Architecture et micro-architecture des processeurs

Bernard Goossens

Je dédie ce livre à la mémoire de Louis Nolin, mon maître.

Sheet number 4

Préface

de Jean-Luc Gaudiot
Professeur à University of California, Irvine (UCI)

Une fois de plus, l'architecture des ordinateurs se trouve à une croisée des chemins. Le succès du modèle RISC et les progrès apportés par les modèles superscalaires nous ont donné des machines individuelles aussi puissantes que ce qui était considéré comme un superordinateur il y a seulement quelques années. Ce spectaculaire gain de performance n'a été possible qu'en combinant les avancées technologiques (comme la réalisation de circuits VLSI avec une finesse de gravure submicronique) à de nouveaux modèles de mémoires (à la RAMBUS), à des stratégies modernes d'organisation, ainsi qu'à des architectures innovantes. Cependant, plusieurs approches architecturales sont en compétition pour les processeurs du futur, allant de machines superscalaires à réordonnancement dynamique de degré élevé, aux concepts VLIW (*Very Long Instruction Word*) et EPIC (*Explicitly Parallel Instruction Computing*). Il est intéressant de remarquer que des approches hier totalement impraticables pour des raisons de restriction de ressources matérielles sont soudain devenues réalistes, voire nécessaires, alors que des constructions complexes du passé sont devenues obsolètes sans doûte par le seul fait de la profusion de ressources.

C'est cette jungle de machines, de techniques architecturales et d'implantations existantes que Bernard Goossens vient mettre en ordre en se basant sur le principe selon lequel l'architecture des ordinateurs ne peut exister pour elle-même, étant fermement ancrée aux technologies d'implantation. Il s'est ainsi efforcé de vous emmener vous, le lecteur, pour un voyage partant du plus profond de l'implantation, au niveau des transistors, et menant aux concepts architecturaux les plus avancés, pour décrire la microarchitecture d'un puissant microprocesseur d'aujourd'hui. Cette approche et l'accent mis sur la compréhension des couches basses de la construction donnent à l'ouvrage un caractère unique, très pédagogique et même, d'une certaine façon, divertissant.

L'auteur commence sa description en construisant une hypothétique machine "typique", comparable à celle qui nous est familière et qui est devenu un produit de consommation courante. Il fait ainsi allusion à l'interaction entre les divers éléments (processeur, mémoire et différents contrôleurs), aux standards des composants mémoires et des bus d'interconnexion, à la structure des entrées-sorties et à la fonctionnalité globale telle qu'elle est perçue par l'utilisateur, décrivant par exemple la suite des événements internes qui interviennent à la suite d'un déplacement brusque et aléatoire de la souris. Cela met certainement en appétit pour les deux chapitres qui suivent: une description exhaustive des principes actuels de conception des circuits et de la façon dont les transistors CMOS peuvent être tout d'abord optimalement utilisés pour construire les portes usuelles et ensuite assemblés en circuits de complexité variable (tels que les additionneurs, les contrôleurs, les mémoires, etc.). Ce chapitre montre aussi l'organisation des contrôleurs de mémoire cache avec un admirable luxe de détails, ainsi que l'organisation complète d'un processeur simple.

Les chapitres suivants s'appuient sur ces connaissances fondamentales et décrivent d'abord un jeu d'instructions (*Instruction Set Architecture*, la machine telle que la perçoit le programmeur ou le concepteur de compilateur) et l'optimisation d'un ensemble d'instructions dans un microprocesseur moderne. L'auteur introduit ensuite un exemple d'implantation (micro-architecture) du processeur ainsi défini et de son système mémoire.

Ce fut un honneur d'écrire cette préface. Entre autre, j'ai pris un immense plaisir à lire le manuscrit. Les exercices judicieusement disséminés dans le texte sont perspicaces et en même temps incitent à la réflexion. Le livre de Bernard Goossens vient à propos et profitera aux étudiants comme aux praticiens de l'architecture des ordinateurs.

Jean-Luc Gaudiot.

Table des matières

3: Des portes aux circuits *47*

Remerciements

Je tiens à remercier mes quatre relecteurs qui ont tous effectué un travail que je sais ingrat: lire de multiples fois, à différents niveaux, le même texte pour tenter d'en déceler les différents types d'erreurs. Par ordre alphabétique:

- Jean-Luc Gaudiot, Professeur à l'Université de California Irvine (UCI) qui m'a en outre fait l'amitié, mais surtout l'honneur, d'accepter de préfacer l'ouvrage.
- Bernard Lecussan, Professeur à SupAéro, qui a gentiment accepté que je présente son enfant, le petit processeur LUC3 comme illustration et synthèse du chapitre sur la circuiterie.
- Daniel Litaize, Professeur à Toulouse.
- Maurice Margenstern, Professeur à Metz, qui en plus de ce *pensum* a été dès l'origine à mes côtés sur ce projet de longue haleine.

Je remercie également André Seznec, Directeur de recherche à l'IRISA, pour avoir relu un manuscrit antérieur. Ses remarques ont fortement influencé l'aspect de ce livre.

Je remercie aussi Daniel Etiemble, Professeur à l'université de Toronto, de l'intérêt qu'il a manifesté pour cet ouvrage et de son expertise sur le chapitre des transistors.

Je remercie encore Daniel Krob, Directeur de recherche au CNRS et Directeur du LIAFA-Paris 7, et Michel Morvan, Professeur à l'ENS Lyon, tous deux codirecteurs de collection chez Springer-France, pour la confiance qu'ils m'ont constamment accordée concernant ce projet éditorial.

Je remercie enfin la correctrice de Springer-France pour son travail méticuleux (moi qui croyais maîtriser le français).

Sheet number 12

Avant-propos

Vous qui passez par ces lignes, acheteur potentiel, en quoi ce livre peut-il vous être utile? Si vous êtes simple utilisateur de l'ordinateur, surfeur sur l'Internet, médecin utilisateur du logiciel de gestion de votre cabinet ou encore passionné de jeux vidéos 3D, le fonctionnement précis de votre machine ne peut relever que de la curiosité. Cet ouvrage va alors très certainement bien au-delà de votre attente.

En revanche, si vous êtes informaticien de métier, ou apprenti, alors tout ou partie de ce livre vous est indispensable. Même si par le passé vous avez reçu une formation en architecture, il y a fort à parier pour que vos connaissances ne soient plus d'actualité. Même si, étudiant que vous êtes, vous n'avez pas l'ambition de travailler dans le domaine des machines, mais de vous spécialiser dans le logiciel de haut niveau, vous ne pouvez faire l'impasse sur le matériel, au risque de ne pas très bien comprendre comment s'exécute un programme. Enfin, si vous êtes, ou rêvez de devenir, concepteur de processeurs ou de logiciels de bas niveaux, par exemple de compilateurs ou de systèmes d'exploitation, la connaissance de tout ce qui se trouve dans ce livre et même au-delà, vous est indispensable.

Ce livre est subdivisé de manière à former le matériau de base de trois cours sur le matériel. L'introduction est un survol rapide de la structure et du fonctionnement de l'environnement d'un processeur dans une machine de type PC. Cette entrée en matière peut servir d'initiation à l'informatique côté matériel soit une ou deux leçons de niveau DEUG.

Cet ouvrage comprend ensuite deux chapitres qui peuvent former un cours de circuiterie. De façon ascendante, en partant des transistors, on construit un processeur rudimentaire. En implantant le processeur en circuits, on peut se faire une idée très précise de l'exécution d'un programme en machine. Le langage machine que le processeur définit permet de faire le lien avec les langages évolués que tout informaticien manipule. Le public visé est l'ensemble des étudiants en informatique. Un cours de ce type dure environ un semestre (une trentaine d'heures de cours et autant d'exercices; cette partie s'achève par la description précise d'un sujet de projet de synthèse).

Une fois ce premier palier franchi, le livre propose un chapitre consacré à l'architecture. Il forme un cours qui complète les enseignements de compilation et de système. On y découvre ce que sont les langages machines d'aujourd'hui: calcul entier, calcul flottant, calcul scalaire, calcul vectoriel; instructions prédicatives, instructions de contrôle, instructions systèmes; architectures RISC et EPIC. Typiquement, il s'agit d'un enseignement qui prend sa place en licence d'informatique, sur une durée pouvant aller de 20 à 30 heures de cours (et autant pour les exercices), selon que l'on se contente de présenter le calcul entier et le calcul flottant ou que l'on y inclut le calcul SIMD (unités vectorielles).

En plus de l'architecture, le livre présente également un chapitre sur la micro-architecture. C'est là la partie des spécialistes. Elle étudie l'implantation de l'architecture à partir des circuits présentés dans la première partie du livre. Il s'agit de l'état de l'art, en l'occurrence une micro-architecture spéculative et superscalaire. La description inclut les unités de calculs, y compris les opérateurs flottants et vectoriels. Les caches, le renommage des registres, la prédiction des sauts et le traitement en parallèle des instructions en sont les thèmes centraux. Le chapitre s'achève sur la présentation d'un sujet de TER (Techniques d'Étude et de Recherche) typique de l'activité de recherche d'un micro-architecte. Ce cours a sa place comme option de maîtrise ou encore en DESS ou DEA orienté vers le matériel, sur une durée d'environ 25 heures.

Enfin, le livre se termine par un chapitre consacré à l'environnement du processeur: sa mémoire hiérarchisée et les bus qui le relient aux périphériques.

1

Introduction

Dans cette introduction, en nous appuyant sur la description sommaire d'un micro-ordinateur actuel, nous montrons comment un processeur interagit avec son environnement pour exécuter des programmes ainsi que les communications d'informations que cela induit. Ensuite, nous définissons l'architecture et la micro-architecture et nous faisons apparaître les liens entre ces deux domaines et les autres domaines de l'informatique. Enfin, nous passons en revue les différents aspects de l'architecture et de la micro-architecture d'un processeur en précisant dans quel chapitre du livre ils sont traités.

1 La carte mère d'un PC

La figure 1.1 p.4 représente la *carte mère* d'un micro-ordinateur de bureau actuel. La carte mère est la colonne vertébrale de l'ordinateur. Il s'agit d'un support imprimé de pistes métalliques reliant ses circuits et ses connecteurs. Parmi ces circuits, on trouve le processeur, la mémoire et des circuits de contrôle. Les liaisons entre les circuits forment des bus, dont la caractéristique principale est la bande passante, produit de la fréquence par la largeur (la bande passante se mesure en octets par seconde; la largeur d'un bus doit donc être exprimée en octets ou fraction d'octet). Les bus peuvent déboucher sur des connecteurs et des ports, qui servent de prise pour brancher une carte ou un périphérique. Ces connecteurs et ces ports respectent différents standards (qui évoluent avec la technologie) auxquels se conforment les constructeurs de périphériques.

Le processeur est relié à un contrôleur de mémoire qui prend à sa charge les liaisons entre le processeur et la mémoire centrale (chargement des caches), entre le processeur et la mémoire vidéo (construction d'image pour l'écran graphique), entre la mémoire vidéo et la mémoire centrale (rangement/chargement d'image sans passer par le processeur), entre le processeur et le contrôleur d'entrées-sorties (entrée-sortie du processeur), entre le contrôleur d'entrées-sorties et la mémoire centrale (accès en DMA: *Direct Memory Access*; par DMA, le contrôleur d'entrées-sorties peut lire ou écrire un bloc de données en mémoire centrale sans intervention du processeur) et enfin entre le contrôleur d'entrées-sorties et la carte graphique (transfert d'images de ou vers l'extérieur, sans intervention du processeur). Le bus reliant le processeur au contrôleur de mémoire est le bus système, parfois appelé bus FSB (*Front-Side Bus*). Il est cadencé à 266 Mhz et a une largeur de huit octets (cas d'un Athlon XP; on commence à voir des bus FSB à 333 Mhz), ce qui lui donne un débit crête de 2,128 GO/s ou 400 Mhz (cas d'un Pentium IV). La bande passante est alors de 3,2 GO/s.

Le bus mémoire, qui relie le contrôleur de mémoire à la mémoire centrale, est conforme au brochage et au cadencement des composants mémoires. Aujourd'hui, la mémoire centrale est composée de SDRAM (*Synchronous Dynamic Random Access Memory*) 133 Mhz ou de DDRAM (*Double data rate DRAM*) 266 Mhz (apparition récente de DDRAM 333 Mhz). Le bus d'accès occupe quatre octets de large, ce qui assure un débit de 533 MO/s pour la SDRAM et 1,066 GO/s pour la DDRAM. Pour les deux types de mémoire, le bus est cadencé au même rythme. Mais pour la DDRAM, la transmission de données peut s'effectuer deux fois par cycle. Enfin, on peut également réaliser la mémoire centrale avec des composants RDRAM (*Rambus DRAM*). Ces composants peuvent être cadencés à 800Mhz, ce qui donne au bus mémoire un débit crête de 3,2 GO/s.

En ce qui concerne le domaine graphique, le standard actuel est le bus AGP-4X (*Accelerated Graphics Port*). Il relie le contrôleur de mémoire à la carte grahique qui contient de la mémoire vidéo (sous forme de SDRAM ou de DDR-SDRAM). Le bus AGP est cadencé à 66 Mhz, avec quatre accès maximum par cycle (pour AGP-4X. On trouve de l'AGP, AGP-2X, AGP-4X et bientôt AGP-8X) et une largeur de quatre octets. Sa bande passante est donc de 1,066 GO/s (266 MO/s pour la version de base AGP, 533 MO/s pour la version 2X et 2,128 GO/s pour la version 8X).

Étant donné la position centrale du bus système, il est clair que sa bande passante doit être la plus élevée possible pour lui permettre de faire face aux diverses demandes du processeur. Néanmoins, il ne faut pas perdre de vue qu'une part très importante des échanges d'informations qui ont lieu au sein de la carte mère ne concerne pas le processeur et s'effectue directement entre un périphérique et la mémoire. Les contrôleurs mémoire et les deux bus mémoire (mémoire centrale et mémoire graphique) sont alors mis à contribution. Leur bande passante est donc tout aussi importante.

Derrière le contrôleur de mémoire se trouve le contrôleur d'entrées-sorties. Celui-ci gère le pilotage des différents bus de liaisons avec la grande variété de périphériques que l'on peut connecter à un ordinateur: périphériques accédés par blocs (disques, CD/DVD-ROM), périphériques accédés par caractères transmis sur une voie d'un octet (transmission parallèle; lecteur de disquette), périphériques accédés par caractères transmis sur une voie d'un bit (transmission série; clavier/souris).

Pour les disques, le standard sur PC est le bus IDE/UDMA-ATA/100 (IDE: *Integrated Drive Electronics*; ATA: *AT Attachment*; Ultra ATA ou Ultra DMA ou ATA/33; ATA/66 ou UDMA/66; ATA/100 ou UDMA/100). Contrairement au bus AGP, le bus IDE n'est pas synchronisé par une horloge. Le protocole d'échange est établi à partir de signaux de contrôles asynchrones. La technologie et la longueur employées pour la transmission des signaux définissent la vitesse d'échange maximale. Le bus IDE véhicule des données de deux octets. La norme ATA/100 permet d'atteindre un débit crête de 100 MO/s (33 MO/s pour ATA/33 et 66 MO/s pour ATA/66).

Pour les liaisons séries (transmission d'un bit à la fois), le standard est le bus USB (*Universal Serial Bus*). Le bus est cadencé à 12 Mhz ce qui autorise un débit crête de 1,2 MO/s. À chaque extrémité du bus, un circuit est chargé de sérier les bits de l'octet à transmettre et de réagglomérer ceux de l'octet reçu. Sur le bus USB se connectent des claviers, des souris, mais aussi des modems (un modem 56 K a un débit crête de 5,6 KO/s). Le standard USB 2.0 est une extension de l'USB. Le bus est cadencé à 480 Mhz pour un débit de 48 MO/s.

On peut connecter d'autres périphériques à l'ordinateur par l'intermédiaire d'une carte enfichée sur l'un des connecteurs d'extensions de la carte mère. La carte fille sert à la fois de contrôleur et de point de branchement du périphérique (la carte fille dispose d'une prise qui débouche sur l'arrière du boîtier du PC). Les connecteurs d'extensions de la carte mère respectent tous le format PCI (*Peripheral Component Interface*). Les connecteurs PCI sont reliés au contrôleur d'entrées-sorties par le bus PCI, qui est cadencé à 33 Mhz et fait quatre octets de largeur. Nous obtenons un débit crête de 133 MO/s. Au format PCI, on trouve des cartes sons, des cartes SCSI (*Small Computer System Interface*) etc. Cette interface est de même nature que l'IDE. On lui connecte principalement des disques. Le bus SCSI est cadencé à 80Mhz -Ultra SCSI160- et fait deux octets de large pour un débit crête de 160 MO/s. Cependant, sur un PC, le bus SCSI étant dépendant du bus PCI, le débit de ce dernier bride celui du premier. Mais sur PC, l'avantage du SCSI n'est pas la vitesse mais l'extensibilité. Il permet en effet de connecter des périphériques supplémentaires au-delà des quatre offerts par l'interface IDE, ce qui est utile par exemple sur un serveur doté de plus de quatre disques. Le standard PCI-2 est une extension du standard PCI. Le bus passe à 64 bits au lieu de 32 et est cadencé à 66 Mhz au lieu de 33, ce qui lui donne un débit de 528 MO/s.

Après le contrôleur d'entrées-sorties, on trouve le contrôleur de configuration et du démarrage. Ce circuit n'est en relation avec le processeur que pendant la phase de démarrage de l'ordinateur (ce qui explique sa position éloignée du processeur). Il contrôle la PROM (*Programmable Read Only Memory*) qui renferme le programme de démarrage. Pendant cette phase, les accès mémoires du processeur sont acheminés au travers du contrôleur de mémoire et du contrôleur d'entrées-sorties jusqu'au contrôleur de configuration. Celui-ci délivre en particulier le programme de démarrage qui commence par la procédure POST (*Power-On Self Test*). Celle-ci se base sur le descriptif de la configuration de l'ordinateur qui est conservé dans la RAM flash pour démarrer et tester les contrôleurs et circuits de la machine: par exemple, démarrage du rafraîchissement de la mémoire centrale, vérification de son bon fonctionnement... Le démarrage se poursuit par la recherche d'un volume contenant le système. Le rôle du contrôleur de configuration s'achève dès que le système a été installé en mémoire centrale (il reprend du service lors de chaque réinitialisation de l'ordinateur).

Les trois contrôleurs - de mémoire, d'entrées-sorties et de configuration - forment le jeu de circuits d'accompagnement du processeur (le *chipset*). À chaque nouvelle génération de processeur, le *chipset* change. En revanche, les bus - bus mémoire, bus AGP, bus IDE, bus PCI, bus USB - ne changent pas aussi fréquemment et doivent être compatibles avec leurs versions antérieures. Dans le cas contraire, les constructeurs de périphériques devraient brutalement modifier l'interfaçage de leurs produits avec le PC.

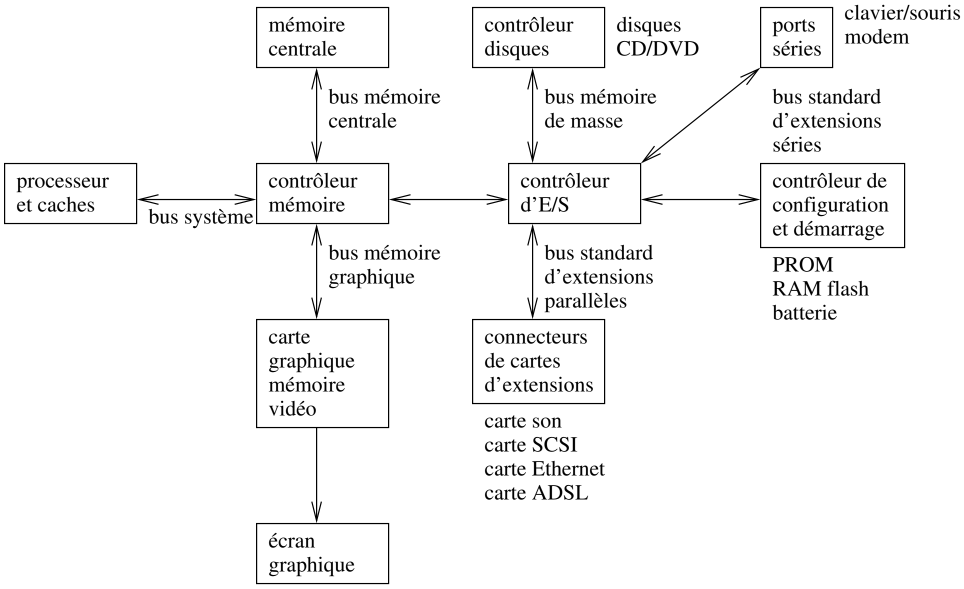

Figure 1.1 : La carte mère d'un micro-ordinateur.

Exercice 1.1: un film est lu à partir d'un disque connecté sur le bus IDE. On suppose que le transfert se fait à vitesse maximum (ATA/100). Le film est présenté dans une fenêtre de 512 par 384 pixels en 256 couleurs. Les images ne sont pas compressées. Quels sont les bus utilisés pour le transfert? Quelle part (pourcentage) de leur bande passante est consommée quand on transfère 24 images par seconde?

Exercice 1.2: cette fois, le film est lu à partir d'un navigateur Internet et via un modem assurant un débit moyen de 4 KO/s. Quels sont les bus utilisés pour le transfert? Combien de temps faut-il pour transférer une image? Comment peut-on améliorer la vitesse d'affichage?

2 L'exécution d'un programme sur un PC

À partir de la figure 1.1, on peut montrer le cheminement des informations induit par l'exécution d'un programme. Tout commence par un clic de souris sur une icône. Plus précisément, supposons que le PC soit dans un état de veille, caractérisé soit par un écran noir, soit par un économiseur d'écran en exécution. Dans ces deux cas, le bureau (l'ensemble des icônes définissant les liens avec les programmes directement accessibles depuis le terminal) n'est pas apparent à l'écran.

En accéléré, le lancement d'un programme nécessite un ordre en provenance de la souris (grâce à un double-clic sur une icône représentant le programme à exécuter ou en provenance du clavier, grâce à la saisie de son nom). Le programme est alors chargé en mémoire centrale depuis le disque et devient un processus. Le processeur en exécute les instructions et l'effet du programme apparaît, soit en mémoire, soit sur les périphériques. Mais pour mieux apprécier la sophistication du procédé, il faut regarder les choses en détail.

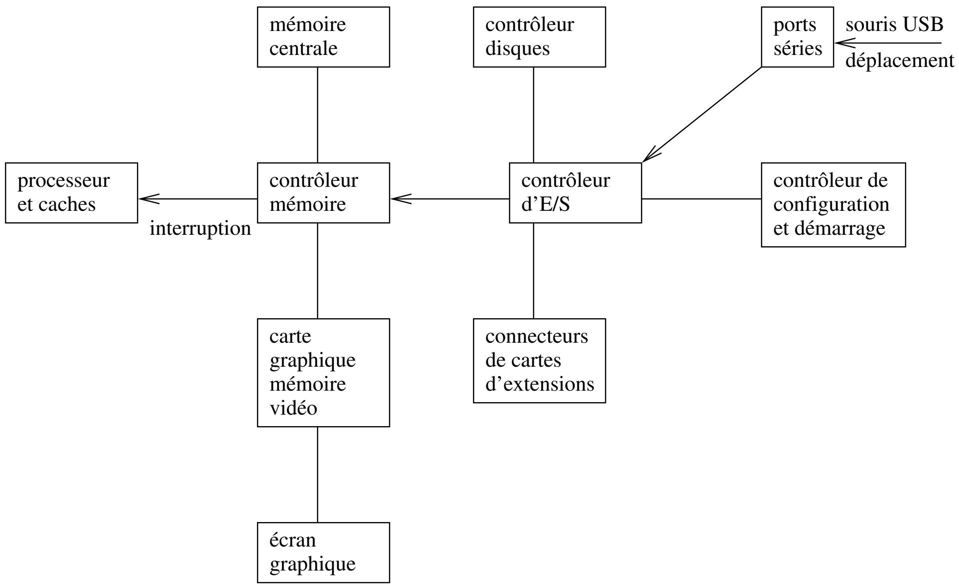

Figure 1.2 : Effet du déplacement de la souris.

Le mouvement de la souris sur son tapis va avoir un certain nombre d'effets (on ne parle pas encore du mouvement du pointeur sur l'écran; cela ne viendra que plus tard, en réaction). Le mouvement de la souris équivaut dans un premier temps à un déplacement mécanique. En effet, une première roue dentée, représentant le déplacement horizontal, et une seconde roue, représentant le déplacement vertical, tournent dans un certain sens et se décalent d'un certain nombre de crans. D'autres dispositifs équivalents peuvent transformer le mouvement en quantité. Un mouvement de grande amplitude se traduit par une succession de déplacements partiels. Ce déplacement mécanique est ensuite converti en une information numérique qui est transmise bit par bit à l'un des ports USB auquel est relié la souris. De là, l'information passe dans le contrôleur d'entrées-sorties via le bus USB (voir la figure 1.2), puis dans le contrôleur de mémoire (qui ne sert que de relais). En émettant une interruption, le contrôleur de mémoire signale au processeur l'arrivée d'une donnée en provenance de la souris. Cette interruption a pour effet de dérouter l'exécution du programme en cours sur le processeur vers un sous-programme d'accueil de la donnée. Le processus interrompu est par exemple l'économiseur d'écran (le processeur peut également être en état de veille, pour ne pas gaspiller d'énergie; l'interruption l'active donc). Le processeur, en exécutant le gérant d'interruption, récupère du contrôleur de mémoire ce qui provient de la souris et le dépose en mémoire. Dans un premier temps, cela reste interne au processeur, celui-ci disposant d'un cache. Un mot dans le cache n'est évacué vers la mémoire centrale que lorsqu'il faut l'y remplacer. Ce qui provient de la souris est placé dans une zone appartenant au processus gestionnaire du terminal graphique. Ce processus est pour l'instant en sommeil, attendant le client. Le gérant d'interruption, en même temps qu'il dépose l'information en provenance de la souris, réveille le gestionnaire de terminal graphique, qui est son destinataire.

Exercice 1.3: à quel débit s'effectue la transmission des informations concernant le déplacement de la souris?

Ce processus, une fois réveillé, récupère le déplacement de la souris et s'en sert pour calculer sa nouvelle position. Pour que l'utilisateur ne s'impatiente pas trop, le gestionnaire ordonne un déplacement équivalent au pointeur sur l'écran: voir la figure 1.3. Cette figure représente une succession d'ordres partant du processeur vers la carte graphique via le contrôleur de mémoire. Chaque ordre modifie les pixels formant l'image affichée: le pointeur disparaît car l'on donne à ses pixels la couleur du fond à cet endroit, puis il réapparaît un cran plus loin dans la direction de sa cible, et ainsi de suite jusqu'à ce que celle-ci soit atteinte. Après plusieurs déplacements successifs de la souris et du pointeur (USB vers processeur et processeur vers écran), le pointeur de la souris atteint l'icône du programme à lancer. L'utilisateur, voyant désormais son pointeur au-dessus de l'icône, effectue un double-clic (voir la figure 1.4 p.7). L'ordre est transmis au gestionnaire de terminal. Le terminal reproduit le double-clic en grisant l'image de l'icône à l'écran (figure 1.5 p.7). Le gestionnaire de terminal, par la position courante du pointeur, détermine le nom et le type associé à l'icône (c'est-à-dire le nom du programme à lancer et son type: programme exécutable, donnée associée à un programme manipulateur, dossier ou catalogue, ...). Le gestionnaire de terminal réveille l'interpréteur de commandes et lui communique une phrase signifiant un ordre d'exécution du programme (en employant le clavier plutôt que la souris, on compose soi-même cette phrase, que l'on peut enrichir de paramètres destinés au programme lancé).

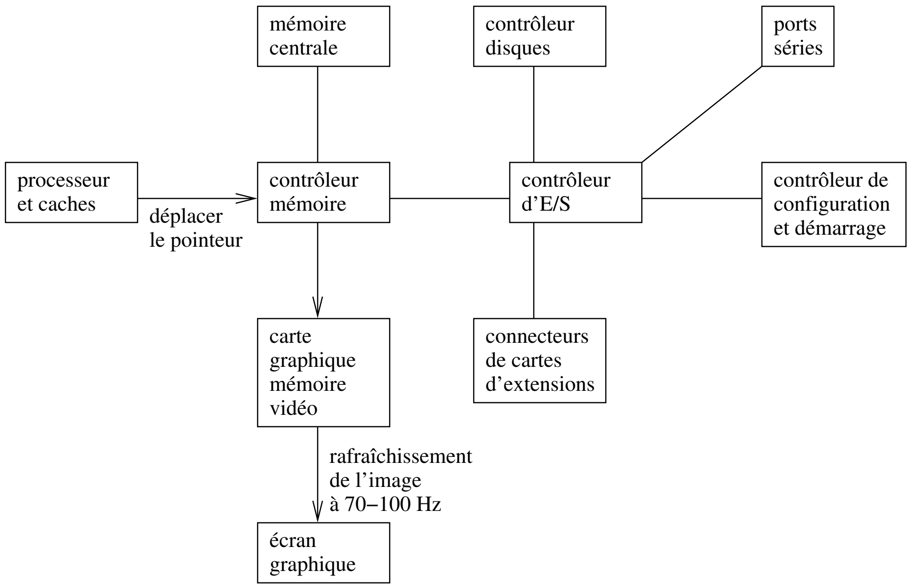

Figure 1.3 : Effet du déplacement de la souris sur l'écran.

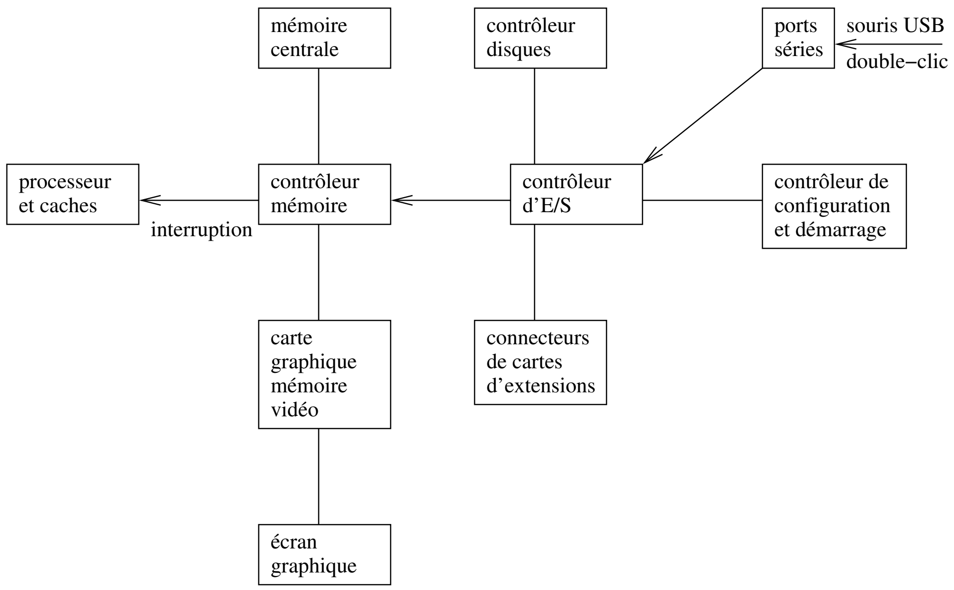

Figure 1.4 : Effet du double-clic de la souris sur le processeur.

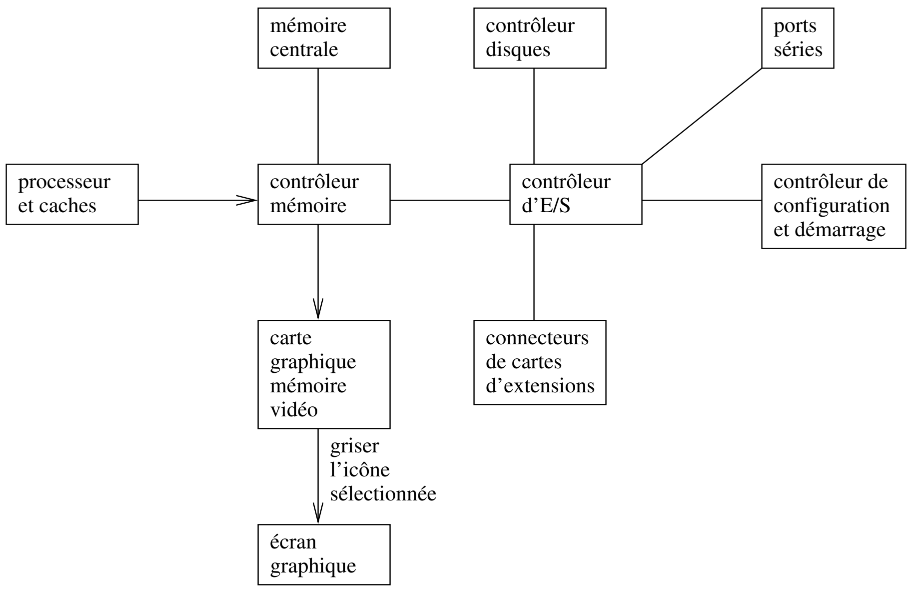

Figure 1.5 : Effet du double-clic de la souris sur l'écran.

Exercice 1.4: à quel débit s'effectue la transmission des informations concernant le déplacement du pointeur sur l'écran?

L'interpréteur de commandes charge une partie du programme en mémoire. Pour cela, il communique au noyau du système le nom du fichier exécutable correspondant au programme (ce que l'on appelle la référence absolue). Le noyau en déduit le support physique où se trouve le fichier (disque, CD-ROM, disquette) et son emplacement sur ce support (pour un disque: partition, cylindre, piste et secteurs). En fonction du support concerné (nous supposons dans la suite qu'il s'agit d'un disque), le noyau fait appel à un pilote adapté. Le pilote contacte le contrôleur d'entrées-sorties par l'intermédiaire du contrôleur de mémoire (figure 1.6) et lui réclame la lecture de différents blocs (ceux qui forment le début du fichier exécutable). Cette lecture se fait en DMA, sans intervention du processeur (figure 1.7 p.9). Le contrôleur d'entrées-sorties et le contrôleur de mémoire se chargent du transfert du disque vers la mémoire centrale. Un ordre de lecture est transmis au contrôleur IDE par le bus IDE. Le contrôleur IDE pilote alors le disque et lui ordonne d'effectuer le déplacement radial de la tête pour atteindre le cylindre visé à partir du cylindre courant. Les secteurs souhaités sont lus et leur contenu est transmis par le bus IDE au contrôleur d'entrées-sorties, deux octets à la fois. Le contrôleur de mémoire place ces données dans une page mémoire choisie par le pilote du noyau (écriture par mot de quatre octets via le bus mémoire). Cette page fait partie de l'espace mémoire réservé aux entrées-sorties par le système et est liée à l'interpréteur de commandes à l'origine de la lecture. Une fois l'amorce du programme installée en mémoire, le contrôleur de mémoire signale par interruption la fin de l'échange DMA au processeur (figure 1.8 p.10). Entre temps, celui-ci a pu effectuer d'autres tâches ou à défaut se mettre en veille.

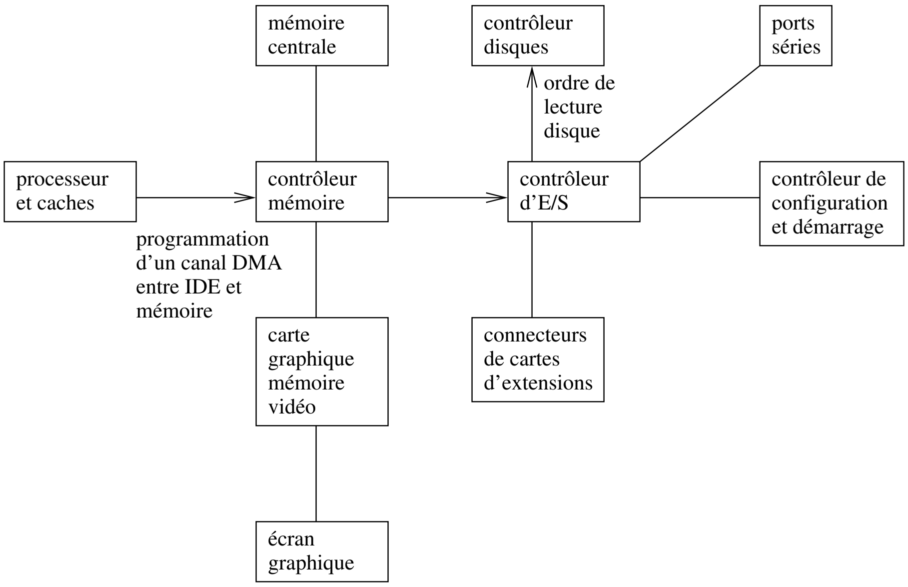

Figure 1.6 : Chargement d'un programme en mémoire: initialisation du canal DMA.

<u>Exercice 1.5</u>: à quel débit s'effectue le chargement du programme en mémoire?

L'interpréteur de commandes peut désormais ajouter un nouveau processus à ceux déjà lancés. Ce processus est relié aux pages de la mémoire centrale contenant son code et ses données initiales. Son exécution peut enfin démarrer. Au cours de cette exécution, de nouvelles entrées-sorties peuvent être nécessaires (affichage à l'écran, lecture au clavier, extension de l'espace mémoire résident par ajout de code, de pile ou de tas, lecture ou écriture de fichiers locaux ou distants, etc ...). Dans tous les cas, le processus s'en remet au système en appelant un pilote adapté au périphérique visé. Le pilote dialogue avec les contrôleurs pour lancer l'entrée-sortie. Comme celle-ci est de longue durée en regard de la vitesse du processeur, le système met le processus en sommeil. Il est réveillé par l'interruption du contrôleur de mémoire qui signale la fin de l'échange. Plus précisément, dans un système multi-tâches préemptif comme Unix, le gérant d'interruption remet le processus à l'état "prêt" et reprend l'exécution du processus interrompu. Le processus en entrée-sortie ne redevient actif qu'à son tour.

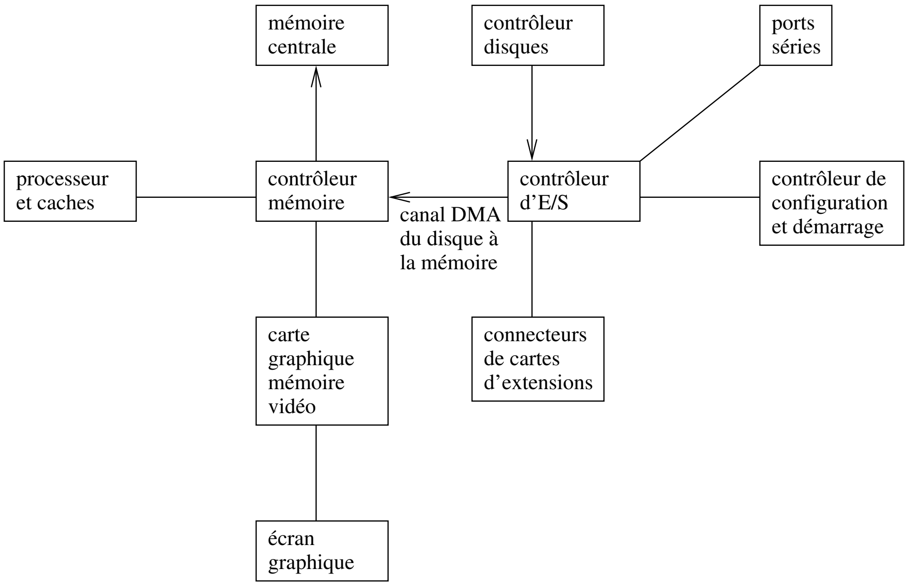

Figure 1.7: Chargement d'un programme en mémoire: transfert par le canal DMA.

Exercice 1.6: à quel débit s'effectue l'exécution du programme (cette fois, le débit doit être exprimé en instructions par seconde plutôt qu'en octets par seconde)?

3 L'interaction entre logiciel et matériel

Si l'on se contente d'utiliser un ordinateur pour exécuter des applications toutes faites, il n'est pas nécessaire d'en connaître plus sur l'informatique. Dans ce cas, l'ordinateur n'est qu'un outil, au même titre qu'une automobile pour une personne désirant se déplacer. En revanche, si l'on veut soi-même développer des applications, il en va tout autrement. Mis à part quelques cas particuliers d'applications à l'aspect et au comportement standardisés (par exemple une fenêtre, un menu sous forme de boutons et un ensemble de réactions simples aux événements comme le clic de la souris: applications Windows ou pages Web enrichies d'*applets*), le développement nécessite des connaissances dans des domaines divers. D'abord en programmation: maîtrise d'un langage, de ses outils (librairies, classes d'objets) et de sa traduction (compilation, semi-compilation ou interprétation). Ensuite en système: exécution du processus, allocation de sa mémoire, traitement de ses entrées-sorties, prise en charge des exceptions. L'écriture du code source nécessite également de bonnes compétences dans le domaine traité par l'application et, d'une façon générale, en algorithmique.

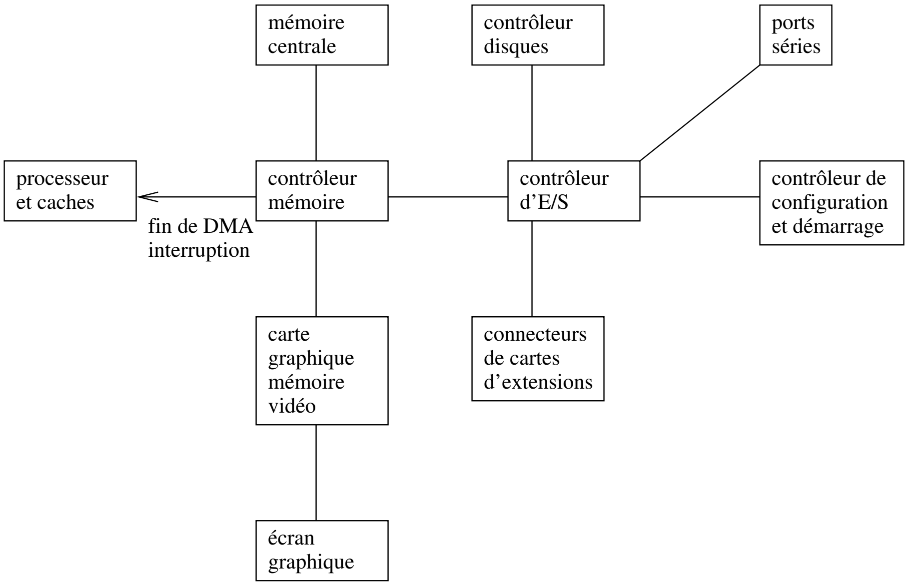

Figure 1.8 : Chargement d'un programme en mémoire: fin du transfert DMA.

L'architecture d'un processeur est le niveau d'abstraction formant la cible des compilateurs (le jeu d'instructions du langage machine, ses types de données - essentiellement entier et flottant, voire booléen - et ses variables - les registres et la mémoire -). La micro-architecture est l'implantation de l'architecture dans un processeur. Par exemple, les PC sont aujourd'hui construits avec des processeurs Pentium IV, Pentium III, Celeron, Duron et Athlon qui sont autant de micro-architectures d'une seule et même architecture, l'IA-32 (que l'on appelait 80x86 par le passé). Les iMacs sont construits avec des Power G3 ou G4, qui sont deux micro-architectures de l'architecture PowerPC.

L'informaticien doit-il avoir des connaissances en architecture? Si le domaine traité par l'application qu'il souhaite développer est le matériel lui-même (cas d'un logiciel de base), la réponse est clairement oui. Il en va de même si l'application est un compilateur. Mais l'architecture des processeurs contient des notions qui sont essentielles à la bonne programmation, quelle que soit l'application visée. Cela commence par la représentation des types de données: le signe des entiers avec la représentation du complément à deux (chapitre 3) et le format flottant IEEE-754 (chapitre 4). La connaissance de ces représentations est indispensable chaque fois que l'on fait emploi de fonctions de conversions, qu'elles soient explicites ou implicites (en C, une déconvenue classique de tout programmeur est de déclarer *unsigned* une variable qui à l'exécution prend une valeur négative; l'anomalie n'est en général détectée que tardivement, après que diverses comparaisons aient été fondées sur une interprétation positive de la valeur négative). Pour les flottants, savoir que les calculs ne sont pas exacts car arrondis est vital pour ne pas accorder une confiance aveugle aux chiffres composant un résultat, voire pour simplement contrôler une boucle de convergence (il se peut que la fonction en calcul exact converge, mais qu'en calcul arrondi on cesse de converger à quelques pas du résultat; on contrôle ce type de boucle en achevant le calcul lorsqu'une précision donnée est atteinte: écart entre la valeur précédente et la nouvelle valeur inférieur en valeur absolue à un terme constant ϵ).

En plus de leurs représentations, il est également utile de connaître les opérations de base applicables aux entiers (chapitre 3 pour la réalisation de l'addition et des décalages en circuits, chapitre 5 pour la multiplication et la division entières) et aux flottants (chapitre 5). Les opérations de base sont celles que l'on sait câbler dans une micro-architecture. Elles sont d'emploi suffisamment fréquent pour figurer dans le jeu d'instructions machine. Par exemple, certaines architectures ne considèrent pas la division entière comme une opération de base. Elle est émulée à partir de la multiplication ou calculée dans l'unité flottante après conversion. Ce peut même être le cas de la multiplication entière, qui est fréquente surtout sous la forme $c * v$ où c est une constante puissance de deux. Dans ce cas, un simple décalage suffit. Associées aux opérateurs, il y a leurs exceptions: le débordement d'intervalle dans les calculs entiers (chapitre 3), ignoré dans un langage comme C, mais traitable par un gestionnaire d'exception en C++; les exceptions de la norme IEEE-754 pour les calculs flottants (chapitre 4) avec les nombres dénormalisés, les NaN et les infinis.

Après les types de données, l'architecture définit les variables. Le modèle universellement adopté est celui d'un ensemble de registres (chapitre 2 pour la description d'une cellule mémoire multi-port, chapitre 3 pour la description d'un banc de registres, chapitre 4 pour la définition des registres dans une architecture). On parle d'architecture registre-registre. Seule une étude détaillée de l'architecture d'un processeur, et même de sa micro-architecture, permet d'expliquer pourquoi on n'emploie pas un modèle organisé autour de la mémoire en accès direct ou en accès par empilement/dépilement. De tels modèles ont existé par le passé mais ont été abandonnés. En particulier, il faut oublier la justification reposant sur la vitesse: un ensemble de n registres de m bits est plus lent d'accès qu'une mémoire de n mots de m bits car le banc de registres occupe une plus grande surface de silicium que la mémoire. Il existe un banc de registres pour chaque type de données (entier, flottant, booléen, ...). Le banc de registres entiers et le banc de registres flottants peuvent éventuellement communiquer. Le jeu d'instructions qui prévoit une instruction de transfert dans un sens ou dans l'autre précise ce point (transfert sans conversion; la conversion est une opération de l'unité flottante). Sans une telle communication, une conversion d'entier vers flottant ou de flottant vers entier nécessite un passage par la mémoire (le compilateur produit les instructions nécessaires quand ce cas apparaît dans un programme, par exemple lors des conversions implicites en C).

Vient ensuite le jeu d'instructions proprement dit. Le modèle CISC (*Complex Instruction Set Computer* qui est le modèle sous-jacent de l'architecture IA-32) est en compétition avec le modèle RISC (*Reduced ISC* qui est le modèle de toutes les autres architectures: PowerPC, DEC Alpha AXP, HP PA, Sun SPARC, MIPS). En réalité, il n'y a plus guère de compétition. En effet, les processeurs d'architecture CISC actuels transforment leurs instructions en micro-instructions RISC. Avec l'architecture IA-64, Intel et HP tentent d'imposer un nouveau type d'architecture et de jeu d'instructions:

le modèle EPIC (*Explicitly Parallel Instruction Computer* qui est présenté à la fin du chapitre 4).

Que l'architecture soit de type CISC, RISC ou EPIC, le jeu d'instructions peut être divisé en cinq grandes catégories (voir le chapitre 4). Il existe des instructions de contrôles principalement composées de sauts conditionnels, des instructions de calculs (que l'on peut encore subdiviser en calculs entiers et flottants) et des instructions d'entrées-sorties, c'est-à-dire d'accès à la mémoire (instructions dites de chargement et de rangement. Dans les systèmes modernes, l'espace des entrées-sorties est mappé en mémoire (chapitre 6)). Par conséquent, lire ou écrire sur un périphérique revient à charger ou ranger une donnée en mémoire (à l'extérieur du processeur, l'adresse émise est reconnue comme identifiant le périphérique visé). Il existe en outre des instructions SIMD (*Single Instruction Multiple Data*; voir le chapitre 4 pour leur sémantique et le chapitre 5 pour leur implantation). Celles-ci permettent le traitement en parallèle des éléments d'un vecteur. Elles s'apparentent aux instructions de calculs (entiers et flottants), en en proposant une extension vectorielle et en ajoutant quelques opérations spécifiques (en particulier, sur les entiers, une arithmétique à saturation en plus de la traditionnelle arithmétique modulo). Enfin, il existe quelques instructions dévolues au système (chapitre 4): accès aux différentes parties de la micro-architecture pour permettre le changement de contexte dans les systèmes préemptifs comme Unix ou Windows-NT (sauvegarde-restauration des registres, mise à jour de la MMU (*Memory Management Unit*), mode noyau et mode utilisateur), mais aussi le traitement des exceptions (défaut de page par exemple, voir le chapitre 6).

La principale différence entre les modèles CISC, RISC et EPIC est la richesse de chacune des cinq catégories et le mode de codage des instructions (chapitre 4). Le modèle CISC repose sur un codage de taille variable. Les instructions les plus fréquentes qui ne nécessitent pas d'autre référence que le banc de registres, sont codées sur deux octets. Cela limite la taille du banc de registres à un maximum de huit registres. Les autres instructions nécessitent des compléments, soit avec un préfixe d'extension, soit avec des données en suffixe. Cela permet une grande souplesse de codage, à tel point que le jeu d'instructions est même extensible (ce qui a permis par exemple d'ajouter les instructions MMX au jeu de base). En revanche, cela induit une grande complexité de décodage. Par exemple, le jeu d'instructions permet de manipuler aussi bien des registres que des variables en mémoire (pour pallier le manque de registres, on dispose d'instructions de calcul ayant une source en mémoire; c'est une autre différence notable avec les architectures RISC et EPIC). Il contient aussi de nombreux modes d'adressage (ce qui s'apparente à du traitement de pointeur en C). Les instructions de calcul comprennent des opérations d'arithmétique décimale et de traitement de chaînes de caractères.

Le modèle RISC implique la codification des instructions en un mot de 32 bits découpé en quelques champs (une instruction est à peu de choses près un quadruplet: deux sources, une destination et une opération). Comme chaque champ est de taille réduite, sauf l'un d'entre eux qui peut servir pour coder les constantes, le modèle RISC limite la taille du banc de registre à 32 registres. Il limite également le nombre d'opérations (de l'ordre d'une centaine, tout type confondu). Les instructions prennent leurs sources et ont leur destination dans le banc de registre, à l'exception des instructions de chargement et de rangement. Les chargements ont une source en mémoire. Les rangements ont une destination en mémoire (le modèle exclut les instructions de calcul ayant une source en mémoire). En général, l'architecture ne prévoit qu'un seul mode d'adressage de type "base" plus "déplacement".

Le modèle EPIC regroupe plusieurs instructions en un paquet (*bundle*). Dans l'architecture IA-64, un paquet regroupe trois instructions de 41 bits chacune accompagnées de cinq bits exprimant leurs dépendances. Cette taille supérieure de codage des instructions permet à la fois de disposer d'un plus grand nombre de registres (128 au lieu de 32) et d'un plus grand nombre d'opérations (à mi-chemin entre CISC et RISC). Le jeu d'instructions IA-64 introduit en particulier de nouveaux concepts, tous orientés vers le parallélisme d'instructions: instructions conditionnelles gardées par un prédicat pour éviter les sauts de faible amplitude, gestion en pile des registres pour faciliter le déroulage des boucles en pipeline logiciel, chargements anticipés et conditionnels pour absorber la latence des accès mémoire. Le découpage en champs réguliers inspiré du modèle RISC simplifie le décodage.

La micro-architecture enfin se compose d'un certain nombre de détails d'implantation qui sont essentiels à une bonne organisation des calculs. Parmi ceux-ci figure la hiérarchisation de la mémoire qui repose sur le principe de localité (chapitres 3 et 6; il est important de connaître les caractéristiques des caches composant la hiérarchie pour fixer leur taux d'échec et leur temps d'accès). En font également partie, l'exécution en parallèle et en recouvrement des instructions (organisation superscalaire et pipelinée du processeur, voir le chapitre 5), la prédiction des sauts avec la pénalité et le taux de mauvaises prédictions, le renommage des registres pour augmenter le parallélisme d'instructions et l'exécution en désordre des instructions, selon un modèle à flot de donnée (chapitre 5; une instruction démarre son exécution quand ses données sont disponibles et non pas lorsque l'instruction précédente a démarré). S'y ajoutent, la latence des opérations (opération courte, opération longue, opération de durée variable comme les chargements dont la latence dépend de la hiérarchie mémoire; les longues latences peuvent être absorbées par l'exécution en désordre) et enfin le fonctionnement des opérations vectorielles.

Si tous les processeurs actuels sont construits à partir d'une micro-architecture de modèle unique (exécution superscalaire, spéculative et en désordre, voir chapitre 5), cette unanimité risque d'être remise en cause par la micro-architecture de l'Itanium, le nouveau processeur Intel-HP fondé sur l'architecture IA-64. Comme cette architecture fait apparaître le parallélisme d'instructions au sein des paquets, il n'est pas nécessaire que la micro-architecture augmente le parallélisme d'instructions à l'exécution. La micro-architecture de l'Itanium exécute les instructions en ordre, à charge pour le compilateur de préparer un ordonnancement des instructions au sein des paquets qui recouvre au maximum les latences.

4 La structure du livre

Ce livre compte six chapitres (y compris cette introduction), qui décrivent un processeur et son environnement de façon ascendante. La figure 1.9 p.14 montre comment s'enchaînent les chapitres. On commence (deuxième chapitre) par les transistors servant à former les cellules de base: portes, séparateurs et cellules mémorisantes.

Ensuite (troisième chapitre), on présente un ensemble de circuits: circuits logiques, arithmétiques, de séparation, de mémorisation. Ces circuits constituent un ensemble suffisant pour construire n'importe quelle micro-architecture actuelle. Ce troisième chapitre représente un premier palier. Il se termine par quelques exemples d'assemblage de circuits permettant notamment de former un cache, mais il présente en outre un exemple d'architecture et de micro-architecture élémentaires, le processeur LUC3. Ce premier palier s'achève par la description d'un projet de synthèse consacré à l'implantation d'un simulateur de circuits devant servir à la réalisation d'un simulateur du processeur donné en exemple.

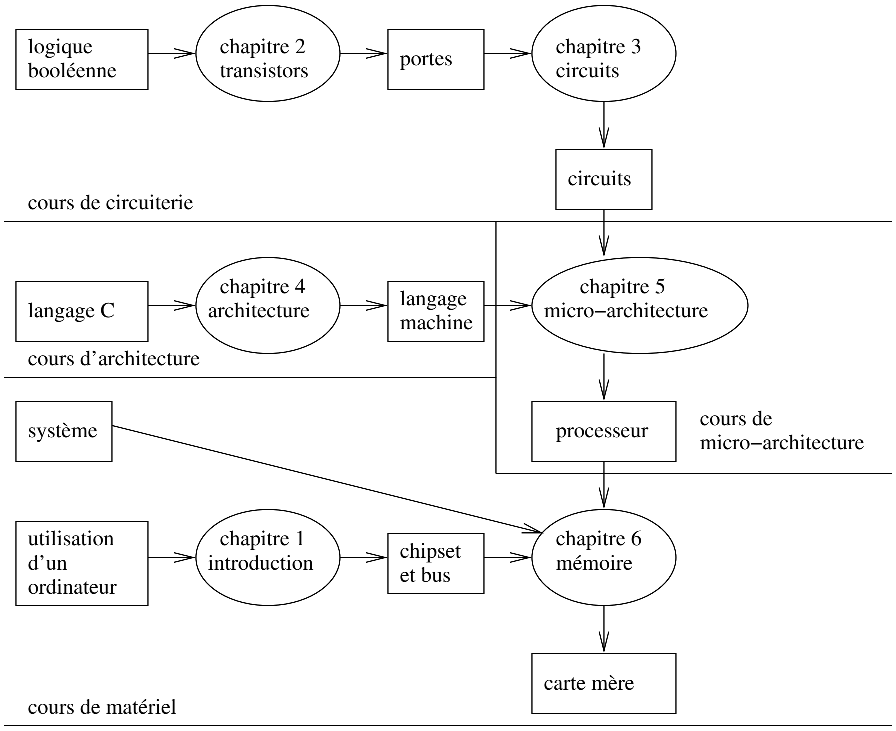

Figure 1.9 : Enchaînement logique des chapitres.

Au quatrième chapitre, on définit une architecture de processeur réel, à mi-chemin entre RISC et EPIC. Elle inclut un jeu d'instructions flottantes respectant la norme IEEE-754. On définit également un jeu d'instructions vectorielles entières et flottantes (équivalent des jeux MMX, SSE1 et SSE2 de l'architecture IA-32/64). Le cinquième chapitre est consacré à la micro-architecture à proprement parler. On y présente la prédiction de saut, le renommage des registres, l'exécution en désordre, le réordonnancement des accès mémoire, le matériel des opérateurs arithmétiques complexes (multiplieur, diviseur, opérateurs flottants, opérateurs vectoriels), l'organisation superscalaire (parallélisme de simultanéité) et enfin l'organisation pipelinée (parallélisme de recouvrement). Ce chapitre s'achève par un sujet de TER (Technique d'Étude et de Recherche; il s'agit des sujets d'initiation à la recherche que les cursus de maîtrise doivent proposer aux étudiants). Le sixième et dernier chapitre traite de l'environnement du processeur: sa hiérarchie mémoire et le fonctionnement des caches, son unité de gestion mémoire (MMU) et la mémoire virtuelle ainsi que les différents types de bus et les entrées-sorties.

Il n'est pas indispensable de lire ce livre linéairement. Le deuxième chapitre ne nécessite pas d'autre connaissance qu'un peu de logique booléenne (tables de vérité des fonctions booléennes de deux variables). Le troisième chapitre présuppose l'existence des portes répertoriées à la fin du deuxième chapitre. Il présente le fonctionnement général d'un processeur simple du point de vue matériel et de se familiariser avec un chemin de données et un chemin d'instructions. Le quatrième chapitre suppose la connaissance d'un langage de programmation comme C. Il est préférable de l'aborder après avoir franchi le premier palier que constituent les trois premiers chapitres. Le cinquième chapitre présuppose la lecture des chapitres 2 et 3. On peut néanmoins emprunter un raccourci en se faisant une idée la plus précise possible de l'ensemble des circuits récapitulés en fin de troisième chapitre. La connaissance de la totalité du quatrième chapitre n'est pas indispensable, mais la première section (généralités sur le langage machine) et la partie de la seconde section concernant les instructions entières sont nécessaires. Pour le sixième chapitre, il faut posséder des connaissances préalables en système (espace mémoire d'un processus, système d'entrées-sorties), mais la lecture des chapitres 4 et 5 n'est pas indispensable.

Pour résumer, on peut bâtir à partir de ce livre trois cours orientés vers le matériel. Tout d'abord, un cours de circuiterie, permettant de passer des transistors à un petit processeur (chapitres 2 et 3, ce qui représente environ 30 heures de cours et autant d'exercices en travaux dirigés et constitue une entrée en matière à la portée d'un étudiant de DEUG). Ensuite, un cours d'architecture (chapitre 4, qui représente autant de temps mais s'adresse cette fois à des étudiants de licence en raison qu'il présuppose des connaissances en programmation). Enfin, un cours de micro-architecture (chapitre 5). Ce cours de spécialiste, qui occupe la même durée que les précédents, peut s'insérer dans un cursus de maîtrise, de DESS ou de DEA. Par ailleurs, cette présente introduction suivie de la dernière section du chapitre 6 peuvent servir de support à quelques heures de cours et de travaux dirigés formant une initiation à l'informatique, en ce qui concerne le matériel.

Comme dans ce chapitre, environ 300 exercices sont disséminés dans les différentes sections du livre, et pas seulement en fin de chapitre. Le plus souvent, ce sont des applications immédiates. Il va sans dire qu'il est important de prendre le temps de les faire (mais cela va mieux en le disant).

Par ailleurs, chaque chapitre se termine par quelques références bibliographiques, certaines accessibles sur le Web. Voilà du surf utile. On consultera en particulier le site suivant qui fournit une liste (dernière mise à jour en août 2000) des adresses URL de tous les articles de recherches et documentations de constructeurs qu'il faut lire en matière d'architecture.

`http://www.mkp.com/architecture-readings/wc/`

On notera également l'adresse URL du site central des architectes:

`http://www.cs.wisc.edu/~arch/www/`

Concernant plus particulièrement les différents éléments présentés dans cette introduction, on consultera avec profit les sites Web des constructeurs ou organismes suivants (le site “www.PCGuide.com” contient une description exhaustive de tout ce que renferme un PC):

- Carte mère:
  `www.PCGuide.com/ref/mbsys/mobo`
  `www.asus.com`
- Processeur Pentium, Celeron, Athlon, Duron, HP8500 et Itanium; architecture IA-32, IA-64; architecture HP-PA:
  `developer.intel.com`
  `www.hp.com`
  `www.amd.com`
- Mémoires SDRAM, DDR-SDRAM et RDRAM:
  `www.elpida-memory.com`
  `www.rambus.com`
- Carte et co-processeur graphique TnT et GeForce:
  `www.nvidia.com`
- Bus AGP, PCI, IDE/ATA, USB, SCSI:
  `developer.intel.com/technology/agp (norme du bus AGP)`
  `www.pcisig.com (norme du bus PCI)`
  `www.usb.org (norme du bus USB)`
  `www.PCGuide.com/ref/hdd/if/scsi (bus SCSI)`
  `www.PCGuide.com/ref/hdd/if/ide (bus IDE ATA)`
- Périphériques: consulter le site “www.PCGuide.com”.

2

Des transistors aux portes

Dans ce chapitre, nous allons définir les opérateurs élémentaires qui interviennent dans la construction des composants électroniques digitaux. Il s'agit des *portes logiques*, des *séparateurs* et des *cellules mémoires*. Pour cela, nous nous appuierons sur la technologie CMOS aujourd'hui dominante.

Dans la suite du texte, les opérateurs logiques *et* et *ou* seront désignés par les symboles $\wedge$ et $\vee$. Le surlignement indiquera la complémentation. L'opérateur *ou exclusif* sera symbolisé par $\oplus$.

Sur les figures, les noms des opérateurs (*et*, *ou*) seront employés éventuellement sous une forme abrégée (*xou* pour le *ou exclusif*). La complémentation sera indiquée par le caractère '#' ajouté en préfixe de l'expression à complémenter (par exemple, *#et* désignera ce que l'on a coutume d'appeler le *nand*).

Dans la suite du texte, la valeur 0 représentera le terme “faux” et la valeur 1 représentera le terme “vrai”.

1 Les transistors

1.1 Le transistor nMOS

Un transistor nMOS (voir figure 2.1 p.18) peut être considéré comme un transmetteur qui, selon la commande, est passant ou non. Lorsqu'il est passant, la sortie a la même valeur logique que l'entrée. Lorsqu'il ne l'est pas, les deux extrémités ont des valeurs logiques indépendantes l'une de l'autre.

Afin de faire abstraction des détails électriques quand ils ne sont pas nécessaires, on peut représenter le transistor nMOS par une fonction de *multiplexage* (que l'on peut encore appeler *si alors sinon*):

$$\textit{sortie} = \text{si } (\textit{commande}) \text{ alors } \textit{entrée} \text{ sinon Z}$$

La variable *commande* prend sa valeur dans {0,1} (0 pour faux et 1 pour vrai). Les variables *sortie* et *entrée* prennent leur valeur dans {0, Z, 1}. La valeur Z sera électriquement et logiquement définie un peu plus loin afin de présenter l'effet de la connexion en série ou en parallèle des transistors. Pour le moment, contentons-nous de dire que ce n'est ni 0, ni 1.

Par définition, le transistor nMOS est passant lorsque sa commande vaut 1 (vrai). On peut s'en souvenir en associant le *n* de nMOS à *négatif*, le signe '-' étant lui-même usuellement représenté par 1.

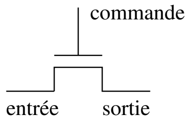

Figure 2.1 : Transistor nMOS.

Deux transistors nMOS placés en série (la sortie de l'un est l'entrée de l'autre; voir figure 2.2) ne passent une donnée d que si les deux commandes sont à 1:

$$\begin{aligned} s &= \text{si } (p') \text{ alors } (\text{si } (p) \text{ alors } d \text{ sinon Z}) \text{ sinon Z} \\ &= \text{si } (p \wedge p') \text{ alors } d \text{ sinon Z} \end{aligned}$$

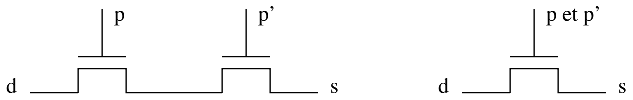

Figure 2.2 : Transistors nMOS en série.

S'ils sont placés en parallèle (les entrées et les sorties sont communes; voir figure 2.3 p.19), ils sont passants si l'une des deux commandes est à 1.

$$\begin{aligned} s &= (\text{si } (p) \text{ alors } d \text{ sinon Z}) \parallel (\text{si } (p') \text{ alors } d \text{ sinon Z}) \\ &= \text{si } (p \vee p') \text{ alors } d \text{ sinon Z} \end{aligned}$$

Nous voyons que pour donner un sens logique à la connexion de transitors en parallèle, il faut préciser la valeur de l'opérateur $\parallel$ quand il est appliqué à Z. L'opérateur $\parallel$ utilisé pour le calcul de la valeur d'une sortie commune à plusieurs transistors est ainsi défini:

$$\begin{aligned} &\forall x \in \{0,\text{Z},1\} \quad x \parallel \text{Z} = \text{Z} \parallel x = x, \\ &\forall x \in \{0,\text{Z},1\} \quad x \parallel x = x \end{aligned}$$

On peut remarquer que $0 \parallel 1$ et $1 \parallel 0$ ne sont pas définis. Si deux transistors reliés par leur sortie sont simultanément passants et qu'ils passent l'un 1 et l'autre 0, la sortie n'est pas définie. Il faut attendre que l'un des deux transistors cesse d'être passant pour que la valeur passée par l'autre s'impose.

D'un point de vue électrique, Z est l'état de *haute impédance*, que l'on représente graphiquement par une valeur à mi-chemin entre 0 et 1. Souvent, ce troisième état logique, ni 0 ni 1, n'a pas besoin d'être précisé car un autre dispositif connecté en parallèle impose un 0 ou un 1 (0‖Z= 0 et 1‖Z= 1). Par exemple (voir la figure 2.4), on peut partager une sortie s en y connectant n dispositifs fournissant n entrées e_0 à e_{n-1}. L'accès de chaque dispositif à la sortie commune est filtré par un transistor commandé par un signal c_i. Supposons qu'un transistor au plus soit passant ($c_i = 1$ et $c_j = 0$, $\forall j \neq i$), la sortie commune s reçoit Z en sortie des $n-1$ dispositifs non passants ($c_j = 0$, $j \neq i$) et soit 0, soit 1 du seul dispositif passant (e_i). C'est lui, par son entrée e_i, qui établit la valeur logique de la sortie commune ($s = e_i$).

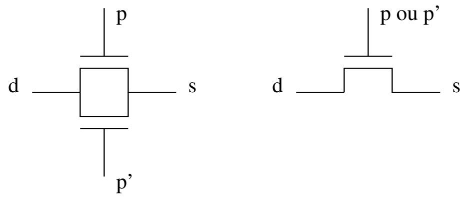

Figure 2.3 : Transistors nMOS en parallèle.

Nous verrons cependant des cas dans ce chapitre où il est indispensable d'associer une valeur logique (0 ou 1) à l'état de haute impédance (voir la figure 2.5 p.20; la sortie du premier transistor commande le second; rappelons que par définition, la commande d'un transistor prend sa valeur dans {0,1}). Pour cela, il faut considérer Z comme la mémoire de ce que passait le transistor avant de devenir non passant. Une fois le transistor non passant, il reste une charge en sortie, qui représente un 0 ou un 1 porté par une très faible intensité (un courant de fuite, très faible, passe par le transistor). Cette charge peut suffire à commander un autre transistor pendant un temps limité (la charge disparaît progressivement à cause du courant de fuite): si sa valeur logique est 1, le transistor nMOS qu'elle commande reste passant. Si c'est un 0, le transistor qu'elle commande reste non passant. Dans l'exemple de la figure 2.5, selon a, la charge en x après que le transistor commandé par b ait été rendu non passant ($b = 0$) est soit 0 ($a = 0$), soit 1 ($a = 1$), Dans le second cas, le second transistor continue de passer (c et d sont reliés).

D'autre part, Z‖Z=Z. Si deux transistors reliés par leur sortie sont non passants, la charge accumulée en sortie (Z) provient de l'entrée du dernier transistor à être devenu non passant.

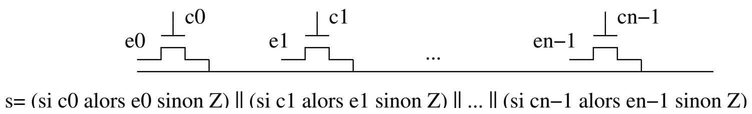

Figure 2.4 : Partage de l'accès à une sortie commune.

<u>Exercice 2.1</u>: démontrer les égalités:

$$\text{(si } (p') \text{ alors (si } (p) \text{ alors } d \text{ sinon Z) sinon Z)} \quad = \quad \text{(si } (p \wedge p') \text{ alors } d \text{ sinon Z)}$$

$$(\text{si } (p) \text{ alors } d \text{ sinon Z}) \parallel (\text{si } (p') \text{ alors } d \text{ sinon Z}) \quad = \quad (\text{si } (p \vee p') \text{ alors } d \text{ sinon Z})$$

Exercice 2.2: établir l'équation associée au montage de la figure 2.5 pour une sortie en d. Que vaut x quand $b = 0$ (attention au domaine de définition de x qui commande un transistor)?

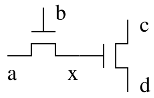

Figure 2.5 : Deux transistors nMOS.

1.2 Le transistor pMOS

Un transistor pMOS (voir figure 2.6) peut être représenté par la fonction de multiplexage suivante:

$$\textit{sortie} = \text{si } (\textit{commande}) \text{ alors Z sinon } \textit{entrée}$$

La variable *commande* prend sa valeur dans {0,1}. Les variables *sortie* et *entrée* prennent leur valeur dans {0, Z, 1}.

Par définition, le transistor pMOS est passant lorsque sa commande vaut 0 (faux). On peut s'en souvenir en associant le p de pMOS à *positif*, le signe '+' étant lui-même usuellement représenté par 0.

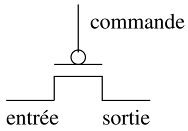

Figure 2.6 : Transistor pMOS.

Deux transistors pMOS en série s'expriment ainsi (voir figure 2.7):

$$\begin{aligned} s &= \text{si } (p') \text{ alors Z sinon (si } (p) \text{ alors Z sinon } d) \\ &= \text{si } (p \vee p') \text{ alors Z sinon } d \end{aligned}$$

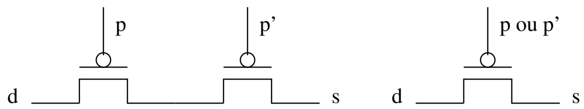

Figure 2.7 : Transistors pMOS en série.

En parallèle (voir figure 2.8 p.21):

$$\begin{aligned} s &= (\text{si } (p) \text{ alors Z sinon } d) \parallel (\text{si } (p') \text{ alors Z sinon } d) \\ &= \text{si } (p \wedge p') \text{ alors Z sinon } d \end{aligned}$$

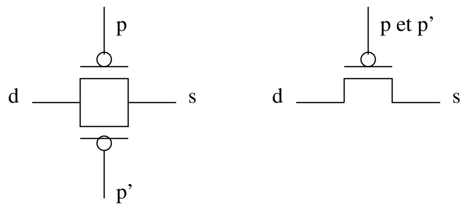

Figure 2.8 : Transistors pMOS en parallèle.

Exercice 2.3: démontrer les égalités suivantes:

$$\begin{aligned}(\text{si } (p') \text{ alors Z sinon } (\text{si } (p) \text{ alors Z sinon } d)) &= (\text{si } (p \vee p') \text{ alors Z sinon } d)\\(\text{si } (p) \text{ alors Z sinon } d) \parallel (\text{si } (p') \text{ alors Z sinon } d) &= (\text{si } (p \wedge p') \text{ alors Z sinon } d)\end{aligned}$$

Exercice 2.4: établir l'équation associée au montage de la figure 2.5 p.20 en remplaçant les transistors nMOS par des transistors pMOS. Que vaut x quand $b = 1$?

1.3 La porte de transfert CMOS

Pour des raisons d'ordre électrique que nous ne préciserons pas, le transistor nMOS passe correctement un 0 (c'est-à-dire rapidement et sans l'affaiblir), mais médiocrement un 1. Inversement, un transistor pMOS passe efficacement un 1 mais pas un 0. Pour passer bien un 0 comme un 1, on utilise donc deux transistors dont les sorties sont reliées, l'un de type nMOS pour passer le 0 et l'autre de type pMOS pour passer le 1. Ce type de montage est une porte de transfert CMOS (voir figure 2.9):

$$s = (\text{si } (p) \text{ alors } d \text{ sinon Z}) \parallel (\text{si } (\overline{p}) \text{ alors Z sinon } d)$$

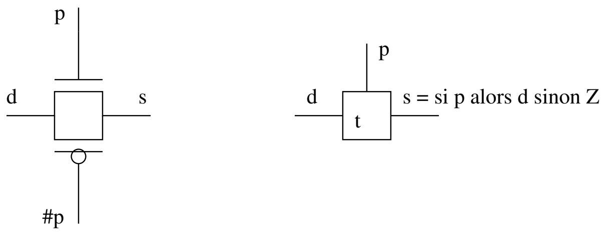

Figure 2.9 : Porte de transfert CMOS.

Quand $p = 1$, la porte de transfert CMOS est passante. Si $d = 1$, le transistor pMOS passe un "bon" 1. Si $d = 0$, le transistor nMOS passe un "bon" 0. Dans les deux cas, la sortie s correspond à l'entrée d non dégradée. À droite de la figure proprement dite se trouve la représentation symbolique de la porte de transfert (dénommée t). Attention, le symbole ne permet plus de voir que le signal p doit également être fourni sous forme complémentée.

On exprime comme suit deux portes de transfert CMOS en série (voir figure 2.10):

$$\begin{aligned} s &= (\text{si } (p') \text{ alors } (\text{si } (p) \text{ alors } (d) \text{ sinon Z}) \text{ sinon Z})\| \\ &\quad (\text{si } (\overline{p'}) \text{ alors Z sinon } (\text{si } (\overline{p}) \text{ alors Z sinon } (d))) \\ &= (\text{si } (p \wedge p') \text{ alors } (d) \text{ sinon Z})\| \\ &\quad (\text{si } (\overline{p} \vee \overline{p'}) \text{ alors Z sinon } (d)) \\ &= (\text{si } (p \wedge p') \text{ alors } (d) \text{ sinon Z})\| \\ &\quad (\text{si } (\overline{p \wedge p'}) \text{ alors Z sinon } (d)) \end{aligned}$$

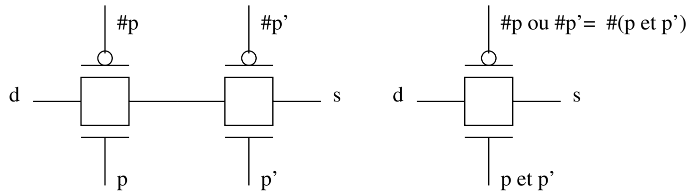

Figure 2.10 : Portes de transfert CMOS en série.

Deux portes en parallèle s'expriment ainsi (voir figure 2.11):

$$\begin{aligned} s &= (\text{si } (p) \text{ alors } (d) \text{ sinon Z})\| \\ &\quad (\text{si } (\overline{p}) \text{ alors Z sinon } (d))\| \\ &\quad (\text{si } (p') \text{ alors } (d) \text{ sinon Z})\| \\ &\quad (\text{si } (\overline{p'}) \text{ alors Z sinon } (d)) \\ &= (\text{si } (p \vee p') \text{ alors } (d) \text{ sinon Z})\| \\ &\quad (\text{si } (\overline{p} \wedge \overline{p'}) \text{ alors Z sinon } (d)) \\ &= (\text{si } (p \vee p') \text{ alors } (d) \text{ sinon Z})\| \\ &\quad (\text{si } (\overline{p \vee p'}) \text{ alors Z sinon } (d)) \end{aligned}$$

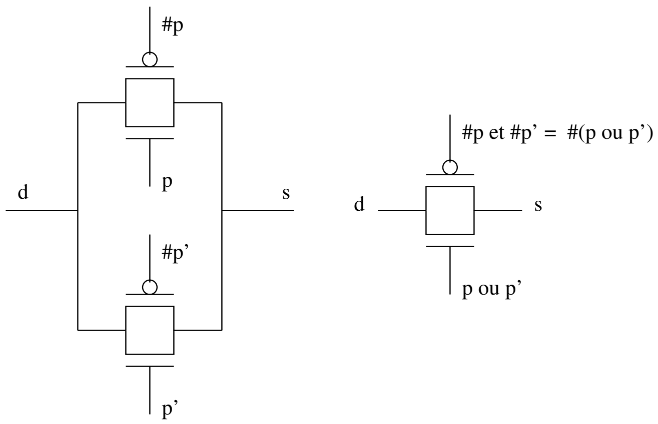

Figure 2.11 : Portes de transfert CMOS en parallèle.

Exercice 2.5: démontrer l'égalité suivante:

$$\begin{array}{rl}(\text{si } (p') \text{ alors } (\text{si } (p) \text{ alors } (d) \text{ sinon Z}) \text{ sinon Z})\| & \\ (\text{si } (\overline{p'}) \text{ alors Z sinon } (\text{si } (\overline{p}) \text{ alors Z sinon } (d))) & = \\ (\text{si } (p \wedge p') \text{ alors } (d) \text{ sinon Z})\|(\text{si } (\overline{p \wedge p'}) \text{ alors Z sinon } (d)) & \end{array}$$

Exercice 2.6: démontrer l'égalité suivante:

$$\begin{array}{rl}(\text{si } (p) \text{ alors } (d) \text{ sinon Z})\|(\text{si } (\overline{p}) \text{ alors Z sinon } (d))\| & \\ (\text{si } (p') \text{ alors } (d) \text{ sinon Z})\|(\text{si } (\overline{p'}) \text{ alors Z sinon } (d)) & = \\ (\text{si } (p \vee p') \text{ alors } (d) \text{ sinon Z})\|(\text{si } (\overline{p \vee p'}) \text{ alors Z sinon } (d)) & \end{array}$$

2 Les portes logiques

2.1 La logique complémentaire ou le passage de constantes

2.1.1 L'inverseur CMOS

En reliant l'entrée du transistor nMOS à 0 et celle du transistor pMOS à 1, puis les deux commandes à une même source p, on obtient un inverseur CMOS (voir figure 2.12; à droite se trouve la représentation symbolique de l'inverseur, dénommé #):

$$s = (\text{si } (p) \text{ alors 0 sinon Z}) \ \| \ (\text{si } (p) \text{ alors Z sinon 1})$$

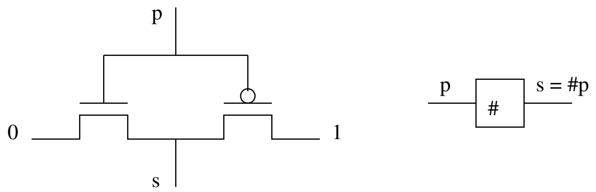

Figure 2.12 : Inverseur CMOS.

Lorsque la commande commune p vaut 1, elle est passante pour le transistor nMOS et non passante pour le transistor pMOS. Le premier laisse passer son entrée (0). La sortie vaut 0. Lorsque la commande commune vaut 0, c'est le transistor pMOS qui laisse passer son entrée (1) et la sortie vaut 1. Ainsi, $\forall p \in \{0,1\}$, on a $s \in \{0,1\}$ ($s \neq$Z).

On peut noter que l'inverseur CMOS est monté de telle manière que le transistor nMOS ne passe jamais de 1 (son entrée est reliée à 0) et que le transistor pMOS ne passe jamais de 0 (son entrée est reliée à 1). On peut également remarquer que la structure de la porte montée en inverseur, avec la partie nMOS reliée à 0 et la partie pMOS reliée à 1, est l'implantation exacte de la table de vérité de la fonction booléenne de complémentation.

Ce principe peut être appliqué à la construction d'une porte CMOS représentant une fonction booléenne quelconque. Un réseau de transistors nMOS est chargé de fournir les 0 de la fonction tandis qu'un réseau de transistors pMOS est chargé de fournir les 1.

Dans chaque construction, il doit exister au moins un chemin de transistors nMOS passants, de 0 à la sortie, pour chaque combinaison d'entrées donnant à la fonction la valeur 0. Pour ces combinaisons d'entrées, il ne doit exister aucun chemin de transistors pMOS passants, reliant 1 à la sortie.

Inversement, il doit exister au moins un chemin de transistors pMOS passants, de 1 à la sortie, pour chaque combinaison d'entrées donnant à la fonction la valeur 1 et aucun chemin de transistors nMOS passants, reliant 0 à la sortie.

D'un point de vue électrique, ce type de porte présente la particularité de posséder une sortie provenant de l'alimentation ou reliée à la masse. On utilise les commandes pour ouvrir ou fermer les chemins menant de l'alimentation à la sortie ou de la sortie à la masse. Nous appellerons ces montages des *transmetteurs de constantes*.

2.1.2 La construction d'un transmetteur de constantes

D'une manière générale, une fonction booléenne f se réalise à partir de:

$$f = (\text{si } (\overline{f}) \text{ alors } 0 \text{ sinon Z}) \parallel (\text{si } (\overline{f}) \text{ alors Z sinon } 1)$$

En exprimant f sous forme normale disjonctive ($f = mt_1 \vee ... \vee mt_n$, les mt_i étant des *min-termes*, c'est à dire de la forme $mt_i = e_x \wedge ... \wedge e_y$), l'équation précédente devient:

$$\begin{aligned} f &= (\text{si } (\overline{mt_1 \vee ... \vee mt_n}) \text{ alors } 0 \text{ sinon Z}) \parallel \\ &\quad (\text{si } (\overline{mt_1 \vee ... \vee mt_n}) \text{ alors Z sinon } 1) \\ &= (\text{si } (\overline{mt_1} \wedge ... \wedge \overline{mt_n}) \text{ alors } 0 \text{ sinon Z}) \parallel \\ &\quad (\text{si } (\overline{mt_1} \wedge ... \wedge \overline{mt_n}) \text{ alors Z sinon } 1) \\ &= (\text{si } (\overline{mt_1}) \text{ alors (si } (\overline{mt_2}) \text{ alors (...) sinon Z) sinon Z}) \parallel \\ &\quad (\text{si } (\overline{mt_1}) \text{ alors Z sinon } 1 \parallel ... \parallel \text{ si } (\overline{mt_n}) \text{ alors Z sinon } 1) \end{aligned}$$

On obtient un réseau nMOS et un réseau pMOS commandés par les min-termes mt_i complémentés. On peut obtenir ces réseaux directement en partant de la table de vérité de f. On construit les chemins nMOS produisant les 0 de la fonction, puis les chemins pMOS produisant les 1.

Soit par exemple la fonction suivante (voir figure 2.13 p.25):

$$s = (\overline{a}b) \vee (a\overline{b}) \text{ (l'opération } \oplus)$$

De la table de vérité, on tire le réseau nMOS (0 de la fonction):

$$\begin{aligned} nMOS(s) &= (\text{si } (\overline{a} \wedge \overline{b}) \text{ alors } 0 \text{ sinon Z}) \parallel \\ &\quad (\text{si } (a \wedge b) \text{ alors } 0 \text{ sinon Z}) \end{aligned}$$

De la table de vérité, on tire également le réseau pMOS (1 de la fonction):

$$\begin{aligned} pMOS(s) &= (\text{si } (a \vee \overline{b}) \text{ alors Z sinon } 1) \parallel \\ &\quad\ (\text{si } (\overline{a} \vee b) \text{ alors Z sinon } 1) \end{aligned}$$

En appliquant les règles de transformation des paragraphes 1.1 et 1.2 (transistors en série et en parallèle), on obtient pour le réseau nMOS:

$$\begin{aligned} nMOS(s) &= (\text{si } (\overline{a}) \text{ alors } (\text{si } (\overline{b}) \text{ alors 0 sinon Z}) \text{ sinon Z}) \parallel \\ &\quad\ (\text{si } (a) \text{ alors } (\text{si } (b) \text{ alors 0 sinon Z}) \text{ sinon Z}) \end{aligned}$$

On obtient pour le réseau pMOS:

$$\begin{aligned} pMOS(s) &= (\text{si } (a) \text{ alors Z sinon } (\text{si } (\overline{b}) \text{ alors Z sinon } 1)) \parallel \\ &\quad\ (\text{si } (\overline{a}) \text{ alors Z sinon } (\text{si } (b) \text{ alors Z sinon } 1)) \end{aligned}$$

Le montage CMOS s'exprime ainsi:

$$\begin{aligned} s &= (\text{si } (a) \text{ alors } (\text{si } (b) \text{ alors 0 sinon Z}) \text{ sinon Z}) \parallel \\ &\quad\ (\text{si } (\overline{a}) \text{ alors } (\text{si } (\overline{b}) \text{ alors 0 sinon Z}) \text{ sinon Z}) \parallel \\ &\quad\ (\text{si } (a) \text{ alors Z sinon } (\text{si } (\overline{b}) \text{ alors Z sinon } 1)) \parallel \\ &\quad\ (\text{si } (\overline{a}) \text{ alors Z sinon } (\text{si } (b) \text{ alors Z sinon } 1)) \end{aligned}$$

Le montage apparaît en figure 2.13. On aperçoit dans le bas du schéma le symbole de la porte "ou exclusif" dénommé *xou*.

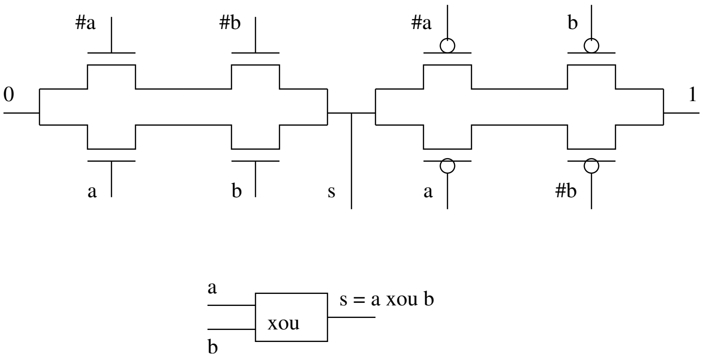

Figure 2.13 : $s = a \oplus b$.

Voyons comment construire des portes $\wedge$ et $\vee$ en logique complémentaire.

Voici $a \wedge b$:

$$\begin{aligned} a \wedge b &= (\text{si } (\overline{a} \wedge \overline{b}) \text{ alors 0 sinon Z}) \parallel \\ &\quad\ (\text{si } (\overline{a} \wedge b) \text{ alors 0 sinon Z}) \parallel \\ &\quad\ (\text{si } (a \wedge \overline{b}) \text{ alors 0 sinon Z}) \parallel \\ &\quad\ (\text{si } (\overline{a} \vee \overline{b}) \text{ alors Z sinon } 1) \end{aligned}$$

En appliquant au réseau nMOS la règle de simplification suivante:

$$\begin{aligned} & (\text{si } (a \wedge b) \text{ alors } 0 \text{ sinon } Z)\| \\ & (\text{si } (a \wedge \overline{b}) \text{ alors } 0 \text{ sinon } Z) \\ = \; & (\text{si } (a) \text{ alors } 0 \text{ sinon } Z) \end{aligned}$$

on simplifie le montage $a \wedge b$:

$$\begin{aligned} a \wedge b \; = \; & (\text{si } (\overline{a}) \text{ alors } 0 \text{ sinon } Z)\| \\ & (\text{si } (\overline{b}) \text{ alors } 0 \text{ sinon } Z)\| \\ & (\text{si } (\overline{a}) \text{ alors } Z \text{ sinon } (\text{si } (\overline{b}) \text{ alors } Z \text{ sinon } 1)) \end{aligned}$$

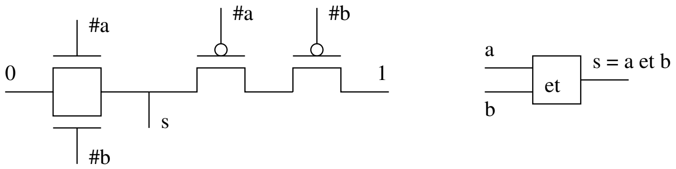

Figure 2.14 : $s = a \wedge b$.

Voici $a \vee b$:

$$\begin{aligned} a \vee b \; = \; & (\text{si } (\overline{a} \wedge \overline{b}) \text{ alors } 0 \text{ sinon } Z)\| \\ & (\text{si } (a \vee \overline{b}) \text{ alors } Z \text{ sinon } 1)\| \\ & (\text{si } (\overline{a} \vee b) \text{ alors } Z \text{ sinon } 1)\| \\ & (\text{si } (\overline{a} \vee \overline{b}) \text{ alors } Z \text{ sinon } 1) \end{aligned}$$

En appliquant au réseau pMOS la simplification suivante:

$$\begin{aligned} & (\text{si } (a \vee b) \text{ alors } Z \text{ sinon } 1)\| \\ & (\text{si } (a \vee \overline{b}) \text{ alors } Z \text{ sinon } 1) \\ = \; & (\text{si } (a) \text{ alors } Z \text{ sinon } 1) \end{aligned}$$

on simplifie le montage $a \vee b$:

$$\begin{aligned} a \vee b \; = \; & (\text{si } (\overline{a}) \text{ alors } (\text{si } (\overline{b}) \text{ alors } 0 \text{ sinon } Z) \text{ sinon } Z)\| \\ & (\text{si } (\overline{b}) \text{ alors } Z \text{ sinon } 1)\| \\ & (\text{si } (\overline{a}) \text{ alors } Z \text{ sinon } 1) \end{aligned}$$

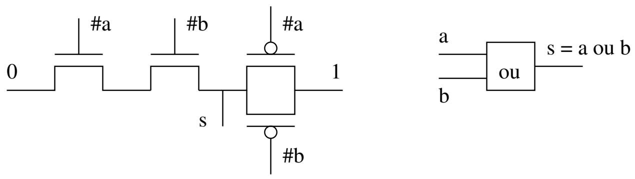

Figure 2.15 : $s = a \vee b$.

De ces portes, on déduit $\overline{\wedge}$ et $\overline{\vee}$ en appliquant les égalités de *de Morgan*: $\overline{ab} = \overline{a} \vee \overline{b}$ et $\overline{a \vee b} = \overline{a}\,\overline{b}$.

On peut remarquer que les portes $\wedge$ et $\vee$ utilisent les compléments des variables mais pas les portes $\overline{\wedge}$ et $\overline{\vee}$. Ainsi, si ces compléments ne sont pas fournis, le $\wedge$ et le $\vee$ peuvent être construits soit en calculant explicitement ces compléments, soit en inversant la sortie d'un $\overline{\wedge}$ pour le $\wedge$ et d'un $\overline{\vee}$ pour le $\vee$.

Exercice 2.7: avec les équations de définition des portes $\wedge$, $\vee$ et $\oplus$, calculer les tables de vérités de ces trois opérateurs.

Exercice 2.8: même question à partir des figures 2.13 (p.25), 2.14 (p.26) et 2.15 (p.26).

Exercice 2.9: construire la porte $a \Rightarrow b$.

Exercice 2.10: démontrer que le schéma de la figure 2.16 est l'opération $a \oplus b$.

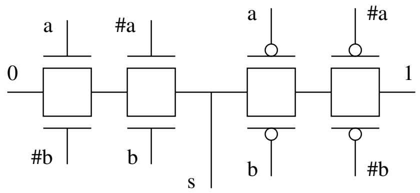

Figure 2.16 : $s = a \oplus b$.

Exercice 2.11: construire la porte $a \oplus b \oplus c$.

2.2 Les portes de transmission ou la transmission de variables

2.2.1 La transmission de variables

On peut simplifier les montages en faisant passer par les transistors des variables booléennes plutôt que les constantes 0 et 1. Afin d'obtenir des portes rapides, il est utile que tous les 0 de la fonction soient issus du réseau nMOS et tous les 1 du réseau pMOS. Cela laisse la possibilité de passer un 0 simultanément par les deux réseaux, le "bon" 0 provenant du réseau nMOS; ou encore de passer un 1 simultanément par les deux réseaux, le "bon" 1 provenant du réseau pMOS. D'une façon générale, une fonction f et son complément $\overline{f}$ peuvent commander deux transistors, l'un nMOS (commande f) et l'autre pMOS (commande $\overline{f}$), pour laisser passer une variable (le transistor nMOS fait passer le 0 et le transistor pMOS le 1). Le passage n'est ouvert que si f est à 1 (les deux transistors sont simultanément passants). Réduit à l'extrême, un tel montage est une simple porte de transfert CMOS, représentant un verrou. La commande n'est plus une fonction, mais un simple signal et son complément (voir la figure 2.9 p.21).

Le même principe s'applique à tous les montages. Nous appellerons les transistors qui passent des variables des *transmetteurs de variables*.

2.2.2 La construction d'un transmetteur de variables

On peut déduire un transmetteur de variables à partir du transmetteur de constantes associé à la même fonction.

Voici les règles de passage des transmetteurs de constantes aux transmetteurs de variables:

$$\begin{aligned}
\text{(si } (a) \text{ alors } (b) \text{ sinon Z)} \| \text{(si } (\overline{a}) \text{ alors Z sinon } (b)) &= \\
\text{(si } (a \wedge \overline{b}) \text{ alors 0 sinon Z)} \| \text{(si } (\overline{a} \vee \overline{b}) \text{ alors Z sinon 1)} & \\
\\
\text{(si } (a) \text{ alors } (\overline{b}) \text{ sinon Z)} \| \text{(si } (\overline{a}) \text{ alors Z sinon } (\overline{b})) &= \\
\text{(si } (a \wedge b) \text{ alors 0 sinon Z)} \| \text{(si } (\overline{a} \vee b) \text{ alors Z sinon 1)} & \\
\\
\text{(si } (\overline{a}) \text{ alors } (b) \text{ sinon Z)} \| \text{(si } (a) \text{ alors Z sinon } (b)) &= \\
\text{(si } (\overline{a} \wedge \overline{b}) \text{ alors 0 sinon Z)} \| \text{(si } (a \vee \overline{b}) \text{ alors Z sinon 1)} & \\
\\
\text{(si } (\overline{a}) \text{ alors } (\overline{b}) \text{ sinon Z)} \| \text{(si } (a) \text{ alors Z sinon } (\overline{b})) &= \\
\text{(si } (\overline{a} \wedge b) \text{ alors 0 sinon Z)} \| \text{(si } (a \vee b) \text{ alors Z sinon 1)} &
\end{aligned}$$

Voyons par exemple comment on peut réaliser l'opération $\oplus$ en se servant de transmetteurs de variables:

$$\begin{aligned}
a \oplus b &= \text{(si } (a \wedge b) \text{ alors 0 sinon Z)} \| \\
& \quad \text{(si } (\overline{a} \wedge \overline{b}) \text{ alors 0 sinon Z)} \| \\
& \quad \text{(si } (a \vee \overline{b}) \text{ alors Z sinon 1)} \| \\
& \quad \text{(si } (\overline{a} \vee b) \text{ alors Z sinon 1)} \\
&= \text{(si } (a) \text{ alors } (\overline{b}) \text{ sinon Z)} \| \\
& \quad \text{(si } (\overline{a}) \text{ alors } (b) \text{ sinon Z)} \| \\
& \quad \text{(si } (\overline{a}) \text{ alors Z sinon } (\overline{b})) \| \\
& \quad \text{(si } (a) \text{ alors Z sinon } (b))
\end{aligned}$$

Le montage de la porte $\oplus$ ne comprend plus que quatre transistors. Il est représenté sur la figure 2.17. On peut remarquer que le même symbole est employé pour les portes construites par transmission de constantes ou de variables.

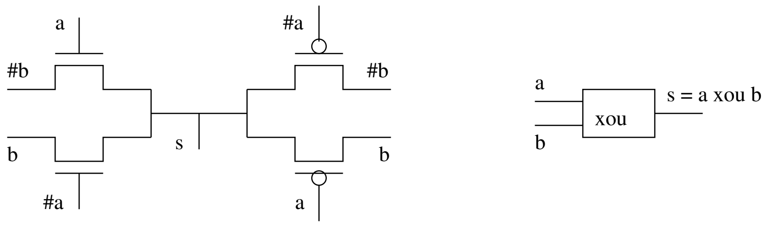

Figure 2.17 : Porte $\oplus$.

Voici maintenant $a \wedge b$ (voir figure 2.18) avec des transmetteurs de variables (trois transistors):

$$\begin{aligned}
a \wedge b &= (\text{si } (\overline{a} \wedge \overline{b}) \text{ alors } 0 \text{ sinon Z})\| \\
& (\text{si } (\overline{a} \wedge b) \text{ alors } 0 \text{ sinon Z})\| \\
& (\text{si } (a \wedge \overline{b}) \text{ alors } 0 \text{ sinon Z})\| \\
& (\text{si } (\overline{a} \vee \overline{b}) \text{ alors Z sinon } 1) \\
&= (\text{si } (\overline{a}) \text{ alors } 0 \text{ sinon Z})\| \\
& (\text{si } (a) \text{ alors } (b) \text{ sinon Z})\| \\
& (\text{si } (\overline{a}) \text{ alors Z sinon } (b))
\end{aligned}$$

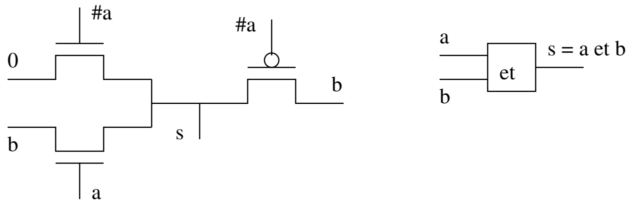

Figure 2.18 : Porte $\wedge$.

Voici $a \vee b$ (voir figure 2.19) avec des transmetteurs de variables (trois transistors):

$$\begin{aligned}
a \vee b &= (\text{si } (\overline{a} \wedge \overline{b}) \text{ alors } 0 \text{ sinon Z})\| \\
& (\text{si } (a \vee \overline{b}) \text{ alors Z sinon } 1)\| \\
& (\text{si } (\overline{a} \vee b) \text{ alors Z sinon } 1)\| \\
& (\text{si } (\overline{a} \vee \overline{b}) \text{ alors Z sinon } 1) \\
&= (\text{si } (\overline{a}) \text{ alors Z sinon } 1)\| \\
& (\text{si } (a) \text{ alors Z sinon } (b))\| \\
& (\text{si } (\overline{a}) \text{ alors } (b) \text{ sinon Z})
\end{aligned}$$

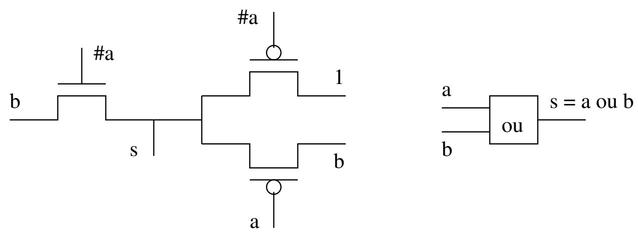

Figure 2.19 : Porte $\vee$.

De ces montages, on déduit ceux de $\overline{\wedge}$ et $\overline{\vee}$.

Le *multiplexeur* est une porte qui sélectionne l'une des deux entrées. En somme, il s'agit de l'opération:

si a alors b sinon c

Voici le multiplexeur avec des transmetteurs de variables (voir figure 2.20):

$$\begin{aligned} m &= (\text{si } (\overline{a} \wedge \overline{c}) \text{ alors 0 sinon Z})\| \\ &\quad (\text{si } (a \wedge \overline{b}) \text{ alors 0 sinon Z})\| \\ &\quad (\text{si } (a \vee \overline{c}) \text{ alors Z sinon 1})\| \\ &\quad (\text{si } (\overline{a} \vee \overline{b}) \text{ alors Z sinon 1}) \\ &= (\text{si } (a) \text{ alors } (b) \text{ sinon Z})\| \\ &\quad (\text{si } (\overline{a}) \text{ alors } (c) \text{ sinon Z})\| \\ &\quad (\text{si } (\overline{a}) \text{ alors Z sinon } (b))\| \\ &\quad (\text{si } (a) \text{ alors Z sinon } (c)) \end{aligned}$$

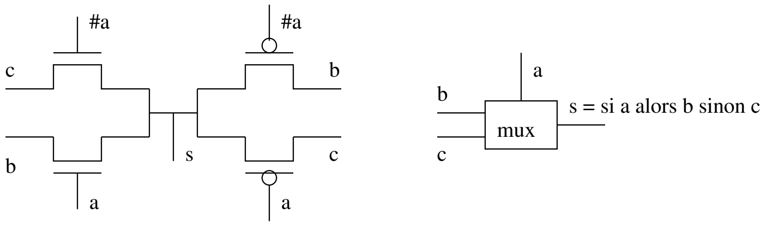

Figure 2.20 : Porte multiplexeur.

Exercice 2.12: à partir des équations de définition des portes $\wedge$, $\vee$, $\oplus$ et mux par transmission de variables, calculer la table de vérité des fonctions booléennes correspondantes.

Exercice 2.13: même question à partir des figures 2.17 à 2.20 (p.28 à 30).

Exercice 2.14: démontrer chacune des quatre règles de passage des transmetteurs de constantes aux transmetteurs de variables.

Exercice 2.15: construire une porte $a \Rightarrow b$ avec des transmetteurs de variables.

Exercice 2.16: construire une porte $a \oplus b \oplus c$ avec des transmetteurs de variables.

2.3 La logique dynamique

On désigne ainsi les montages contrôlés par un signal périodique (par exemple une horloge). Ainsi, quand le signal périodique est actif, la sortie d'une porte dynamique est chargée (valeur booléenne). Une fois que le signal périodique s'est inversé, la charge est isolée car la porte n'est plus passante (valeur Z que l'on peut interpréter comme la mémoire de la valeur booléenne). La logique dynamique peut être employée pour réaliser des portes, mais elle sert surtout à construire des *séparateurs*.

2.3.1 Le contrôle temporel par un signal unique: la logique TSPC

La figure 2.21 p.31 montre une porte dynamique TSPC (*True Single Phase Clock*: portes dynamiques commandées par un signal périodique unique). La porte proprement dite est représentée par ses réseaux nMOS et pMOS. Plus précisément, si la porte est la fonction booléenne f, les réseaux nMOS et pMOS sont l'implantation de la fonction $\overline{f}$ car la logique dynamique TSPC ajoute un inverseur (sur la figure, les deux transistors connectés aux alimentations). La partie dynamique se compose de deux transistors nMOS. La porte est passante quand $h = 1$. On dit qu'elle est *active* ou *transparente* à l'état haut. Symétriquement, on dit qu'elle est *opaque* quand h est à l'état bas.

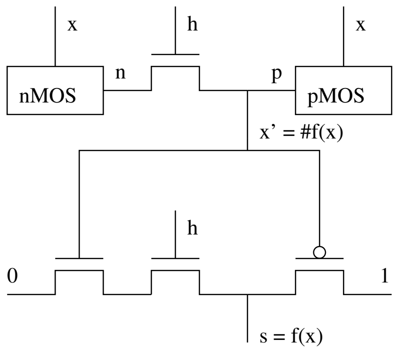

Figure 2.21 : Porte dynamique TSPC.

Le *chronogramme* de la figure 2.22 p.32 montre l'évolution de s en fonction des variations de x et du cycle de h. Les variations de h sont représentées en niveau (trait haut pour 1 et trait bas pour 0). Celles de l'entrée et de la sortie apparaissent en valeur arbitraire (double trait pour 0 ou 1, croisement pour un changement de valeur). Le chronogramme montre qu'une nouvelle entrée x arrive un peu après la descente de h à chaque cycle. Il montre aussi que la sortie s s'établit un peu après la montée de h. Sa valeur pendant la phase basse de h est une charge (Z). On peut la représenter par un trait unique à mi-chemin entre le trait haut du 1 et le trait bas du 0 (représentation usuelle de Z). Dans le cas présent, puisque la charge en s quand $h = 0$ est la mémoire de la valeur booléenne $f(x)$ et qu'elle est renouvelée tous les cycles, elle peut être confondue avec la valeur en s quand $h = 1$.

En supposant que x reste stable à la valeur x_i quand $h = 1$, les deux transistors commandés par h étant passants, on a $s = f(x_i)$. En effet, si $\overline{f}(x_i) = 0$, c'est le transistor pMOS de l'inverseur qui est passant et $s = 1$. Si $\overline{f}(x_i) = 1$, c'est le transistor nMOS de l'inverseur qui est passant et $s = 0$. Quand h s'inverse (x restant stable à la valeur x_i), les transistors nMOS commandés par h deviennent non passants. Si $\overline{f}(x_i) = 1$, le réseau nMOS qui fixe s n'est plus passant. Donc $s = Z$, ce Z étant une charge représentant la valeur logique 0. Si $\overline{f}(x_i) = 0$, le réseau nMOS établissant x' n'est plus passant. La commande x' est à Z, qui représente 0. Cette charge laisse le transistor pMOS de l'inverseur passant et $s = 1$. En conclusion, $s = f(x_i)$.

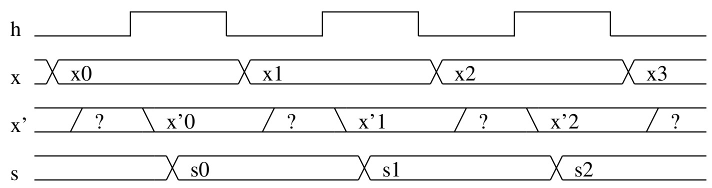

Figure 2.22 : Chronogramme d'une porte TSPC.

Pendant que $h = 0$, on substitue x_{i+1} à x_i. Si $\overline{f}(x_{i+1}) = 1$, on a $x' = 1$ avant même que h ne s'inverse car le réseau pMOS établissant x' reste passant (il est indépendant de h). Si au contraire $\overline{f}(x_{i+1}) = 0$, le terme x' continue de représenter $\overline{f}(x_i)$ jusqu'à ce que h monte, puisque le réseau nMOS fixant x' dépend de h (en x' il y a la charge $\overline{f}(x_i)$, isolée de $\overline{f}(x_{i+1})$ par le transistor commandé par h et par le réseau pMOS, tous deux non passants). C'est ce qu'indique le chronogramme en montrant que la valeur de x' est incertaine entre le changement de x et la montée de h.

Malgré tout, tant que h est bas, la possible variation de x' (de 0 à 1) est sans effet sur s. Elle rend non passant le transistor pMOS de l'inverseur, ce qui isole s (s = Z; ce Z représente un 1). Quand h monte, la nouvelle valeur de x', soit $\overline{f}(x_{i+1})$, passe en s en s'inversant (si $\overline{f}(x_{i+1}) = 1$, l'entrée 0 de l'inverseur franchit le réseau nMOS établissant s; si $\overline{f}(x_{i+1}) = 0$, le transistor pMOS de l'inverseur fixe s à 1).

En intercalant des transistors pMOS commandés par h entre les réseaux nMOS et pMOS, plutôt que des transistors nMOS, on forme une porte dynamique transparente à l'état bas et opaque à l'état haut. La figure 2.23 montre le montage.

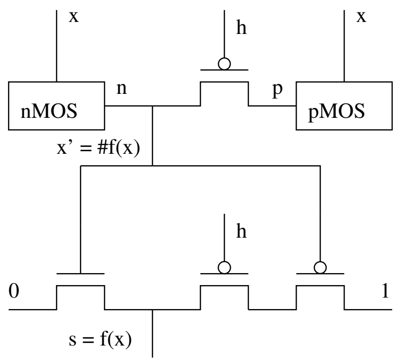

Figure 2.23 : Porte TSPC transparente à l'état bas.

<u>Exercice 2.17</u>: construire une porte $\wedge$ TSPC transparente à l'état bas.

2.3.2 La connexion de portes dynamiques TSPC

Pour relier des portes dynamiques TSPC en série, on alterne les portes actives à l'état haut avec celles actives à l'état bas. La figure 2.24 p.33 montre le montage.

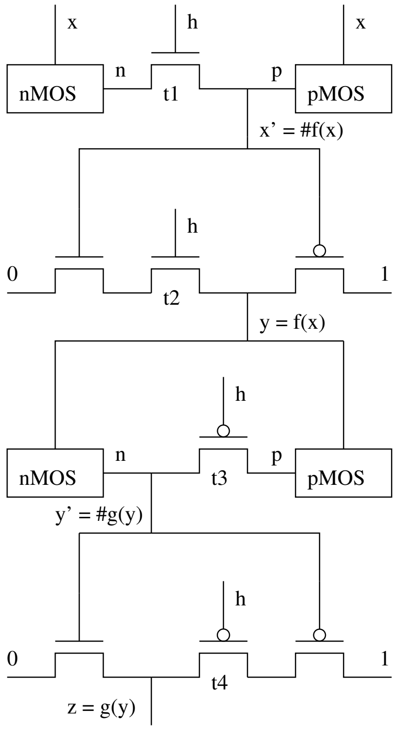

Figure 2.24 : Portes dynamiques TSPC en série.

Quand $h = 1$, les transistors $t1$ et $t2$ sont passants tandis que les transistors $t3$ et $t4$ sont non passants. Les rôles s'inversent quand $h = 0$. Le chronogramme de la figure 2.25 montre les évolutions de x, x', y, y' et z en fonction de h. Comme nous l'avons remarqué ci-dessus, la variation de x quand $h = 0$ peut avoir un effet sur x' en le faisant passer de 0 à 1 (ce qui est représenté par un trait oblique montant sur le chronogramme). On a vu que cela était sans effet sur y. Il en est de même pour y' qui peut passer de 1 à 0 quand $h = 1$, sans effet sur z.

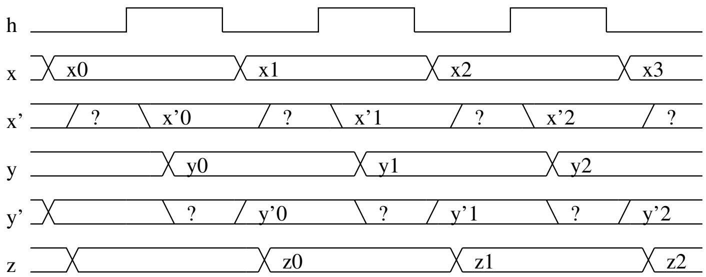

Figure 2.25 : Chronogramme de deux portes TSPC en série.

Dans la réalité, les quatre transistors ne reçoivent pas la variation de h au même instant. Supposons que le signal h les atteigne en ordre décroissant (ce qui peut se réaliser en faisant progresser h dans le sens inverse des portes). Le chronogramme de la figure 2.26 montre les évolutions de x, x', y, y' et z en fonction de h_i (h_i est le signal h commandant le transistor t_i). On constate que le montage fonctionne avec pour seule différence par rapport à la figure précédente des périodes d'instabilité pour y et z (sur la figure, elles sont représentées par les zones hachurées).

En effet, quand $t2$ devient passant ($h_2 = 1$), $t1$ reste non passant pendant un instant ($h_1 = 0$). La valeur en x' (indéterminée: déjà $\overline{f(x_i)}$ ou encore $\overline{f(x_{i-1})}$) passe en y. Ce n'est pourtant pas forcément la valeur définitive pour le cycle. Quand h_1 monte à son tour, x' s'établit définitivement pour le cycle à $\overline{f(x_i)}$, qui s'inverse et passe en y qui à son tour est stabilisé jusqu'au prochain cycle. La situation est la même pour z entre la descente de h_4 et celle de h_3.

Il faut également noter que pendant un court instant, les quatre transistors t_i sont simultanément passants ($h_3 = h_4 = 0$ et $h_1 = h_2 = 1$). Pendant cet instant, il est crucial que x ne varie pas encore. Le changement de valeur ne peut intervenir qu'une fois h_1 redescendu (donc h_2 aussi puisque l'on suppose qu'il descend avant). Si x variait de x_i à x_{i+1}, il ne serait pas impossible de retrouver $g(f(x_{i+1}))$ en z au lieu de $g(f(x_i))$.

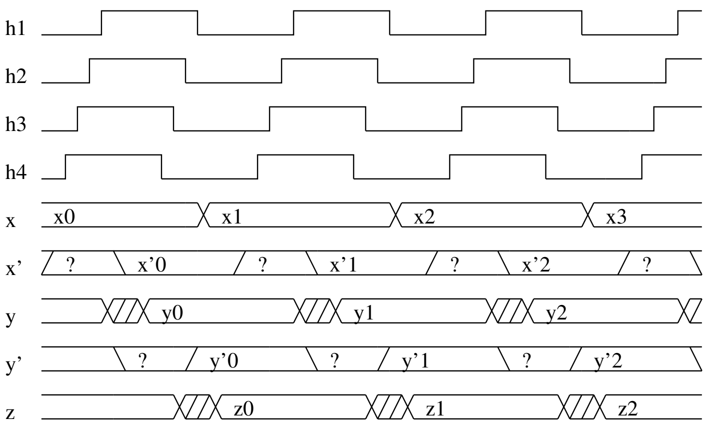

Figure 2.26 : Chronogramme détaillé de deux portes TSPC en série.

<u>Exercice 2.18</u>: dessiner le chronogramme de deux portes TSPC en série, l'une transparente à l'état haut calculant $y = a \wedge b$, l'autre transparente à l'état bas calculant $z = y \vee c$.

2.3.3 Les portes à préchargement

On peut utiliser la logique dynamique pour réaliser des portes CMOS économiques pour le nombre de transistors. Plutôt que de construire à la fois le réseau nMOS et le réseau pMOS d'une porte, la logique dynamique permet de ne construire qu'un seul des deux réseaux (en général, c'est le réseau nMOS) et de remplacer l'autre par un transistor de préchargement. La figure 2.27 p.35 montre une porte $\overline{a \vee b \vee c \vee d}$ construite selon ce principe.

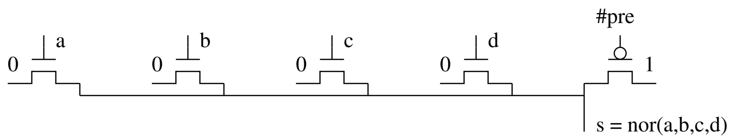

Figure 2.27 : Porte $\overline{a \vee b \vee c \vee d}$ *à préchargement.*

La phase de préchargement ($\#pre = 0$) fixe la sortie à 1. On initialise en quelque sorte s à 1 par défaut. Pendant ce temps, les entrées a, b, c et d sont stabilisées, de sorte que les transistors nMOS commandés par les entrées soient stables avant que $\#pre$ ne s'inverse. Une fois $\#pre$ mis à 1, si le réseau nMOS est passant (au moins un parmi a, b, c et d est un 1), la charge accumulée en s lorsque le transistor pMOS était passant s'évacue et s passe à 0. Dans le cas contraire, la charge reste et $s = 1$.

Exercice 2.19: construire une porte $a \wedge b \wedge c \wedge d$ à préchargement.

Exercice 2.20: construire une porte $\overline{a \wedge b \wedge c \wedge d}$ à préchargement.

2.4 Les séparateurs: verrous et bascules

2.4.1 Les verrous

En logique TSPC, si l'on remplace les réseaux nMOS et pMOS de la porte TSPC par des transistors uniques (un transistor nMOS et un transistor pMOS activés par une commande commune), on obtient un *verrou* (*latch* en anglais) en logique TSPC. Ainsi, la figure 2.28 représente un verrou transparent quand $h = 1$.

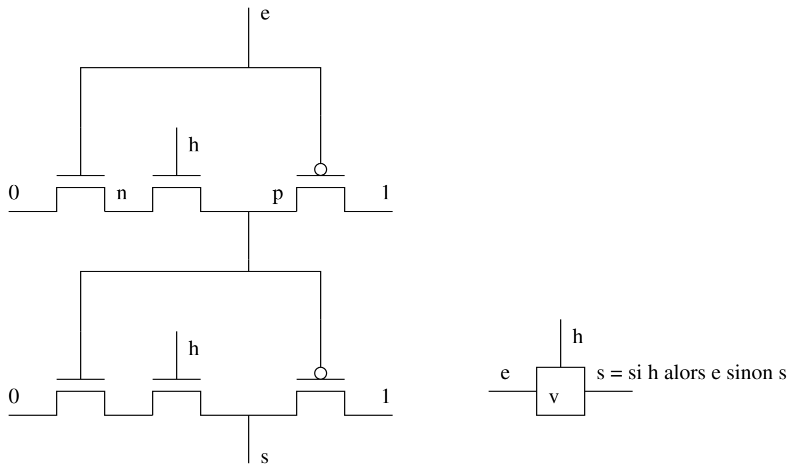

Figure 2.28 : Verrou TSPC.

La figure 2.29 p.36 représente un verrou transparent quand $h = 0$, ce que nous dénommerons dans la suite du texte un *#verrou*.

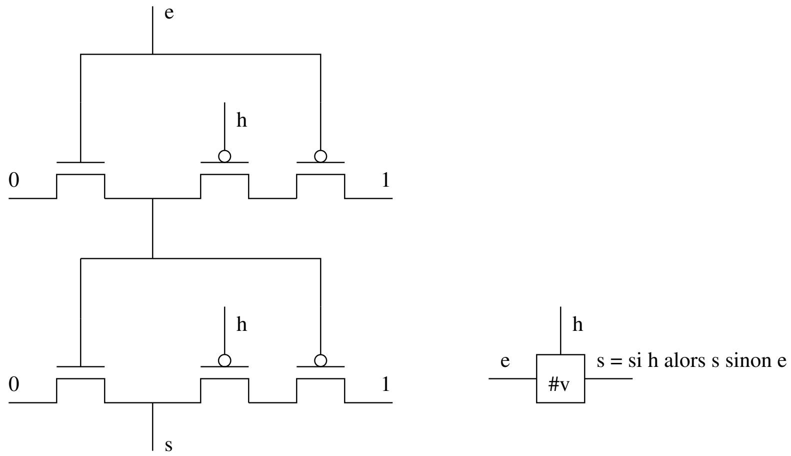

Figure 2.29 : #Verrou TSPC.

Exercice 2.21: dessiner un chronogramme de fonctionnement d'un verrou. À partir de la figure 2.28 p.35, étudier ce qui se passe quand la variation de l'entrée e est proche de la montée de h.

2.4.2 Les bascules

En plaçant un #verrou et un verrou en série (voir la figure 2.30), on obtient une *bascule* (*flip-flop* en anglais). Si le verrou précède le #verrou (voir figure 2.31 p.37), il s'agit d'une bascule *active sur front descendant*, que nous appellerons une *#bascule* dans la suite.

Dans le cas d'une bascule, une nouvelle entrée ne se retrouve en sortie qu'après la montée de h qui suit. Ceci est toujours vrai, que la nouvelle donnée arrive pendant que $h = 0$ ou que $h = 1$. Dans ce cas, il faudra que h descende puis remonte pour que la sortie reflète la nouvelle entrée. La bascule est un séparateur passant son entrée en sortie sur le front montant de l'horloge, alors que le verrou est un séparateur passant son entrée en sortie pendant que l'horloge est au niveau haut. On dit que les bascules fonctionnent *sur front* et les verrous *sur niveau.*

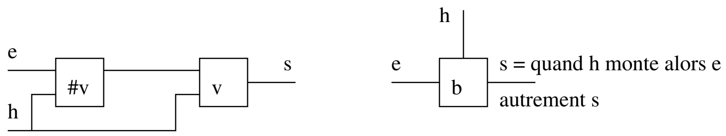

Figure 2.30 : Bascule TSPC.

Pour garantir le fonctionnement d'une bascule, il faut que la nouvelle entrée soit stable un peu avant que le front montant de h ne rende le verrou transparent (ce que l'on appelle le temps d'*établissement* ou en anglais *set up time*). À cet instant, h étant encore bas, le #verrou est transparent et la nouvelle entrée patiente devant le verrou.

Elle doit rester stable au moins tant que le #verrou n'est pas redevenu opaque (ce qui correspond au temps de *maintien* ou en anglais *hold time*).

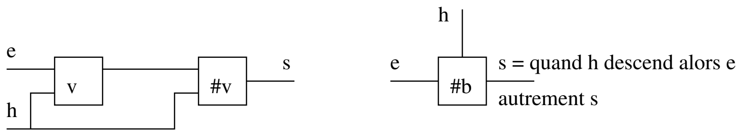

Figure 2.31 : #Bascule TSPC.

En plaçant une porte *et* devant le #verrou d'une bascule, on permet sa remise à zéro synchrone (voir figure 2.32). Notons que le signal #z de remise à zéro, actif à l'état bas et neutre à l'état haut, peut se réduire à une impulsion autour du front montant de h pour provoquer $s = 0$.

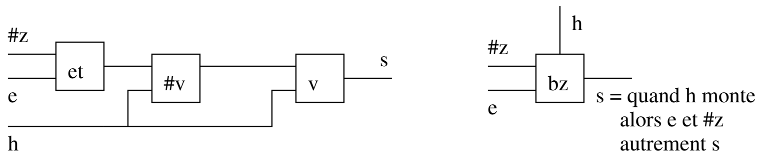

Figure 2.32 : Bascule TSPC à remise à zéro synchrone.

<u>Exercice 2.22</u>: quelles différences y a-t-il entre une porte de transfert CMOS et un verrou?

<u>Exercice 2.23</u>: étudier un montage composé de deux verrous en série, transparents sur niveau haut, le premier commandé par h et le second par $\overline{h}$. On suppose que le signal $\overline{h}$ arrive un peu après h, si bien que les quatre transistors du montage commutent dans l'ordre $t2$, $t1$, $t4$ puis $t3$.

<u>Exercice 2.24</u>: dessiner un chronogramme du fonctionnement d'une bascule et d'une #bascule.

<u>Exercice 2.25</u>: comment peut-on appeler une bascule dont la porte *et* de remise à zéro est placée entre le #verrou et le verrou?

<u>Exercice 2.26</u>: dessiner le chronogramme de la sortie h' du montage de la figure 2.33. Quelle est la fréquence de h' par rapport à celle de h?

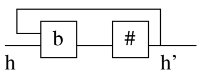

Figure 2.33 : Deux signaux h et h'.

<u>Exercice 2.27</u>: pour rendre une bascule inactive on emploie l'un des deux montages de la figure 2.34 p.38. Lequel est préférable? Pourquoi?

Figure 2.34 : Rendre une bascule inactive.

2.5 Les cellules mémoire

En logique CMOS, la fonction de mémorisation nécessite un mécanisme de conservation *ad hoc*. Les séparateurs (verrous et bascules) n'en contiennent pas puisque leur sortie est périodiquement mise à jour par l'activation de l'horloge.

2.5.1 Le maintien d'une valeur verrouillée

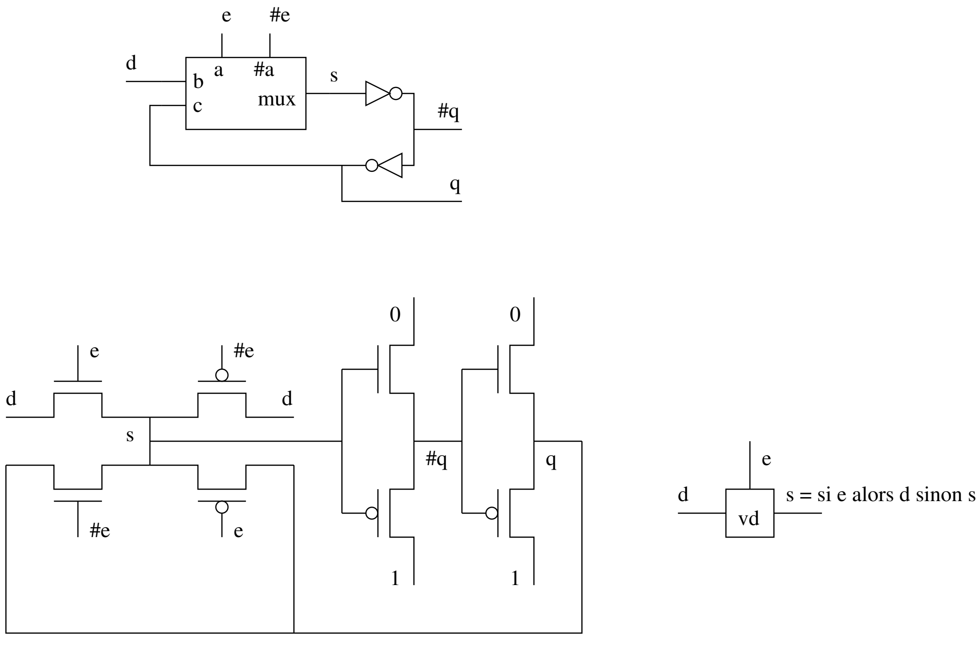

Figure 2.35 : Verrou-d.

La figure 2.35 présente un *verrou-d* (*d-latch* en anglais). Ce montage CMOS mémorise une valeur booléenne. Le signal d'écriture e permet de mémoriser la donnée d. Elle est conservée tant que e reste à l'état bas (verrou-d opaque), elle est ensuite remplacée par la nouvelle valeur de d quand e est haut (verrou-d transparent).

$$e = 1 \Rightarrow s = d \Rightarrow \overline{q} = \overline{d} \Rightarrow q = d$$

L'entrée d doit rester stable tant que $e = 1$.

$$e = 0 \Rightarrow s = q$$

Si on substitue d' à d, s ne change pas tant que $e = 0$. On voit que la mémorisation est assurée par une paire d'inverseurs dont les alimentations maintiennent la dernière valeur entrée.

Exercice 2.28: dessiner le chronogramme de fonctionnement d'un verrou-d.

2.5.2 La cellule de mémoire statique

Le montage précédent emploie huit transistors. On peut en réduire le nombre à six, ce qui est particulièrement avantageux si une telle cellule doit être dupliquée, comme c'est le cas pour un composant mémoire. On remplace le multiplexeur par deux transistors transmetteurs de variables comme indiqué dans la figure 2.36 qui présente une cellule de *sram*.

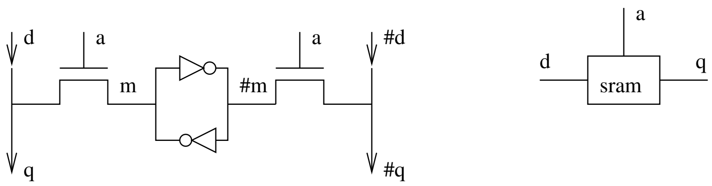

Figure 2.36 : Cellule de mémoire statique à six transistors.

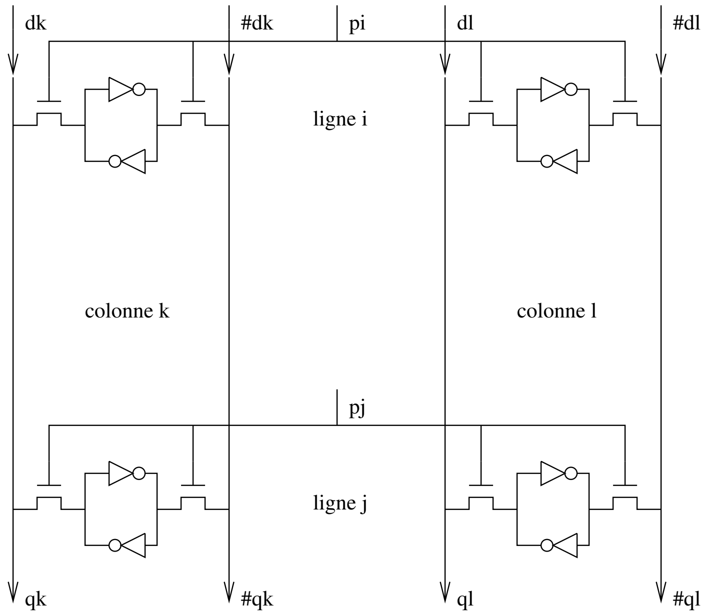

Figure 2.37 : Matrice de cellules de mémoires statiques.

Le montage tire parti de la symétrie de l'entrée et de la sortie d'un transistor nMOS. En lecture, les entrées sont en m et $\#m$ et les sorties en q et $\#q$. En écriture, le sens s'inverse: entrées en d et $\#d$, sorties en m et $\#m$. C'est la différence d'intensité des courants appliqués en d et m (respectivement en $\#d$ et $\#m$) qui détermine le sens.

En lecture, le signal d'accès a est activé, rendant passants les transistors d'accès à la cellule. La valeur mémorisée en m passe en q et symétriquement son complément passe de $\#m$ en $\#q$.

En écriture, les charges en d et $\#d$ sont portées par un courant de plus forte intensité que celui portant les charges en m et $\#m$. Dès que les transistors d'accès sont passants, les charges en d et $\#d$ (la valeur à écrire et son complément) l'emportent sur celles en m et $\#m$ (la valeur à remplacer et son complément).

La différence entre lecture et écriture est marquée par une commande externe à la cellule mémoire dont le rôle est de laisser passer (écriture) ou pas (lecture) une donnée d.

Notons que d'un point de vue logique, un seul des deux transistors d'accès est suffisant pour procéder à la lecture comme à l'écriture. Cependant, d'un point de vue électrique l'écriture est facilitée (accélérée) par l'emploi des deux accès (cas des mémoires statiques). On peut également en tirer avantage pour bénéficier de deux voies d'accès séparées, l'une pour la lecture et l'autre pour l'écriture (cas des registres; dans ce cas, les commandes des deux transistors d'accès sont distinctes).

La figure 2.37 p.39 montre comment un ensemble de cellules se partagent des voies d'accès et des signaux de commandes.

On lit ou (exclusif) on écrit une ligne à la fois. La ligne visée est sélectionnée par p. Si c'est la ligne i, $p_i = 1$ et $p_j = 0$, $\forall j \neq i$.

En ajoutant des transistors d'accès à la cellule, on fournit des voies d'accès supplémentaires. La figure 2.38 montre une cellule à huit transistors offrant un port d'écriture (accès commandé par e_0) et trois ports de lecture (accès au port i commandé par l_i). Le symbole qui l'accompagne désigne une cellule multiport offrant m ports de lecture (commandes l_0 à l_{m-1} et données lues q_0 à q_{m-1}) et n ports d'écriture (commandes e_0 à e_{n-1} et données à écrire d_0 à d_{n-1}). On y lit et y écrit à des instants distincts. En outre, en écriture, un port au plus est passant.

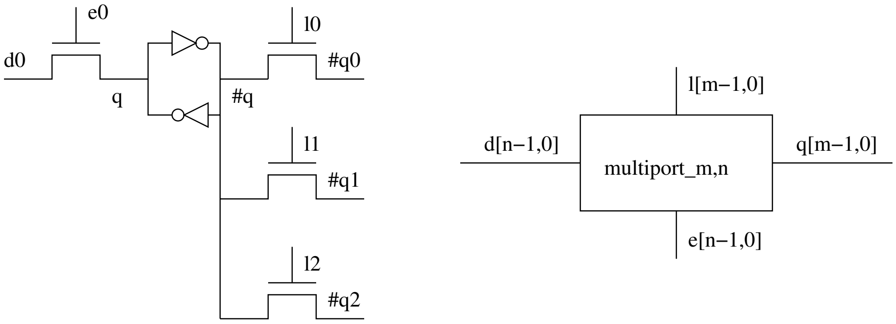

Figure 2.38 : Cellule de mémoire statique multiport.

<u>Exercice 2.29</u>: construire une cellule de mémoire statique à quatre ports d'accès: deux

ports en lecture, deux en écriture. Que se passe-t-il si les deux ports en écriture sont simultanément passants?

2.5.3 La cellule de mémoire dynamique

Figure 2.39 : Cellule de mémoire dynamique.

Un seul transistor suffit (voir figure 2.39), dont l'une des extrémités sert de capacité (c'est-à-dire qu'elle conserve la charge de la valeur mémorisée). Le transistor est rendu passant par une demande d'accès a (lecture ou écriture). En écriture, une charge, correspondant à la valeur à écrire, est appliquée en entrée d. Elle est maintenue jusqu'au chargement de la capacité en q. En lecture, la charge sortant du transistor en d quand $a = 1$ commande un amplificateur (des inverseurs en série), dont la sortie est rebouclée sur la cellule. Ainsi, la charge s'évacue, est amplifiée et rechargée (la lecture est destructrice et nécessite une ré-écriture).

Même lorsque la cellule n'est pas en accès ($a = 0$), la charge accumulée dans le transistor s'évacue par un courant de fuite. Il faut donc à intervalles réguliers recharger la cellule. C'est le *rafraîchissement* (on procède à une lecture qui effectue la ré-écriture).

Le symbole de la cellule fait apparaître que la voie de donnée d est bidirectionnelle. Il s'agit d'une cellule de *dram*.

2.5.4 La cellule de mémoire associative

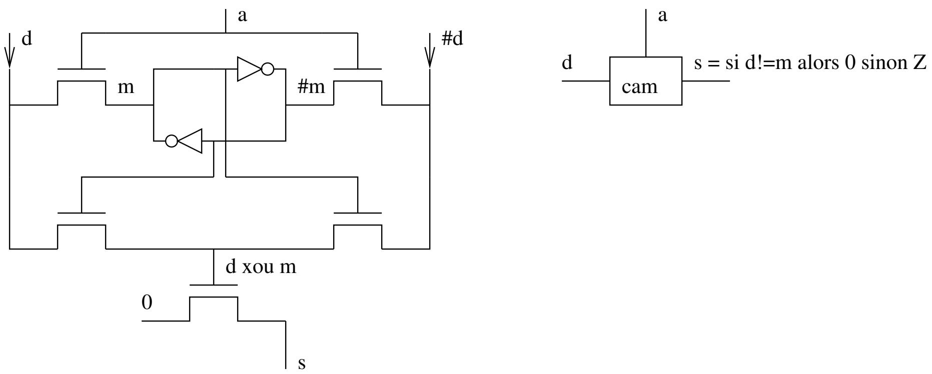

Figure 2.40 : Cellule de mémoire associative.

Une mémoire *associative* (CAM ou *Content-Adressable Memory*: mémoire atteinte par contenu) est une mémoire dans laquelle on recherche. On y accède donc non pas par adressage, mais en fournissant une donnée qui y est recherchée. La mémoire peut aussi être utilisée en mode de remplissage: la donnée n'est plus recherchée, mais écrite. L'entrée modifiée est désignée par un adressage, l'adresse étant fournie par un algorithme de sélection de l'entrée à remplacer.

Une cellule de mémoire associative (voir figure 2.40 p.41) se compose d'une cellule *sram* (pour la mémorisation et l'écriture) complétée d'un comparateur d'égalité (pour la recherche). Pour conserver à la cellule *cam* la compacité de la cellule *sram*, le comparateur est implanté sous la forme de deux transistors nMOS (réseau nMOS d'une porte $\oplus$). Sa sortie est filtrée par une porte de transfert (transistor nMOS) pour permettre la réunion de plusieurs comparaisons (comparaison de mot).

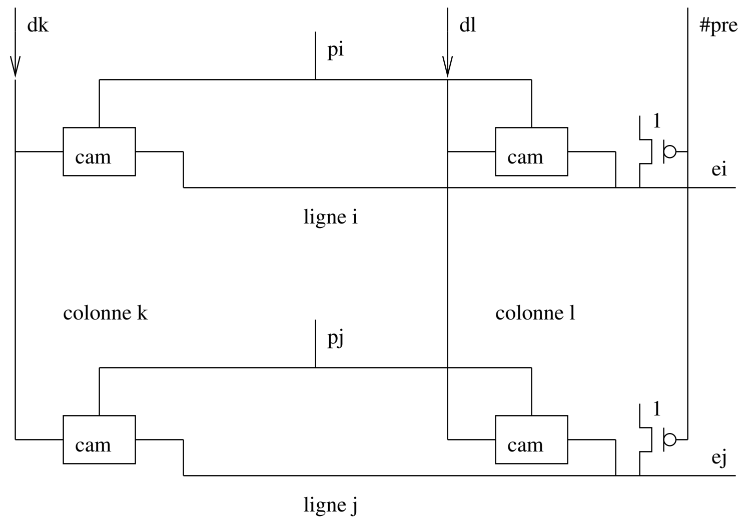

Figure 2.41 : Mémoire associative.

On emploie la cellule soit en recherche soit en écriture. En recherche, on fournit d et son complémentaire $\#d$ et on laisse a à 0 (les transistors d'accès de la cellule *sram* restent non passants). La sortie s vaut 0 si le contenu m de la cellule n'est pas égal à d. Dans le cas contraire, elle vaut Z. En écriture, on procède comme pour une cellule *sram*, en ouvrant l'accès à la cellule à écrire en activant a.

Une mémoire *cam* (voir la figure 2.41) se compose de cellules *cam* organisées en lignes. Les cellules d'une même ligne sont solidaires: elles sont atteintes ensemble. Ainsi, la recherche concerne le mot formé par les cellules d'une ligne. L'écriture concerne toutes les cellules d'une même ligne. Pour cela, les transistors d'accès des cellules d'une même ligne sont reliés à une même commande (commande p_i pour la ligne i). De plus, les sorties de comparaison d'une même ligne sont reliées, formant un signal d'égalité de ligne (signal e_i pour la ligne i). Ce signal est préchargé à 1 en activant la commande $\#pre$ ($\#pre$ = 0). Parallèlement, le mot $d = d_0 ... d_{n-1}$ et son complémentaire sont fournis. Puis on désactive $\#pre$ ($\#pre = 1$). Chaque ligne i dont le mot mémorisé est égal à d laisse son signal e_i à 1. Chaque ligne j dont le mot mémorisé est différent de d voit son signal e_j remis à 0 par une cellule *cam*.

Remarquons qu'en écriture on modifie une et une seule ligne (l'adressage permet de n'activer qu'un et un seul des signaux p_i). En revanche, en recherche, plusieurs lignes peuvent être égales à la donnée d.

<u>Exercice 2.30</u>: construire une cellule *cam* permettant de comparer son contenu simultanément à deux entrées d_1 et d_2, avec un port d'écriture unique. En déduire le nombre de transistors d'une cellule *cam* multiports à n ports de comparaisons et un port d'écriture.

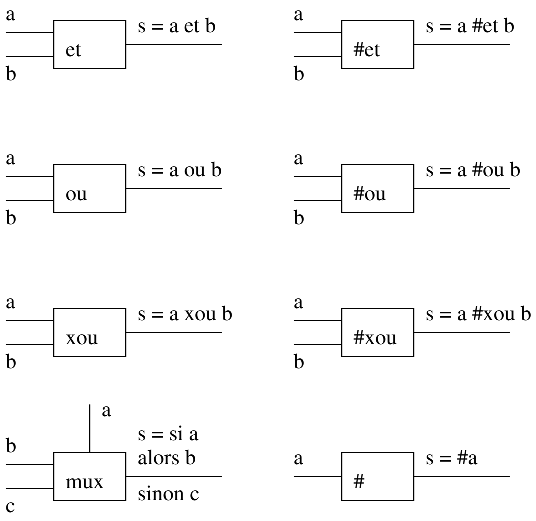

Figure 2.42 : Représentation symbolique des portes.

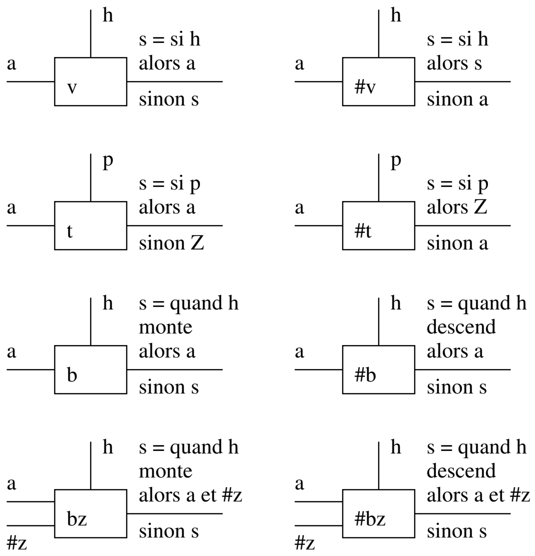

Figure 2.43 : Représentation symbolique des séparateurs.

3 Représentation symbolique des opérateurs élémentaires

Nous allons récapituler l'ensemble des portes, séparateurs et cellules mémoires vus jusqu'ici et qui forment la première couche de la construction des circuits digitaux.

La figure 2.42 p.43 montre la représentation symbolique des portes, faisant abstraction du mode de transmission (constantes ou variables).

La figure 2.43 p.43 montre les représentations symboliques des séparateurs.

La figure 2.44 montre les représentations symboliques des cellules de mémoire. Notons que pour une *sram* ou une *dram*, la différenciation entre lecture et écriture se fait de façon externe à la cellule. Pour une cellule *cam*, la commande *e/#r* est une écriture quand elle vaut 1 et une recherche quand elle vaut 0.

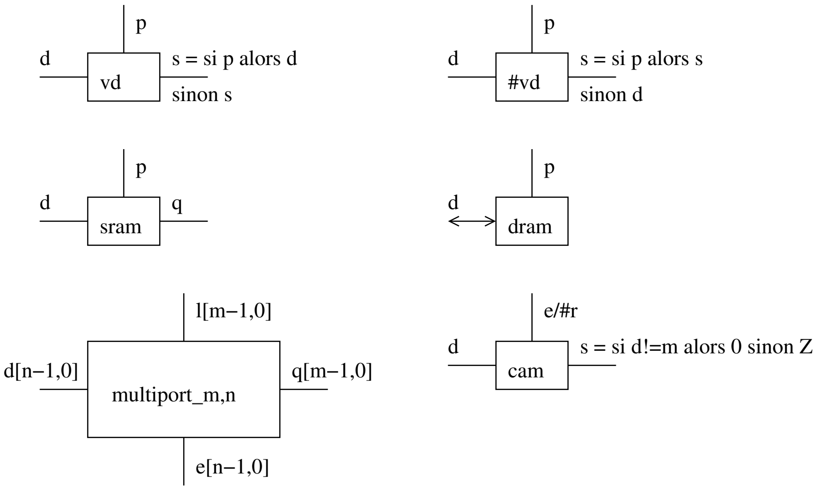

Figure 2.44 : Représentation symbolique des cellules de mémoires.

Références

- N. Weste et K. Eshraghian: *Principles of CMOS VLSI Design: A Systems Perspective*, deuxième édition, 1993, Addison-Wesley, Reading, Mass.
  Voilà un excellent ouvrage sur la technologie CMOS. On y trouve toutes les caractéristiques électriques des transistors nMOS et pMOS qui ne sont pas décrites ici. Y sont également traitées des réalisations détaillées de toutes les portes et de bon nombre des circuits qui seront décrits au chapitre suivant.
- Daniel W. Dobberpuhl, Richard T. Witek: *A 200-MHz 64-bit Dual-Issue CMOS Microprocessor. Digital Technical Journal*, Vol. 4/4, 1992.
  Cet article technique présente le processeur DEC Alpha 21064 à sa sortie en 1992. On y trouve une réalisation de verrous TSPC et la référence à l'article de J. Yuan et C. Svensson qui ont introduit cette logique en 1989. L'article (ainsi

que tous ceux du *Digital Technical Journal*) peut être obtenu (sans les schémas malheureusement) à l'adresse Web suivante:
`http://www.digital.com/info/DTJ800/`
À défaut, on peut également trouver cet article dans la revue IEEE JSSC:
Daniel W. Dobberpuhl, Richard T. Witek: *A 200-MHz 64-bit Dual-Issue CMOS Microprocessor. IEEE Journal of Solid State Circuits*, vol. 27, n° 11 (novembre 1992), pages 1555-1565.

- J. Yuan et C. Svensson: *High-Speed CMOS Circuit Techniques. IEEE Journal of Solid-State Circuits*, vol. 24, n° 1 (février 1989), pages 62-70.
  Cet article introduit la logique TSPC sur laquelle sont fondés les séparateurs d'aujourd'hui.
- Peter Van Zant: *Microchip Fabrication: A Practical Guide to Semiconductor Processing*, Mac Graw Hill, 4e édition, avril 2000.
  Ce livre traite de la fabrication des circuits intégrés et de leur coût.