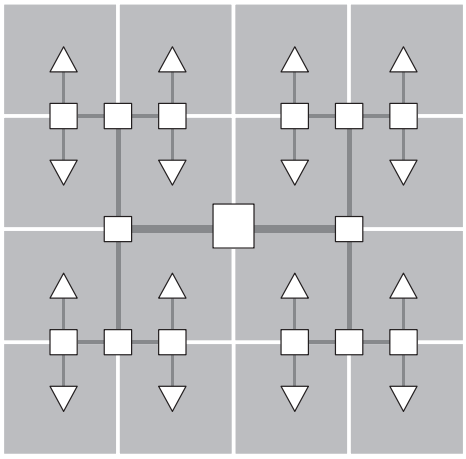


Figure 10.12 Distribution de l'horloge à l'aide d'un réseau en « H »

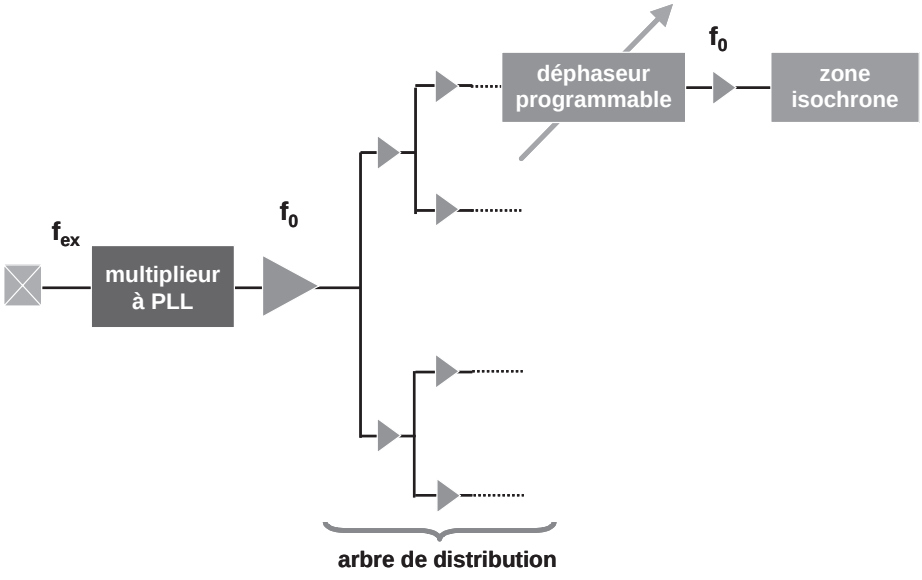


Figure 10.13 Resynchronisation de l'horloge distribuée au niveau de chaque zone isochrone

rateurs de phase et les déphaseurs pour asservir continuellement les phases des horloges des zones isochrones pour qu'elles restent en phase [SAI01, SWA01]. Un tel asservissement global est toutefois difficile à mettre en œuvre (figure 10.14).

Comme la distribution d'une horloge représente l'un des facteurs principaux de la puissance dissipée par les circuits de haute performance, il est possible de ne distribuer qu'une fréquence, plus basse, de synchronisation qui sera multipliée par chaque zone

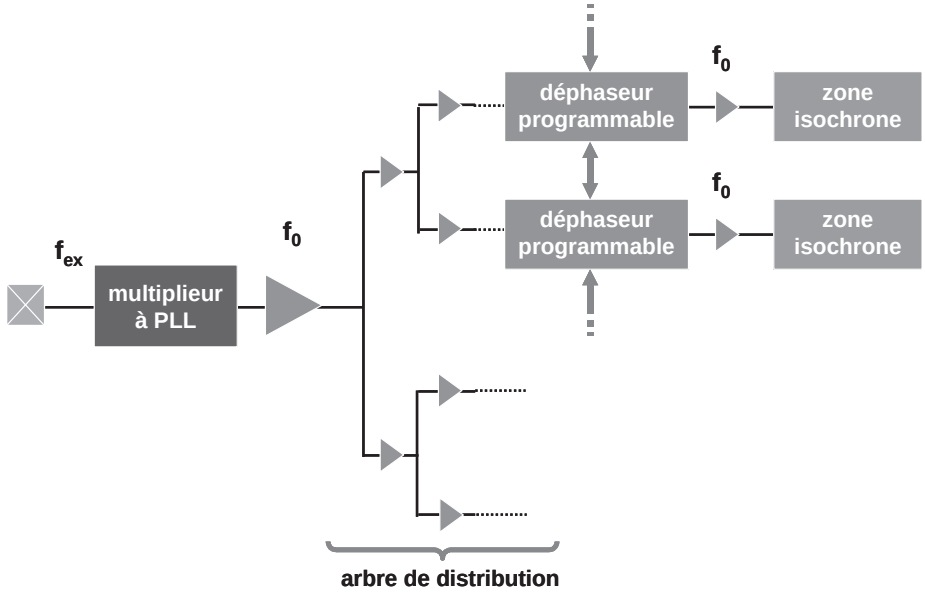


Figure 10.14 Asservissement des phases des horloges entre les zones isochrones

isochrone. L'interconnexion de ces multiplieurs permettra de réaliser leur asservissement en phase. L'énergie dissipée par le réseau de distribution est définie par la précision que l'on souhaite obtenir pour chaque transition. En gardant la même précision, la diminution de la fréquence transmise entraînera une diminution correspondante de la puissance dissipée par le réseau de distribution [ANC03] (figure 10.15).

10.3 VERS LE FUTUR

L'évolution de la vitesse d'horloge des microprocesseurs est un phénomène impressionnant qui bouleverse tous les avis d'expert [FIS98]. Il pourrait encore se poursuivre au moins jusqu'à 2010 par l'utilisation de fréquences qui sont encore impensables pour la synchronisation des ordinateurs.

L'utilisation de telles fréquences ferait intervenir des phénomènes de propagation du type de ceux que l'on rencontre dans le domaine des micro-ondes. Les réseaux de distribution d'horloge devraient alors faire appel à des techniques très particulières pour permettre un fonctionnement synchronisé de ces machines.

Toutefois, l'évolution de la puissance dissipée vers des valeurs inacceptables et le basculement du marché vers le matériel portable, à conduit à une limitation (actuelle) de la fréquence d'horloge aux environs de 3 Ghz, au profit des structures à traitement multiples. Lorsque les puissances dissipées redeviendront raisonnables, il n'est pas exclu que la course à la fréquence d'horloge reprenne.

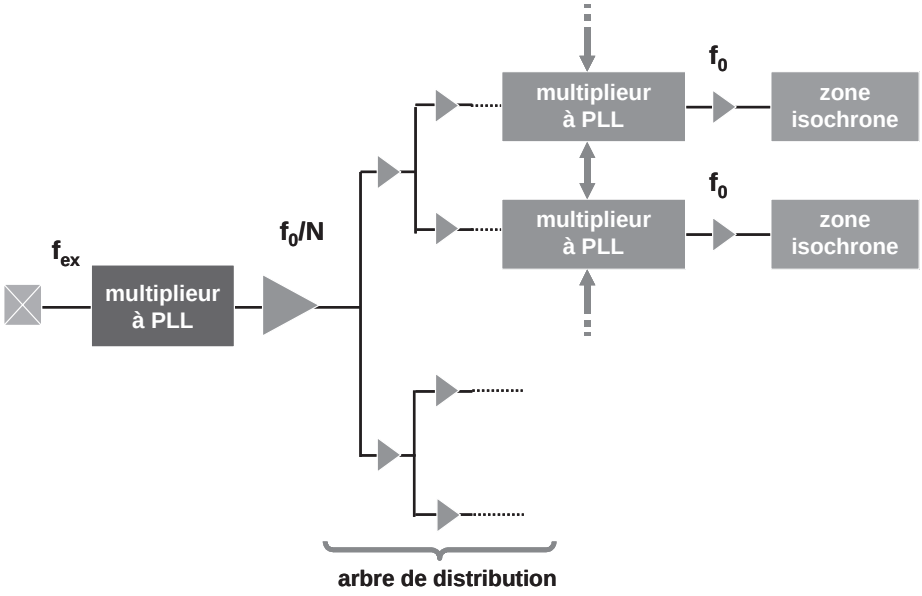


Figure 10.15 Réduction de la puissance dissipée par la distribution d'une fréquence d'horloge réduite

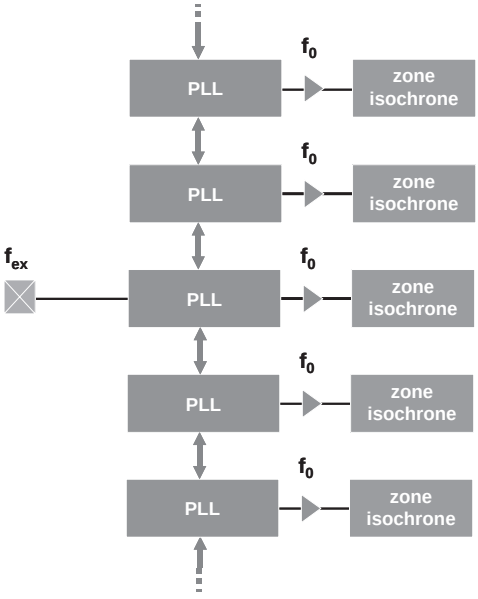


Figure 10.16 Multi-PLL

La distribution d'une horloge centralisée pourrait céder le pas à un réseau de PLL interconnectés générant des horloges locales toutes en phase. Seul, l'un de ces PLL reçoit une horloge de référence pour fixer la fréquence de l'ensemble. De tels réseaux sont appelés des *multi-PLL* [GIL95] (figure 10.16).

BIBLIOGRAPHIE

[ANC82] F. Anceau, A synchroneous approach for clocking VLSI systems, *IEEE journal of Solid State Circuits*, vol. SC-17, n° 1, Feb. 1982.

[GIL95] J. Nguyen Gill, A. Pratt, Distributed Synchronous Clocking, *IEEE transaction on parallel and distributed systems*, Vol. 6, n° 3, march 1995.

[HAE98] V. von Kaenel, D. Aebischer, R. van Dongen and C. Piguet, A 600 Mhz CMOS PLL Microprocessor Clock Generator with a 1.2 Ghz VCO, *ISSCC98 Conference*, session 25, Feb. 1998.

[FIS98] P.D. Fisher and R. Nesbill, The Test of Time, *Circuits & Devices*, March 1998.

[BAI98] D.W. Bailey and Bradley J. Benschneider, Clocking Desing and Analysis for a 600-Mhz Alpha Microprocessor, *IEEE journal of Solid State Circuits*, vol. 33, n° 11, Nov. 1998.

[POL99] F. Pollack, New Microarchitectures Challenges in the Coming Generations of CMOS Process Technologies, *Micro32*, 1999.

[BEN00] B.J. Benschneider *et al.*, A 1Ghz Alpha Microprocessor, *ISSCC2000 Conference*, session 5, Feb. 2000.

[RUS00] S. Rusu and S. Tam, Clock Generation and Distribution for the First IA-64 Microprocessor, *ISSCC2000 Conference*, session 10, Feb. 2000.

[GUT00] V. Gutnick, Active Ghz Clock Network Using Distributed PLLs, *IEEE journal of Solid State Circuits*, vol. 35, n° 11, Nov. 2000.

[GES01] P.P. Geslinger, Microprocessors for the New Millennium : Challenges, Opportunities, and the New Frontiers, *ISSCC2001 Conference*, session 1, Feb. 2001.

[FRI01] E.G. Friedman, Clock Distribution Network in Synchronous Digital Integrated Circuits, *Proceeding of the IEEE*, vol. 89, n° 5 May, 2001.

[RES01] P.J. Restle *et al.*, A Clock Distribution Network for Microprocessors, *IEEE journal of Solid State Circuits*, vol. 36, n° 5, May 2001.

[SAI01] M. Saint-Laurent and M. Swaminathan, A Multi-PLL Clock Distribution Architecture for Gigascale Integration, *IEEE Computer Society Workshop VLSI 2001*, May 2001.

[SWA01] M. Saint-Laurent, M. Swaminathan and James D. Meindl, On the Micro-Architectural Impact of Clock Distribution Using Multiple PLLs, *ICCD 2001 Conference*, Sept. 2001.

[KUR01] N.A. Kurd *et al.*, A Multigigahertz Clocking Scheme for the Pentium 4 Microprocessor, *IEEE Journal of Solid-State Circuits*, vol. 36, n° 11, Nov. 2001.

[ANC03] F. Anceau, Une technique de réduction de la puissance dissipée par l'horlogerie des circuits complexes rapides, 4es journées d'étude Faible Tension Faible Consommation, Paris, 16 mai 2003.

Chapitre **11**

Outils et méthodes de conception des circuits intégrés complexes

11.1 CONTEXTE

La conception de circuits intégrés complexes est une activité en plein développement. Elle est pratiquée chez :

– les fabricants de circuits intégrés (ST Microelectronics, Philips, Infineon, Atmel, Intel, Texas, Freescale….) pour :
  • les circuits de leurs catalogues respectifs,
  • des circuits spécifiques commandés par des entreprises spécialisées dans la conception de systèmes (télécom, militaires, aérospatial, grand public, informatique…) ;
– les grands fabricants de systèmes qui conçoivent des circuits spécifiques pour leurs équipements ;
– de petites entreprises qui vendent des circuits très spécialisés (par exemple, Transmeta) qu'ils conçoivent mais qu'ils font fabriquer chez des *fondeurs* (entreprises qui ne se chargent que de la réalisation des circuits).

La conception de circuits intégrés se divise en deux grands domaines :

– La conception automatisée de circuits, encore appelée la *compilation de silicium*. Dans cette approche, le travail de conception se restreint à écrire une spécification du circuit (en VHDL), à la vérifier, puis à surveiller le travail des outils de conception automatique qui fournissent le dessin des masques. Ce type de conception de circuit intégrés se présente comme une évolution technologique de la conception des cartes électroniques dont elle reprend la même approche méthodologique

(approche monophasée, bibliothèque de portes, peu ou pas de prise en compte des problèmes topologiques). Le travail de conception se trouve fortement allégé, mais les circuits résultants sont faiblement optimisés. Ces approches sont réservées aux circuits produits en faible série (moins d'un million d'exemplaires par an).

– La conception manuelle au niveau des masques, encore appelée *custom* (ou *full custom*). Dans cette approche, le travail de conception consiste à obtenir des circuits très optimisés en exploitant au maximum toutes les spécificités de la circuiterie VLSI. Cette approche est réservée aux circuits produits en très grande série (par exemple les microprocesseurs) ou ceux qui doivent être particulièrement optimisés (en surface ou en vitesse). La conception d'un circuit de ce type est beaucoup plus longue et complexe que celle d'un circuit conçu automatiquement, mais les performances obtenues sont plus importantes, et la surface de silicium occupée plus faible. Toutefois, pour quand même réduire le coût de conception des circuits complexes, certains blocs des circuits custom sont réalisés avec des outils automatiques.

11.2 LA MAÎTRISE DES COÛTS DE CONCEPTION

L'augmentation régulière de la taille des circuits intégrés entraîne naturellement une augmentation du coût de leur conception (figure 11.1).

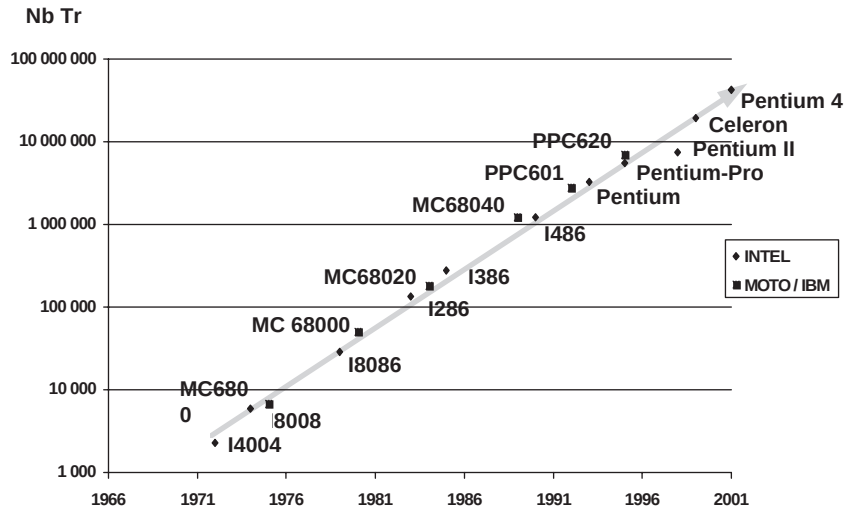


Figure 11.1 Évolution de la complexité des microprocesseurs

Pour éviter que celle-ci devienne prohibitive, les entreprises doivent continuellement innover en terme de méthodologie. L'économie de la conception des circuits intégrés est d'ailleurs très comparable à celle des logiciels. Des solutions très voisines sont utilisées pour maîtriser l'inflation des coûts et des délais dans ces deux domaines.

La conception manuelle d'un circuit au niveau des transistors coûte environ une heure de travail par transistor dessiné en incluant la spécification, la vérification et la documentation. Ce coût devient rapidement prohibitif lorsque la complexité atteint quelques centaines de milliers de transistors, d'autant plus que la relation coût/complexité n'est pas linéaire. En effet, le coût de la conception tend à augmenter avec la complexité des circuits car sa gestion possède elle-même un coût.

Le meilleur moyen de réduire le coût de conception est d'accroître la réutilisation de blocs déjà conçus. Cela se fait à la foi à l'intérieur d'un circuit et entre des circuits différents. Nous verrons que les méthodes modernes de conception s'appuient sur la constitution de bibliothèques de blocs préconçus, vérifiés et documentés. Un marché de tels blocs est d'ailleurs en train de se développer sous le nom d'IP (pour *Intellectual property*).

Il faut également tenir compte d'un facteur spécifique au monde des circuits intégrés qui est une grande variabilité de la technologie. Celle-ci apparaît à la fois chez chaque fondeur qui fait continuellement évoluer sa technologie pour rester compétitif, et entre les fondeurs qui ne disposent pas, à un instant donné, de technologies complètement compatibles entre elles. Il est donc nécessaire de développer des outils et des méthodes de conception qui permettent de porter un circuit, ou simplement des blocs, d'une technologie à une autre sans trop d'efforts. Les techniques de conception automatique, que nous allons évoquer, se sont largement développées en réponse à ces problèmes.

Les techniques, et les coûts de conceptions associés, se sont scindés en fonction de l'importance du marché visé. Par exemple, les microprocesseurs, qui sont à la fois des circuits très complexes et qui doivent aussi être très optimisés en performance et en taille, sont destinés à un vaste marché de plusieurs dizaines de millions d'unités par an. À titre d'exemple, la conception du microprocesseur Itanium® d'Intel (25 Mtr) aurait duré 5 ans et aurait coûté 4 500 personnes × années. La majorité des autres circuits très complexes sont des circuits dédiés à des applications spécifiques (GSM, GPS, photographie-cinéma, périphérie d'ordinateurs…). Leur marché est numériquement beaucoup plus faible et leurs contraintes d'optimisation ne sont pas aussi importantes. Toutefois, le coût de conception de ces circuits doit rester de l'ordre de quelques personnes × années. Ils doivent, en plus, être conçu très rapidement pour s'adapter aux contraintes commerciales qui pèsent sur ce genre de produits. Les méthodes de conception de ces deux domaines seront donc très différentes, bien que des blocs issus de la première catégorie puissent être réutilisés dans la seconde.

11.3 CIRCUITS COMPILÉS

Le point de départ d'un processus de conception automatique consiste en l'écriture d'une spécification fonctionnelle du circuit à réaliser. Suivant les outils dont on dispose, le niveau de cette spécification peut être :

– un *schéma logique* donnant l'interconnexion de composants extraits d'une bibliothèque ;
– une *description fonctionnelle* (par exemple dans un sous-ensemble de VHDL) ;

– une *description procédurale* (dite comportementale) sous la forme d'un algorithme (par exemple, écrit sous la forme d'un `process` VHDL ou d'un programme en Système C) qui simule le comportement que devra avoir le circuit.

À partir de ces données, un compilateur de silicium va générer un schéma logique qui représente l'interconnexion de cellules prises dans une bibliothèque.

La suite du processus dépend du type de circuit visé. Ceux-ci peuvent être :

– soit la paramétrisation électrique de circuits pré-existants (FPGA, CPLD). Bien que très utilisée, cette approche n'entre pas dans le domaine de la conception de circuits intégrés ;
– soit la définition de l'interconnexion de circuits dont les premières étapes technologiques sont déjà réalisées de manière standard (circuits dits *gate arrays*). Ces circuits comportent initialement des transistors qui sont interconnectés à la demande pour réaliser des blocs de base (portes, bascules) qui sont ensuite câblées pour obtenir le schéma logique ;
– soit la réalisation de circuits « standardisés » (aussi appelés *pré-caractérisés*). La conception (automatique) de ce type de circuits se déroule de la manière suivante :
  • le placement en ligne des cellules sur le circuit en ménageant l'espace estimé pour les interconnexions. Les cellules sont prises dans une bibliothèque qui contient leurs dessins et toutes les informations nécessaires à leur utilisation. Ces bibliothèques sont développées et mises à jour par les fondeurs,
  • l'interconnexion des cellules en utilisant des canaux entre les lignes de cellules ou en utilisant les couches métalliques au dessus des cellules,
  • la finition du circuit en le munissant d'une couronne de plots et d'alimentations.

Le coût de réalisation des circuits pré-caractérisés fait que cette approche n'est presque plus utilisée pour la conception automatique de circuits intégrés. Toutefois, l'efficacité de l'outillage mis au point fait qu'il est réutilisé pour accélérer la conception

Figure 11.2 Dessin d'un séquenceur câblé en approche pré-caractérisée (voir chapitre 9) synthétisé avec le système Alliance

des circuits conçus manuellement en fournissant une réalisation « pré-caractérisée » pour les blocs logiques.

Certains compilateurs de silicium sont spécialisés pour réaliser des circuits d'un type particulier (par exemple, ceux basés sur le modèle d'un microprocesseur). Ces outils travaillent en adaptant un modèle générique (fonctionnel et topologique) de circuit au problème particulier. Les circuits obtenus sont beaucoup plus optimisés que ceux issus d'un outil général. Toutefois, la prise en compte des optimisations « fines » que l'on réalise lors d'une conception manuelle reste souvent difficile.

11.4 CIRCUITS « CUSTOM »

La conception d'un circuit custom est un travail très complexe qui doit être mené avec beaucoup de rigueur et de méthode. Les systèmes logiciels de conception disposent tous d'outils de gestion de projet qui évitent de s'égarer dans la complexité du travail. Le travail suit une démarche descendante.

– La première étape consiste en l'écriture de la spécification globale du circuit. Celle-ci peut être un schéma logique ou un programme VHDL, ce qui permet de la valider. À ce niveau du processus de conception, la spécification est très globale.
– La seconde étape consiste à diviser le futur circuit en gros blocs qui pourront être conçu indépendamment. Un « plan de masse » du futur circuit sera dessiné. Si le circuit est vraiment très gros, ce découpage se poursuivra jusqu'à l'obtention de blocs dont la conception relèvera d'une approche spécifique. Nous pouvons distinguer :
  • les blocs *compilés* qui réutilisent les outils de conception des circuits pré-caractérisés ;

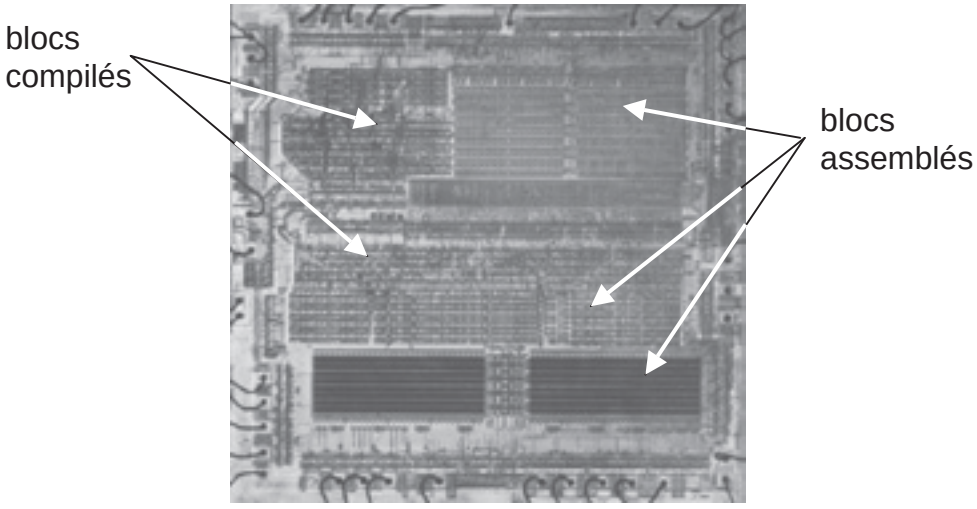


Figure 11.3 Microcontrôleur Intel 80C51®

- les blocs *assemblés*. Ces blocs sont obtenus par la simple juxtaposition de cellules conçues pour s'emboîter exactement les unes dans les autres (et réaliser ainsi les connexions inter-cellules). Certains de ces blocs (ROM, PLA, chemins de données…) sont générés par des outils spécifiques appelés *générateurs*. Le travail d'assemblage des cellules est souvent piloté par l'exécution d'un algorithme qui gère les irrégularités (par exemple dans le contenu d'un PLA).

Avant leur conception, la fonctionnalité de chacun de ces blocs doit être soigneusement décrite pour permettre leur vérification.

En parallèle de la conception des blocs, les différentes cellules nécessaires doivent être dessinées à la main. Leur complexité ne doit pas excéder une trentaine de transistors.

– L'assemblage des blocs d'après le plan de masse, pour obtenir le circuit se fait ensuite à l'aide d'un outil de câblage automatique. Leur placement est généralement manuel.

11.4.1 Styles de conception

La conception manuelle des circuits complexes est un art. Il existe différentes approches pour la mener à terme, chacune ayant ses avantages et ses inconvénients. Par exemple, les caissons pourront être disposés verticalement ou horizontalement, ce qui correspondra d'ailleurs à l'usage de transistors minimums ou « grands ». Les concepteurs qui se rapprochent des techniques automatiques ont tendance à dessiner des cellules très générales qui peuvent être utilisées dans toutes les configurations et donc à utiliser de grands transistors pour obtenir des portes de sortance élevées (mais qui ont des capacités d'entrées et des consommations plus élevées !) tandis que les concepteurs qui favorisent un style plus manuel, dessineront des transistors justes dimensionnés à leur besoin. De même, le croisement des couches se fera avec plus ou moins de rigueur suivant que l'on visera, ou non, un processus d'assemblage basé sur la simple juxtaposition des cellules.

11.5 VÉRIFICATION DE LA CONCEPTION

En parallèle de ces étapes de conception se situent les étapes de vérification. Une seule erreur peut rendre un circuit inutilisable. Le coût et le délai de réalisation d'un circuit prototype sont tels qu'il est préférable d'augmenter l'effort de vérification plutôt que de multiplier les prototypes. La norme actuelle devrait être : « bon du premier coup ! ». Le coût de réalisation d'un nouveau jeu de masques est de l'ordre du million de dollars et le temps de réalisation d'un nouveau jeu de circuits varie entre quelques semaines à quelques mois. Pour réduire le coût des prototypes, les fondeurs et les centres de recherche utilisent la technique des circuits « multiplexés », c'est-à-dire que la surface de la tranche est occupée par plusieurs projets qui ne sont représentés que par quelques exemplaires. Les coûts de fabrication des circuits sont ainsi partagés entre le nombre de projets qui partagent le même lot.

La vérification de la conception des circuits intégrés utilise maintenant des outils très avancés.

– Pour la vérification des cellules de base : celles-ci sont vérifiées soigneusement à partir de leur masques.
  • le dessin de leurs masques est vérifié par un outil appelé DRC (*Design Rules Checker*) qui vérifie la bonne application des règles géométriques de dessin ;
  • leur comportement électrique (normal et aux limites de la technologie) est vérifié en extrayant automatiquement leur schéma électrique. La vérification électrique commence par s'assurer que le schéma est sain à l'aide d'un outil appelé ERC (*Electrical Rules Checker*) qui vérifie que le circuit obéit à certaines règles électriques de bonne construction (vérification du rapport dimensionnel des transistors, détection des portes non alimentées, des court-circuits…). Puis la cellule est simulée dans toutes les configurations possibles à l'aide d'un simulateur électrique (par exemple, SPICE) ;
  • leur fonctionnalité est ensuite vérifiée en extrayant automatiquement une description fonctionnelle à partir de leur schéma électrique. Cette description extraite est ensuite comparée avec leur spécification fonctionnelle à l'aide d'un outil de vérification formelle [MAD88] (V-Formal ou Chrysalys).
– Pour la vérification des blocs : puisque les cellules sont vérifiées, il ne reste théoriquement qu'à s'assurer que l'interconnexion des cellules est correcte. Toutefois, il faut se mettre en garde contre des modifications manuelles du câblage et de l'ajout manuel de transistors « annexes ». Pour cela, il est préférable de ré-extraire le schéma électrique, d'identifier automatiquement les cellules, puis d'extraire la fonctionnalité globale et de la comparer *formellement* avec la spécification fonctionnelle du bloc. Les outils actuels permettent d'effectuer des vérifications formelles de blocs jusqu'à 100 000 transistors.
– Les spécifications des blocs ont elles-mêmes été vérifiées en les assemblant pour obtenir une description du circuit complet sur lequel on peut simuler l'application.
– Pour la vérification du circuit complet : il n'est pas (encore) possible de réaliser une vérification formelle de l'ensemble d'un circuit de plusieurs millions de transistors. Il est toutefois possible de simuler une description fonctionnelle globale, extraite automatiquement, sur un jeu d'essai réduit.

Une autre vérification finale est souvent pratiquée. Elle consiste en une vérification globale des règles de dessin et en une comparaison du schéma électrique extrait avec celui obtenu en assemblant les schémas des cellules telles qu'elles sont utilisées dans les blocs et jusqu'au circuit complet. Ces dernières opérations sont très coûteuses en volume de calcul. Pour des circuits très complexes, elles peuvent mobiliser des machines très performantes pendant plusieurs semaines.

11.6 SYSTÈMES INTÉGRÉS SOC (*SYSTEMS ON CHIP*)

Une nouvelle famille de circuit est maintenant en production. Il s'agit de circuits qui réalisent un système complet (figure 11.4). De tels circuits sont utilisés pour les téléphones portables, les appareils grand public, etc. Généralement, un tel circuit se compose d'un petit système informatique spécialisé qui peut comprendre :

– un ou plusieurs processeurs plus ou moins performants ;
– de la mémoire programme (morte ou inscriptible ou électriquement reprogrammable) ;
– de la mémoire de travail ;
– des dispositifs périphériques standard ou spécialisés.

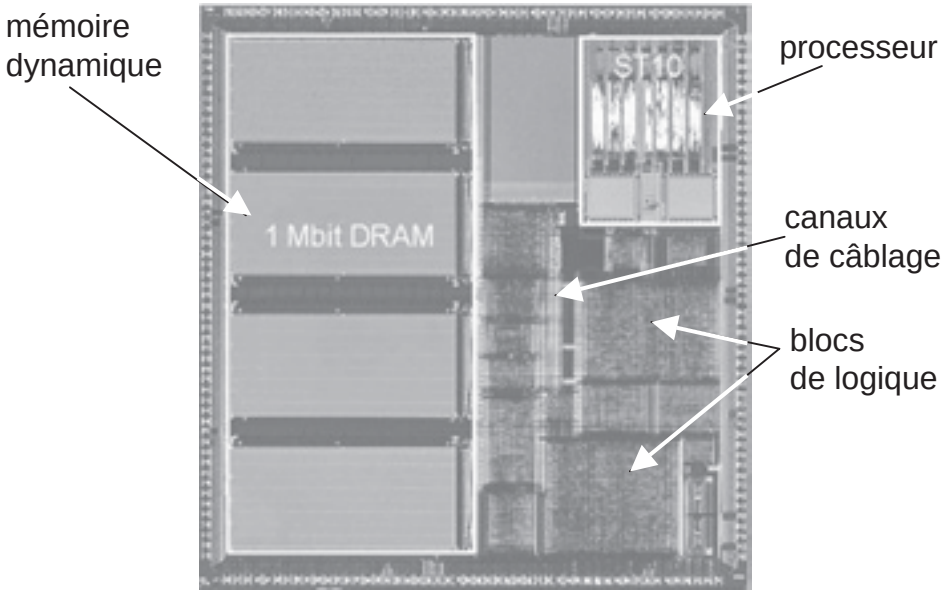


Figure 11.4 Circuit spécialisé pour la commande de disques durs (STMicroelectronics, technologie 0,35µ, 1997)

Tous ces organes sont assemblés à l'aide de bus intégrés.

Un marché de blocs pré-conçus (IP) se développe pour ce genre d'applications.

La réalisation d'un tel circuit est fortement simplifiée par l'utilisation d'outils qui permettent la simulation de l'ensemble (avec son logiciel). La conception physique du circuit est réalisée en sélectionnant les blocs dans une bibliothèque, en les paramétrant, en les plaçant sur le circuit et en les interconnectant. Seuls les blocs spécifiques et le logiciel sont à concevoir avec des méthodes « traditionnelles ». Les bus sont conçus automatiquement par des outils automatiques qui les dimensionnent, les connectent et les dessinent.

11.7 LA SUITE…

La diminution de la taille des motifs de dessin permet la réalisation de circuits de plus en plus complexes. Le seuil des 100 millions de transistors va bientôt être atteint.

Il devient de plus en plus important de réduire le coût de conception de tels circuits, d'autant plus que la demande est très importante.

La solution qui se dessine consiste à réaliser de tels circuits en assemblant des sous-systèmes (processeur + mémoires + périphériques) déjà conçus pour réaliser des fonctions bien déterminées comme, par exemple un module GPS complet qui effectue le calcul d'une position géographique, en incluant la réception des satellites, ou encore un module qui effectue une connexion radio Bluetooth à courte distance (figure 11.5). L'approche devient alors très comparable à la conception d'un système par l'assemblage de cartes qui réalisent de telles fonctions.

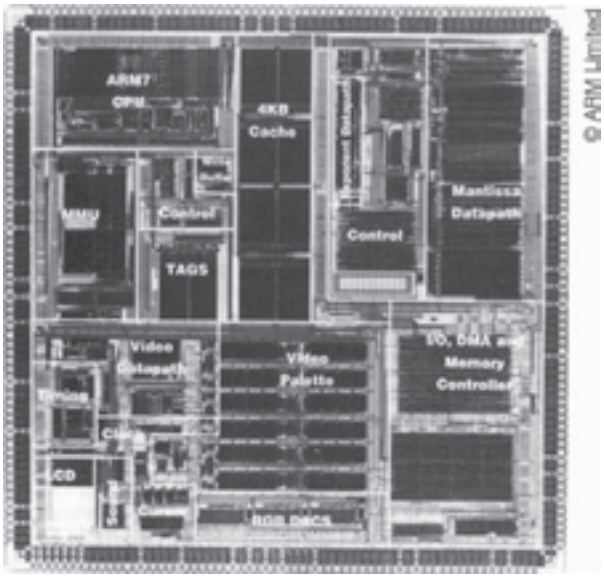


Figure 11.5 Un ordinateur personnel sur un seul circuit (ARM7500FE®)

L'offre de tels modules commence à se dessiner sous la forme de macro-blocs IP, avec la création d'entreprises spécialisées dans un domaine applicatif (comme par exemple la conceptions de tels blocs spécialisés pour la connexion au réseau Internet).

Une approche « ultime » consiste à développer des circuits « standard » pour une famille de produits et de les particulariser par logiciel.

L'augmentation constante du nombre de transistors disponibles, ainsi que de la vitesse d'horloge des circuits amène à réaliser des circuits SOC de plus en plus complexes qui amènent de plus en plus d'« intelligence » au niveau des applications qui les utilisent (comme par exemple dans le cas des téléphones portables) (figure 11.6).

Comme ces circuits géants vont quelquefois se trouver dans des applications critiques (comme par exemple dans les systèmes de freinage des automobiles) la responsabilité des différents intervenants (fournisseurs de blocs IP, concepteur du circuit, fondeur) risque d'être difficile à départager en cas de problème.

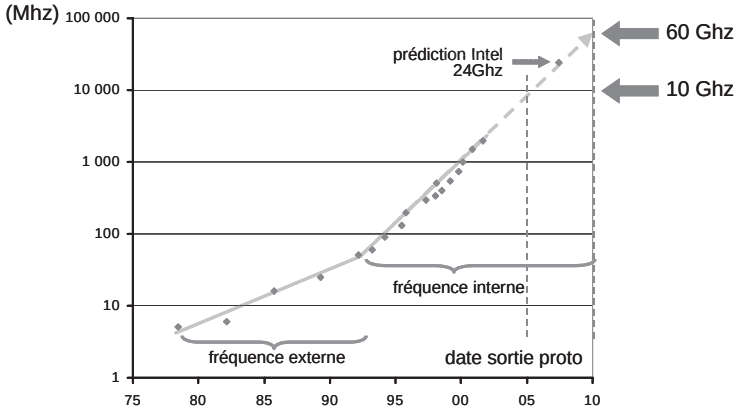


Figure 11.6 Évolution de la fréquence d'horloge des microprocesseurs X86

BIBLIOGRAPHIE

[MAD88] J.-C. Madre et J.-P. Billon, *Proving Circuit Correctness using Formal Comparison Between Expected and Extracted Behaviour*, 25th ACM/IEEE Design Automation Conference, Anaheim, juin 1988.

Chapitre **12**

En guise de conclusion…

Nous espérons que la lecture de cet ouvrage a montré au lecteur les grandes lignes de force de la conception des circuits intégrés complexes. Pour ce faire, nous avons choisi de montrer ce qu'est un transistor MOS et comment il est possible d'en tirer le maximum d'intérêts. La couverture complète du sujet nécessiterait toute une bibliothèque. Beaucoup de choses restent donc à apprendre, mais si nous avons réussi à transmettre au lecteur la passion de la conception manuelle, alors notre effort n'aura pas été vain.

À l'époque où certains cours se targuent du titre de microélectronique parce qu'ils enseignent le VHDL et la paramérisation des FPGA, il nous a paru utile de montrer ce qu'est le véritable paysage des circuits intégrés, et surtout comment ils sont dessinés. Dans ce sens, notre démarche s'apparente à celle qui consiste à apprendre la programmation en commençant par l'assembleur.

Cet ouvrage est aussi un message de nostalgie. Il cherche à participer à la conservation de ce savoir qui est en voie de disparition en Europe, sous les coups de boutoir de la déesse de la productivité. Passées de mode, ces techniques qui présentaient pourtant de nombreux avantages, pourraient peut être encore servir à améliorer l'efficacité des circuits modernes, au lieu d'être simplement condamnées à disparaître.

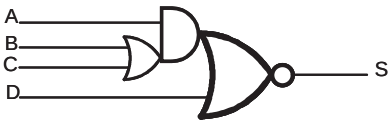
Passionnés de longue date par l'efficacité de la circuiterie polyphasée, nous avons fait un gros effort pour également présenter, sur un pied d'égalité, la circuiterie monophasée qui est devenue quasiment la seule à être maintenant utilisée. Après cet exercice, nous avons constaté que sous une apparence de principes rigoureux. Cette approche conduit à réaliser des circuits beaucoup plus complexes et cache des points délicats qui amènent souvent les concepteurs à faire des « bidouilles » peu recommandables. Maintenant que la conception « presse-bouton » des circuits intégrés amène les concepteurs à ne plus regarder les transistors, il serait peut-être possible d'introduire quelques principes de la conception polyphasée dans les outils automatiques.

Un autre élément de notre nostalgie réside dans la beauté que dégagent les gros circuits dessinés manuellement. Il est encore possible, en les regardant, d'y voir un style, une intention, une école, tout ce que l'on ne retrouve que dans l'architecture ou dans les objets artisanaux. L'organisation topologique de ces circuits dégage un effet de réalisation pensée, tant au niveau des cellules qui apparaissent comme les résultats de puzzles complexes dont l'emboîtement réussi est synonyme de surface gagnée, qu'au niveau global qui s'apparente à l'organisation territoriale d'un immense pays microscopique. Ces considérations, qui n'intéressent pas l'industrie, ne sont là que pour montrer la passion avec laquelle certains concepteurs réalisaient ces chefs-d'œuvre, que les outils automatiques modernes dessinent maintenant sans art.

EXERCICES

Conception d'une porte CMOS complexe

Soit la porte complexe suivante :



On supposera qu'elle est réalisée en CMOS « classique ».

1. Déterminer son schéma « en transistors ».

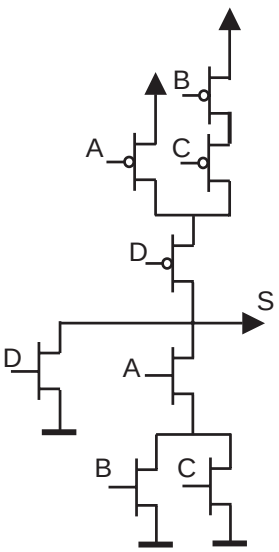
2. Dimensionner la taille de ses transistors par rapport à ceux de l'inverseur « minimum » pour obtenir des performances électriques comparables (temps de montée et de descente).

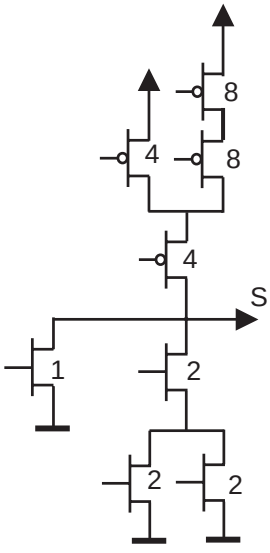
3. Déterminer les charges capacitives des entrées, par rapport à celle de l'inverseur minimum.

4. Cette porte peut-elle être attaquée par des inverseurs minimaux ? (justifiez votre réponse.)

5. Proposer un dessin « squelettique » des masques de cette porte.

1. et **2.** Dessin en transistor et leurs dimensionnement :



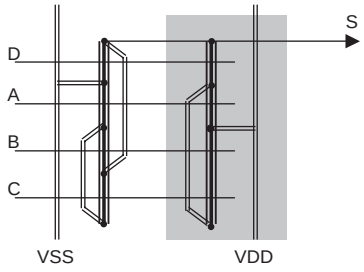


3. Charge des entrées

entrées	Charge tr N	Charge tr P	Total	Charges equ
A	2	4	6	2
B	2	8	10	3,3
C	2	8	10	3,3
D	1	4	5	1,6

4. Un inverseur minimal possède une impédance d'entrée égale à 3 fois la capacité de grille de son transistor N et une sortance de 8. La charge apportée par chacune de ses entrées peut être évaluée relativement à celle d'un inverseur, ce qui montre que dans tous les cas, elle peut être attaquée par un inverseur minimal.

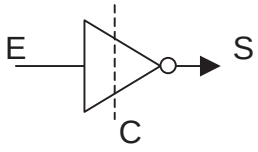
5. Dessin squelettique des masques de cette porte :



Le transistor N excité par l'entrée A peut être permuté avec les transistors N excités par les entrées B et C de manière à simplifier le dessin des masques.

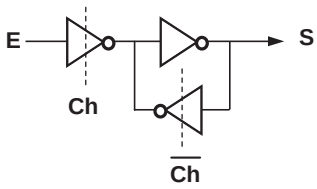
Étude d'un latch statique

Soit un inverseur trois états :



1. Dessiner le schéma « squelettique » des masques de cet inverseur en respectant le croisement des flux de donnée (l'entrée) et de commande (de chargement).

Soit un latch statique utilisant deux inverseurs trois états :

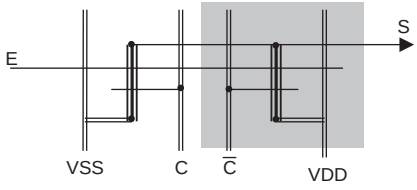


2. Rappeler son schéma en transistors.

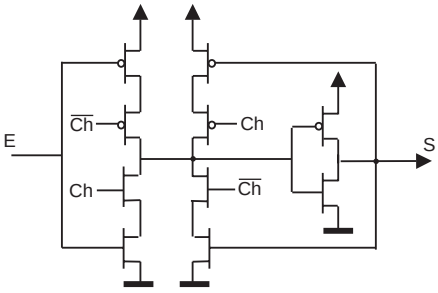
3. Dessiner le schéma « squelettique » de ses masques en respectant le même croisement des flux. On réduira la longueur de cette cellule en disposant les portes sur deux niveaux superposés. On évitera aussi la duplication des lignes de commande.

4. Comment peut-on ajuster la sortance de ce latch ?

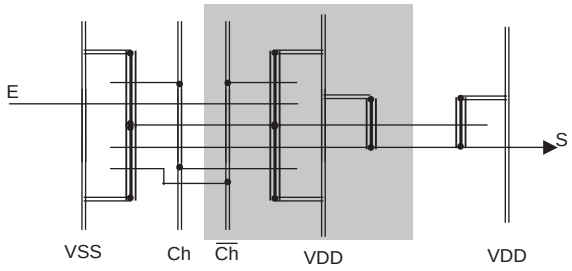
1. Dessin squelettique des masques d'un inverseur 3 états :



2. Dessin du latch en transistors :



3. Dessin squelettique des masques du latch :

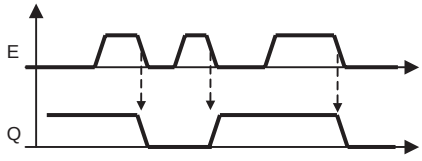


4. Ajustement de la sortance du latch :
La sortie du latch est pilotée par son seul inverseur qui n'est pas 3 états. Pour Ajuster la sortance du latch, il suffit de modifier la taille de ses transistors (dans la limite de la sortance des inverseurs 3 états qui l'attaquent).

Étude de la synthèse d'une bascule T

Une bascule T asynchrone est un diviseur de fréquence par deux.

Elle ne possède pas d'horloge. Ses sorties s'inversent à chaque transition négative de l'entrée E.



Ce sujet propose l'étude de cette bascule T vue comme un automate asynchrone.

1. Construire le diagramme d'état asynchrone de la bascule T.

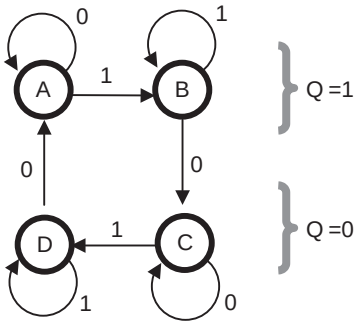
2. Construire le tableau des transitions de cette bascule, en indiquant ses états stables.

3. Donner le tableau codé des transitions donnant le code de l'état suivant, en fonction du code de l'état en cours et de l'entrée E.

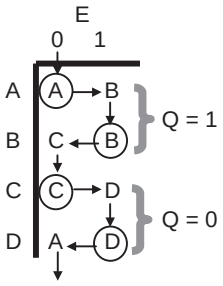
4. En déduire les équations booléennes définissant les bits de l'état suivant.

5. Traduire ces équations sous forme de NOR, en cherchant à obtenir des termes communs.

1. Diagramme d'état (asynchrone) :

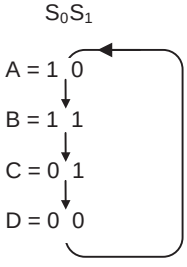


2. Tableau des transitions asynchrones :



L'état (asynchrone) S de la bascule sera codé sur plusieurs bits. Le bit de poids faible correspondra à la sortie Q. Pour éviter les aléas, le codage des états sera tel qu'il ne variera que d'un bit lors de l'enchaînement normal (codage asynchrone).

3. Les états seront codés sur deux bits en respectant les contraintes précédentes.



Avec $Q = S_0$.

Soit le tableau codé des états :

		E = 0	E = 1
S_0	S_1		
1	0	10	11
1	1	01	11
0	1	01	00
0	0	10	00

4. Équations booléennes des bits d'état :

$$S_0 = (\overline{E} \wedge \overline{S_1}) \vee (E \wedge S_0)$$

$$S_1 = (\overline{E} \wedge S_1) \wedge (E \wedge S_1)$$

5. Réalisation en NOR

En couvrant les compléments dans le tableau, on obtient :

$$S_0 = \overline{((E \wedge \overline{S_0}) \vee (\overline{E} \wedge S_1))}$$

$$S_1 = \overline{((E \wedge \overline{S_0}) \vee (\overline{E} \wedge \overline{S_1}))}$$

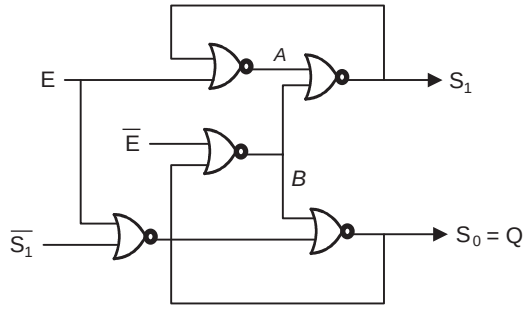
Soit, en appliquant le théorème de De Morgan :

$$S_0 = \left(\overline{\overline{(\overline{E} \vee S_0)} \vee \overline{(E \vee \overline{S_1})}}\right)$$

$$S_1 = \left(\overline{\overline{(\overline{E} \vee S_0)} \vee \overline{(E \vee S_1)}}\right)$$

Les deux expressions possèdent le terme $\overline{(\overline{E} \vee S_0)}$ en commun.

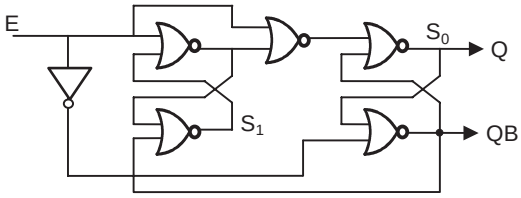
D'où un premier schéma pour la bascule T :



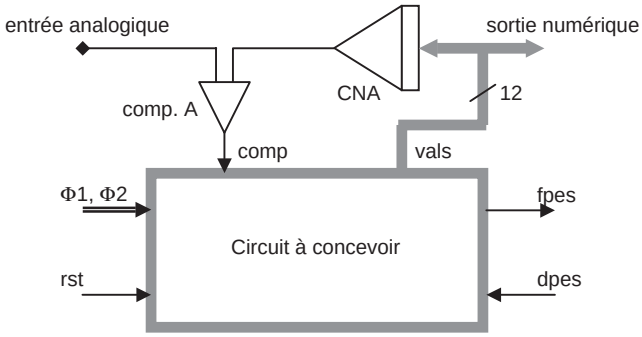
On reconnaît deux bascules RS. Il reste à générer économiquement $\overline{S_1}$. Pour cela, nous allons essayer d'utiliser la branche complémentaire *A* du RS supérieur.

Pour cela il faut vérifier que $\overline{S_1}$ peut être remplacé par $(\overline{E \vee S_1})$ (valeur en *A*) dans $(\overline{E \vee \overline{S_1}})$, ce qui ne pose aucun problème.

De même, *B* sera appelé QB selon la tradition des RS.
D'où le schéma final de la bascule T :



Étude d'un circuit de commande d'un convertisseur analogique/numérique



Le circuit à étudier doit réaliser la commande d'un convertisseur analogique → numérique à approximations successives (par pesée). La précision de ce convertisseur sera de 12 bits.

Ce circuit fonctionnera avec des composants externes :

Un convertisseur numérique → analogique (CNA de 12 bits) et un comparateur analogique, (supposés fournis).

Les connexions du circuit de commande avec l'extérieur seront :

– Une sortie numérique vals de 12 bits servant à la fois de sortie et d'excitation du convertisseur numérique → analogique.

– Une entrée comp recevant la sortie (binaire) du comparateur analogique.

(comp = 1 si l'entrée analogique est > à la sortie du CNA).

– Une entrée dpes dont l'excitation déclenche la pesée.

– Une sortie fpes indiquant que la pesée est terminée et que la sortie vals est disponible pour l'extérieur.

– Deux entrées d'horloge (Φ1 et Φ2).

– Une entrée d'initialisation rst.

La vitesse du circuit de commande est limitée par les circuits analogiques externes.

L'algorithme de pesée est très classique. Il consiste à construire le mot de sortie bit à bit à partir des poids forts. On commence par proposer un 1, si la sortie du comparateur indique que la valeur proposée est trop grande, ce bit est remplacé par un 0, sinon il est gardé, et l'on passe au bit suivant. L'algorithme se termine lorsque les 12 bits sont déterminés.

Une attente devra être prévue dans cet algorithme pour permettre au convertisseur numérique → analogique et au comparateur de fonctionner.

Les performances souhaitées suggèrent de réaliser un chemin de données de 12 bits et un séquenceur à base d'un PLA.

1. Écrire l'algorithme à réaliser sous la forme d'un organigramme.

2. Proposer une forme standard.

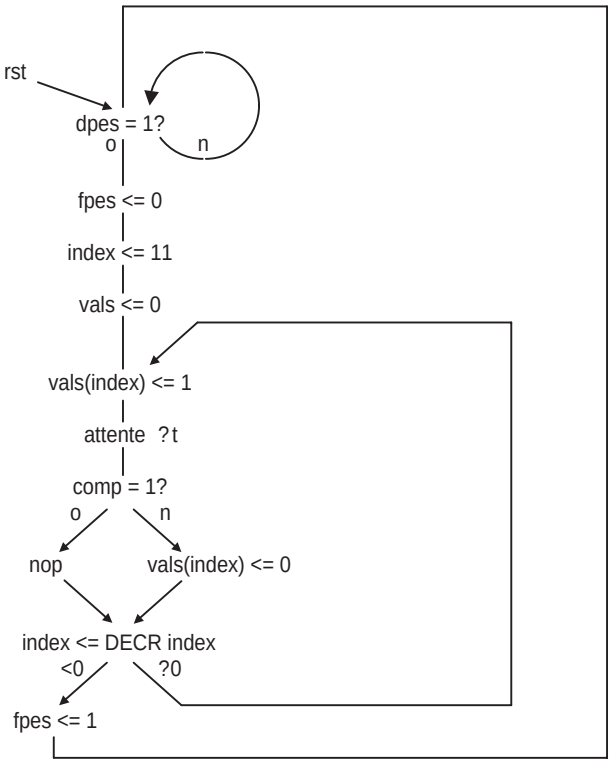
3. Réécrire l'algorithme optimisé avec des instructions standardisées.

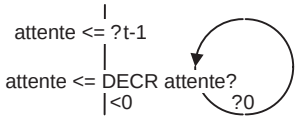
4. Proposer un chemin de données pour ce circuit.

5. Proposer une organisation pour son séquenceur.

6. Proposer un contenu pour le PLA.

1. Algorithme : voir ci-contre.
Une autre solution aurait consisté à utiliser un masque pour sélectionner un bit de vals. Puis à décaler ce masque à droite.
2. Liste des opérations utilisées :
fpes <= 1
fpes <= 0
index <= 11
index <= DECR index
vals <= 0
vals(index) <= 1
vals(index) <= 0
nop
L'attente peut être réalisée par une petite boucle : voir ci-contre.

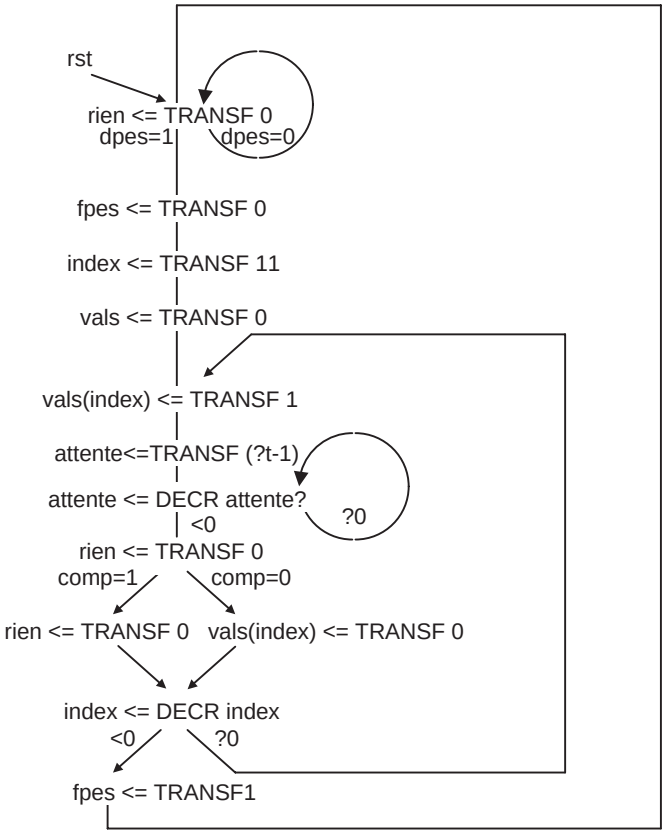




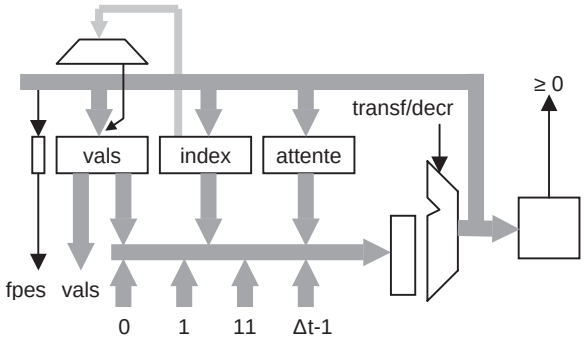
Tests :
dpes =1? ET comp = 1? seront réalisés directement par le séquenceur.
Seul le test du signe du résultat sera testé dans le chemin de données : result < 0 / ≥ 0
Forme standard :

dest	=	op	source
rien		transf	0
fpes		decr	1
vals			11
vals(index)			Δt-1
index			index
attente			attente

3. Algorithme optimisé :

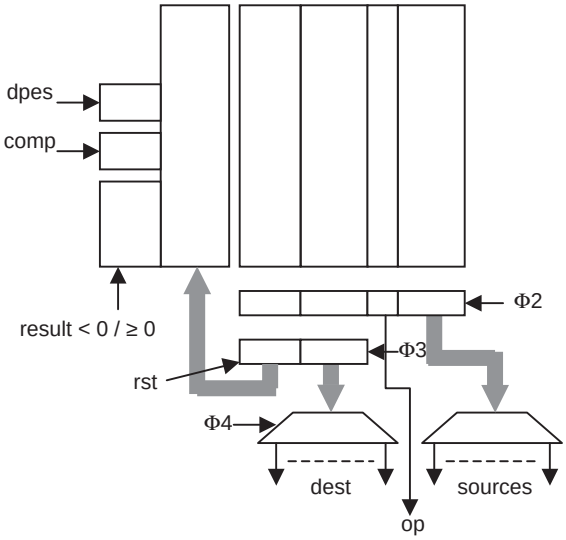


4. Chemin de données (bi-phasé) :



Le choix d'opérations (transfert/décrémentation) peut être réalisé en modifiant la retenue initiale d'un décrémenteur.

5. Organisation du séquenceur (à base de PLA).



La machine est supposée quadriphasée avec un fonctionnement on superposé du séquenceur et du chemin de données.

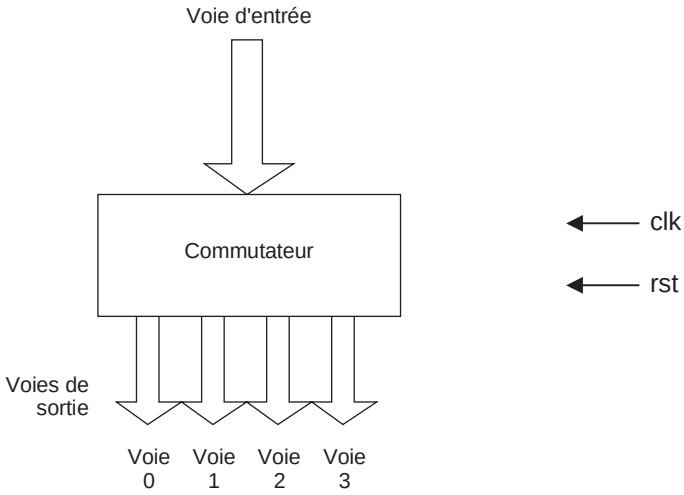
6. Contenu du PLA du séquenceur :

test	état	dest	op	source	état suiv
dpes=0	0	rien	TRANSF	0	0
dpes=1	0	fpes	TRANSF	0	1
	1	index	TRANSF	11	2
	2	vals	TRANSF	0	3
res = 0	3	vals(index)	TRANSF	1	4
res < 0	3	fpes	TRANSF	1	0
	4	attente	TRANSF	Ät-1	5
res = 0	5	attente	DECR	attente	5
res < 0	5	rien	TRANSF	0	6
comp=1	6	rien	TRANSF	0	7
comp=0	6	vals(index)	TRANSF	0	7
	7	index	DECR	index	3
matrice ET		matrice OU			

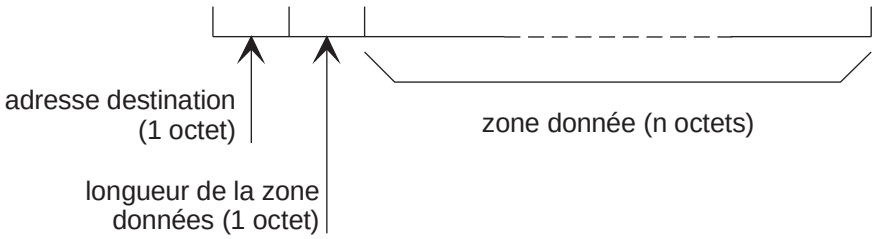
Étude d'un petit commutateur de messages

> Le sujet consiste à étudier un commutateur (routeur) de messages (très simplifié) pour un mini réseau hiérarchique. Cet organe reçoit des messages sur une voie d'entrée et réémet **intégralement** leur contenu sur l'une de ses

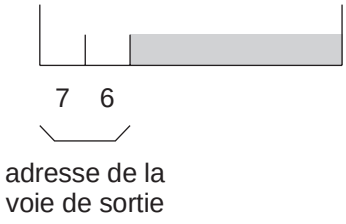
4 voies de sortie, en fonction de l'adresse de destination située dans l'entête de chaque message. Toutefois, l'octet d'adresse sera ré-émis avec un décalage de deux positions à gauche pour préparer l'aiguillage suivant.



Les messages sont constitués d'une suite d'octets. Leur trame, simplifiée, est la suivante :

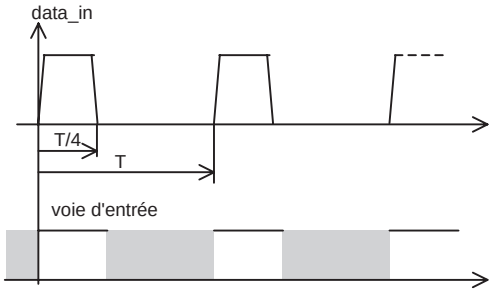


Le format de l'octet d'adresse de destination est le suivant (en entrée) :



Les messages arrivent octet par octet sur 8 lignes d'entrée. Les octets d'un message sont séparés par un intervalle de temps d'environ T. La présence

d'un octet sur la voie d'entrée sera indiquée par l'activation d'un signal externe *Data_in* mis à 1 au moment de l'arrivée de chaque octet et remis à 0 au bout d'un temps d'au moins T/4. Les octets seront valides sur la voie d'entrée pendant que *Data_in* est à 1. Les messages ré-émis sur les voies de sortie respecteront les mêmes contraintes temporelles. Pour cela, des signaux *Data_out* identiques à *Data_in* seront générés sur chaque voie de sortie. Il est toutefois normal que les messages réémis soient retardés par rapport à ceux d'entrée, mais on cherchera à minimiser ce décalage.



Ce commutateur sera construit à l'aide d'un chemin de données et d'un séquenceur. Son fonctionnement sera rythmé par une horloge de période T/16. Pour permettre l'acheminement de messages rapides, cette fréquence sera la plus élevée possible compte tenu des possibilités de la technologie utilisée.

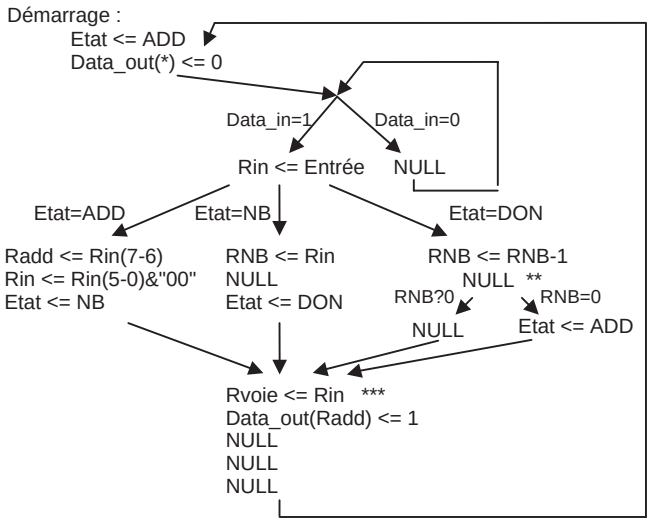
1. Écrire l'algorithme décrivant le fonctionnement du commutateur sous la forme d'un organigramme. On suggère d'utiliser une variable d'état auxiliaire pour distinguer l'étape courante dans le traitement du message d'entrée (adresse, nombre d'octets de donnée, données). On utilisera des instructions ineffectives (NULL) pour assurer la synchronicité de la lecture des octets sur la voie d'entrée et l'émission des octets sur la voie de sortie sélectionnée.

2. Proposer un schéma pour le chemin de données du commutateur. La voie d'entrée sera lue octet par octet dans un registre et les voies de sorties issues de registres spécialisés capables de contenir un octet.

3. Proposer un séquenceur mono-PLA. On proposera un format pour la partie de la matrice de génération du PLA qui fournira les commandes au chemin de données, et une organisation symbolique pour l'ensemble du PLA.

4. Proposer une version monophasée du circuit avec un séquenceur câblé réalisé à partir d'un générateur de temps et d'un réseau combinatoire générant les commandes pour le chemin de données.

1.



Pour éviter de relire l'octet précédent, la boucle de l'algorithme devra comporter au moins 5 instructions.

Pour attendre l'octet suivant, cette boucle devra comporter moins de 16 instructions.

L'instruction `Data_out(*)<=0` signifie une remise à 0 de toutes les sorties `Data_out`.

Remarques :

– Comme le circuit doit être le plus rapide possible, les fonctionnements de son chemin de données et de son séquenceur seront superposés, ce qui entraîne que les conditions devront être calculées au moins une instruction avant d'être testées.

– Comme la prise en compte des octets des messages ne se fait que lorsque le signal `Data_out` est actif, il n'y a aucun inconvénient à exciter toutes les sorties en parallèle. L'optimisation du brochage du circuit suggère de ne mettre qu'une seule sortie Voie, mais en conservant 4 sorties `Data_out`.

2. Conception du chemin de données :

Variables utilisées :

octet :	`Rin, Rnb, Rvoie`	2 bits :	`Radd(1,0)`
bit :	`data_out(0 to 3)`	Taille non précisée (≤ octet) :	`Etat`

Instructions :

```
Rin <= Entrée
Radd <= Rin(7-6)
Rin <= Rin(5-6)&"00"
Rnb <= Rin
Rnb <= Rnb - 1
Rvoie <= Rin
data_out(Radd) <= 1
data_out(*) <= 0
Etat <= ADD
```

```
Etat <= NB
Etat <= DON
NULL (rien <= x"00")
```

Tests :

```
Data_in=1/0
Etat=ADD/NB/DON
Rnb=0/?0
```

Format standard :

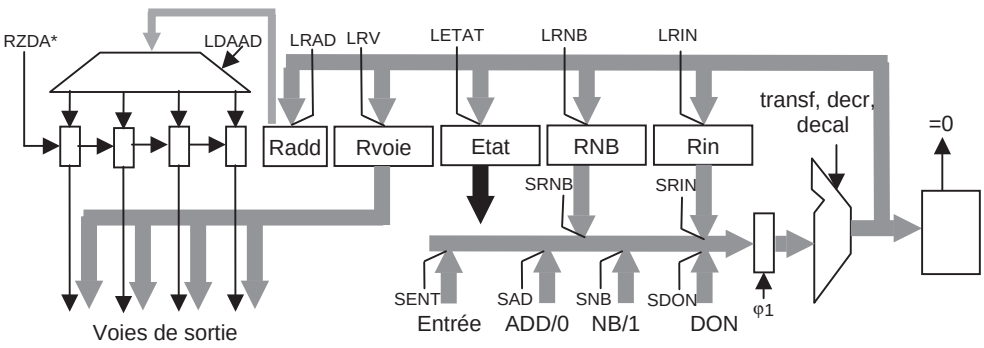
```
<Rdest> <= <op>< Rsource>
```

L'instruction `NULL` sera réalisée en ne chargeant aucun registre (`Rdest` = rien) avec la valeur 0.

Avec : `Rdest : rien / Rin / Rnb / Radd / Rvoie / data_out(Radd) / data_out(*) / Etat`
`op : transf / decr / decal_gauche_2`
`Rsource : Entrée / Rin / Rnb / ADD / NB / DON / 0 / 1`

(Il est possible de coder `ADD=0` et/ou `NB=1` pour diminuer le nombre de sources à réaliser.)

Schéma du chemin de données :



Les fonctions `decr` et `transf` de l'opérateur seront obtenues à partir d'un décrémenteur dont on fera varier la valeur de la retenue initiale.

Un registre tampon a été inséré pour éviter les rebouclages asynchrones. Le chemin de données travaille avec une horloge bi-phasée.

3. Étude d'un séquenceur à base de PLA

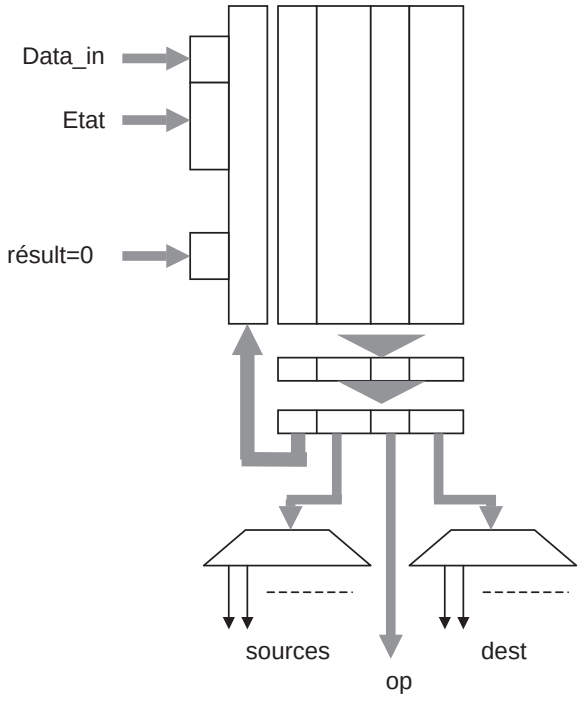
Format de la partie génération des commandes :

3 champs : `Rdest` 3 bits, `op` 2 bits, `Rsource` 3 bits.

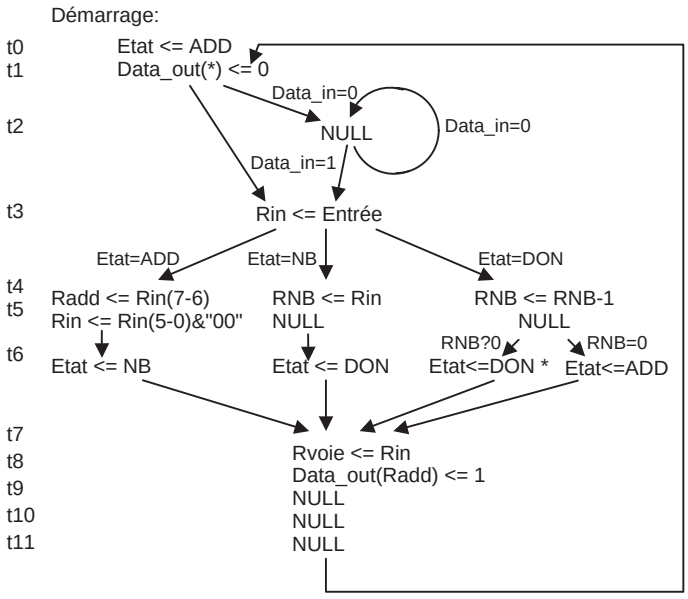
Contenu symbolique du PLA :

Data_in	=0	État	étape	étape suiv	action
			0	1	`Etat <= ADD`

Data_in	=0	État	étape	étape suiv	action
			1	2	Data_out(*) <= 0
0			2	2	rien <= 0
1			2	3	Rin <= Entrée
		ADD	3	4	Radd <= Rin(7-6)
		NB	3	5	Rnb <= Rin
		DON	3	6	Rnb <= Rnb -1
			4	7	Rin <= Rin(5-6)&"00"
			7	10	Etat <= NB
			5	8	rien <= 0
			8	10	Etat <= DON
			6	9	rien <= 0
	non		9	10	rien <= 0
	oui		9	10	Etat <= ADD
			10	11	Rvoie <= Rin
			11	12	Data_out(Radd) <= 1
			12	13	rien <= 0
			13	14	rien <= 0
			14	1	rien <= 0



4. Conception d'un séquenceur câblé :
Repérage des instants de séquencement de l'algorithme :



L'algorithme peut être rythmé par 12 instants. L'attente de l'impulsion `Data_in` est réalisée au niveau du générateur de temps, tandis que les autres temps ne concernent pas ce générateur ;

L'instruction `NULL` dans la branche `Etat=DON / RNB?0` est remplacée par `Etat <=DON` qui est aussi ineffective, mais qui permet d'uniformiser toutes les destinations des opérations déclenchées en `t6` à être des chargements du registre `Etat`, ce qui simplifiera le circuit combinatoire de génération des commandes.

Le générateur d'instants réalise la séquence suivante, de laquelle on peut déduire son schéma en logique monophasée (voir figure page 286).

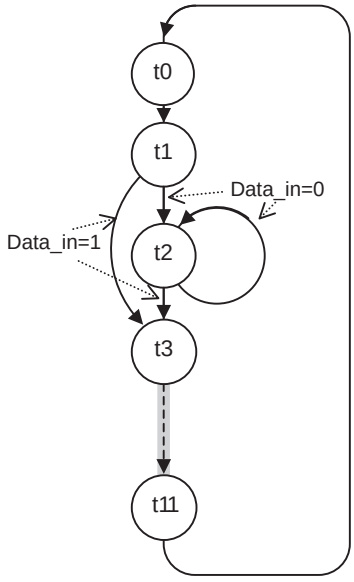
Le signal `rst` provoquera la mise à 1 de la bascule to et la mise à 0 de toutes les autres bascules, permettant ainsi le démarrage du générateur de temps.

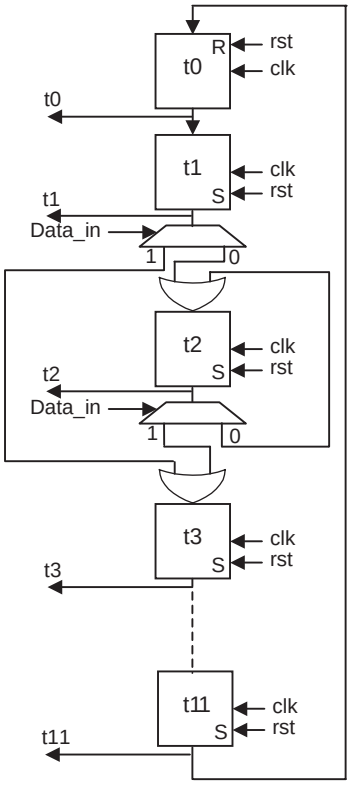
Pour réaliser un circuit monophasé, le chemin de données devra être légèrement modifié (voir figure page 287).

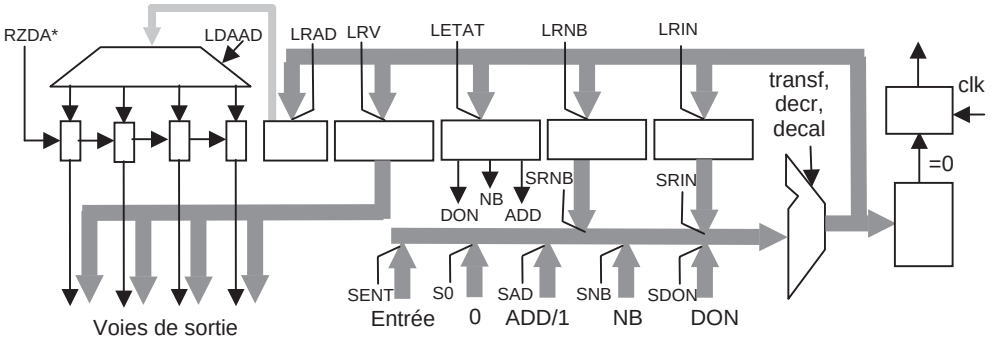
Le registre additionnel mis pour éviter le rebouclage en logique biphasée sera supprimé. Tous les autres registres deviendront des maîtres-esclaves. Le codage des états sera modifié pour qu'ils soient représentés par de simples bits du registre `Etat` (`ADD=Etat(1)`, `NB=Etat(2)`, `DON=Etat(4)`), ce qui impose de rajouter la constante 0.

Le circuit combinatoire de génération des commandes pour le chemin de données, supposé monophasé, sera obtenu par sa compilation à partir de ses équations VHDL :

```
LETAT <= '1' when t0='1' or t6='1' else '0';
LRAD <='1' when t4='1' else '0';
```





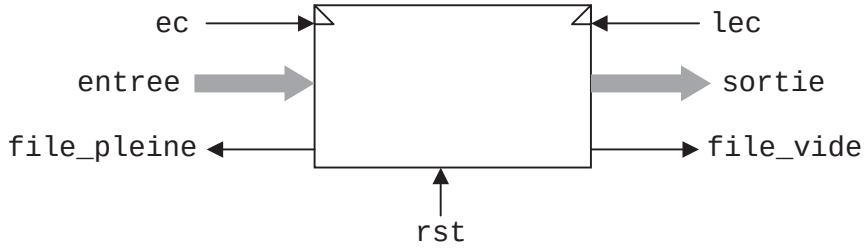



```
LRIN <= '1' when t3='1' or (t5='1' and ETAT=ADD) else '0';
LRNB <= '1' when t4='1' and (ETAT=NB or ETAT=DON) else '0';
LRV <='1' when t7='1' else '0';
LDAAD <='1' when t8='1' else '0';
RZDA <='1' when t1='1' else '0';
SENT <='1' when t3='1' else '0';
SRNB <='1' when t4='1' and ETAT=DON else '0';
SRIN <='1' when (t4='1' or t5='1') and ETAT=ADD else '0';
```

```
SAD <='1' when t0='1' or (t6='1' and ETAT=DON and result='1')
or t8='1' else '0';
SNB <='1' when t6='1' and ETAT=ADD else '0'
SDON <='1' when t6='1' and (ETAT=NB or (ETAT=DON and result='0'))
else '0';
S0 <='1' when t1='1' or t2='1' or (t5='1' and (ETAT=DON or ETAT=DON)
or t9='1' or t10='1' or t11='1' else '0';
commande UAL <= decr when t4='1' and ETAT=DON else
decal when t5='1' and ETAT=ADD else transf;
```

Toutes les commandes issues du séquenceur seront stockées dans un registre pour leur utilisation dans le cycle suivant. Les sorties de ce registre qui commandent les chargements serviront à générer des sous horloges spécifiques à chaque registre.

Conception d'une FIFO En VHDL



Brochage :

ecr – un front descendant sur cette ligne provoque l'écriture dans la FIFO de l'information présente sur les lignes **entree**. L'écriture est impossible si la ligne **file_pleine** est active.

entree – bus d'entrée, les informations devront être présentes avant et au moment du front descendant de **ecr**.

lec – un front descendant sur cette ligne provoque la lecture de l'information dans la FIFO. L'information lue sera ensuite considérée comme effacée (sauf si la ligne **file_vide** est active).

sortie – bus de sortie, les informations (le mot à lire) seront présentes avant et au moment du front descendant de **lect.** (l'information ne sera pas pertinente si la ligne **file_vide** est active)

file_pleine – cette ligne sera activée lorsque la FIFO est plein. Elle sera activée juste après la dernière écriture possible et désactivée après la première lecture qui suit.

file_vide – cette ligne sera activée lorsque la FIFO est vide. Elle sera activée juste après la dernière lecture possible et désactivée après la première écriture qui suit.

rst – un niveau haut sur cette ligne provoque l'initialisation de la FIFO.

La FIFO sera réalisée à l'aide d'une petite mémoire constituée d'une batterie de registres. Cette mémoire sera gérée de manière circulaire à l'aide de deux index. L'un repèrera la position dans laquelle se fera la prochaine écriture, l'autre repèrera la position dans laquelle se fera la prochaine lecture. Ces deux index se suivront dans leur déplacement. Si l'index d'écriture rattrape celui de lecture, alors la FIFO est pleine. Si l'index de lecture rattrape celui d'écriture, alors la FIFO est vide.

Il n'y a pas d'horloge, le circuit n'est rythmé que par les signaux `ecr` et `lec`.

On écrira quatre descriptions de la FIFO : dans les trois premières on supposera les écritures et les lectures non simultanées).

On écrira l'interface de l'entité FIFO et quatre architectures.

Pour s'affranchir de la largeur des bus `entree` et `sortie` et de la profondeur de la FIFO, on réalisera un modèle de FIFO générique.

1. La première description utilisera des variables entières pour représenter les index.

Les informations seront stockées, à une place fixe, dans une mémoire `mem` constituée d'un tableau de registres. Les index d'écriture et de lecture sont représentés par deux signaux entiers.

2. La seconde description, plus proche d'une réalisation matérielle, utilisera des registres à décalage circulaires contenant un seul 1 pour représenter les index.

Les index d'écriture et de lecture seront réalisés par des registres à décalage rebouclés.

3. La troisième description différera de la seconde par le fait que les détections de file pleine et de file vide se feront d'une manière correspondant à une réalisation matérielle.

Il devient nécessaire de conserver la mémoire de la dernière opération effectuée pour savoir si l'égalité des deux index signifie que la file est pleine ou vide. Pour cela, nous introduirons un signal `operation` qui prendra la valeur 0 à la suite d'une lecture et 1 à la suite d'une écriture. Ce signal servira à aiguiller l'information de coïncidence des index vers les indicateurs de file pleine et de file vide

4. La quatrième description différera de la troisième par le fait que les lectures et les écritures peuvent être simultanées (mais calées sur une même horloge de référence).

5. Sur quels principes pourrait-on écrire une description complètement conforme à une réalisation physique ?

1. Il faudra initialiser les index à 0 par `rst` pour s'affranchir du fait qu'ils peuvent prendre n'importe quelle valeur avant l'initialisation du circuit et provoquer des sorties de bornes des tableaux.

Écriture dans la FIFO :
L'index d'écriture désigne la position dans laquelle on va écrire. Il est incrémenté après l'écriture effective sur le front arrière de `ecr` (sauf si la file est pleine). L'info doit être présente sur le bus `entree` avant le front arrière de `ecr`. L'écriture est bloquée si la file est pleine. La détection de file pleine s'effectue par la détection de l'égalité des deux index après une écriture. Toute écriture a pour conséquence que la file n'est pas vide.
Lecture dans la FIFO :
L'index de lecture désigne la position dans laquelle on va lire. Il est incrémenté après la lecture effective pendant que la ligne `lec` est active (sauf si la file est vide). L'info est présente sur le bus `sortie` au moins pendant que la ligne `lec` est active. L'information est supposée extraite de la file après une lecture. La détection de file pleine s'effectue par la détection de l'égalité des deux index après une lecture. Toute lecture a pour conséquence que la file n'est pas pleine.
Les états de file pleine et de file vide apparaissent consécutivement aux ordres d'écriture et de lecture. Pour cela, ils doivent être générés respectivement dans les processus d'écriture et de lecture. Comme la valeur finale des index n'apparaît qu'après un temps δt, il faut retarder d'un petit instant la comparaison de ces index. Cette comparaison ne peut pas se faire en dehors des processus car sa signification dépend de l'action qui l'a provoquée.
Remarque VHDL :
Dans certains simulateurs VHDL l'instruction :

```
R<=R(7)&R(0 to 6) when clk'event;
```

provoque le décalage de R de deux positions à chaque transition de `clk`.
Tandis que :

```
R<=R(7)&R(0 to 6) after 1ps when clk'event;
```

ou

```
process (clk)
begin
R<=R(7)&R(0 to 6);
end process;
```

ne provoquent que le décalage de R d'une position. Dans le premier cas, le `clk'event` doit vraisemblablement durer plus de δt et permettre deux décalages, tandis que le `process` ne fait qu'un tour.

➤ Première description VHDL

```
library vector;
use vector.functions.all;
entity fifo is
 generic(larg: integer=8;
         prof: integer);
 port(
      ecr :in bit:= '0';
```

```
      lec :in bit:= '0';
      rst :in bit:= '0';
      file_pleine :buffer bit;    --bascules en sortie
      file_vide:buffer bit;
      entree :in bit_vector(larg-1 downto 0);
      sortie :out bit_vector(larg-1 downto 0));
end fifo;
architecture arch1 of fifo is
 type mat is array(0 to prof-1) of bit_vector(larg-1 downto 0);
 signal mem:mat;
 signal clec:integer:=0;    --compteur de lecture
 signal cecr:integer:=0;    --compteur d'écriture
 begin
 process (rst)     --initialisation de la file
 begin
    clec <= 0;
    cecr <= 0;
    file_vide <= '1';
    file_pleine <= '0';
 end process;
 process    --ecriture dans le fifo
 begin
  --attente transition neg ecr et file non pleine
  wait until ecr'event and ecr='0'and file_pleine='0';
   mem(cecr)<= entree; --ecriture dans la fifo
  --incrementation cecr modulo mem'length
   cecr<= (cecr+1)mod mem'length(1);
   file_vide<='0'; --la fifo ne peut plus etre vide
   wait on cecr;  -- attente positionnement de cecr
   if cecr=clec then
     file_pleine <= '1';
   end if;
 end process;
 process    --lecture dans le fifo
 begin
  --attente transition neg lec et file non vide
  wait until lec'event and lec='0' and file_vide='0';
  --incrementation clec modulo mem'length
  clec<= (clec+1)mod mem'length(1);
   file_pleine <= '0'; -- La fifo ne peut plus etre pleine
   wait on clec;  -- attente positionnement de clec
   if cecr=clec then
     file_vide <= '1';
   end if;
 end process;
 sortie<= mem(clec); --l'info est disponible le plus longtemps
possible
end arch1;
```

2. Un problème VHDL apparaît : la sélection des registres de la mémoire par les bascules de ses registres à décalage n'est pas une opération élémentaire de VHDL.

Nous devons écrire une fonction dans un `package` préliminaire pour calculer l'indice correspondant à la position d'un 1 dans un registre à décalage. Pour se prémunir du fait que ce registre possède une valeur quelconque avant l'initialisation par `rst`, cette fonction retourne un 0 lorsque ce registre est à 0.

➤ Seconde description

```
package complements is
 function select_dir(a:bit_vector) return integer;
end complements;
package body complements is
 function select_dir(a:bit_vector) return integer is  --fonction
pour
  alias av:bit_vector(1 to a'length) is a;            --simuler la
  begin                                            --selection directe
   for i in 1 to a'length loop
    if av(i)='1' then return i-1;end if;
   end loop;
   return 0; -- cas ou le vecteur est a 0 (initialisation!!)
 end select_dir;
end complements;
library vector;
use vector.functions.all;
use work.complements.all;
entity fifo is
 generic(larg: integer=8;
         prof: integer);
port(
      ecr :in bit:= '0';
      lec :in bit:= '0';
      rst :in bit:= '0';
      file_pleine :buffer bit;
      file_vide:buffer bit;
      entree :in bit_vector(larg-1 downto 0);
      sortie :out bit_vector(larg-1 downto 0));
end fifo;
architecture arch2 of fifo is
 type mat is array(0 to prof-1) of bit_vector(larg-1 downto 0);
 signal mem:mat;
 signal Rlec:bit_vector(0 to prof-1);
 signal Recr:bit_vector(0 to prof-1);
 begin
 process (rst)     --initialisation de la file
 begin
  if rst='0' then  --sur front descendant de rst
    Rlec(0) <= '1';
    Recr(0) <= '1';
    for i in 1 to prof-1 loop
      Rlec(i) <= '0';
      Recr(i) <= '0';
```

```
    End loop;
    file_vide <= '1';
    file_pleine <= '0';
  end if;
 end process;
 process    --ecriture dans le fifo
 begin
  --attente transition neg ecr et file non pleine
  wait until ecr'event and ecr='0'and file_pleine='0';
   mem(select_dir(Recr)) <= entree; --ecriture dans la fifo
   Recr<= Recr ror 1;  --decalage circulaire droit de Recr
   file_vide<='0'; --la fifo ne peut plus etre vide
   wait on Recr;  -- attente positionnement de Recr
   if Recr=Rlec then
     file_pleine <='1';
   else
     file_pleine <='0';
   end if;
 end process;
 process    --lecture dans le fifo
 begin
  --attente transition neg lec et file non vide
  wait until lec'event and lec='0' and file_vide='0';
   Rlec<= Rlec ror 1;  --decalage circulaire droit de Rlec
   file_pleine <= '0'; -- la fifo ne peut plus etre pleine
   wait on Rlec;  -- attente positionnement de Rlec
   if Recr=Rlec then
     file_vide <='1';
   else
     file_vide <='0';
   end if;
 end process;
 sortie<= mem(select_dir(Rlec));
end arch2;
```

3.

➤ Troisième description

```
package complements is
 function select_dir(a:bit_vector) return integer;
end complements;
package body complements is
 function select_dir(a:bit_vector) return integer is  --fonction
pour
  alias av:bit_vector(1 to a'length) is a;      --simuler la
  begin                                         --selection directe
   for i in 1 to a'length loop
    if av(i)='1' then return i-1;end if;
   end loop;
   return 0; -- cas ou le vecteur est a 0 (initialisation!!)
```

```
 end select_dir;
end complements;
library vector;
use vector.functions.all;
use work.complements.all;
entity fifo is
 generic(larg: integer=8;
         prof: integer);
port(
      ecr :in bit:= '0';
      lec :in bit:= '0';
      rst :in bit:= '0';
      file_pleine :buffer bit;
      file_vide:buffer bit;
      entree :in bit_vector(larg-1 downto 0);
      sortie :out bit_vector(larg-1 downto 0));
end fifo;
architecture arch3 of fifo is
 type mat is array(0 to prof-1) of bit_vector(larg-1 downto 0);
 signal mem:mat;
 signal Rlec:bit_vector(0 to prof-1);
 signal Recr:bit_vector(0 to prof-1);
 signal operation:bit:='0'; -- 0=> lec / 1=> ecr
 begin
 --initialisation de la file
 process (rst)     --initialisation de la file
 begin
  if rst='0' then  --sur front descendant de rst
    Rlec(0) <= '1';
    Recr(0) <= '1';
    for i in 1 to prof-1 loop
      Rlec(i) <= '0';
      Recr(i) <= '0';
    End loop;
    file_vide <= '1';
    file_pleine <= '0';
    operation <= '0';
  end if;
 end process;
--ecriture
 Recr<= Recr ror 1 after 1ps
             when ecr'event and ecr='0'and file_pleine='0';
 mem(select_dir(Recr)) <= entree
             when ecr'event and ecr='0'and file_pleine='0';
 operation<='1'after 1ps when ecr'event and ecr='0'and
           file_pleine='0';
 file_vide <= '0'when ecr'event and ecr='0'and file_pleine='0';
 file_pleine <= '1' when Recr=Rlec and operation='1';
 --lecture
 Rlec<= Rlec ror 1 after 1ps
```

```
              when lec'event and lec='0'and file_vide='0';
 operation<='0'after 1ps when lec'event and lec='0'and
           file_vide='0';
 file_pleine <= '0'when lec'event and lec='0'and file_vide='0';
 file_vide <= '1' when Recr=Rlec and operation='0';

 sortie<= mem(select_dir(Rlec));
end arch3;
```

4. La coïncidence entre les écritures et les lectures n'est vraiment problématique que lorsque l'une d'entre elles risque de provoquer l'état file pleine ou file vide. Toutefois, pendant la coïncidence la bascule `operation` est sollicitée simultanément dans deux états différents (ce qui devrait provoquer un problème de simulation). Une analyse simple nous permet de remarquer qu'une coïncidence de change pas l'état de remplissage de la file car les deux pointeurs conservent leur écart relatif qui ne peut être nul. En effet, si cet écart était nul, il y aurait déjà l'état de pile pleine ou de pile vide qui bloquerait l'une des commandes et il n'y aurait plus de coïncidence. Le seul problème à résoudre concerne le remplacement de la bascule `operation`. Pour cela on utilisera deux signaux (bascules) `op_ecr` et `op_lec` qui seront respectivement mis à 1 par les fronts descendants de `ecr` et de `lec` et qui seront, tous les deux, remis à 0 par les fronts montants d'`ecr` et de `lec`.

➤ Quatrième description

```
package complements is
 function select_dir(a:bit_vector) return integer;
end complements;
package body complements is
 function select_dir(a:bit_vector) return integer is  --fonction
pour
  alias av:bit_vector(1 to a'length) is a;      --simuler la
  begin                                         --selection directe
   for i in 1 to a'length loop
    if av(i)='1' then return i-1;end if;
   end loop;
   return 0; -- cas ou le vecteur est a 0 (initialisation!!)
 end select_dir;
end complements;
library vector;
use vector.functions.all;
use work.complements.all;
entity fifo is
 generic(larg: integer=8;
         prof: integer);
 port(
      ecr :in bit:= '0';
      lec :in bit:= '0';
      rst :in bit:= '0';
      file_pleine :buffer bit;
      file_vide:buffer bit;
```

```
      entree :in bit_vector(larg-1 downto 0);
      sortie :out bit_vector(larg-1 downto 0));
end fifo;
architecture arch4 of fifo is
 type mat is array(0 to prof-1) of bit_vector(larg-1 downto 0);
 signal mem:mat;
 signal Rlec:bit_vector(0 to prof-1);
 signal Recr:bit_vector(0 to prof-1);
 signal op_ecr:bit:='0';
 signal op_lec:bit:='0';
 begin
 --initialisation de la file
 process (rst)     --initialisation de la file
 begin
  if rst='0' then  --sur front descendant de rst
    Rlec(0) <= '1';
    Recr(0) <= '1';
    for i in 1 to prof-1 loop
      Rlec(i) <= '0';
      Recr(i) <= '0';
    End loop;
    file_vide <= '1';
    file_pleine <= '0';
    op_ecr <= '0';
    op_lec <= '0';
  end if;
 end process;
 --ecriture
 Recr<= Recr ror 1 after 1ps
          when ecr'event and ecr='0'and file_pleine='0';
 mem(select_dir(Recr))<= entree
          when ecr'event and ecr='0'and file_pleine='0';
 op_ecr<='1' after 1ps when ecr'event and ecr='0'and
file_pleine='0';
 file_vide<='0' when ecr'event and ecr='0'and file_pleine='0';
 op_ecr<='0'when (ecr'event and ecr='1')
               or(lec'event and lec='1');
 file_pleine<='1'when (Recr=Rlec and op_ecr='1');
 --lecture
 Rlec<= Rlec ror 1 after 1ps
           when lec'event and lec='0'and file_vide='0';
 op_lec<='1' after 1ps when lec'event and lec='0'and file_vide='0';
 file_pleine <='0' when lec'event and lec='0'and file_vide='0';
  op_lec<='0'when (ecr'event and ecr='1')
                or(lec'event and lec='1');
 file_vide<='1'when (Recr=Rlec and op_lec='1');

 sortie<= mem(select_dir(Rlec));
end arch4;
```

5. Dans ce cas, le problème à résoudre concerne l'élimination des fonctions `select_dir` et = ainsi que le `process` d'initialisation. Ceci ne peut se faire qu'en remplaçant le tableau `mem` par un ensemble de registres qui peuvent être sélectionnés directement par les différentes bascules des registres à décalage `Recr` et `Rlec`. La suppression de la fonction = nécessite de réaliser un `and` vectoriel entre les deux registres à décalage puis un `or` entre tous les bits du résultat. L'élimination du `process` d'initialisation suppose la possibilité d'initialiser les registres `Rlec` et `Recr` de `longeur` paramétrable.

Annexe **1**

Rappels d'algèbre de Boole

Cette structure algébrique a été étudiée par le mathématicien anglais Georges Boole (1815-1864) pour formaliser les règles de la logique des propositions. Elle a été publiée dans son ouvrage : « The Mathematical Analysis of Logic » en 1847.

A1.1 DÉFINITION

On appelle *algèbre de Boole*, un quadruplet $(B, \overline{\ }, \wedge, \vee)$ composé d'un ensemble $B = \{0, 1\}$, d'une opération unaire telle que $\forall a \in B \rightarrow \overline{a} \in B$ appelée *complémentation*, et de deux opérations binaires : $\wedge, \vee \in B \times B \rightarrow B$ appelées respectivement « `et` » et « `ou` ».

Pour tout $a, b, c \in B$, on a les égalités suivantes :

$$(a \wedge b) \wedge c = a \wedge (b \wedge c)$$
$$a \wedge c = c \wedge a$$
$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$
$$a \vee 0 = a$$
$$a \wedge \overline{a} = 0$$
$$(a \vee b) \vee c = a \vee (b \vee c)$$
$$a \vee c = c \vee a$$
$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$
$$a \wedge 1 = a$$
$$a \vee \overline{a} = 1$$

En partant de ces dix axiomes on constate que 0 est l'élément neutre de l'opération $\vee$ et 1 celui de l'opération $\wedge$.

De même, on montre que :

$$(\overline{a \wedge b}) = \overline{a} \vee \overline{b}$$
$$(\overline{a \vee b}) = \overline{a} \wedge \overline{b}$$

appelée la loi de De Morgan.

Il existe plusieurs formes possibles d'écriture de l'algèbre de Boole. Les éléments de l'ensemble B peuvent, par exemple, être appelés V, F et les opérations $\cap$, $\cup$ ou $\bullet$, $+$. La complémentation, notée ici par un surlignement est quelquefois notée par $\neg$ ou un $-$. Il est aussi possible de représenter une algèbre de Boole avec un seul opérateur binaire, par exemple le *NI*. On montre l'équivalence de cette forme en écrivant :

$$NI(a, b) = (\overline{a \vee b})$$

d'où :

$$\overline{a} = NI(a, a)$$
$$a \vee b = NI(NI(a, b), NI(a, b))$$
$$a \wedge b = NI(NI(a, a), NI(b, b))$$

Une autre forme est l'*anneau booléen* noté $(B, \oplus, \vee)$ dans lequel l'opérateur $\oplus$ est le *ou-exclusif* qui peut être défini par :

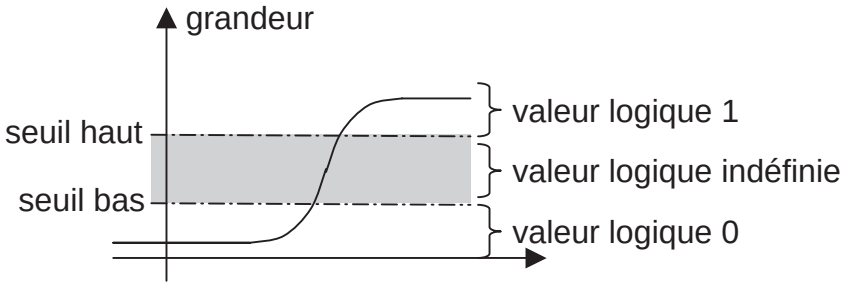
$$a \oplus b = (a \wedge \overline{b}) \vee (\overline{a} \wedge b)$$

A1.2 INTERPRÉTATION

L'algèbre de Boole, initialement développée pour formaliser les problèmes de la logique des propositions, s'applique à de très nombreux domaines dans lesquels l'ensemble des valeurs se réduit à deux éléments. Dans le cas de l'électronique, ces valeurs pourront être deux niveaux de tension, deux intensités ou bien le fait qu'un élément soit conducteur ou isolant. Il faut remarquer que dans le cas des fonctions logiques, les interprétations des arguments et du résultat pourront être identiques ou différentes. Par exemple, des niveaux de tension en argument et une conduction comme résultat.

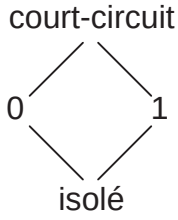
Comme beaucoup de ces grandeurs physiques sont continues, la définition des deux niveaux correspondant à des valeurs booléennes se fera *via* l'introduction de *seuils* hauts et bas.

Lorsque la valeur du signal est inférieure au seuil bas, on dira que sa *valeur logique* est 0, elle sera dite égale à 1 lorsque la valeur du signal est supérieure au seuil haut (ou réciproquement !). Lorsque la valeur du signal est entre les deux seuils, on dira que sa valeur logique n'est pas définie.



En fait, beaucoup de montages électroniques se modélisent à l'aide de fonctions incomplètement définies. En plus des valeurs logiques, il est nécessaire de représenter soit une valeur non définie, soit un état isolé. Ceci peut être formalisé en utilisant des *logiques multivaluées*, ternaires, ou d'arité plus importante, opérant sur des treillis de valeurs (c'est le cas du type `std_ulogic` de la librairie IEEE de VHDL).

Exemple : Treillis des états booléens et isolé.



De telles approches nous permettent de parler d'interrupteurs qui peuvent isoler ou connecter des sources logiques.

A1.3 FONCTIONS BOOLÉENNES

Nous définirons des *fonctions booléennes* à n arguments comme des applications : $f \in B^n \rightarrow B$. Les n arguments d'une telle fonction constituent un vecteur booléen qui peut prendre 2^n valeurs distinctes. Une fonction booléenne revient à attribuer des valeurs booléennes à ces valeurs des arguments. Elle peut donc être représentée par le tableau des 2^n valeurs de ces arguments auxquelles on associe la valeur de la fonction. Ce tableau est appelé la *table de vérité* de la fonction.

A1.3.1 Terme

On appelle *terme* un produit de variables booléennes directes ou complémentées.

Exemple : $a \wedge \overline{b} \wedge c$

A1.3.2 Forme canonique d'une fonction booléenne

On appelle *forme canonique* d'une fonction booléenne son écriture sous la forme d'une somme de termes, appelés *monômes*, car contenant l'ensemble des variables directes ou complémentées.

Exemple : $f = (a \wedge \overline{b} \wedge c) \vee (a \wedge \overline{b} \wedge \overline{c}) \vee (a \wedge b \wedge \overline{c})$

Cette écriture est assez lourde car elle comporte autant de termes que de valeurs à 1 dans la table de vérité de la fonction.

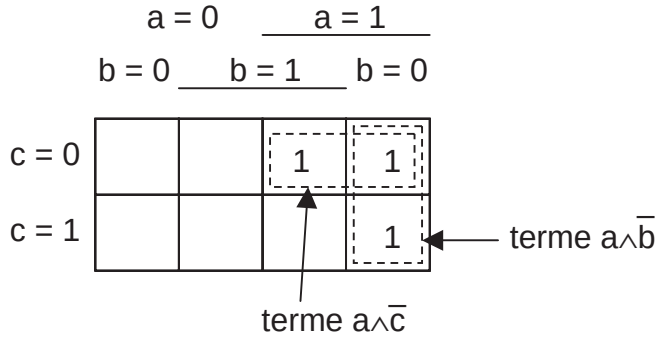
A1.3.3 Simplification d'une fonction booléenne

Il est possible de réduire la complexité de l'écriture précédente en regroupant des monômes pour obtenir un nombre plus réduit de termes. Par exemple, les monômes $a \wedge \overline{b} \wedge c$ et $a \wedge \overline{b} \wedge \overline{c}$ pourront être condensés dans le terme $a \wedge \overline{b}$.

Exemple : f peut ainsi s'écrire : $(a \wedge \overline{b}) \vee (a \wedge \overline{c})$

Si la fonction ne comporte que quelques variables, ce processus de simplification pourra être réalisé sur une représentation tabulée de cette fonction appelée *table de Karnaugh*.

Exemple : Table de Karnaugh de la fonction précédente :



Les zones rectangulaires pointillées dans ce tableau représentent des termes.

L'écriture comportant le minimum de termes sera appelée *forme minimale* et ses termes des *mintermes*. Elle sera utilisée pour minimiser la complexité de certains organes comme des PLA.

Il existe des outils informatiques [1, 2] très puissants, capables de minimiser des fonctions booléennes très complexes.

A1.3.4 Duale d'une fonction booléenne

On appelle *duale* d'une fonction booléenne $f(a, b, c)$ la fonction $f^* = \overline{f(\overline{a}, \overline{b}, \overline{c})}$. On montre que la fonction f^* s'obtient simplement en remplaçant les occurrences des opérateurs $\wedge$ par $\vee$ et $\vee$ par $\wedge$.

Exemple : si $f = a \vee b$ alors $f^* = a \wedge b$;

si $f = a \wedge (b \vee \overline{c})$ alors $f^* = a \vee (b \wedge \overline{c})$.

La notion de duale intervient dans l'étude des montages CMOS.

Fonctions auto-duales

On appelle *auto-duale* une fonction booléenne égale à sa duale.

Exemple : $f = a \oplus b \oplus c = \overline{(\overline{a} \oplus \overline{b} \oplus \overline{c})} = f^*$.

A1.3.5 Propriétés du OU-exclusif

Le ou-exclusif est une fonction booléenne très importante, douée de plusieurs propriétés très intéressantes :

commutativité : $a \oplus b = b \oplus a$
associativité : $a \oplus (b \oplus c) = (a \oplus b) \oplus c = a \oplus b \oplus c$
élément « neutre » : $a \oplus 0 = a$
et aussi : $a \oplus 1 = \overline{a}$
$\overline{(a \oplus b)} = \overline{a} \oplus b = a \oplus \overline{b}$
$a \oplus b = \overline{a} \oplus \overline{b}$
si $c = a \oplus b$, alors $b = a \oplus c$.

Toutes ces propriétés sont très utiles pour la réalisation de portes ouex et non-ouex.

A1.3.6 Vision dissymétrique des fonctions booléennes

Il est possible de voir de manière dissymétrique certaines fonctions booléennes :

Par exemple : $a \wedge b$ peut être vu comme *si* $b = 1$ *alors* a *sinon* 0, c'est-à-dire que b joue le rôle d'une sorte de commutateur entre la valeur de a et 0. Cette propriété est utilisée soit pour sélectionner une partie d'un mot par un masque, soit pour forcer à 0 une partie de ce mot.

Exemple :

	a 10010110	
$\wedge$	b 000**1111**0	(masque de sélection)
	s 000**1011**0	(zone sélectionnée)

Exemple :

	a 10010110	
$\wedge$	b 1111**0000**	(masque de forçage)
	s 1001**0000**	(zone forcée à 0)

$a \wedge b$ peut être vu comme *si* $b = 1$ *alors* 1 *sinon* a, c'est-à-dire que si b vaut 1 alors la valeur de sortie est forcée à 1. Cette propriété est utilisée pour forcer à 1 une partie d'un mot.

Exemple :

a 10010110
$\vee$ b 0000**1111** (masque de forçage)
s 1001**1111** (zone forcée à 1)

$a \vee b$ peut aussi être vu comme *si* $b = 1$ *alors* 1 *sinon* a, c'est-à-dire un commutateur entre l'un des opérandes et son complémentaire.

BIBLIOGRAPHIE

[1] R.E. Bryant: *Graph-Based Algorithms For Boolean Function Manipulation*, IEEE Transaction on Computers, Vol C-35 n° 8, August 1986.

[2] J.-C. Madre, J.-P. Billon: *Proving Circuit Correctness by Formaly Comparing their Expected and Extracted Behaviour*, Proceeding of the 25th Design Automation Conference, Anaheim, june 1988.

Annexe **2**

Étude d'une montre avec affichage

L'objectif de cette étude est de montrer l'application de la méthode de conception présentée au chapitre 9 sur un exemple plus complexe. Ceci permet de montrer l'influence de ces extensions sur la structure du circuit obtenu et en particulier sur l'introduction d'un décodeur dans le chemin de données. Cet exemple est une extension de l'étude de la montre simple donnée comme exemple de la méthode de conception développée dans le chapitre 9.

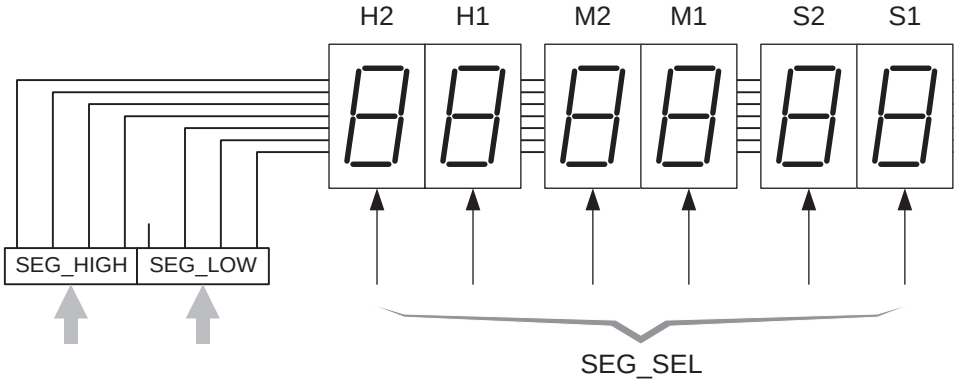
A2.1 ORGANISATION DE L'AFFICHAGE

L'affichage de la montre sera réalisé à l'aide de six afficheurs « 7-segments » sans électronique. Ces afficheurs recevront la même information concernant les segments à afficher. Celle-ci proviendra de deux variables de quatre bits appelés : `seg_low` et `seg_high`. Ils seront individuellement sélectionnés par une variable de six bits appelée `seg_sel`.

Pour minimiser le matériel utilisé, les six chiffres seront affichés séquentiellement dix fois par seconde pour donner l'impression d'un affichage continu. Le temps d'affichage de chaque chiffre sera donc de 1/60 seconde.

A2.2 NOUVEL ALGORITHME

Le soixantième de seconde est une ancienne unité de temps appelée *tierce*. Le nouvel algorithme de la montre effectuera un tour par tierce. Pendant un cycle de six tierces,



il effectuera l'affichage des six chiffres. Dix cycles se succèderont dans chaque seconde.

Nous ajouterons donc deux décomptages par six puis par dix à l'algorithme de la montre.

```
package complements is
   subtype quad is bit_vector(3 downto 0); -- definition format
   subtype hex is bit_vector(0 to 5);
end complements;
library vector;
use vector.functions.all;
use work.complements.all;
entity montre_aff is
 port(seg_low,seg_high:out quad;
      seg_sel:out hex;
      clk, rst:in bit);
end montre_aff;
architecture arch1 of montre_aff is
 signal t1,t2,s1,s2,m1,m2,h1,h2:quad;
 signal valid:bit:='0';
function decode_sel(entree:quad) return hex is
 variable res:hex;
 begin
   res:="000000";
   if to_natural(entree)<6 then
      res(to_natural(entree)):='1';
   end if;
   return res;
 end decode_sel;
function decode_low(entree:quad) return quad is
 begin
   case entree is
     when x"0" => return "0111";
     when x"1" => return "0001";
     when x"2" => return "0110";
```

```
      when x"3" => return "0011";
      when x"4" => return "0001";
      when x"5" => return "0011";
      when x"6" => return "0111";
      when x"7" => return "0001";
      when x"8" => return "0111";
      when x"9" => return "0011";
    end case;
 end decode_low;
function decode_high(entree:quad) return quad is
 begin
   case entree is
     when x"0" => return "1011";
     when x"1" => return "0010";
     when x"2" => return "0111";
     when x"3" => return "0111";
     when x"4" => return "1110";
     when x"5" => return "1101";
     when x"6" => return "1101";
     when x"7" => return "0011";
     when x"8" => return "1111";
     when x"9" => return "1111";
   end case;
 end decode_high;
begin
 seg_sel<= decode_sel(t1) when valid='1' else "000000";

 process(rst)  -- mise dans un etat initial connu (23h 59m 40s)
 begin
  t1 <= x"0";
  t2 <= x"0";
  s1 <= x"0";
  s2 <= x"4";
  m1 <= x"9";
  m2 <= x"5";
  h1 <= x"3";
  h2 <= x"2";
  valid <= '0';
 end process;

 process
  variable tv1,tv2,sv1,sv2,mv1,mv2,hv1,hv2:quad;
  variable validv:bit;
  begin
   wait until clk'event and clk='1'; --front montant de clk
-- transferts signaux => variables
   tv1:=t1;
   tv2:=t2;
   sv1:=s1;
   sv2:=s2;
```

```
   mv1:=m1;
   mv2:=m2;
   hv1:=h1;
   hv2:=h2;
   validv:=valid;
-- description de l'algorithme
validv:= '0'; -- extinction de l'affichage
case tv1 is  -- decodage des segments
   when x"0" => seg_low <= decode_low(sv1);
                seg_high <= decode_high(sv1);
   when x"1" => seg_low <= decode_low(sv2);
                seg_high <= decode_high(sv2);
   when x"2" => seg_low <= decode_low(mv1);
                seg_high <= decode_high(mv1);
   when x"3" => seg_low <= decode_low(mv2);
                seg_high <= decode_high(mv2);
   when x"4" => seg_low <= decode_low(hv1);
                seg_high <= decode_high(hv1);
   when x"5" => seg_low <= decode_low(hv2);
                seg_high <= decode_high(hv2);
end case;
tv1:= tv1+x"1";
validv:= '1'; -- allumage de l'affichage
if tv1 = x"6" then  -- test a 6
  tv1:= x"0";
  tv2:= tv2+x"1";
  if tv2 = x"A" then -- test a 10
   tv2:= x"0";
   sv1:=sv1+x"1";
   if sv1 = x"A" then
    sv1:=x"0";
    sv2:=sv2+x"1";
    if sv2 = x"6" then
     sv2:=x"0";
     mv1:=mv1+x"1";
     if mv1 = x"A" then
      mv1:=x"0";
      mv2:=mv2+x"1";
      if mv2 = x"6" then
       mv2:=x"0";
       hv1:=hv1+x"1";
       if (hv2 = x"2") and (hv1 = x"4") then
        hv1:=x"0";
        hv2:=x"0";
       elsif hv1 = x"A" then
        hv1:=x"0";
        hv2:=hv2+x"1";
       end if;
      end if;
     end if;
```

```
        end if;
      end if;
    end if;
  end if;
-- transferts variables => signaux
   t1 <= tv1;
   t2 <= tv2;
   s1 <= sv1;
   s2 <= sv2;
   m1 <= mv1;
   m2 <= mv2;
   h1 <= hv1;
   h2 <= hv2;
   valid <= validv;
 end process;
end arch1;
```

Le décodage des chiffres BCD en 7 segments sera effectué en deux temps par deux fonctions `decode_low()` et `décode_high()` pour utiliser les moyens du chemin de données de 4 bits.

Les afficheurs seront sélectionnés par la variable `seg_sel` issue du décodage de la variable `t1` conditionné par le signal `valid`. La mise à 0 de ce signal pendant le changement de la valeur de `t1`, de `seg_low` et de `seg_high` permet d'éviter une perturbation temporaire de l'affichage.

A2.3 OPTIMISATION DE L'ALGORITHME

Plusieurs interprétations sont possibles pour donner un sens matériel à l'instruction case. Celle que nous choisissons consiste à effectuer un adressage indexé des registres à afficher. Pour cela nous renommeront ces registres comme une nappe de fils unique `R(0 to 5)` par les instructions « toujours vraies » suivantes :

```
R(0) <= h2;
R(1) <= h1;
R(2) <= m2;
R(3) <= m1;
R(4) <= s2,
R(5) <= s1;
```

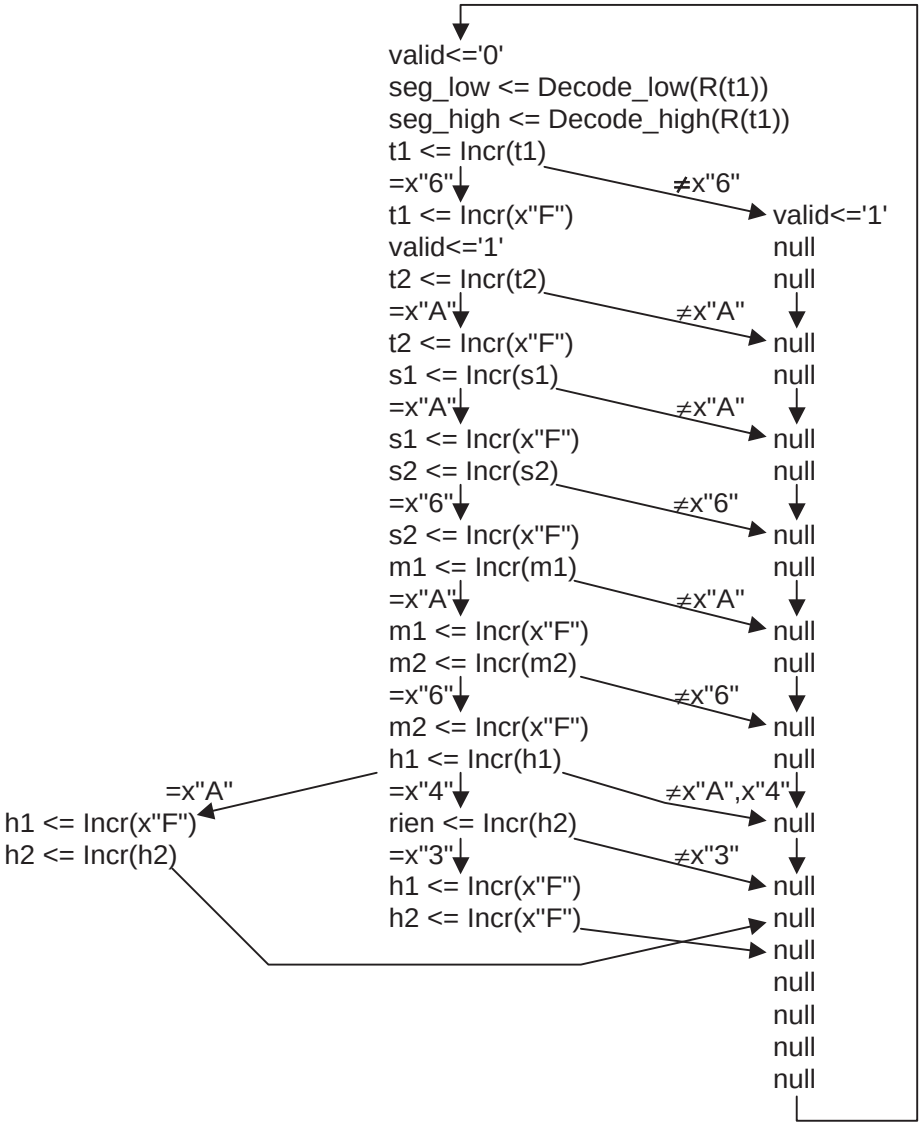
Ceci permet de transformer l'instruction `case` en un couple d'instructions suivantes:

```
seg_low <= Decode_low(to_natural(R(t1)));
seg_high <= Decode_high(to_natural(R(t1)));
```

Qui se traduira par un simple décodeur.

A2.3.1 Organigramme

Celui-ci ressemble beaucoup à l'organigramme de la montre simple. La boucle est allongée à 25 instructions, cela signifie que cette montre fonctionnera avec une horloge plus rapide de 1 500 hz.



A2.4 CONCEPTION DU CHEMIN DE DONNÉES

Cette étape reprend tous les éléments du chemin de données de la montre simple. Il ne faut lui ajouter que :

– Les registres `t1`, `t2`, `seg_low` et `seg_high` de 4 bits.
– La bascule `valid`
– Un décodeur qui sélectionne :
  • Les registres `h2`, `h1`, `m2`, `m1`, `s2`, `s1` pour les connecter sur le bus `source`.
  • Les afficheurs `h2`, `h1`, `m2`, `m1`, `s2`, `s1` conditionnés par la bascule `valid`.

A2.4.1 Forme standard

La nouvelle forme standard étend celle de la montre classique. Elle s'écrira :

```
<dest> <= <ope> <source>
```

Les champs <dest>, <ope> et <source> de la nouvelle forme standard pourront contenir :

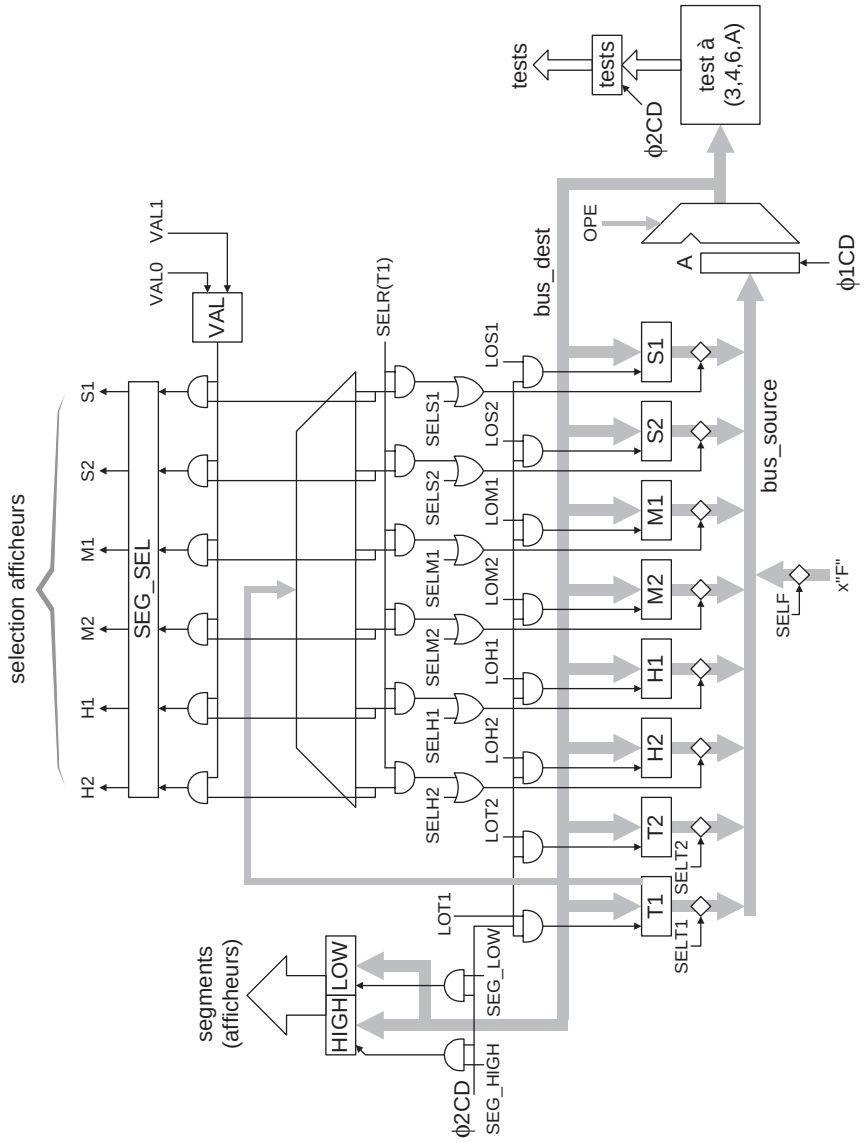
<dest>	<ope>	<source>
rien	Incr	X"F"
t1	Decode_high	t1
t2	Decode_low	t2
s1		s1
s2		s2
m1		m1
m2		m2
h1		h1
h2		h2
seg_high		R(t1)
seg_low		
valid<=0		
valid<=1		

L'opérateur reste unaire, mais il doit maintenant être capable de réaliser trois opérations :

– l'incrémentation comme celui de la montre simple ;
– le décodage des 4 segments hauts pour l'affichage 7 segments ;
– le décodage des 3 segments bas pour l'affichage 7 segments.

Les lignes `valid<=0` et `valid<=1` de la colonne <dest> ne sont pas vraiment des destinations, leur décodage provoquera la mise à `0` et à `1` de la bascule de validation de la sélection des afficheurs.

A2.4.2 Schéma du chemin de données



Le décodeur et sa logique associée peuvent être dessinés comme des tranches supplémentaires élargissant le chemin de données et simplifiant son dessin. Cette extension latérale comportera un nouveau bus véhiculant le contenu du registre `t1`.

A2.5 CONCEPTION DU SÉQUENCEUR

Comme pour la montre simple, le séquenceur de la montre avec affichage peut être réalisé soit à l'aide d'un PLA, soit avec des portes.

Dans le cas d'une réalisation avec un PLA, le nombre de micromots passera de 16 à 25. Le nombre de champs de chaque micro-mot passera de trois à quatre et la largeur des champs <dest> et <source> passera de trois à quatre bits.

Le décodeur du champ <dest> devra comporter deux irrégularités qui correspondront aux pseudo-destinations `valid<='0'` et `valid<='1'`. Les sorties de leur décodage serviront à positionner la bascule `valid`, tandis que le chemin de données effectuera une opération `null`.

Contenu du PLA :

Matrice ET		Matrice OU			
etat	**conditions**	**dest**	**ope**	**source**	**et. suiv.**
E0		Valid<='0'	Incr	x"F"	E1
E1		Seg__Low	Decode_low	R(T1)	E2
E2		Seg__High	Decode_high	R(T1)	E3
E3		T1	Incr	T1	E4
E4	=x"6"	T1	Incr	x"F"	E5
E4	≠x"6"	Valid<='1'	Incr	x"F"	E7
E5		Valid<='1'	Incr	x"F"	E6
E6		T2	Incr	T2	E9
E7		rien	Incr	x"F"	E8
E8		rien	Incr	x"F"	E9
E9	=x"A"	T2	Incr	x"F"	E10
E9	≠x"A"	rien	Incr	x"F"	E11
E10		S1	Incr	S1	E12
E11		rien	Incr	x"F"	E12
E12	=x"A"	S1	Incr	x"F"	E13
E12	≠x"A"	rien	Incr	x"F"	E14
E13		S2	Incr	S2	E15
E14		rien	Incr	x"F"	E15
E15	=x"6"	S2	Incr	x"F"	E16
E15	≠x"6"	rien	Incr	x"F"	E17
E16		M1	Incr	M1	E18
E17		rien	Incr	x"F"	E18
E18	=x"A"	M1	Incr	x"F"	E19
E18	≠x"A"	rien	Incr	x"F"	E20
E19		M2	Incr	M2	E21
E20		rien	Incr	x"F"	E21
E21	=x"6"	M2	Incr	x"F"	E22
E21	≠x"6"	rien	Incr	x"F"	E23
E22		H1	Incr	H1	E24
E23		rien	Incr	x"F"	E24

Matrice ET		Matrice OU			
etat	**conditions**	**dest**	**ope**	**source**	**et. suiv.**
E24	=x"A"	H1	Incr	x"F"	E25
E24	=x"4"	rien	Incr	H2	E26
E24	≠x"4", ≠x"A"	rien	Incr	x"F"	E26
E25		H2	Incr	H2	E28
E26	=x"3"	H1	Incr	x"F"	E27
E26	≠x"3"	rien	Incr	x"F"	E28
E27		H2	Incr	x"F"	E29
E28		rien	Incr	x"F"	E29
E29		rien	Incr	x"F"	E30
E30		rien	Incr	x"F"	E31
E31		rien	Incr	x"F"	E32
E32		rien	Incr	x"F"	E33
E33		rien	Incr	x"F"	E0

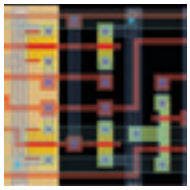
Index

50036 – (I) – (1,4) – OSB 80° – ALL – JME

Achevé d'imprimer sur les presses de
SNEL Grafics sa
Z.I. des Hauts-Sarts - Zone 3
Rue Fond des Fourches 21 – B-4041 Vottem (Herstal)
Tél +32(0)4 344 65 60 - Fax +32(0)4 289 99 61
mars 2007 – 41205

Dépôt légal : mars 2007
Imprimé en Belgique

SCIENCES SUP



François Anceau, Yvan Bonnassieux

CONCEPTION DES CIRCUITS VLSI

DU COMPOSANT AU SYSTÈME

La maîtrise de la conception des circuits intégrés VLSI *(Very Large Scale Integration)* est nécessaire au développement d'une industrie électronique performante. Cet ouvrage présente les techniques de conception des circuits intégrés CMOS complexes, du composant jusqu'à l'aspect système. Il aborde par conséquent les grands principes de la micro-électronique.

Les auteurs présentent les méthodes et les techniques sous-jacentes au travail de conception de circuits *« full custom »*. Les approches modernes de conception par « compilation de silicium » sont également abordées. Un exemple complet de conception à partir d'une description comportementale est traité.

Destiné aux élèves ingénieurs et aux étudiants en Master d'électronique et d'informatique, ce cours est complété par de nombreux exercices de conception avec corrigés. Cet ouvrage intéressera également les chercheurs et les ingénieurs.

FRANÇOIS ANCEAU

est professeur au Conservatoire National des Arts et Métiers, chercheur au laboratoire SOC/Lip6 de l'université Pierre et Marie Curie, fondateur du service CMP de réalisation de circuits intégrés pour l'enseignement et la recherche.

YVAN BONNASSIEUX

est maître de conférences à l'École Polytechnique, agrégé, ancien élève de l'Ecole Normale Supérieure de Cachan.

MATHÉMATIQUES
PHYSIQUE
CHIMIE
SCIENCES DE L'INGÉNIEUR
INFORMATIQUE
SCIENCES DE LA VIE
SCIENCES DE LA TERRE

LICENCE 1 2 3 | MASTER 4 5 | DOCTORAT 6 7 8

9 782100 500369

6494454
ISBN 978-2-10-050036-9

www.dunod.com

DUNOD