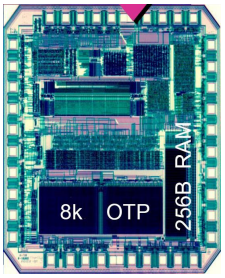
Polytech' Orléans

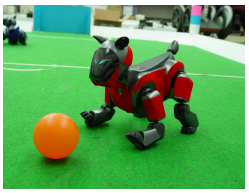


ELECTRONIQUE NUMERIQUE

G. Lamarque

Objectif

A partir de la connaissance
de l'architecture interne
d'un ordinateur
comprendre son fonctionnement



2

Glossaire

Ø ADC (Analog to Digital Converter) : Convertisseur Analogique Numérique (CAN).

Ø ALU (Arithmetic Logic Unit) : Unité arithmétique et logique. Cette unité permet d'effectuer toutes les opérations arithmétiques et logiques à l'intérieur d'un processeur.

Ø Antifuse : un petit circuit électronique qui peut passer de façon irréversible de l'état non conducteur à l'état conducteur (avec une impédance d'environ 100 ohm).

Ø ASIC (Application Specific Integrated Circuit) : Circuit numérique ou analogique contenant une application spécifique. La programmation est réalisée par marquage.

Ø CLB (Configurable logic Block) : nom donné par Xillinx aux blocs logiques.

3

Glossaire

Ø CMOS (Complementary Metal Oxide Silicon) : Technologie prédominante dans les circuits logiques et les mémoires. Cette technologie a remplacé l'ancienne technologie TTL (Transistor transistor Logic) dans la plupart des applications. la technologie CMOS permet d'obtenir des circuits faible consommation de très petite taille qui sont aujourd'hui aussi rapide voire plus rapide que les circuits en technologie TTL.

Ø CPLD (Complex Programmable Logic Device) : Circuit composé de plusieurs blocs de type PAL.

Ø CPU (Central Processing Unit) : Unité centrale de traitement.

Ø DAC (Digital to Analog Converter) : Convertisseur Numérique Analogique (CNA).

Ø Densité (density) : Quantité de logique interne à un composant programmable. Souvent mesuré en nombre de portes. Une meilleure mesure, pour les FPGA est de donner le nombre de cellules logiques. La nature d'une cellule logique diffère d'un constructeur à l'autre.

4

Glossaire

Ø FPD (Field Programmable Device) : Tout circuit, utilisé pour implémenter des fonctions logique, programmable par l'utilisateur. La programmation peut se faire soit en utilisant un programmateur
soit directement sur la carte.

Ø FPGA (Field Programmable Gate Array) : Circuit logique très complexe intégrant un très grand nombre de cellules logiques standard mais aussi des blocs spécialisés (multiplieurs, mémoires, etc...).

Ø GAL (Generic Array Logic) : Se sont des circuits identiques aux PAL mais la technologie utilisée permet de les effacer et de les reprogrammer électriquement.

Ø IOB (Input/Outpu Block) : Blocs logiques dédiés aux entrées/sorties.

Ø IP (Intelectual Property) : Fonction complexe (comme par exemple une FFT, un codeur MPEG, un microprocesseur, ...) appellée aussi "core" qui aide le concepteeur de circuits programmables) à réaliser des fonctions encore plus complexes.

5

Glossaire

Ø IR (Instruction register) : Le registre instruction contient l'instruction en court d'éxécution.

Ø ISP (In System Programmable device) : C'est un circuit logique programmable qui peut être programmé directement sur la carte après qu'il est été soudé.

Ø JTAG (Joint Test Action Group) : Ancien nom pour "IEEE 1149.1 Boundary Scan", un moyen de test des composants sur une carte.

Ø LUT (Look Up table) : Aussi appelé générateur de fonction. C'est une mémoire avec N (souvent entre 2 et 6) entrées et une sortie qui contient tous les états de sortie. Elle permet d'implémenter n'importe qu'elle fonction logique.

Ø LSI (Large Scale Integration) : Circuit intégré à grande échelle environ 1000 composants sur la même puce.

Ø MAR (Memory Adress Register) : Registre contenant l'adresse de la donnée à traiter.

6

Glossaire

Ø MBR (Memory Buffer Register) : Registre contenant un mot de donnée à écrire en mémoire ou la dernière donnée lue.

Ø OTP (One Time Programmable) : Mémoires ou circuits logiques programmable une seule fois par l'utilisateur. Les circuit en technologie fusibles (Fuses) ou anti-fusibles (Anti-fuses) sont implicitement des circuits OTP. Les circuits utilisant des EPROM deviennent des OTP lorsqu'il sont dans des boîtiers plastiques. en effet, il n'est plus possible d'intégrer de fenêtre pour pouvoir les effacer avec des UV.

Ø PAL (Programmable Array Logic) : Petit circuit programmable composé d'une matrice ET programmable et d'une matrice OU fixe. Ces circuits ne sont programmables qu'une seule fois par l'utilisateur.

Ø PC (Program Counter) : Le compteur programme contient l'adresse de l'instruction suivante à éxécuter.

7

Glossaire

Ø PLA (Programmable Logic Array) : Petit circuit programmable composé d'une matrice ET programmable et d'une matrice OU programmable. Ces circuits ne sont programmable qu'une seule fois par l'utilisateur.

Ø PSW (Program Status Word) : Registre contenant l'état du processeur suite à l'exécution d'une instruction.

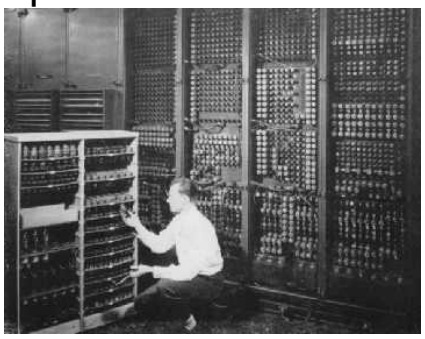
Ø SPLD (Simple Programmable Logic Device) : Petit circuit généralement de la complexité d'un PAL ou PLA.

Ø SSI (Small scale Integration) : Circuit intégré à petit échelle environ 100 composants sur une même puce.

Ø VLSI (Very Large Scale Integration) : Circuit intégré à grande échelle environ 10 000 composants sur une puce.

8

Historique : Premier ordinateur



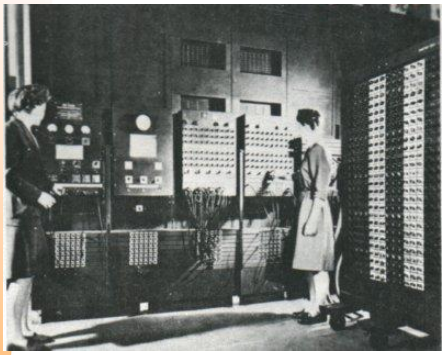
Remplacement d'une lampe

Ø L'ENIAC (Electronic numerical Integrator And Computer) a été conçu et construit sous la direction de john Mauchly et Prosper Eckert à l'université de Pennsylvanie. C'est le premier ordinateur électronique numérique généraliste de la planète.

- La construction de l'ENIAC a commencé en 1943.
- La construction de l'ENIAC c'est achevé en 1946.
- L'ENIAC a fonctionné jusqu'en 1955 date à laquelle il a été désassemblé.

9

Historique : Premier ordinateur



Programmation de l'ENIAC

- Poids : 30 tonnes.
- Encombrement : 500m² au sol.
- 18 000 tubes à vide.
- Consommation : 140KW.
- Performances : 5 000 additions par seconde.

- C'était un ordinateur décimal et non pas binaire.
- L'ENIAC était programmé manuellement en positionnant des commutateurs et en branchant et débranchant des câbles.

10

Historique : le transistor

- Le premier changement majeur à affecter l'ordinateur électronique a été le remplacement du tube à vide par le transistor.
- Le transistor est :
  - Plus petit.
  - Moins cher.
  - Produit moins de chaleur.

  qu'un tube à vide.
- Le transistor a été inventé dans les laboratoires Bell en 1947.
- Dés les années 1950 il révolutionne le monde de l'électronique.
- Les premiers ordinateurs entièrement transistorisés ont vu le jour en 1950.
- Pendant 10 ans de 1950 à 1960 les équipements électroniques étaient composés essentiellement de composants discrets :
  - Transistors.
  - Résistances.
  - Condensateurs.

11

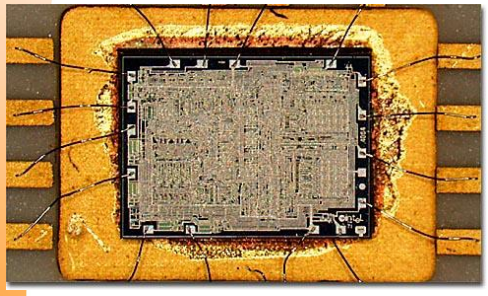
Historique : du transistor au circuit intégré

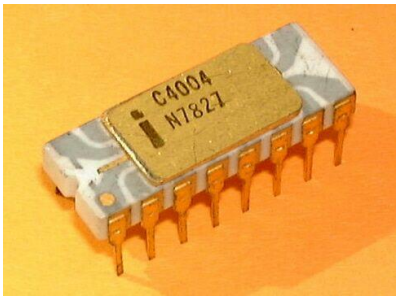
- Les tous premiers ordinateurs contenaient environ 10 000 transistors.
  - L'ensemble du processus de fabrication, depuis le transistor jusqu'à la carte à circuits imprimés était coûteux et complexe.
  - Chaque composants discrets étaient fabriqués séparément, intégrés à son propre boîtier.
  - Les différents composants étaient alors soudés et câblés ensemble sur des cartes à circuits imprimés que l'on installait à l'intérieur des ordinateurs.
- En 1958, nouvelle révolution dans le monde de l'électronique avec la naissance de l'ère de la micro électronique : l'invention du circuit intégré.

12

Historique : le microprocesseur

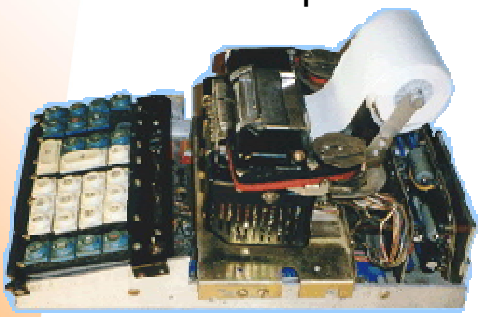
- 1971 : Innovation majeure, développement par Intel du premier microprocesseur le 4004.
- Le 4004 pouvait ajouter 2 nombres de 4 bits et ne pouvait multiplier que par répétition d'additions.

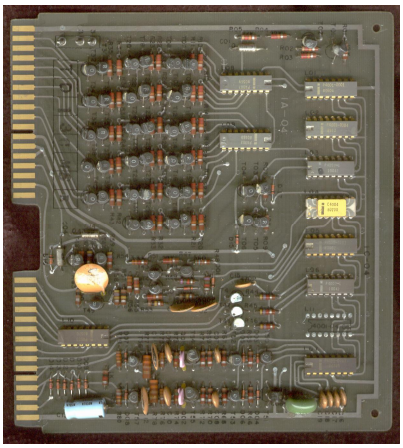




13

Historique : le BUSICOM

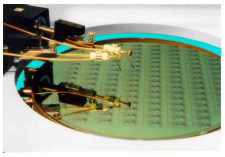




14

Historique : du Silicium au microprocesseur

Transistor



Wafer

Porte Logique
Plusieurs transistors

Microprocesseur
Plusieurs millions de transistors

15

Historique : le microprocesseur

- Un microprocesseur se compose :
  - De portes logiques.
  - De cellules mémoires.
  - D'interconnexions entre les différents éléments.
- Les portes et les cellules mémoires sont construites à l'aide de composants électroniques numériques simples.
- Un circuit intégré exploite le fait que des composants tels que :
  - Les transistors.
  - Les résistances.
  - Les condensateurs.

  Peuvent être fabriqués à partir d'un semi conducteur comme le silicium.

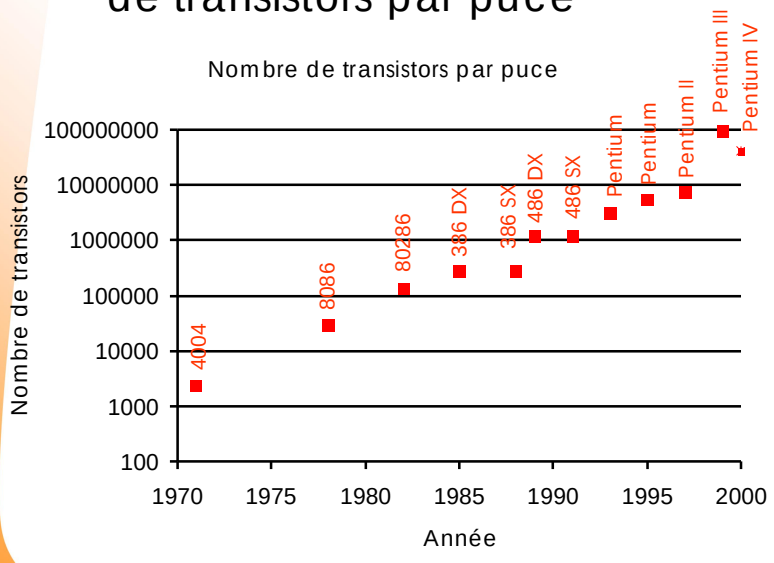
16

Historique : le microprocesseur

- Le fait que l'on se soit mis à fabriquer un circuit entier dans un petit morceau de silicium au lieu d'assembler dans un même circuit des composants discrets fabriqués à partir de différents morceaux de silicium n'est qu'une évolution des technologies de l'état solide.
- On peut placer de nombreux transistors sur une tranche de silicium.
- Ces transistors peuvent être connectés par un processus de métallisation afin de former des circuits.
- Au début on arrivait à fabriquer et à assembler de façon fiable que quelques portes ou cellules mémoire.
- Avec le temps, il a été possible d'intégrer de plus en plus de composants au sein d'une même puce.

17

Historique : Évolution du nombre de transistors par puce



18

Historique : Évolution du nombre de transistors par puce

- L'augmentation rapide du nombre de transistors intégrés sur une même puce à eu des conséquences importantes :
  - Le coût d'une puce est resté pratiquement inchangé au cours de cette période d'augmentation rapide de la densité.
    - Le prix de la logique de traitement et des circuits de mémoire a chuté à un rythme impressionnant.
  - Comme les éléments logiques et les éléments mémoires sont de plus en plus denses.
    - Le chemin que doit parcourir l'impulsion électrique est raccourci.
    - Accroissement de la vitesse de fonctionnement.
  - Les interconnexions sur les circuits intégrés sont bien plus fiables que les connexions par soudure.
    - Augmentation de la fiabilité des cartes électronique.

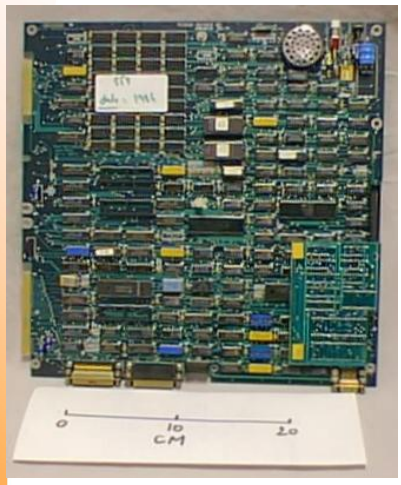
19

Historique : le microprocesseur

- 1972 : Introduction par Intel du 8008, le premier microprocesseur 8 bits.
- 1974 : Lancement par Intel du 8080, le premier microprocesseur «multi-usages». Le 4004 et le 8008 avaient été conçus pour des applications spécifiques.
  - Plus rapide.
  - Jeu d'instructions plus complexe.
- 1978 : Premier microprocesseur 16 bits, le 8086.

20

Historique : le 8086



Carte PC avec un 8086

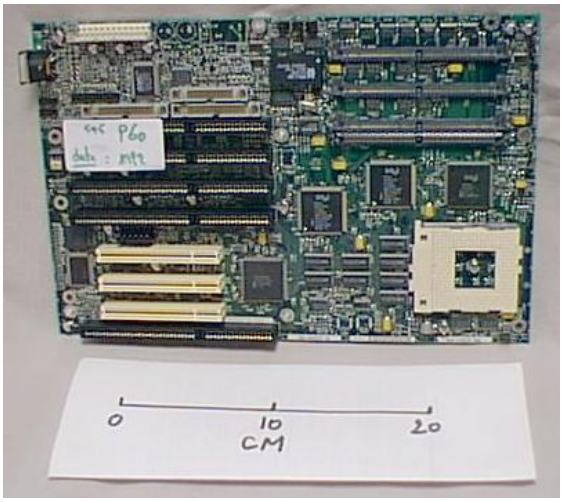
8086

Historique : le 80386

- 1981 : Premier microprocesseur 32 bits en une seule puce développé par Bell Labs et Hewlett Packard.
- 1985 : Intel introduit son microprocesseur 32 bits le 80386.

Historique : le microprocesseur

- 1995 : Intel introduit son microprocesseur 64 bits le Pentium.



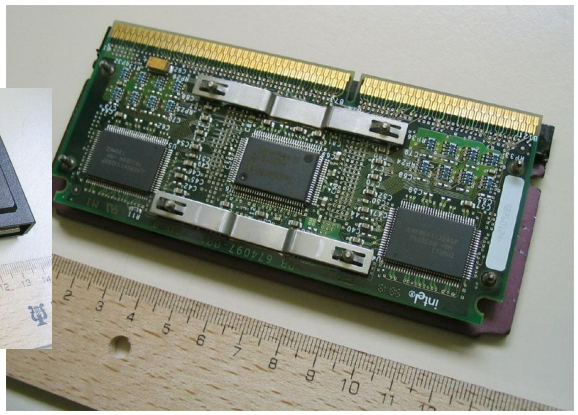
23

Historique : le Pentium

- Les constructeurs de puces de processeurs se sont lancés dans une course acharnée à la performance, ce qui a donné des processeurs comme le Pentium.
- La vitesse brute du microprocesseur ne peut atteindre son plein potentiel que si la puce est alimentée constamment par un flot de travaux à exécuter.
  - Toute entrave à ce flot régulier réduit la puissance du processeur.
  - Aujourd'hui, le principal problème est la vitesse à laquelle il est possible de transférer les données entre la mémoire et le processeur.
  - Difficile dans ces conditions d'alimenter en instructions et en données le processeur.

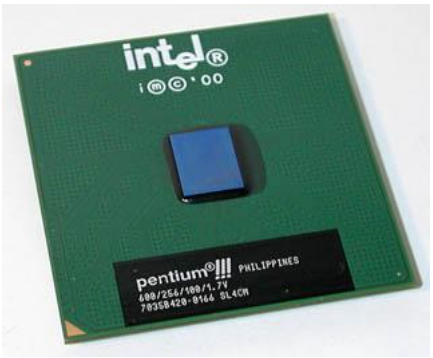
24

Historique : le Pentium II



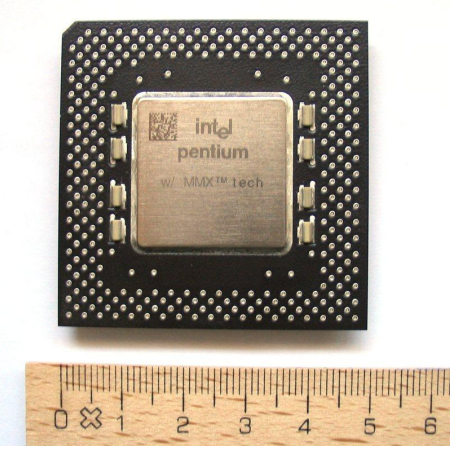
25

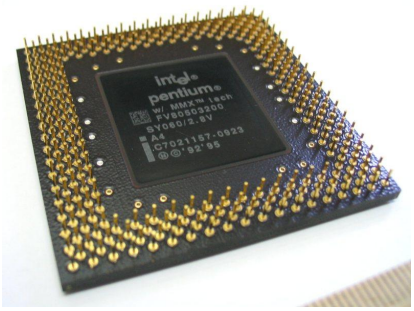
Historique : le Pentium III



26

Historique : le Pentium MMX





27

Historique : le Pentium IV

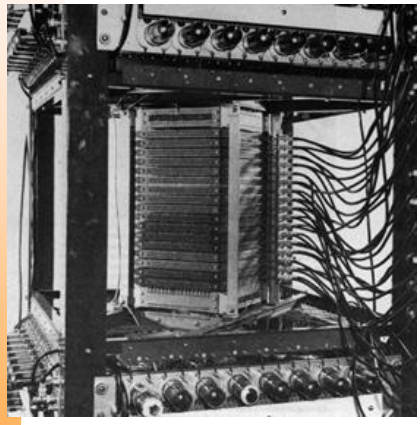
28

Historique : le microprocesseur

- Solution :
  - Augmenter la taille du bus de données : augmenter le nombre de bits récupérés simultanément.
  - Utiliser plusieurs types de mémoires :
    - Des mémoires très rapides mais de faible capacité à proximité du processeur.
    - Des mémoires moins rapides mais de grande capacité.
  - L'histoire et l'avenir des ordinateurs est donc intimement lié à celui des mémoires.

29

Historique : les mémoires



Mémoire à tore magnétique
1024 mots de 16 bits (2Ko)

- Pendant les années 1950 et 1960, la plupart des mémoires étaient constituées de petits anneaux ferromagnétiques d'environ 0.5 cm de diamètre. Ces mémoires à tores magnétiques étaient :
  - Relativement rapide (temps de lecture de 1µs).
  - Destructive, une lecture entraînée la perte du bit stocké en mémoire.
  - Très chère.
  - Très volumineuse.

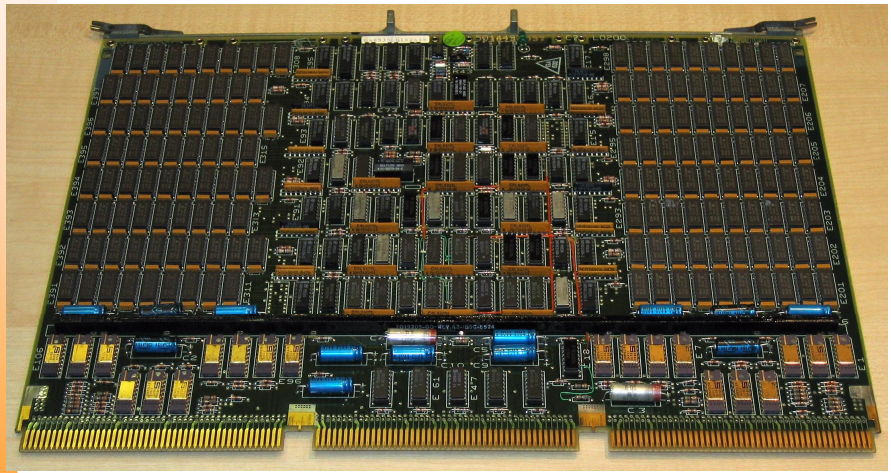
30

Historique : les mémoires

- En 1970, Fairchild a produit la première mémoire à semi-conducteurs relativement élaborée :
  - Dimension : taille d'un anneau (0.5 cm).
  - Capacité : 256 bits.
  - Vitesse de lecture : 70 nS.
  - Coût au bit : supérieur aux mémoires à tores.
- En 1974 : le coût des mémoires à semi conducteurs devient inférieur au coût des mémoires à tores magnétiques.

31

Historique : carte mémoire VAX 4MO (1986)



32

Évolution et performances des ordinateurs

- L'évolution des ordinateurs se caractérise par :
  - Une augmentation constante de la vitesse des processeurs.
  - La minimisation de la taille des composants.
  - L'augmentation de la taille de la mémoire.
  - L'augmentation des débits d'Entrées/Sorties.
- La réduction de la taille des microprocesseurs est largement responsable de l'accroissement des performances des processeurs :
  - La distance réduite entre les composants augmente la vitesse.
  - Aujourd'hui l'accroissement de la vitesse par la réduction des distances atteint ces limites.
  - Aujourd'hui pour accroître la vitesse des processeurs on optimise l'architecture interne des composants.
    - Pentium avec technologie dual core.

33

Évolution et performances des ordinateurs

Processeur	Année	Vitesse d'horloge Max	Largeur du bus	Nombre de transistors	Mémoire adressable	Finesse de la gravure
4004	71	108 KHz	4 bits	2 300	640 octets	10 µm
8086	78	10 MHz	16 bits	29 000	1 Mo	3 µm
80286	82	12.5 MHz	16 bits	134 000	16 Mo	1.5 µm
386 DX	85	33 MHz	32 bits	275 000	4 Go	1 µm
386 SX	88	33 MHz	16 bits	275 000	4 Go	1 µm
486 DX	89	50 MHz	32 bits	1.2 Millions	4 Go	0.8 µm

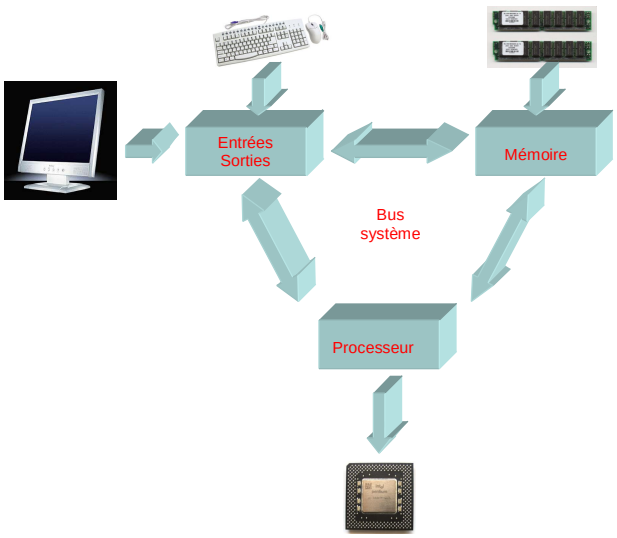
34

Évolution et performances des ordinateurs

Processeur	Année	Vitesse d'horloge Max	Largeur du bus	Nombre de transistors	Mémoire adressable	Finesse de la gravure
486 SX	91	133 MHz	32 bits	1.185 Millions	4 Go	1 µm
Pentium	93	166 MHz	32 bits	3.1 Millions	4 Go	0.8 µm
Pentium	95	200 MHz	64 bits	5.5 Millions	64 Go	0.6 µm
Pentium II	97	300 MHz	64 bits	7.5 Millions	64 Go	0.35 µm
Pentium III	99	600 MHz	64 bits	95 Millions	64 Go	0.25 µm
Pentium IV	2000	1.8 GHz	64 bits	42 Millions	64 Go	

35

Architecture simplifiée d'un ordinateur



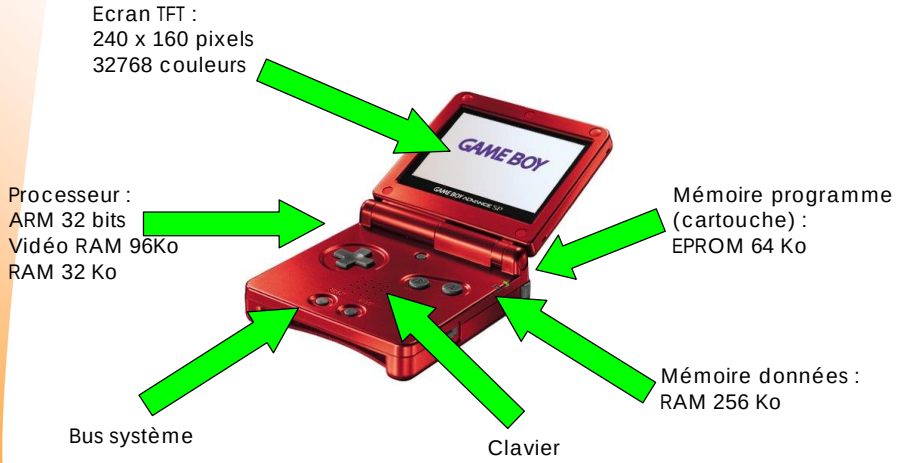
36

Architecture d'un ordinateur

- Un ordinateur est principalement composé :
  - D'un processeur (CPU : Central Processing Unit) : qui contrôle le fonctionnement de l'ordinateur et exécute ses fonctions de traitement des données.
  - De mémoires : qui mémorisent les données et contiennent le programme en cours d'exécution.
  - Les entrées/Sorties : qui permettent le transfert des données entre l'ordinateur et son environnement externe.
  - Les bus système : ensemble de fils sur lesquels circulent des signaux qui prennent en charge la communication entre le processeur, la mémoire et les entrées/sorties.

37

Exemple : la Game BOY



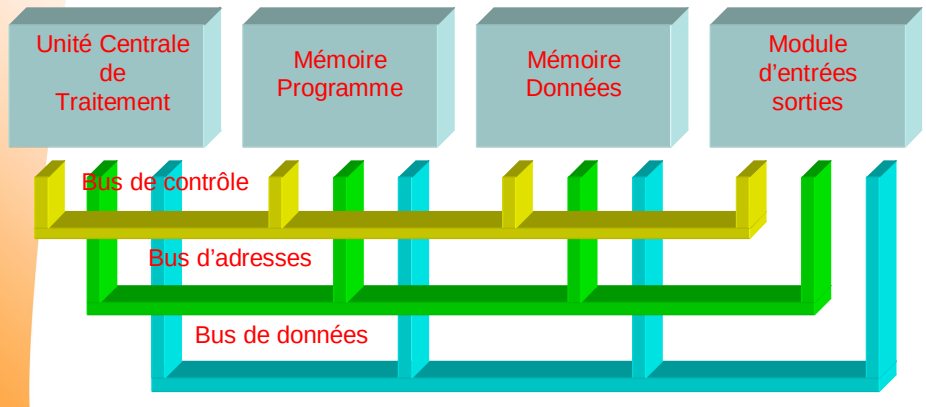
38

Exemple : la Game BOY

- C'est un ordinateur « spécifique » dédié principalement au jeu.
- Mais il n'y a aucune difficulté pour la transformer en un oscilloscope il suffit :
  - De rajouter, en se connectant au bus système, un module d'entrée avec un Convertisseur Analogique Numérique.
  - D'écrire un programme qui permette de faire l'acquisition d'un signal et d'afficher le résultat sur l'écran.
- Le PC est un ordinateur « généraliste » qui peut s'adapter à beaucoup de domaines :
  - Jeu.
  - Bureautique.
  - Calcul scientifique.
  - Multi médiat.

39

Architecture d'un ordinateur



40

Architecture d'un ordinateur

Le bus de données

- Sert à transférer les données entre les différents modules.
- La taille du bus de données permet de caractériser l'architecture d'un microprocesseur (lorsque l'on parle d'un microprocesseur 8 bits cela signifie que son bus de données est de 8 bits).
- Chaque ligne du bus peut transporter un seul bit à la fois. La taille du bus de données détermine le nombre de bits que l'on peut transférer simultanément.
- La taille du bus de données est un des facteurs clef qui détermine les performances globales d'un système.

Taille du bus de données	8 bits	16 bits	32 bits	64 bits
Taille du mot	Octet	2 Octets	4 Octets	8 Octets

41

Architecture d'un ordinateur

Le bus d'adresses

- Sert à désigner la source ou la destination des données.
- La largeur du bus d'adresses détermine la capacité mémoire maximale du système.
- Typiquement on se sert :
  - Des bits de poids fort pour sélectionner un module particulier connecté au bus.
  - Des bits de poids faible pour sélectionner une position particulière d'un module.

Taille du bus d'adresses	8 bits	16 bits	24 bits	32 bits
Mémoire adressable	256 mots	64K mots	16M mots	4G mots

42

Architecture d'un ordinateur

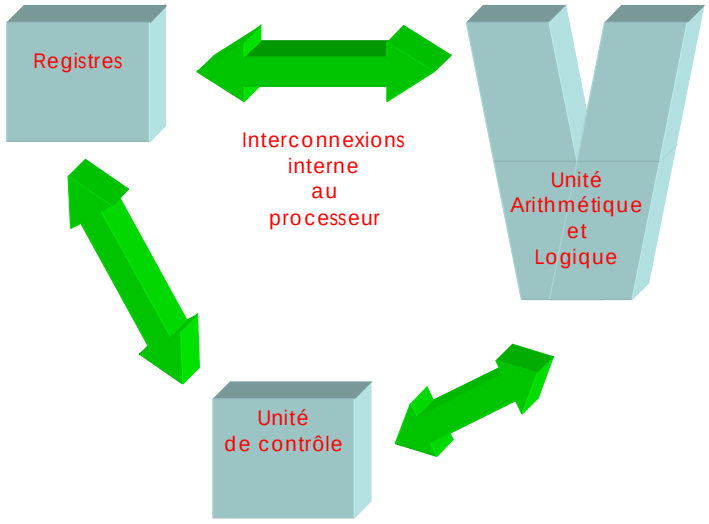
Le bus de contrôle

- Sert à contrôler l'accès et l'utilisation des bus de données et d'adresses.
- Les bus d'adresses et de données étant partagés entre tous les modules, il faut contrôler leur utilisation.
- Exemples de signaux de contrôle :
  - Lecture en mémoire données.
  - Écriture en mémoire données.
  - Lecture en mémoire programme.
  - Lecture d'un module d'entrée (ADC : Analog to Digital Converter).
  - Écriture dans un module de sortie(DAC : Digital to Analog Converter).
  - Horloge.
  - Reset.
  - Etc....

Architecture d'un ordinateur

- Les éléments de base d'un ordinateur numérique doivent être capables de réaliser des fonctions suivantes :
  - Mémorisation.
  - Traitement.
  - Transfert.
  - Contrôle.
- Fondamentalement seuls deux types de composants sont nécessaires pour cela :
  - Les portes.
  - Les cellules mémoire.
- En interconnectant de grandes quantités de ces composants fondamentaux on peut construire un ordinateur.

Architecture simplifiée d'un processeur



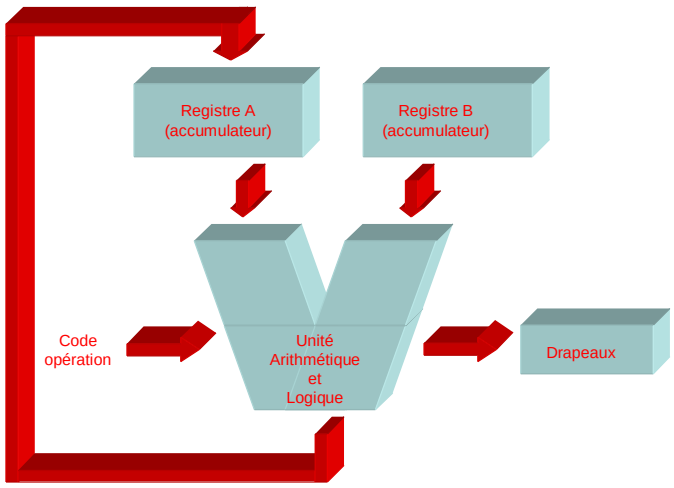
45

Architecture d'un processeur

- Intéressons nous au cœur de cet ordinateur :
  - Le processeur.
- Le processeur est l'élément le plus complexe de ce système il est composé :
  - D'une unité arithmétique et logique (ALU : Arithmetic Logic Unit) : qui exécute les fonctions de traitement sur les données.
  - D'une unité de contrôle : qui contrôle le fonctionnement du processeur et donc de l'ordinateur.
    - Elle supervise le transfert des données et des instructions vers et en provenance de la CPU.
    - Elle contrôle également le fonctionnement de l'ALU.
  - De registres : qui prennent en charge les opérations de mémorisation à l'intérieur du processeur.
  - Les bus processeurs : ensemble de fils sur lesquels circulent des signaux (données, adresses, contrôle) qui prennent en charge la communication entre l'unité de contrôle, l'ALU et les registres.

46

Architecture d'une ALU



47

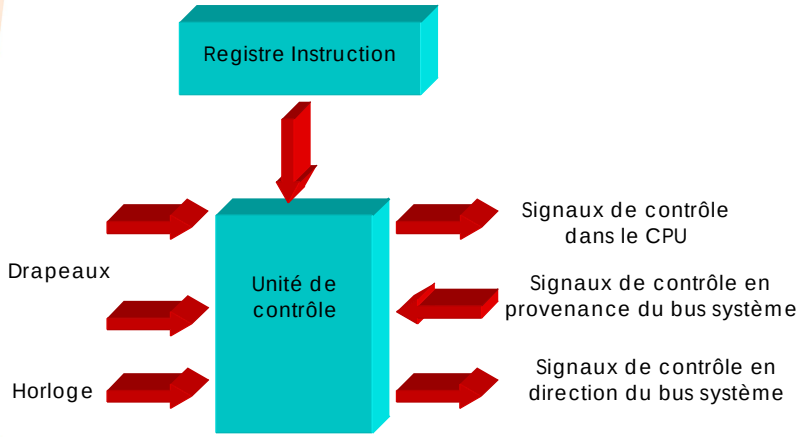
L'unité Arithmétique et Logique (ALU)

- C'est la partie du processeur qui effectue les opérations arithmétiques et logiques sur les données.
- La tâche des autres éléments du processeur (unité de contrôle, Registres, mémoire, Entrées/Sorties) consiste principalement à amener des données à l'ALU pour qu'elle les traite et à retourner le résultat.
- L'ALU est basée sur l'utilisation de dispositifs numériques simples capables de ranger des nombres binaires et d'effectuer des opérations booléennes logiques.
- L'ALU reçoit les données et range les résultats des opérations dans des registres (les registres sont des emplacements de mémorisation temporaire à l'intérieur du processeur).
- Une opération peut également positionner des drapeaux (*flag*) pour indiquer par exemple un dépassement de capacité (*overflow*).

48

Contrôle du processeur

- L'unité de contrôle effectue deux tâches de base :
  - Séquencement : le processeur passe par une série de micro opérations dans l'ordre approprié, en fonction du programme exécuter.
  - Exécution : chaque micro opération est exécutée.



49

Contrôle du processeur

- L'horloge : permet à l'unité de contrôle de « maîtriser le temps ». L'unité de contrôle déclenche l'exécution d'une micro opérations (ou d'un ensemble de micro opérations simultanées) par impulsion d'horloge.
- Registre instruction : le code opération de l'instruction en cours sert à déterminer les micro opérations à effectuer durant le cycle d'exécution.
- Drapeaux : l'unité de contrôle les utilise pour déterminer l'état du processeur et le résultat d'opérations antérieures de l'ALU.
- Signaux de contrôle provenant du bus de contrôle : la partie contrôle du bus système émet des signaux vers l'unité de contrôle : signaux d'interruptions et acquittements.
- Signaux de contrôle dans le processeur : ils sont de deux types ceux qui déclenchent le transfert des données d'un registre à un autre et ceux qui activent des fonctions spécifiques de l'ALU.
- Signaux de contrôle en direction du bus de contrôle : ils sont également de deux types : en direction de la mémoire et en direction des modules d'entrées/sorties.

50

Organisation d'un processeur

- La CPU doit :
  - Lire une instruction en mémoire.
  - Décoder l'instruction pour déterminer l'action à exécuter
  - Exécuter l'instruction. Il peut s'agir :
    - De lire des données dans la mémoire ou depuis un module d'entrée.
    - D'exécuter une opération arithmétique ou logique sur une donnée.
    - D'écrire des données dans la mémoire ou dans un module de sortie.

Organisation d'un processeur

- Pour ce faire la CPU doit :
  - Ranger des données de manière temporaire.
  - Se souvenir de l'emplacement de la dernière instruction pour savoir où récupérer la prochaine.
  - Ranger les instructions et les données de manière temporaire pendant l'exécution d'une instruction.
  - La CPU à donc besoin de petites mémoires internes : Les registres.

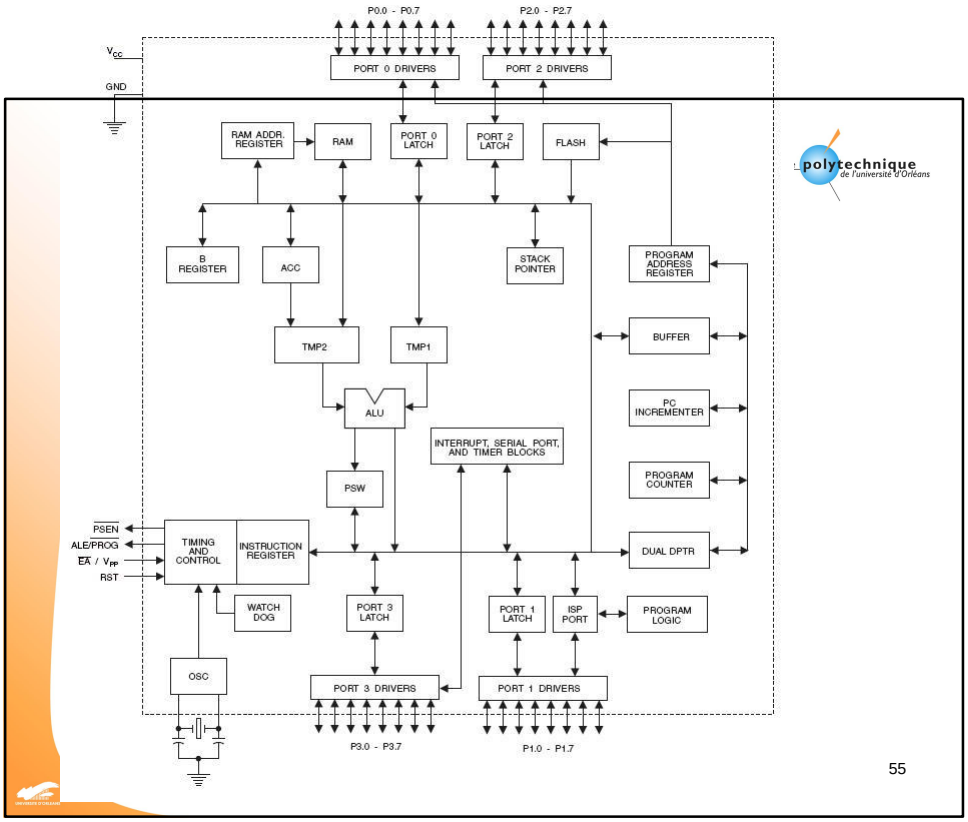
- Quatre registres sont essentiels à l'exécution d'une instruction :
  - Le compteur de programme (PC : Program Counter) : contient l'adresse de l'instruction à lire.
  - Le registre instruction (IR : Instruction Register) : contient la dernière instruction lue.
  - Le registre d'adresse mémoire (MAR : Memory Adress Register) : contient l'adresse d'un emplacement mémoire.
  - Le registre tampon mémoire (MBR : Memory Buffer Register) : contient un mot de données à écrire en mémoire ou le dernier mot lu.

Organisation d'un processeur

- La CPU met à jour le PC après chaque lecture d'instruction afin qu'il pointe toujours vers la prochaine instruction à exécuter.
- L'instruction lue est chargée dans le IR où les spécificateurs de code opération et d'opérande sont analysés.
- Les échanges de données avec la mémoire se font par le biais des registres MAR et MBR est directement connecté au bus de données.
- Dans le processeur, les données doivent être présentées à l'ALU pour y être traitées.
  - L'ALU dispose d'un accès directe au MBR et aux registres utilisateur.

Organisation d'un processeur

- Toute CPU comprend un registre ou un ensemble de registres souvent connus sous le nom de : « mot d'état du programme » (PSW : Program Status Word) contenant les informations d'état. Les drapeaux les plus courants sont :
  - Le bit de signe : indique le signe du résultat de la dernière opération arithmétique.
  - Le bit zéro : indique que le résultat est nul.
  - Le bit de retenu : indique si une opération a provoqué une retenue.
  - Le bit d'égalité : indique si une comparaison logique entraîne une égalité.
  - Le bit de dépassement : indique un dépassement arithmétique.
  - Le bit d'état d'une interruption : permet d'activer et de désactiver les interruptions.



Exemple : registre PSW du 8051

7	6	5	4	3	2	1	0
CY	AC	F0	RS1	RS2	OV	-	P

- P : Bit de parité : nombre paire/impaire de « 1 » dans l'accumulateur
- \- : Bit utilisateur
- OV : Bit d'overflow : indique un dépassement de capacité lors d'une opération arithmétique
- RS1, RS2 : Numéro de la banque de registres active (4 banques de registres différentes)
- F0 : Drapeau utilisateur
- AC : Bit de retenue auxiliaire : indique si une opération a provoquée une retenue entre le 3ème et 4ème bit
- CY : Bit de retenue : indique si une opération a provoquée une retenue

56

Caractéristiques d'une instruction

- Chaque instruction doit contenir les informations dont le CPU a besoin pour l'exécuter.
- A l'intérieur d'un processeur, chaque instruction est représentée par une séquence de bits.
- La CPU doit être capable d'extraire les données des différents champs de l'instruction pour accomplir l'opération attendue.
- Pour les programmeurs, il n'est pas facile de manipuler les représentations binaires des instructions machine. Une pratique consiste à utiliser une représentation symbolique de ces instructions : les mnémoniques.
- Exemple du 8051.

Exemple : Extrait du jeu d'instructions du 8051

Code	mnémonique	Fonction
1110 0100	CLR A	Remise à zéro du contenu de l'accumulateur A
1111 0100	CPL A	Complémente le contenu de l'accumulateur A
0001 0100	DEC A	Décrémente le contenu de l'accumulateur A
1000 0100	DIV AB	Divise le contenu de l'accumulateur A par le contenu de l'accumulateur B
1010 0100	MUL AB	Multiplie le contenu de l'accumulateur A par le contenu de l'accumulateur B
0000 0000	NOP	Aucune opération

Exemple : Extrait du jeu d'instructions du 8051

Code	mnémonique	Fonction
1100 0100	SWAP A	Échange les 4 bits de poids faibles et les 4 bits de poids fort de l'accumulateur A
0100 0100 + Data	ORL A,#Data	OU logique entre le contenu de l'accumulateur A et une donnée (Data)
0101 0100 + Data	ANL A,#Data	ET logique entre le contenu de l'accumulateur A et une donnée (Data)

59

Exemple : Extrait du jeu d'instructions du 8051

Code	mnémonique	Fonction
0111 0101 + Adresse + Data	MOV Adresse Destination, #Data	Transférer la donnée à l'adresse de la destination.
1000 0101 + Adresse destination + Adresse source	MOV Adresse Destination, Adresse Source	Transférer le contenu de l'adresse de la source à l'adresse de la destination.
0010 0100 + Data	ADD A,#Data	Ajouter le contenu de l'accumulateur A et la donnée. Mettre le résultat dans l'accumulateur A.

60

Caractéristiques d'une instruction

- Aujourd'hui, la plupart des programmes sont écrits dans un langage évolué ou exceptionnellement, en langage assembleur.
- Une seule instruction en langage évoluée peut nécessiter plusieurs instructions en langage machine.
- La conception du jeu d'instructions est l'un des aspects les plus intéressants et les plus étudiés de la conception d'un processeur. Le jeu d'instructions définit les nombreuses fonctions de la CPU et a donc une incidence significative sur l'implémentation matérielle du processeur. Les problèmes de conception les plus importants sont :
  - Répertoire des opérations : combien d'opérations prévoir, lesquelles, et quel degré de complexité.
  - Types de données : les différents types de données sur lesquelles il sera possible d'accomplir des opérations.
  - Format des instructions : longueur des instructions (en bits), nombre d'adresses, taille des différents champs.
  - Registres : nombre de registres de la CPU.
  - Adressage : modalité de spécification des adresses d'opérandes.

61

Compilation d'un programme C

```
Void main()
{
    char X, Y, Z ;
    X = 15 ;
    X++ ;
    Y = 2 ;
    Y++ ;
    Z = X + Y ;
}
```

Trois variables de type caractère :
- Espace occupée : 1 octet par variable.

X vaut 15 = 0x0F

On incrémente X, X vaut 16 = 0x10

Y vaut 2 = 0x02

On incrémente Y, Y vaut 3 = 0x03

On ajoute X et Y, le résultat Z vaut 19 = 0x13

62

Compilation d'un programme C

- Après compilation le code précédent devient :
  - La variable X est sauvegardée dans un registre. Ici le compilateur a choisi le registre R7.
  - La variable Y est sauvegardée dans un registre. Ici le compilateur a choisi le registre R6.
  - La variable Z (résultat de l'opération) est sauvegardée dans la mémoire donnée. La phase de compilation n'affecte pas d'adresses physiques en mémoire (programme ou donnée).
- Après transformation en mnémonique assembleur le code précédent devient :

Compilation d'un programme C

Code C	Mnémonique assembleur (pour un 8051)	Commentaires
X = 15 ;	MOV R7,#0FH	Mettre 15 dans le registre R7 (la variable X).
X++ ;	INC R7	Incrémenter le registre R7 (la variable X).
Y = 2 ;	MOV R6,#02H	Mettre 02 dans le registre R6 (la variable Y).
Y++ ;	INC R6	Incrémenter le registre R6 (la variable Y).
Z = X + Y ;	MOV A,R7	Mettre le contenu du registre R7 (variable X) dans l'accumulateur A.
	ADD A,R6	Ajouter le contenu de l'accumulateur A (variable X) et le contenu du registre R6 (variable Y).
	MOV @Z,A	Mettre de contenu de l'accumulateur A (résultat Z de l'opération) à l'adresse de la variable Z (non définie dans cette phase).
}	RET	Retour de sous programme (fonction main).

Compilation d'un programme C

- L'étape d'édition de liens va :
  - Affecter des adresses physiques (adresses mémoire donnée) aux variables non affectées à un registre.
  - Affecter des adresses physiques (adresses mémoire programme) au code.
- Cette étape tient compte de l'architecture de la cible :
  - Adresse mémoire donnée.
  - Adresse mémoire programme.
- Les mnémoniques peuvent alors être traduit en une suite de « zéros » et de « uns ». Le code C devient :

Compilation d'un programme C

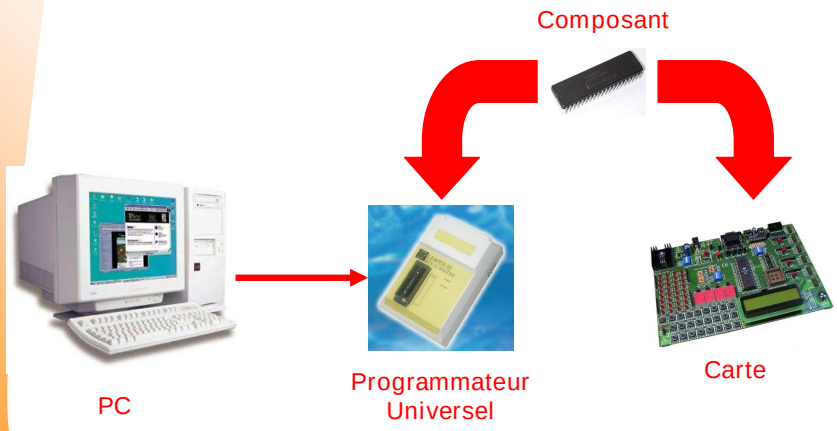
Code C	Mnémonique assembleur (pour un 8051)	Suite de « zéros » et de « uns »	
		Adresse mémoire programme	Code machine
X = 15 ;	MOV R7,#0FH	0000	7F
		0001	0F
X++ ;	INC R7	0002	0F
Y = 2 ;	MOV R6,#02H	0003	7E
		0004	02
Y++ ;	INC R6	0005	0E
Z = X + Y ;	MOV A,R7	0006	EF
	ADD A,R6	0007	2E
	MOV @Z,A	0008	F5
		0009	08
}	RET	000A	22

Chargement du programme

- Une fois le code machine sauvegardé dans la mémoire programme aux adresses définies par l'édition de liens le code peut être charger dans la mémoire programme.
  - Lorsque le code est compilé sur la machine cible il est chargé par l'outil de développement dans la RAM de l'ordinateur.
  - Lorsque le code est compilé sur une autre machine que la machine cible (cas des applications embarquées) le code doit être chargé dans la mémoire programme soit:

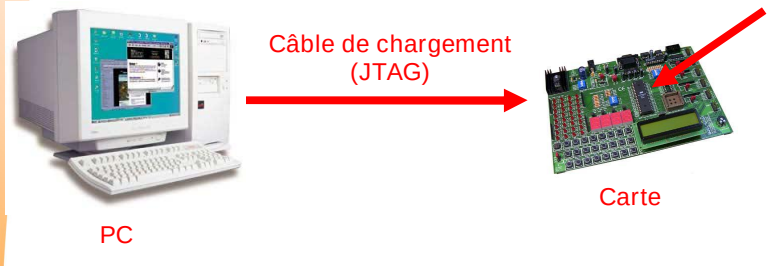
Chargement du programme

- En utilisant un programmateur.



Chargement du programme

v En utilisant une liaison dédiée.



69

En résumé

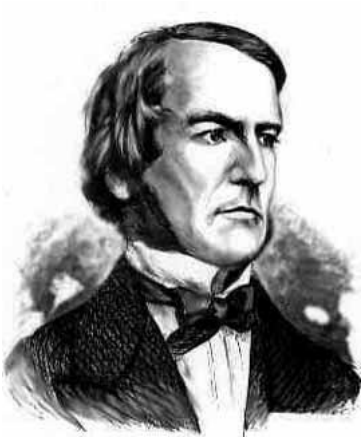
Ø Le fonctionnement d'un ordinateur est basé sur la mémorisation et le traitement de données binaires (suite de « zéros » et de « uns »).
Ø La conception et l'analyse du comportement des circuits numériques des ordinateurs et autres systèmes numériques reposent sur l'algèbre de Boole.

70

Algèbre de Boole

Ø George Boole était un mathématicien anglais qui a posé les principes de base de cette algèbre en 1854 dans son traité :

« *An Investigation of the laws of Thought on which to Found the Mathematical Theories of Logic and Probabilities* »



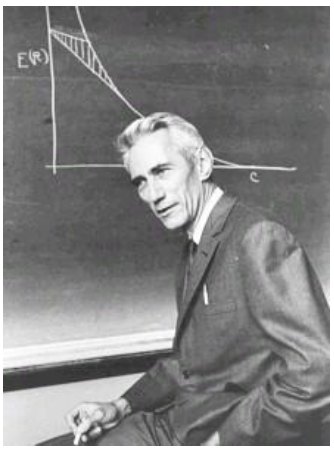
Georges Boole
1815-1864

71

Algèbre de Boole

Ø Claude Shannon, en 1938, alors assistant de recherche au service génie électrique du MIT (Massachusette Institute of Technology) a suggéré l'utilisation de l'algèbre de Boole pour résoudre le problème de conception de circuits de commutation à relais.

Ø Les techniques de Shannon ont ensuite été utilisées dans l'analyse et la conception des circuits numériques électroniques.



Claude Shannon
1916-2001

72

Algèbre de Boole

- L'algèbre de Boole est un outil pratique dans deux domaines :
  - Analyse : c'est une manière de décrire la fonction des circuits numériques.
  - Conception : En fonction de ce que l'on souhaite faire, on peut appliquer l'algèbre de Boole pour développer une implémentation simplifiée de cette fonction.

- Comme toute autre algèbre, l'algèbre de Boole utilise :
  - Des variables : une variable peut prendre :
    - La valeur 0 (Faux, False, Ouvert, Bloqué,...)
    - La valeur 1 (Vrai, True, Fermé, Saturé,...)
  - Des opérateurs : les opérations logiques de base sont :
    - ET (AND) : A . B
    - OU (OR) : A + B
    - NON (NOT) : $\overline{A}$

Table de vérité

- Une table de vérité permet de lister la valeur d'une opération pour chaque combinaison possible de valeurs d'opérandes.

- De manière générale un système à N entrées différentes. Dont chaque entrée peut prendre les valeurs 0 ou 1 de façon indépendante. Aura :
  - $M = 2^N$ états d'entrée différents.
  - La table de vérité aura donc $M=2^N$ lignes.

    - On voit rapidement la limite de ce type de représentation!

- De manière générale deux sorties seront distinctes si leur valeur est différente pour au moins un état d'entrée.

- Avec un système à N entrées on aura au maximum $L = 2^{2N}$ sorties distinctes.

Opérations logiques

Ø INVERSION (NON) : L'opération unaire NON inverse la valeur de cet opérande. La table de vérité est la suivante :

E	S
0	1
1	0

Ø ET : L'opération ET produit VRAI (valeur binaire 1) si et seulement si les deux opérandes sont VRAI. La table de vérité est la suivante :

A	B	S = A.B
0	0	0
0	1	0
1	0	0
1	1	1

75

Opérations logiques

Ø OU : L'opération OU produit VRAI (valeur binaire 1) si l'une ou l'autre des opérandes est VRAI. La table de vérité est la suivante :

A	B	S = A+B
0	0	0
0	1	1
1	0	1
1	1	1

76

Identités de base de l'algèbre de Boole

- Il existe deux classes d'identités :
  - Les règles de base (ou postulats) : qui sont déclarées sans être démontrées.
  - Les autres identités : que l'on peut dériver des postulats de base.

Postulats de base

- Commutativité
  - $A \cdot B = B \cdot A$
  - $A + B = B + A$
- Associativité
  - $A \cdot (B \cdot C) = (A \cdot B) \cdot C = (A \cdot C) \cdot B$
  - $A + (B + C) = (A + B) + C = (A + C) + B$
- Distributivité
  - $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
- Éléments identité
  - $1 \cdot A = A$
  - $0 + A = A$
- Élément d'inversion
  - $A \cdot \overline{A} = 0$
  - $A + \overline{A} = 1$
- En l'absence de parenthèses, l'opération ET a la priorité sur l'opération OU.

Autres identités

- $A + 1 = 1$
  - Évident dans la table de vérité.
- $A + A = A$
  - Évident dans la table de vérité.
- $A \cdot 0 = 0$
  - Évident dans la table de vérité.
- $A \cdot A = A$
  - Évident dans la table de vérité.

A	B	OU
0	0	0
0	1	1
1	0	1
1	1	1

A	B	ET
0	0	0
0	1	0
1	0	0
1	1	1

79

Théorème de De Morgan

- Le théorème de De Morgan permet de calculer des expressions de la forme :
  - $\overline{A \cdot B \cdot C \cdot \ldots \cdot Z}$
  - $\overline{A + B + C + \ldots + Z}$
- Les termes A, B, C, …, Z peuvent regrouper plusieurs termes.
- Le théorème de De Morgan permet de calculer l'inverse de n'importe quelle expression booléenne.
- L'inverse d'une somme est égal au produit des inverses des termes de la somme :
  - $\overline{A + B + C + \ldots + Z} = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \ldots \cdot \overline{Z}$
- L'inverse d'un produit est égal à la somme des inverses des termes du produit :
  - $\overline{A \cdot B \cdot C \cdot \ldots \cdot Z} = \overline{A} + \overline{B} + \overline{C} + \ldots + \overline{Z}$

Augustus De Morgan
1806-1871

80

Autres identités

- $A + A \cdot B = A$
  - $A + A \cdot B = A \cdot (1 + B) = A$
- $A + \overline{A} \cdot B = A + B$
  - $A + \overline{A} \cdot B = A + \overline{A + \overline{B}} = \overline{\overline{A} \cdot (A + \overline{B})} = \overline{\overline{A} \cdot A + \overline{A} \cdot \overline{B}} = \overline{\overline{A} \cdot \overline{B}} = A + B$

Formes canoniques d'une fonction booléenne

- Il faut extraire de la table de vérité les équations de chacune des sorties.
- Une fonction booléenne de plusieurs variables peut toujours être représentée par deux expressions équivalentes (théorème de De Morgan) :
  - Minterm : Somme de produits.
    - Somme de toutes les combinaisons pour lesquelles la fonction vaut 1.
  - Maxterm : Produit de sommes.
    - Produit de toutes les combinaisons pour lesquelles la fonction vaut 0.
- Ces deux formes ne sont pas minimales.

Exemple : additionneur

Ø on veut faire un additionneur binaire de deux nombres (A et B) codés sur 2 bits.

Ø 1ère étape : table de vérité
- q On a 4 bits d'entrée :
  - v 2 pour coder la nombre A.
  - v 2 pour coder le nombre B.
- q On a donc une table de vérité a $2^4 = 16$ lignes.
- q On a 3 sorties :
  - v 2 pour coder le résultat (S).
  - v 1 pour coder la retenue (C).

83

Exemple : additionneur

A		B		C	S	
A1	A0	B1	B0		S1	S0
0	0	0	0			
0	0	0	1			
0	0	1	0			
0	0	1	1			
0	1	0	0			
0	1	0	1			
0	1	1	0			
0	1	1	1			
1	0	0	0			
1	0	0	1			
1	0	1	0			
1	0	1	1			
1	1	0	0			
1	1	0	1			
1	1	1	0			
1	1	1	1			

84

Exemple : additionneur

Ø 2ème étape : Forme canonique de la fonction
  q Retenue (*Minterm*)
  $C = \overline{A1} \cdot A0 \cdot \overline{B1} \cdot B0 + A1 \cdot \overline{A0} \cdot B1 \cdot \overline{B0} + A1 \cdot \overline{A0} \cdot B1 \cdot B0 +$
  $A1 \cdot A0 \cdot \overline{B1} \cdot B0 + A1 \cdot A0 \cdot B1 \cdot \overline{B0} + A1 \cdot A0 \cdot B1 \cdot B0$
  q Poids fort du résultat (*Minterm*)
  $S1 = \overline{A1} \cdot \overline{A0} \cdot B1 \cdot \overline{B0} + \overline{A1} \cdot \overline{A0} \cdot B1 \cdot B0 + \overline{A1} \cdot A0 \cdot \overline{B1} \cdot B0 +$
  $\overline{A1} \cdot A0 \cdot \overline{B1} \cdot \overline{B0} + A1 \cdot \overline{A0} \cdot \overline{B1} \cdot \overline{B0} + A1 \cdot \overline{A0} \cdot \overline{B1} \cdot B0 +$
  $A1 \cdot A0 \cdot \overline{B1} \cdot \overline{B0} + A1 \cdot A0 \cdot B1 \cdot B0$
  q Poids faible du résultat (*Minterm*)
  $S0 = \overline{A1} \cdot \overline{A0} \cdot B1 \cdot B0 + \overline{A1} \cdot A0 \cdot \overline{B1} \cdot B0 + \overline{A1} \cdot A0 \cdot \overline{B1} \cdot \overline{B0} +$
  $\overline{A1} \cdot A0 \cdot B1 \cdot \overline{B0} + A1 \cdot \overline{A0} \cdot \overline{B1} \cdot B0 + A1 \cdot \overline{A0} \cdot B1 \cdot B0 +$
  $A1 \cdot A0 \cdot \overline{B1} \cdot \overline{B0} + A1 \cdot A0 \cdot B1 \cdot \overline{B0}$
Ø On aurait très bien pu donner la forme *Maxterm*.
Ø Le choix de la forme *Minterm* ou *Maxterm* repose uniquement sur le fait que la table de vérité contient plus de 1 ou de 0 pour la fonction de sortie.

85

Exemple : additionneur

Ø 3ème étape : simplifier les équations.
  q Il existe trois méthodes de simplification :
    v Simplification algébrique.
    v Tableaux de Karnaugh.
    v Méthode de Quine et McCluskey

86

Simplification algébrique

Ø La simplification algébrique implique l'application des identités de base de l'algèbre de Boole pour réduire l'expression booléenne à une expression composée de moins d'éléments.

Exemple additionneur (Retenues)

$C = \overline{A1}.A0.B1.B0 + A1.\overline{A0}.B1.\overline{B0} + A1.\overline{A0}.B1.B0 +$
$A1.A0.\overline{B1}.B0 + A1.A0.B1.\overline{B0} + A1.A0.B1.B0$

Ø Après simplification on trouve ...

$C = A1.B1 + A0.B1.B0 + A1.A0.B0$

Avec un peu (beaucoup) d'entraînement on peut devenir très performant.

C'est un jeu comme un autre...

87

Tables de Karnaugh

Ø La table de Karnaugh est la représentation d'une expression Booléenne d'un petit nombre de variables à l'aide d'un tableau de 2^N cases représentant les combinaisons de valeurs de N variables binaires possibles.

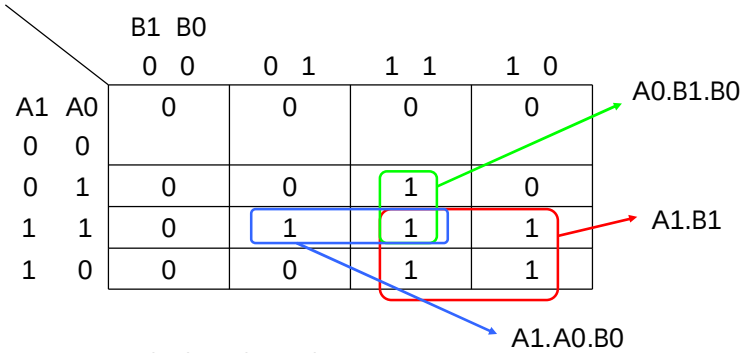
Ø La table de Karnaugh reprend, sous une forme, toutes les informations contenues dans la table de vérité en faisant apparaître les simplifications possibles.

Ø Les termes de la forme $A.\overline{X} + A = A + X$ sont matérialisés par des blocs de 2^N cases contiguës ayant la même valeur 1.

88

Tables de Karnaugh

Ø Exemple additionneur (retenue)



C = A1.B1 + A1.A0.B0 + A0.B1.B0

Ø C'est beaucoup moins laborieux que tout à l'heure mais cette méthode est quasi inexploitable lorsque le nombre de variables augmente (quasi inutilisable au delà de 6 variables d'entrée).

89

Méthode de Quine-McKluskey

Ø La méthode de la table de Karnaugh devient extrêmement compliquée pour plus de 4 variables.

Ø Avec 5 variables, il faut deux tables de 16x16, où l'une se place au-dessus de l'autre.

Ø Avec 6 variables, on passe à 4 tables de 16x16.

Ø Une alternative consiste à utiliser une approche tabulaire appelée méthode de Quine-McKluskey

Ø Elle peut se programmer sur un ordinateur pour fournir un outil automatique de minimisation des expressions booléennes.

Ø Aujourd'hui on ne cherche plus à simplifier la forme canonique on laisse ce travail fastidieux à l'ordinateur qui est :

- q Plus rapide.
- q Sans erreur.

90

Autres Fonctions

NON ET (NAND) : L'opération NON ET produit faux (valeur binaire 0) si et seulement si les deux opérandes sont VRAI.

La table de vérité est la suivante :

A	B	$S = \overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

Minterm : $S = \overline{A} . \overline{B} + \overline{A} . B + A . \overline{B}$

Maxterm : $S = \overline{A} + \overline{B}$

- ê Ces deux formes sont identiques.
- ê Cette fonction peut s'exprimer en fonction des opérateurs déjà étudiés.

91

Autres Fonctions

NON OU (OR) : L'opération NON OU produit VRAI (valeur binaire 1) si et seulement si les deux opérandes sont à FAUX.

La table de vérité est la suivante :

A	B	$S = \overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Minterm : $S = \overline{A} . \overline{B}$

Maxterm : $S = (A + \overline{B}) . (\overline{A} + B) . (\overline{A} + \overline{B})$

- ê Ces deux formes sont identiques.
- ê Cette fonction peut s'exprimer en fonction des opérateurs déjà étudiés.

92

Autres Fonctions

OU EXCLUSIF : OU exclusif (XOR) : L'opération OU exclusif produit VRAI uniquement lorsqu'une seule des deux opérandes (pas les deux) est VRAI.

La table de vérité est la suivante :

A	B	S = A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Minterm : $A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}$

Maxterm : $A \oplus B = (A + B) \cdot (\overline{A} + \overline{B})$

- ê Ces deux formes sont identiques.
- ê Cette fonction peut s'exprimer en fonction des opérateurs déjà étudiés.

93

Théorèmes fondamentaux

- Ø $A \oplus 0 = A$
  - q Évident dans la table de vérité.
- Ø $A \oplus A = 0$
  - q Évident dans la table de vérité.
- Ø $A \oplus 1 = \overline{A}$
  - q Évident dans la table de vérité.
- Ø $A \oplus \overline{A} = 1$
  - q Évident dans la table de vérité.

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

94

Théorèmes fondamentaux

- $A \oplus A \oplus A \oplus \ldots \oplus A = 0$ ê Si on a un nombre pair de variables.
- $A \oplus A \oplus A \oplus \ldots \oplus A = A$ ê Si on a un nombre impair de variables.
- $A \oplus B \oplus C \oplus \ldots \oplus Z = 0$ ê Si on a un nombre pair de variables à 1.
- $A \oplus B \oplus C \oplus \ldots \oplus Z = 1$ ê Si on a un nombre impair de variables à 0.

- Applications :
  - Générateur de parité.
  - C'est un moyen de détecter si une donnée contenue dans la mémoire d'un PC, par exemple, a été altérée (on sauvegarde en plus de la donnée le bit de parité de la donnée).
  - C'est un moyen de détecter si une donnée transmise entre 2 PC par une liaison série, par exemple, a été corrompue (on transmet en plus de la donnée son bit de parité et le récepteur calcul le bit de parité et le compare à celui reçu).
  - C'est un moyen simple et rapide (matériel) de vérifier des erreurs par contre, un nombre pair d'erreurs ne sera pas détecté.

95

Théorèmes fondamentaux

- $A \cdot (A + B) = A$
  - $A \cdot (A + B) = A \cdot A + A \cdot B = A + A \cdot B = A (1 + B) = A$

- $(A + B) \cdot (A + \overline{B}) = A$
  - $(A + B) \cdot (A + \overline{B}) = A \cdot A + A \cdot \overline{B} + B \cdot A + B \cdot \overline{B}$
    $= A + A \cdot \overline{B} + B \cdot A$
    $= A \cdot (1 + \overline{B} + B)$
    $= A$
- $\overline{\overline{A}} = A$
  - Évident.

96

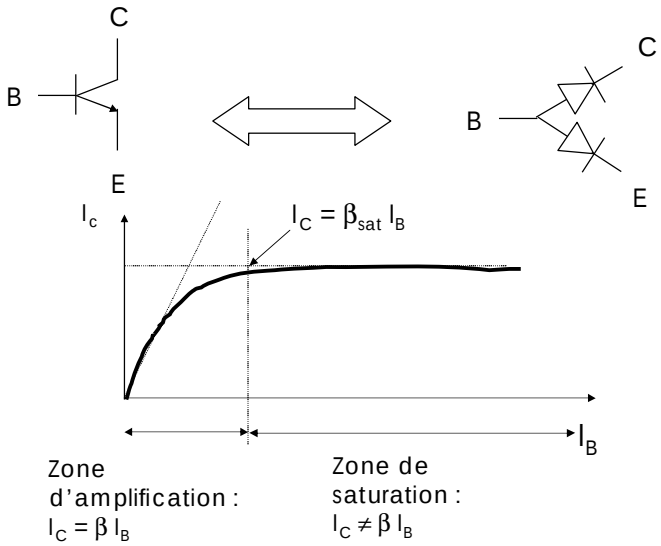
Propriétés du OU Exclusif

- Commutativité :
  - $A \oplus B = B \oplus A$
- Associativité :
  - $A \oplus (B \oplus C) = (A \oplus B) \oplus C = (A \oplus C) \oplus B$
- Distributivité :
  - $A . (B \oplus C) = A . B \oplus A . C$

PORTES

- La porte est l'élément de base de la logique numérique.
- Les fonctions logiques sont implantées par l'interconnexion de portes.
- Une porte est un circuit électronique qui produit un signal de sortie sous forme d'une opération booléenne sur ses signaux d'entrée.
- Si on est capable de trouver un circuit électrique pour chacune des fonctions de base de l'algèbre de boole (ET, OU, NON)
  - On sera capable d'obtenir un circuit électrique pour n'importe quelle fonction booléenne.
  - On pourra créer un ordinateur !

Rappel sur le transistor bipolaire



99

Rappel sur le transistor bipolaire

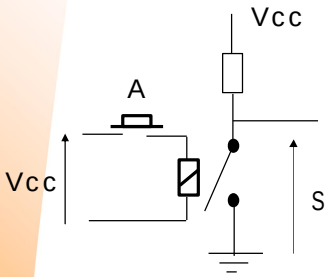
Ø En régime d'amplification : Cas de l'électronique analogique
- q $I_C = \beta I_B$
- q $I_E \approx I_C$

Ø En régime de saturation : Cas de l'électronique numérique
- q I_C et I_B peuvent être du même ordre de grandeur.
- q $I_E = I_C + I_B$
- q On ne peut plus négliger systématiquement I_B .
- q Lorsque le transistor est saturé :
  - q $V_{BE} \approx 0.7V$
  - q $V_{CE} \approx 0.2V$
  - q $V_{BC} \approx 0.7V$

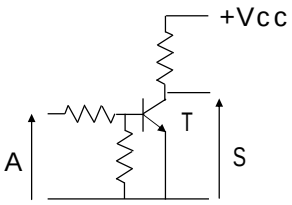
100

Inversion

Schémas classiques



BPA	Relais (contact)	S

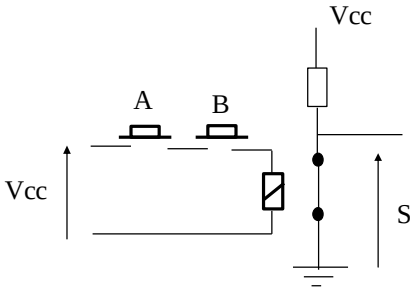


A	T	S

101

ET

Schéma classique

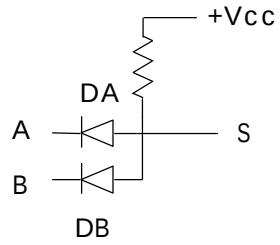


BPA	BPB	Relais (contact)	S

102

ET

Autre schéma classique

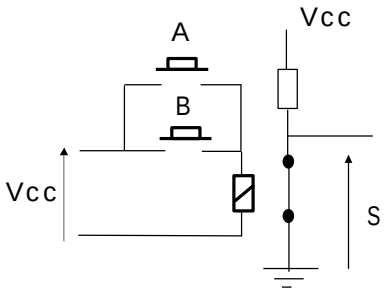


A	B	DA	DB	S

103

OU

Schéma classique

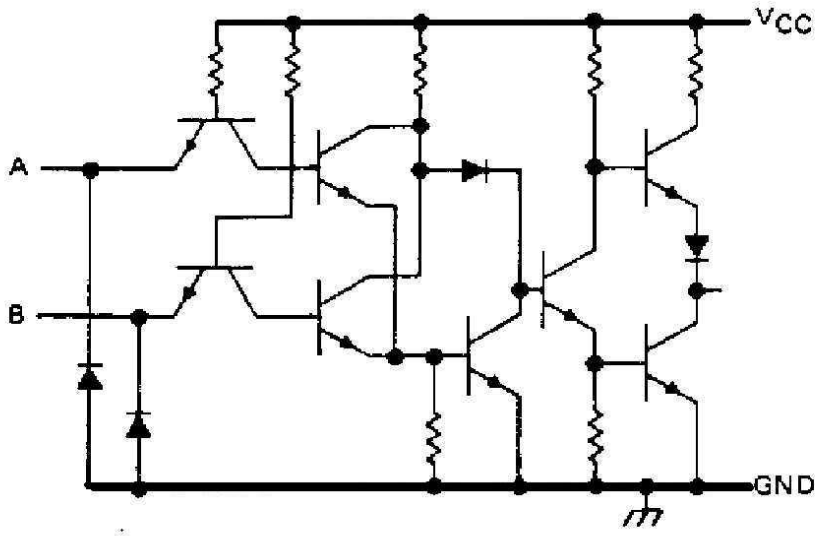


BPA	BPB	Relais (contact)	S

104

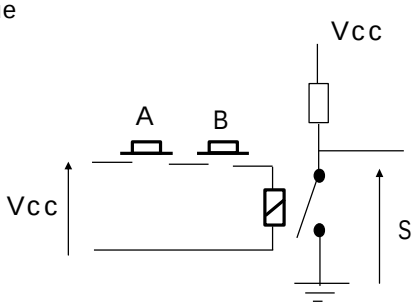
OU

Autre schéma classique



NON ET

Schéma classique

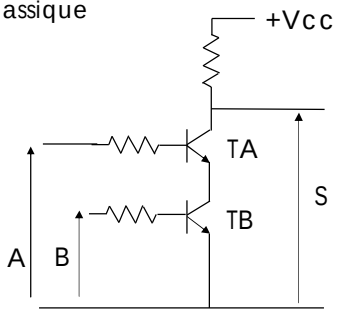


BPA	BPB	Relais (contact)	S

106

NON ET

Autre schéma classique

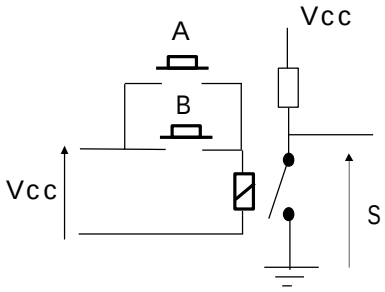


A	B	TA	TB	S

107

NON OU

Schéma classique

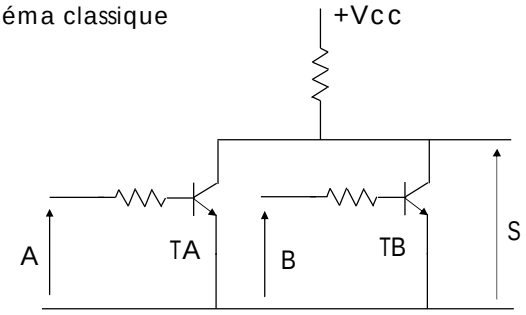


BPA	BPB	Relais (contact)	S

108

NON OU

Autre schéma classique

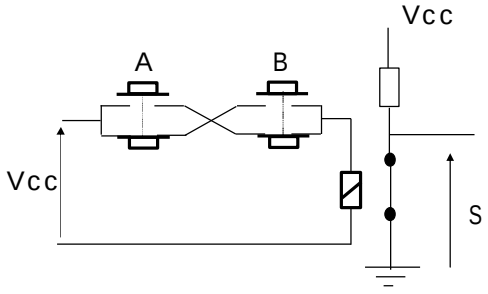


A	B	TA	TB	S

109

OU exclusif

Schéma classique



BPA	BPB	Relais (contact)	S

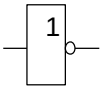
110

Portes

Ø Pour simplifier les schémas on introduit une représentation symbolique des différentes portes.

Inversion

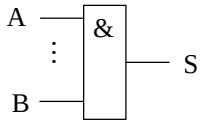
Ø Norme internationale CEI/IEC 60617-12 : 1997



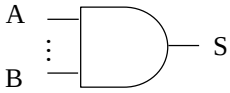
Ø Norme IEEE91

ET

Ø Norme CEI



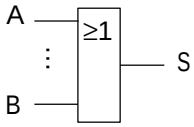
Ø Norme IEEE91



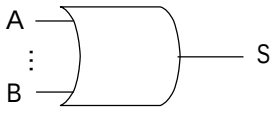
113

OU

Ø Norme CEI



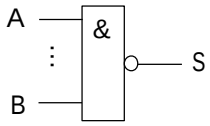
Ø Norme IEEE91



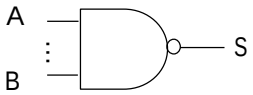
114

NON ET

- Norme CEI



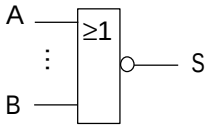
- Norme IEEE91



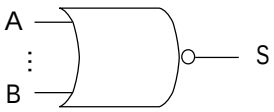
115

NON OU

- Norme CEI



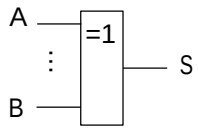
- Norme IEEE91



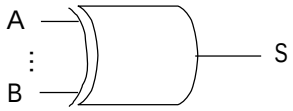
116

OU Exclusif

Ø Norme CEI



Ø Norme IEEE91



117

PORTES

Ø A mesure que le niveau d'intégration augmente les portes n'ont plus étaient réalisée avec des composants discrets mais à partir de circuits intégrés qui utilisaient l'intégration à petite échelle (SSI : Small Scale Integration : intégration à petite échelle).

Ø Lorsqu'une puce intègre plus de 1000 composants on parle de circuit LSI (Large Scale Integration : intégration à grande échelle).

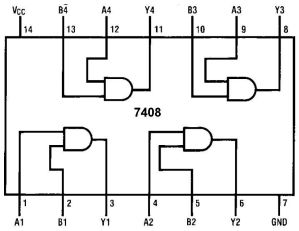
Ø Lorsqu'une puce intègre plus de 10000 composants on parle de circuit VLSI (Very Large Scale Integration : intégration à très grande échelle).

Ø La technologie des circuits intégrés a permis de créer les premiers processeurs.

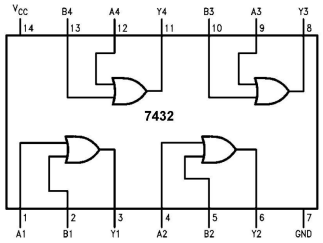
118

Exemple de circuits intégrés

Ø Porte ET (7408)



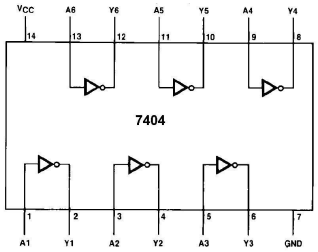
Ø Porte OU (7432)



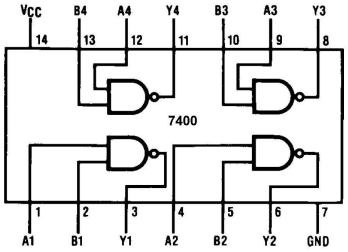
119

Exemple de circuits intégrés

Ø Inverseur (7404)



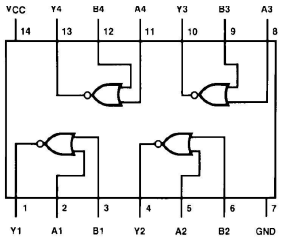
Ø Porte NON ET (7400)



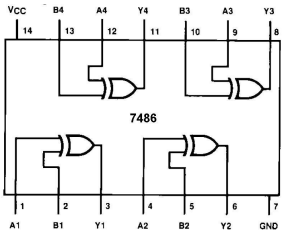
120

Exemple de circuits intégrés

Ø Porte NON OU (7402)



Ø Porte Ou exclusif (7486)



121

PORTES

Ø Généralement on n'emploie pas tous les types de portes dans un design :
- q Réduction des coûts.
- q Simplicité de maintenance.

Ø Il suffit, en effet, d'utiliser un jeu de portes fonctionnelles complet :
- q ET, OU, NON.
- q ET, NON.
- q OU, NON.
- q NON ET.
- q NON OU.

122

PORTES

Ø Exemple avec un circuit NON OU
Table de vérité.

A	B	$\overline{A + B}$	$\overline{A}$	A . B	A + B
0	0	1	1	0	0
0	1	0	-	0	1
1	0	0	-	0	1
1	1	0	0	1	1

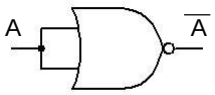
Ø Inverseur : Évident
Ø OU : Évident
Ø ET : De Morgan

$A . B = \overline{\overline{A} . \overline{B}} = \overline{\overline{A} + \overline{B}}$

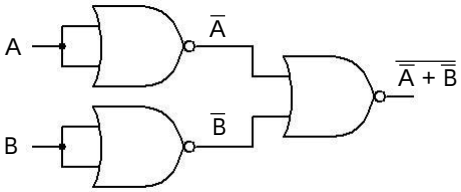
123

PORTES

Ø Inverseur



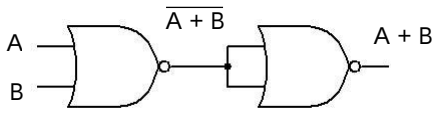
Ø ET



124

PORTES

Ø OU



125

PORTES

Ø Néanmoins, chaque fonction logique de base donne lieu à plusieurs circuits en fonction :
- q Du nombre d'entrées.
- q Du type d'entrée (présence ou non d'un trigger).
- q Du type de sortie (collecteur ouvert ou non, drain ouvert ou non).
- q Du type de technologie (AC, ACT, AHC, AHCT, ALS, ALVC, AS, AVC, AVP, CD4K, F, HC, HCT, LS, LV, LVC, S TTL).

Ø Si on regarde chez Texas Instruments on dénombre pas moins de :
- q 48 portes ET.
- q 103 portes NON ET.
- q 33 portes OU.
- q 5 portes "universelles" ET, OU, Inverseuse.
- q 33 portes OU.
- q 42 portes NON OU.
- q 27 portes OU exclusif
- q 65 portes inverseuses.

126

PORTES

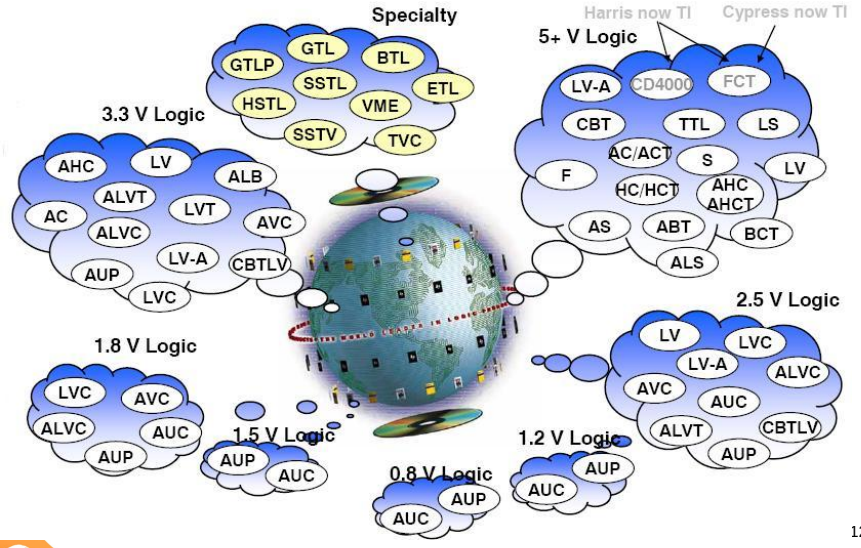
Ø Exemple : Différentes portes ET chez Texas Instruments.

AND Gates

DESCRIPTION	OUTPUT	TYPE	TECHNOLOGY															
			AC	ACT	AHC	AHCT	ALS	ALVC	AS	AUC	AUP	CD4K	F	HC	HCT	LS	LV	LVC	S
Single 2 Input		1G08			✔	✔				✔	✔							✔	
Dual 2 Input		2G08								✔								✔	
Quad 2 Input		08	✔•	✔•	✔	✔	✔	✔	✔		✔		✔	✔	✔	✔	✔	✔	✔
	CP	11008	✔	✔															
	OC	09					✔									✔			✔
		4081										✔							
Quad 2-Input Buffers/Drivers		1008							✔										
Quad 2 Input with Schmitt-Trigger Inputs		7001												✔					
Dual 4 Input		21					✔		✔				✔	✔	✔	✔	✔		
		4082										✔							
Triple 3 Input		11	✔•	✔			✔		✔				✔	✔	✔	✔	✔		
		4073										✔							

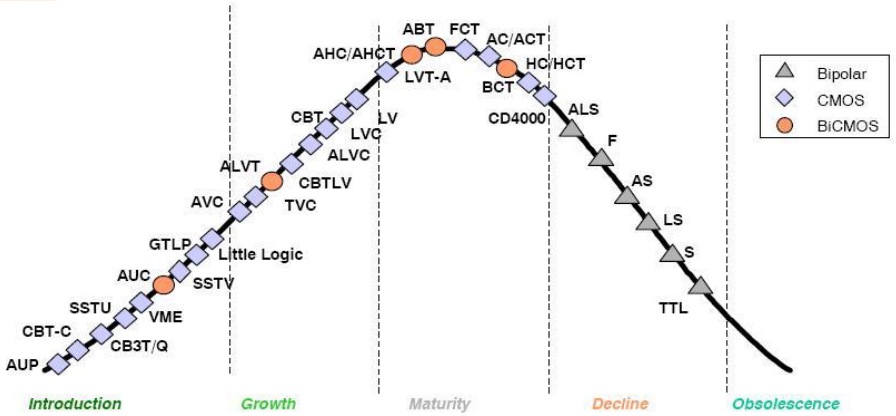
127

Les différentes technologies offertes par TI

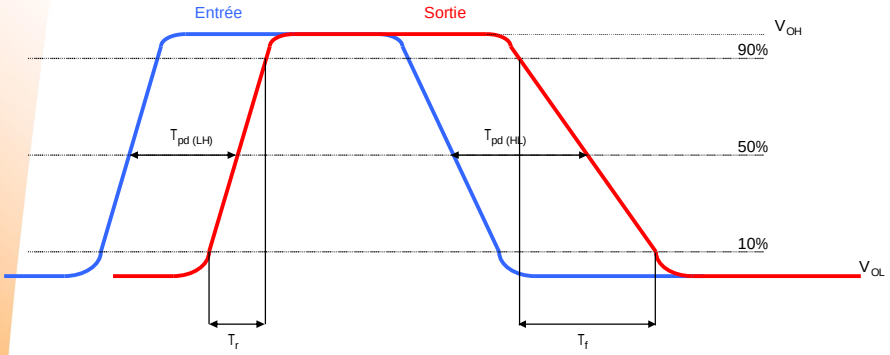


128

Cycle de vie des différentes technologies



Définitions de base



- Temps de propagation : t_{pd}.
- Temps de montée : t_r (rise time).
- Temps de descente : t_f (fall time).
- Tension de sortie à l'état haut : V_{OH}.
- Tension de sortie à l'état bas : V_{OL}.

Définitions de base

Ø Lorsque l'on parle de vitesse il faut avoir 2 notions en tête :
- q Le temps de propagation à l'intérieur de la porte
  - v Plus t_{pd} est petit plus le circuit est rapide.
  - v $t_{pd(LH)}$: temps de propagation pour passer d'un niveau bas à un niveau haut.
  - v $t_{pd(HL)}$: temps de propagation pour passer d'un niveau haut à un niveau bas.
  - v Ces temps sont mesurés en prenant comme instant d'origine et de fin le moment où le signal passe par un potentiel fixé par le constructeur (en général la moitié de l'amplitude de sortie).
  - v Attention la capacité de la charge (C_L) en sortie modifie les temps de propagation (plus C_L est élevée plus le temps de propagation augmente).

Ø On peut définir un temps de propagation moyen :

$$T_{pd}\,(\text{average}) = \frac{1}{2}\,(t_{pd(LH)} + t_{pd(HL)})$$

- v En général $t_{pd(LH)}$ $t_{pd(HL)}$

131

Définitions de base

- q Le slew rate
  - v Le temps de montée et le temps de descente sont mesurés entre 10% et 90% de l'amplitude du signal de sortie.
  - v Slew rate : $\frac{dV}{dt} = \frac{(V_{OH}-V_{OL}) \times 80\%}{t_r \text{ (ou } t_f)}$
  - v Plus le temps de montée ou de descente est faible plus le *slew rate* est important.
  - v Plus le *slew rate* est important plus la technologie est bruyante.

Ø Le temps de propagation et le *slew rate* ne sont pas nécessairement des grandeurs proportionnelles.

132

Définitions de base

Ø Sortance (*fan out*)

q Lors des associations de circuits, la somme des courants d'entrée (I_i) ne doit pas être supérieure au courant de sortie (I_o) du circuit qui la commande.

q La sortance est le nombre maximal de charge qu'une sortie peut commander.

q Elle s'exprime en unité de charge U. L., la référence étant la porte TTL standard.

q Sortance à l'état bas :

$$S = \frac{I_{OL\,min}}{I_{IL\,max}} = \frac{I_{OL\,min}}{1{,}6\ mA} \quad \text{en U. L.}$$

q Sortance à l'état haut :

$$S = \frac{I_{OH\,max}}{I_{IH\,max}} = \frac{I_{OH\,max}}{40\mu A} \quad \text{en U. L.}$$

Définitions de base

Ø Entrance (*fan in*)

q C'est le nombre de charges que présente un circuit en entrée pour une sortie l'alimentant.

q Elle s'exprime en U. L. avec la porte TTL standard comme référence.

q Entrance à l'état bas :

$$E = \frac{I_{IL\,max}}{I_{IL\,max}} = \frac{I_{IL\,max}}{1{,}6\ mA} \quad \text{en U. L.}$$

q Entrance à l'état haut :

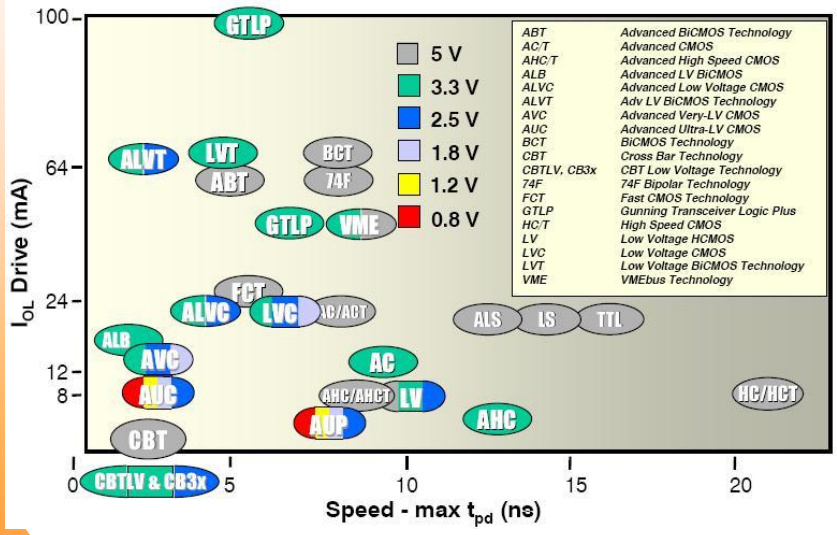
$$E = \frac{I_{IH\,max}}{I_{IH\,max}} = \frac{I_{IH\,max}}{40\mu A} \quad \text{en U. L.}$$

Choix d'une famille logique

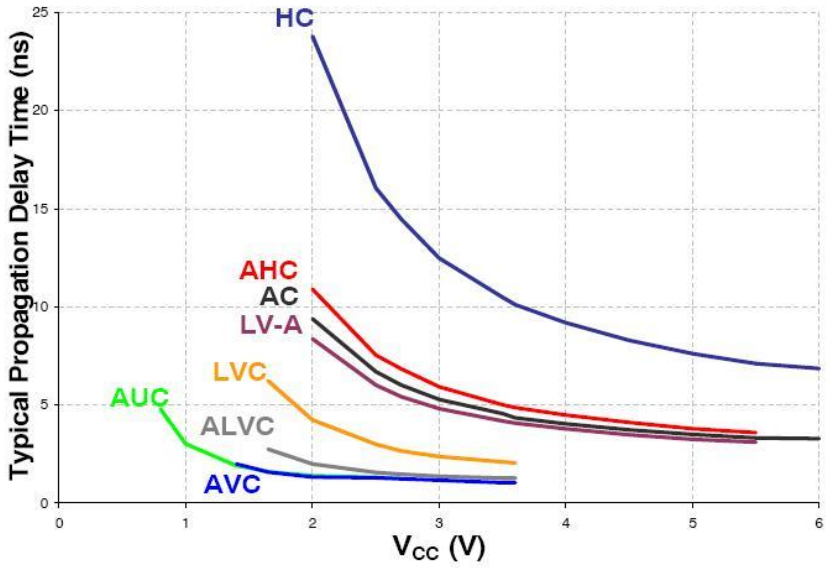
- Le concepteur doit faire attention à :
  - la vitesse.
  - La tension d'alimentation.
  - La possibilité d'avoir un courant de sortie suffisant.
  - La consommation.
  - La facilité d'utilisation (on considère qu'une technologie peu sensible au bruit est plus facile à mettre en œuvre).
  - Le type de boîtier (peut poser des problèmes de réalisation pour des grandes séries).

- En général il faut faire un compromis comme par exemple :
  - Courant de sortie en fonction du temps de propagation.
  - Le temps de propagation en fonction de la tension d'alimentation.

Courant de sortie en fonction du temps de propagation

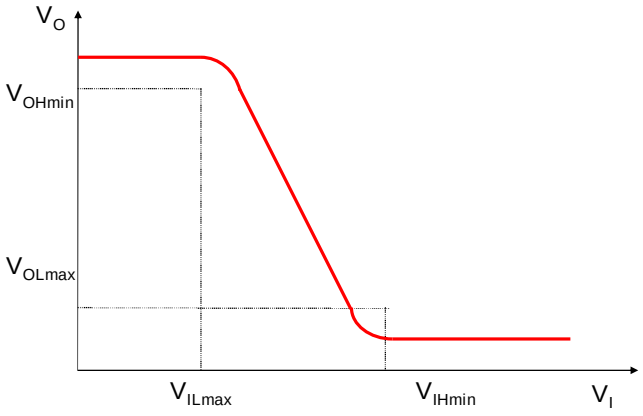


Temps de propagation en fonction de la tension d'alimentation



Fonction de transfert d'une porte inverseuse

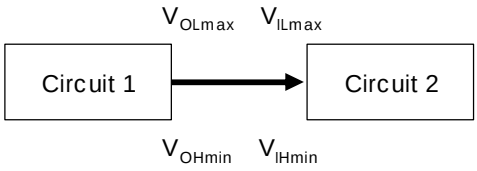
Ø Pour mixer différentes technologies sur une même carte il faut bien prendre en compte les niveaux de fonctionnement.



138

Fonction de transfert d'une porte inverseuse

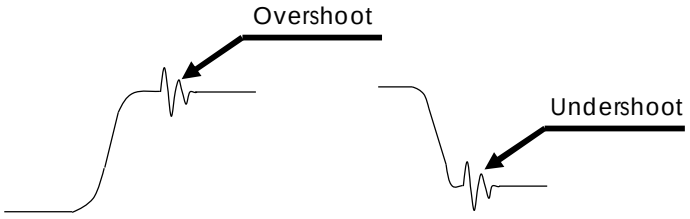
- Fonction de transfert d'une porte inverseuse :
  - Tant que la tension d'entrée au niveau bas est inférieure à un seuil (V_{ILmax}) la tension de sortie est supérieure à un seuil (V_{OHmin}).
  - Dés que la tension d'entrée devient supérieure à un seuil V_{IHmin} la tension de sortie passe à l'état bas et est inférieure à un seuil (V_{OLmax}).
  - Le fonctionnement normal de la porte est obtenu pour :
    - Une tension d'entrée inférieure à V_{ILmax}.
    - Une tension d'entrée supérieure à V_{IHmin}.
  - En réponse on a :
    - Une tension de sortie supérieure à V_{OHmin}.
    - Une tension de sortie inférieure à V_{OLmax}.
- Pour un fonctionnement correct il faut que les niveaux de sortie du circuit 1 soient compatibles avec les niveaux d'entrées du circuit 2.



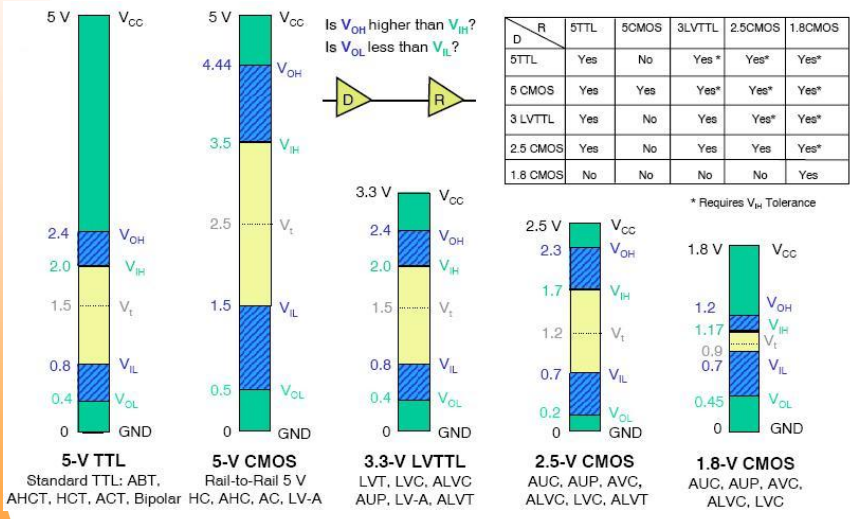
Fonction de transfert d'une porte inverseuse

- Marge de bruit :
  - Niveau haut : $V_{OHmin} - V_{IHmin}$
  - Niveau bas : $V_{ILmax} - V_{OLmax}$

- Il est souhaitable d'avoir :
  - La marge de bruit la plus large possible.
    - Grande immunité aux parasites (*overshoot* et *undershoot*)
  - La région d'incertitude la plus faible possible.
    - $V_{IHmin} - V_{ILmax}$
    - Lorsque la tension d'entrée se situe dans la zone d'incertitude on ne peut pas savoir comment va réagir la sortie.



Compatibilité d'association de différentes technologies

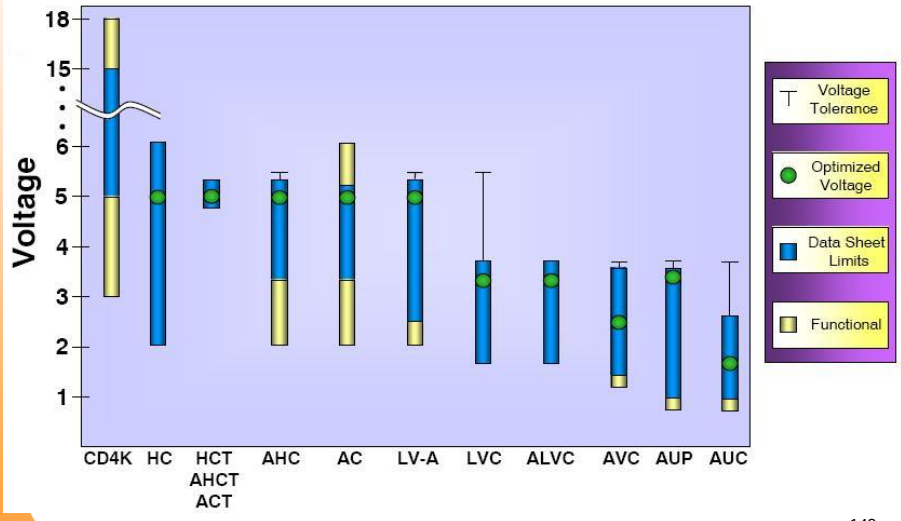


D \ R	5TTL	5CMOS	3LVTTL	2.5CMOS	1.8CMOS
5TTL	Yes	No	Yes *	Yes*	Yes*
5 CMOS	Yes	Yes	Yes*	Yes*	Yes*
3 LVTTL	Yes	No	Yes	Yes*	Yes*
2.5 CMOS	Yes	No	Yes	Yes	Yes*
1.8 CMOS	No	No	No	No	Yes

* Requires V_{IH} Tolerance

141

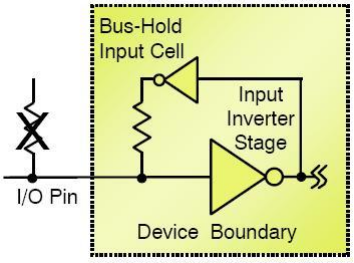
Tension de fonctionnement de divers technologies



142

Maintien de l'entrée

Ø Maintien le dernier état d'entrée connu. Évite les entrées flottantes.
Ø Élimine la nécessité d'une résistance sur les entrées non utilisées ou les broches d'entrée/sortie flottantes.
Ø Augmentation négligeable de la puissance consommée par le système.



Ø Principales familles bénéficiant de cette technologie :
  q ABT, ALVC, ALVT, AVC, AUC, FCT, GTL, GTLP, LVC, LVT, VME.

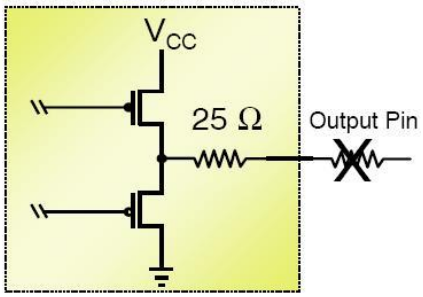
143

Résistance série d'amortissement

Ø Améliore l'intégrité des signaux.

Ø Fournie une meilleure adaptation d'impédance.

Ø Élimine la nécessité d'une résistance série externe.



Ø Principales familles bénéficiant de cette technologie :
  q ABT, ALVC, ALVT, F, GTLP, LVC, LVT, VME.

144

Exemple : Technologie sur le déclin

Arithmetic Logic Units

DESCRIPTION	TYPE	TECHNOLOGY		
		AS	LS	S
Arithmetic Logic Units/Function Generators	181	✔	✔	
	381			✔
Look-Ahead Carry Generators	182			✔

Comparators (identity)

DESCRIPTION	OUTPUT	TYPE	TECHNOLOGY	
			ALS	F
8 Bit Identity (P = Q)		521	✔	✔
8 Bit Identity (P = Q) with Input Pullup Resistors	OC	518	✔	
8 Bit Identity (P = Q) with Input Pullup Resistors		520	✔	✔
12 Bit Address		679	✔	

Comparators (magnitude)

DESCRIPTION	TYPE	TECHNOLOGY						
		ALS	AS	CD4K	HC	HCT	LS	S
4 Bit	85				✔	✔	✔	✔
	4063			✔				
	4585			✔				
8 Bit	682				✔		✔	
	684				✔		✔	
	688	✔			✔	✔	✔	
	885		✔					

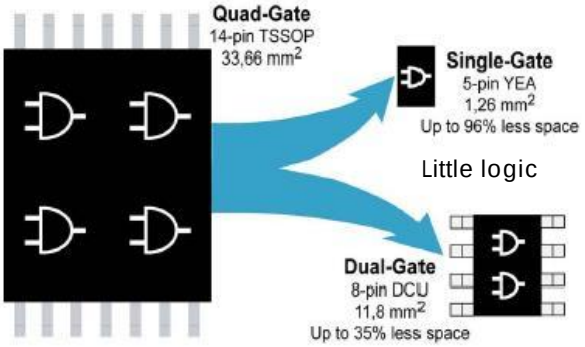
145

Exemple : Technologies récentes

DEVICE	NO. PINS	DESCRIPTION	AVAILABILITY		
			DSBGA	SOP	SOT
SN74AUP1G02	5	Low-Power Single 2-Input NOR Gates		✔	✔
SN74AUP1G04	5	Low-Power Single Inverter Gates		✔	✔
SN74AUP1G08	5	Low-Power Single 2-Input Positive-AND Gates	✔	✔	✔
SN74AUP1G14	5	Low-Power Single Schmitt-Trigger Inverters		✔	✔
SN74AUP1G17	5	Low-Power Single Schmitt-Trigger Buffers		✔	✔
SN74AUP1G32	5	Low-Power Single 2-Input OR Gates		✔	✔
SN74AUP1G57	6	Low-Power Configurable Multiple-Function Gates	✔	✔	✔
SN74AUP1G58	6	Low-Power Configurable Multiple-Function Gates	✔	✔	✔
SN74AUP1G80	5	Low-Power Single Positive-Edge-Triggered D-Type Flip-Flops		✔	✔
SN74AUP1G97	6	Low-Power Configurable Multiple-Function Gates	✔	✔	✔
SN74AUP1G98	6	Low-Power Configurable Multiple-Function Gates	✔	✔	✔

146

Pourquoi des portes uniques (little logic) ?



Logique standard

Ø Rajouter une porte pour modifier une erreur de design sur un circuit ASIC.

147

Historique

Ø Lorsque Philips a introduit les PLAs sur le marché au début des années 1970 leur principaux inconvénients étaient :
- q Le prix.
- q Les vitesse relativement faibles.

Ø Ces deux inconvénients majeurs étaient liés au fait que la matrice ET et la matrice OU étaient programmables.

Ø Pour pallier à ces inconvénients MMI (aujourd'hui AMD) a développé les PALs. Ces circuits disposés d'une matrice ET programmable mais d'une matrice OU fixe.

148

Historique

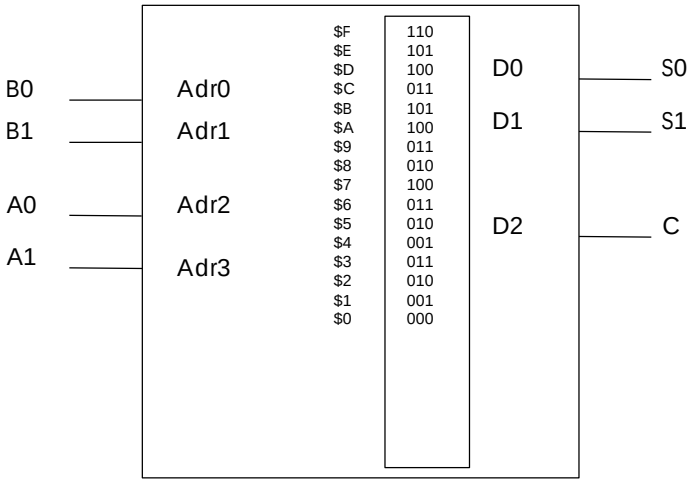
Ø Le premier circuit programmable par l'utilisateur est la mémoire de type ROM (Read Only Memory). Le principe est très simple :
- q Les lignes d'adresses : représentent les entrées de la fonction logique.
- q Les cellules mémoire : contiennent toutes les valeurs de la fonction de sortie.
- q Chaque ligne de donnée : représente la sortie de la fonction logique.

Ø Exemple : additionneur

Ø On a 4 entrées => 4 bits d'adresse.
=> mémoire de $2^4 = 16$ mots.

Ø On a 3 sorties => La taille des mots doit être de 3 bits.

Historique



Historique

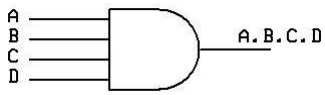
Ø La méthode est très simple mais très fastidieuse lorsque le nombre d'entrées augmente. De plus, les tailles mémoires disponibles font que la plupart des entrées et des sorties ne sont pas utilisées.

Ø Néanmoins, aujourd'hui, c'est une méthode très souvent utilisée dans les FPGA pour coder une fonction quelconque. Ces blocs mémoire portent le nom de LUT (Look Up Table) et sont programmés de façon transparente, pour l'utilisateur, à patir de la forme canonique.

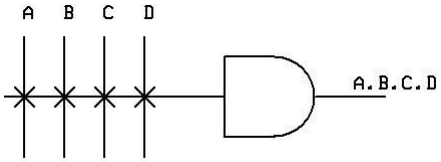
Représentation schématique

Ø Pour simplifier la structure interne des PALs on adopte souvent la représentation simplifiée suivante :

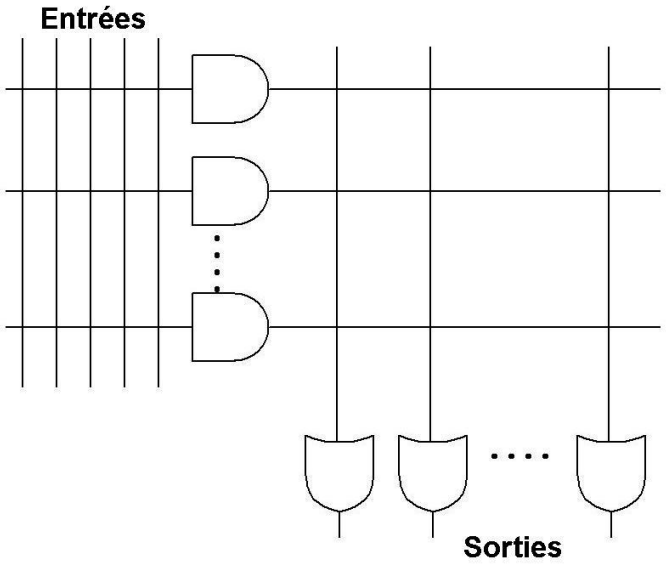
ET a 4 entrées



Représentation simplifiée d'un ET a 4 entrées

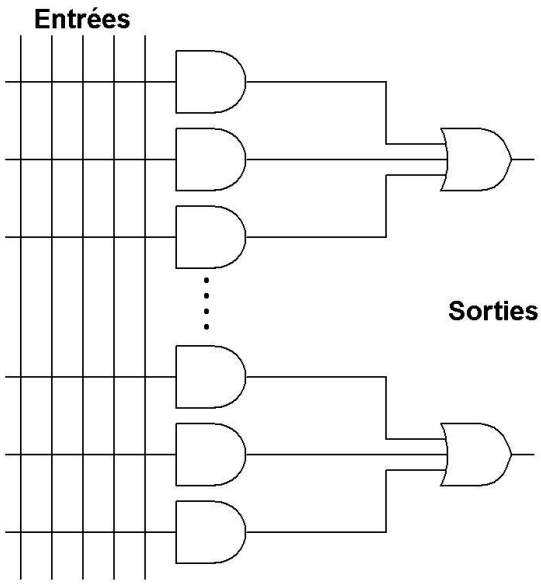


Architecture d'un PLA



153

Architecture d'un PAL



154

Exemple :

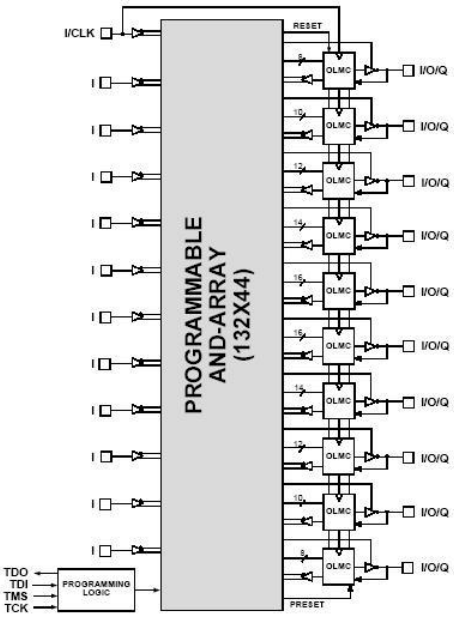
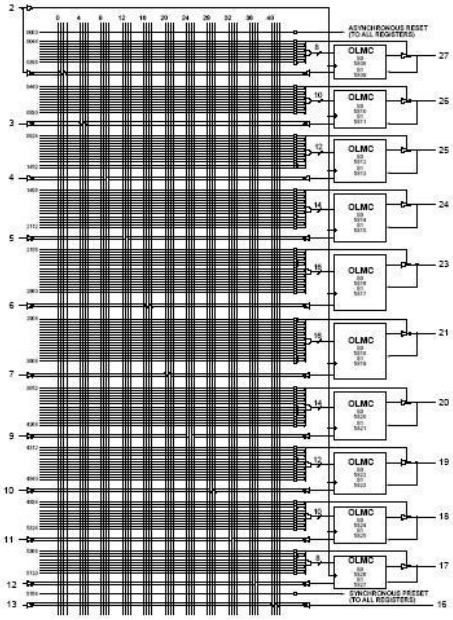


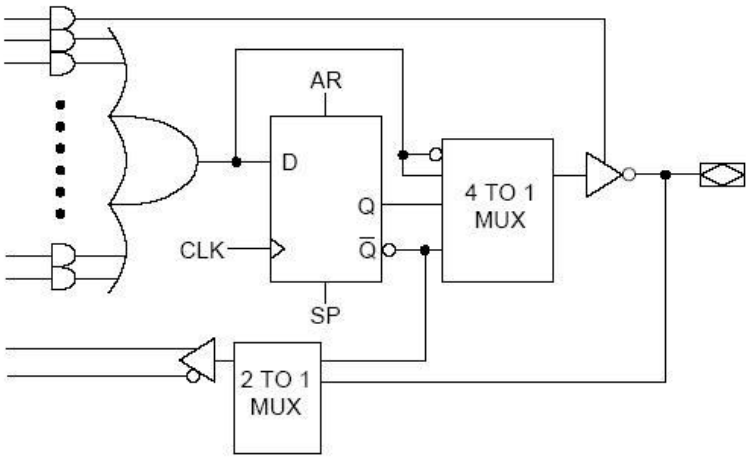
Schéma bloc d'un ispGAL 22LV10

Exemple :



Matrice d'interconnexion d'un ispGAL 22LV10

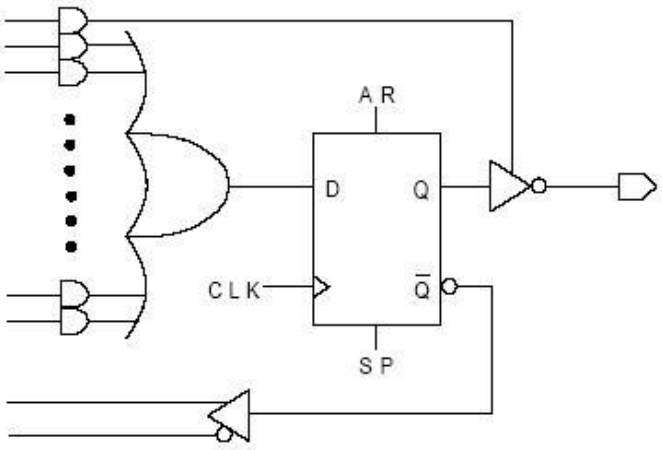
Exemple :



Structure d'un OLMC (Output Logic MacroCell) d'un ispGAL 22LV10

157

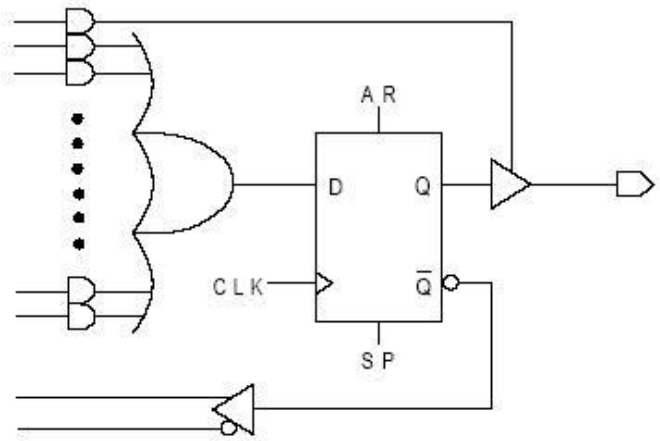
Exemple :



Structure d'un OLMC d'un ispGAL 22LV10 : Mode registre

158

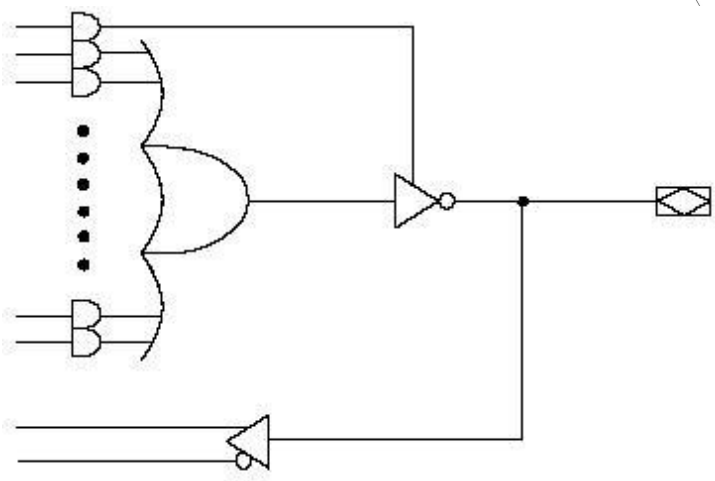
Exemple :



Structure d'un OLMC d'un ispGAL 22LV10 : Mode registre

159

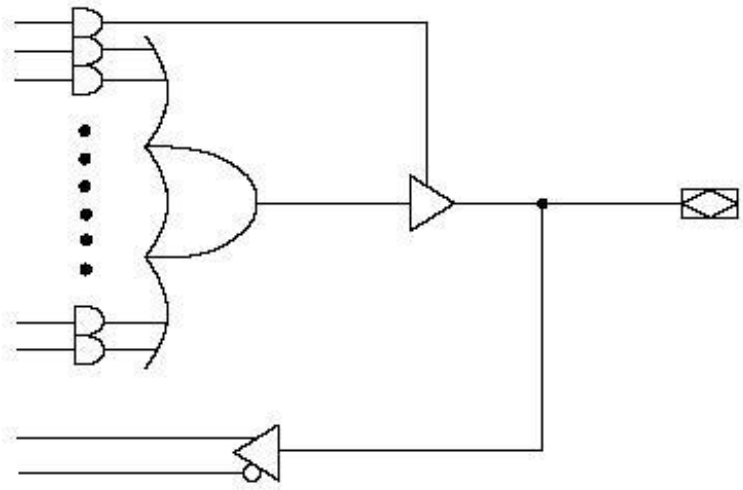
Exemple :



Structure d'un OLMC d'un ispGAL 22LV10 : Mode combinatoire

160

Exemple :



Structure d'un OLMC d'un ispGAL 22LV10 : Mode combinatoire

161

PAL

Ø L'introduction des PALs a profondément révolutionné la manière de concevoir des systèmes numériques avec parallèlement :
Ø L'évolution des outils logiciel :
- q Forme canonique.
- q Schématique.
- q Langage de haut niveau : VHDL (Very Hight Speed Integrated Circuit, Hardware description Language).

Ø L'évolution des outils matériel :
- q Programmateur de circuits.
- q Programmation *in situ*.

Ø L'évolution des technologies :
- q Quelques centaines de portes (PAL).
- q Quelques milliers de portes (CPLD).
- q Plusieurs millions de portes (FPGA).

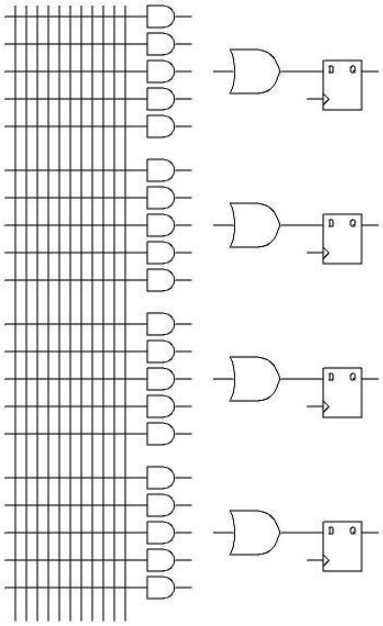
Ø La diminution des coûts :
- q Du logiciel.
- q Du matériel.

162

PAL

- Les PALs fournissent environ 50 fois plus de portes dans un même composant qu'un circuit logique standard.
- Les CPLDs étendent la densité des SPLDs.
- Le concept est d'avoir quelques PLDs dans un seul circuit avec une matrice d'interconnexion.
- Les temps de propagation entrées/sorties sont déterministes.
- La fréquence de travail peut atteindre 200MHz.
- Les coûts de développement sont faibles :
  - Les CPLDs sont reprogrammables il est donc facile et très peu chère de changer ou modifier un design.
  - Les outils de développement sont très souvent gratuits.
- La taille des cartes est réduite :
  - Les CPLDs offrent un haut niveau d'intégration (grand nombre de portes sur une petite surface).
- Maintenance :
  - Avec les PLDs il suffit de maintenir et donc de stocker un seul composant.

Architecture d'un CPLD

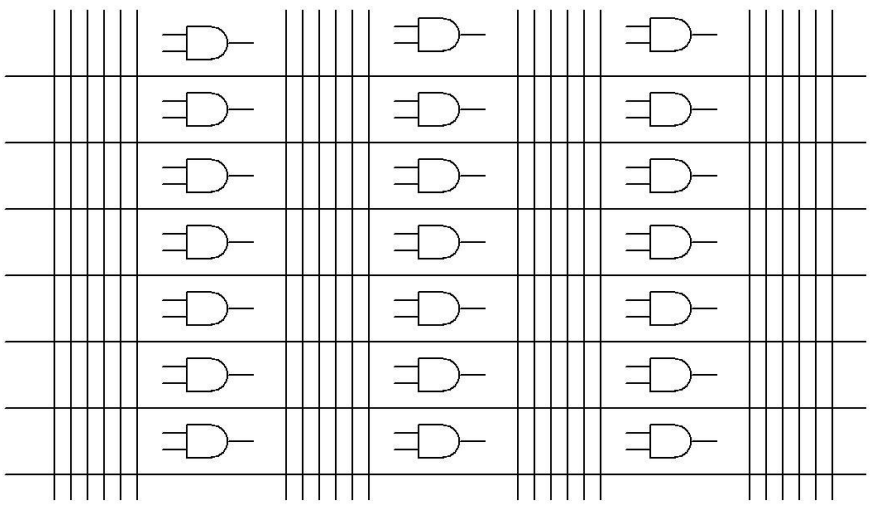


FPGA

- En 1985, Xilinx introduit sur le marché un nouveau type de PLD le FPGA.
- Un FPGA comporend :
  - Des cellules logiques identiques.
  - Un réseau d'interconnexion total.
- Il existe deux types de FPGA :
  - Les FPGAs basés sur des SRAMs : ils peuvent être reprogrammés autant de fois que l'on veut. En faite ils doivent être reprogrammés chaque fois que l'alimentation est coupée. Une EPROM série ou un cordon JTAG permet de recharger l'architecture du système. Pour ces FPGAs :
    - Une SRAM détermine les interconnexions.
    - Des SRAMS définissent la logique dans des LUT.
  - Les FPGAs basés sur des OTPs : Ils sont programmable une seule fois. Pour ces FPGAs :
    - Les interconnexions sont vbasées sur une technologie anti-fusible.
    - La logique est basée sur des portes standards.

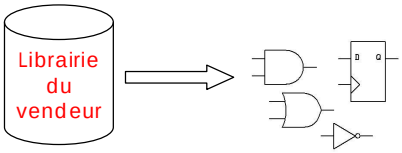
165

Architecture d'un FPGA

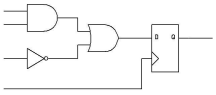


166

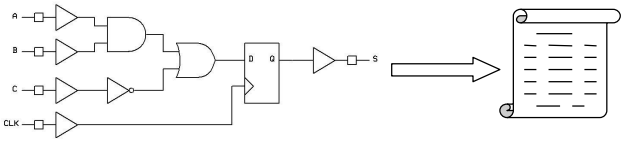
Conception d'un circuit à partir d'un schéma



Choisir des portes



Interconnecter les portes avec des « nets »



Ajouter des buffers d'entrées sorties

Générer une netlist

167

Conception d'un circuit à partir d'un schéma

Ø La netlist est un fichier texte qui décrit le schéma. Elle est générée automatiquement par l'outil de développement.

Ø La netlist est un moyen compacte, pour les autres programmes de comprendre :

- q Quelles portes sont utilisées.
- q Comment les interconnecter.
- q Quel est le nom des entrées/sorties.

Ø Le format EDIF est un format de netlist très répendu dans l'industrie.

Ø Il existe d'autre format comme par exemple le format XNF (Xilinx Netlist Format) utilisé par Xilinx.

168

Conception d'un circuit à partir d'un schéma

Ø L'outil de saisie de schéma trouve rapidement ces limites. Par exemple si l'on veut réaliser un multiplieur de deux nombres de 16 bits. Il faut :
- q 16 bits d'entrée pour l'opérande A.
- q 16 bits d'entrée pour l'opérande B.
- q 32 bits de sortie pour le résultat.
- q Soit un total de 64 entrées/sorties.
- q Environ 6000 portes logiques.
- q Sur une page de schématique on place classiquement 200 portes logiques.
- q Soit environ 30 pages de schématique à saisir.
- q A noter que si l'on travaille avec des portes logiques il faudrait en plus les câbler.

Ø Un outil de saisie de schéma est forcément lié aux outils du vendeur. Si l'on change de fournisseur de composants il faut refaire le schéma.

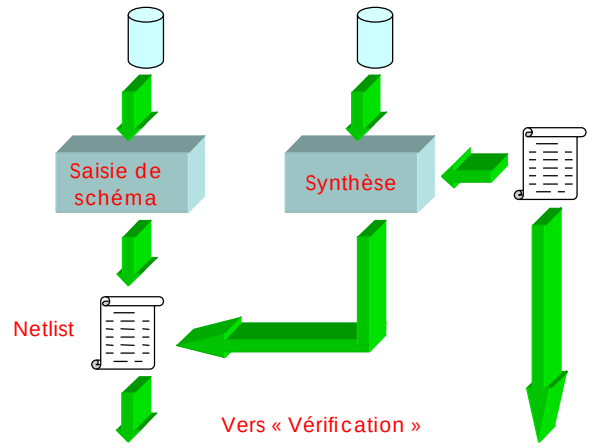
Conception d'un circuit à partir d'un schéma

Ø Pour se genre de design il est préférable d'utiliser un langage de description de haut niveau.

Ø Si l'on souhaite maintenant réaliser un multiplieur de deux nombres de 32 bits il faut tout refaire et un tel design représente environ 90 pages de schématique. Avec un langage de haut niveau il faut simplement changer 2 constantes dans un fichier texte...

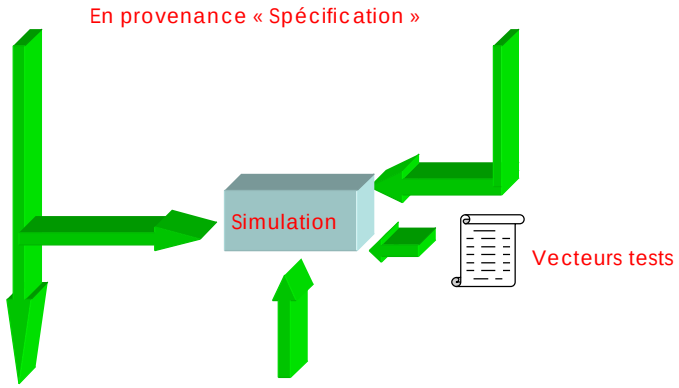
Ø Il y a principalement deux langages de description de haut niveau :
- q Le VHDL.
- q Le Verilog.

Flot de conception : Spécification



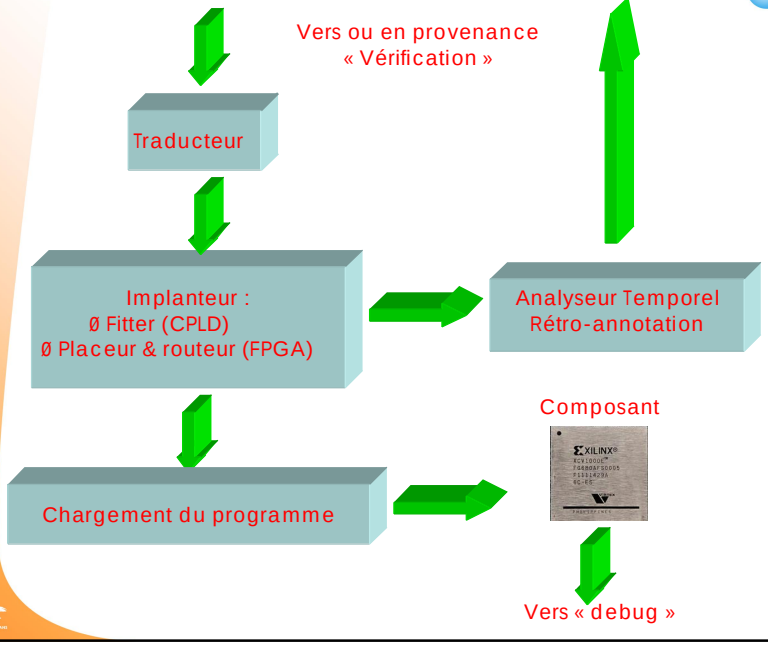
171

Flot de conception : Vérification



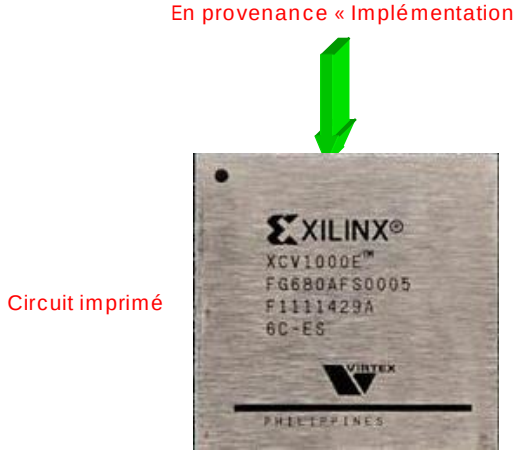
172

Flot de conception : Implémentation



173

Flot de conception : Debug du système



174

Flot de conception

Ø Un outil de synthèse permet d'obtenir une netlist à partir d'un fichier texte écrit en langage de description de haut niveau.

Ø Les conceptions en logique programmable sont vérifiées en utilisant un simulateur. C'est un programme qui permet de vérifier les fonctionnalités du système et d'analyser le comportement temporel d'un circuit.

Ø L'utilisation d'un langage de description matériel (HDL : Hardware Description Language) permet de voir le comportement du système sur des composants de différentes marques en utilisant simplement le synthétiseur des différents constructeurs.

Ø La simulation fonctionnelle vérifie simplement le fonctionnement logique du système sans tenir compte des temps de propagation à l'intérieur du circuit.

Flot de conception

Ø La simulation fonctionnelle s'effectue directement sur le fichier en langage HDL sans passer par l'étape de synthèse (gain de temps).

Ø Une fois que la simulation fonctionnelle donne les résultats attendus. Il faut passer à la synthèse et générer la netlist.

Ø La netlist décrit le système en utilisant les portes particulières contenues dans le circuit choisi.

Ø Une fois le système complètement vérifié on peut l'implanter dans le composant choisi.

Ø Le traducteur comprend plusieurs programmes qui vont transformer la netlist pour les étapes suivantes.

Flot de conception

Ø La phase de traduction comprend :
- q L'optimisation.
- q Le choix des portes réellement présentes dans le composant.
- q La vérification que le nombre de portes nécessaires n'excède pas le nombre réel de portes contenues dans le circuit.

Ø La phase de traduction donne lieu généralement a un rapport qui permet de conclure si le circuit choisi est bien adapté.

Ø La phase d'implantation dure plus ou moins longtemps suivant la nature de la fonction et le type de composant.

Ø Cette phase utilise un installateur (fitter) pour un CPLD.

Flot de conception

Ø La structure régulière d'un CPLD limite le nombre de degrés de liberté au moment de l'implantation. Ce qui n'est pas le cas pour un FPGA. De même, cette phase est rallongée lorsque l'on impose des contraintes :
- q Position des entrées/sorties sur les broches du circuit (*pin locking*).

=> Un mauvais choix du concepteur peut bloquer cette étape.

Ø La phase d'implantation utilise un placeur & router pour un FPGA.
- q Le placeur sélectionne les modules spécifiques ou les blocs logique.
- q Le routeur réalise l'interconnexion entre les différentes portes.

Flot de conception

Ø La plupart des outils de placement routage sont automatiques, l'utilisateur peut donc ne pas s'intéresser, en détail, à l'architecture interne du composant. Néanmoins, dans certains cas critiques, l'utilisateur peut lui même effectuer le placement et/ou le routage de façon manuelle. Ce mode nécessite de la part de l'utilisateur, une parfaite connaissance de l'architecture interne du composant sélectionné.

Ø L'étape de placement routage, en raison de sa complexité, est une étape relativement longue surtout si l'on rajoute des contraintes :

- Temps de propagation maximum (chemins critiques)

=> Une contrainte trop sévère peut bloquer cette étape.

Ø Cette étape fournit des informations relativement précise sur les temps de propagation.

Flot de conception

Ø Pour connaître, très précisément, le fonctionnement dynamique du circuit il est possible de fournir après cette étape des informations (rétro annotation) qui permettront au simulateur de faire une simulation temporelle proche de la réalité. Cette simulation prend en compte le temps passé à la fois dans les portes et au niveau des interconnexions.

Ø Le flot de bits (*bitstream*) peut maintenant être chargé à l'intérieur du composant. Il contient toutes les informations utiles pour définir la logique et les interconnexions entres les différentes portes.

Ø Comme les technologies basées sur des SRAMs perdent leur contenu dès que l'alimentation est coupée le flot de bits peut être sauvegardé dans une PROM série qui servira à recharger le composant programmable chaque fois que la carte est alimentée.

Flot de conception

Ø Pour les technologies basées sur des anti-fusibles le composant est programmé une fois pour toute grâce aux informations contenues dans le flot de bits.

Ø Lors de la phase de mise au point les technologies basées sur des SRAMs peuvent être chargées via un cordon de type JTAG.

Ø Une fois le comportement du composant simulé avant et après routage puis programmé, les causes de non fonctionnement sont principalement du a des spécifications incorrectes où a des parties non simulées.

VHDL Historique

Ø Au départ le langage VHDL n'a pas été conçu comme un langage de description matérielle.

Ø Cette lacune le rendait au départ impropre à la synthèse automatique et inapte à la simulation détaillée de composants réels.

Ø Aujourd'hui c'est un élément incontournable de la panoplie du concepteur de circuits logiques programmables.

Ø Le langage VHDL assure au moins en théorie :
- La pérennité
- La portabilité

des développements

Ø VHDL signifie VHSIC (Very High Speed Integrated Circuits) Harware Description Language.

VHDL Historique

Ø Ce langage est né sous l'impulsion du département de la défense américaine qui souhaitait un outil propre à décrire des cahiers des charges non ambigus.

Ø Le VHDL est donc né du besoin d'outils de spécification de haut niveau dans le but de simplifier le développement de projets très complexes par des équipes importantes.

Ø La publication de la première norme IEEE date de 1987.

Ø La révision fondamentale date de 1993. Elle conserve la compatibilité avec le standard originel de 1987, mais enrichit le langage de nouveaux opérateurs et de nouveaux types.

Ø A méditer :
q - « Adopter le langage VHDL n'est, en soi, absolument pas un gage de qualité de conception ».
q - « Un mauvais concepteur, armé de VHDL, produira des projets qui seront encore plus mauvais ».

183

Structure de base du langage VHDL

PORTE.VHD

Nom du fichier

Référence de la librairie utilisée

Partie de la librairie que l'on va utiliser

```
LIBRARY IEEE ;

USE IEEE.STD_LOGIC_1164.ALL ;

ENTITY Porte_ET IS
  PORT ( A, B : IN STD_LOGIC ;
         C : OUT STD_LOGIC ) ;

END Porte_ET ;

ARCHITECTURE Archi_porte_ET OF Porte_ET IS
BEGIN
   C <= A AND B ;
END Archi_porte_ET ;
```

Entité

Ø Description des objets à synthétiser vue de l'extérieur.
Ø Nom et nature des entrées.
Ø Nom et nature des sorties.
Ø C'est l'équivalent de la boîte du composant.

Architecture

Ø Description du comportement de l'objet.
Ø Une architecture est forcément associée à une entité.
Ø Une même entité peut avoir plusieurs architectures.

184

Structure de base du langage

Ø Les commentaires sont toujours précédés par deux signes moins et se terminent au passage à la ligne.

Ø Chaque instruction VHDL se termine par un point virgule.

Ø Les clauses LIBRARY et USE ne sont pas globales à l'ensemble d'un fichier source. Leur effet s'arrête dès la déclaration d'une nouvelle entité.

Ø Un des problèmes de VHDL, et des langages de haut niveau en général, est que l'on peut très bien avoir :
- q Une description correcte qui se simule parfaitement.
- q Mais dont la synthèse est rigoureusement impossible.

Ø Quand on utilise un simulateur VHDL, on compile un code VHDL. Le résultat de cette compilation devient une librairie qui apporte à son futur utilisateur de nouveaux objets prêts à être réemployés.

Ø Lors de la synthèse, la compilation aboutit à un montage constitué d'éléments logiques de base (portes et bascules) interconnectés. Cette description est ensuite optimisée en fonction de la technologie cible puis placée-routée dans le composant choisi.

Éléments de base du langage

Identificateurs

Ø Le premier caractère doit être une lettre.

Ø Le dernier caractère ne doit pas être un *underscore* « _ ».

Ø Deux *underscores* à la suite sont interdits « __ ».

Ø Le langage n'est pas sensible à la différence majuscules minuscules.

Caractères

Ø Ils sont délimités par des guillemets simple, et certains caractères spéciaux prédéfinis.

Ø Exemples : 'A', 'z', CR (Retour Chariot), LF (*Line feed*).

Éléments de base du langage

Chaînes de caractères

- Elles sont délimitées par des guillemets doubles.
- Exemple : "Bonjour".

Bits

- Deux valeurs prévues : '0' et '1' sont délimitées par des guillemets simples.

Vecteurs de bits

- Ils sont délimités par des guillemets doubles. Le caractère *underscore* est ignoré dans une chaîne de 0 et de 1.
- Exemples : "1101_0011", "11010011", X"D3" représentent la même valeur.

Éléments de base du langage

Il y a ambiguïté entre :

- Le caractère '1' et le bit '1'. Pour lever l'ambiguïté on doit spécifier le type.
  - Exemples : character'('1') ;
    bit'('1') ;
- La chaîne de caractères "0110" et le vecteur de bits "0110". Pour lever l'ambiguïté on doit spécifier le type.
  - Exemples : string'("0110") ;
    bit_vector'("0110") ;

Éléments de base du langage

Types étendus

Ø Pour représenter des états autres que 0 et 1, on fait appel a une extension du langage (la librairie IEEE 1164) qui offre le type STD_LOGIC constitué de neufs états :

- q U : non initialisé
- q X : Inconnu forcé
- q 0 : "zéro" forcé
- q 1 : "Un" forcé
- q Z : Haute impédance
- q W : Inconnu faible
- q L : "zéro" faible (pull-down)
- q H : "Un" faible (pull-up)
- q - : Indifférent

Booléens (boolean)

Ø Bien que le type booléen ne puisse prendre que deux valeurs : true et false, il n'est pas compatible avec le type bit.

Ø Les instructions de comparaison génèrent un résultat booléen.

Éléments de base du langage

Constantes

Ø Permettent de faciliter l'écriture et la modification de valeurs fixes lorsqu'elles sont répétitives.

Ø La portée d'une constante est limitée à la zone où elle est déclarée.

Ø Exemple : CONSTANT taille : INTEGER : 16 ;

Signaux

Ø Ils représentent toujours une équipotentielle.

Ø les signaux peuvent être initialisés lors de leur déclaration.

- q Cette pratique simplifie la simulation.
- q Beaucoup d'outils de synthèse ignorent ou interprètent mal cette initialisation.

Ø L'affectation d'une valeur à un signal se note : Signal <= valeur ;

Éléments de base du langage

Variables

- Leur utilisation est délicate.
- Peu de synthétiseur du marché les traitent correctement.
- Elles ne sont utilisables et n'ont d'existence :
  - Qu'à l'intérieur des Process.
  - Que dans les sous programmes.
- Elles ne sont pas censées conserver leur valeur entre deux activation du process.
- Elles n'ont pas nécessairement de représentation physique cohérente.
- L'affectation d'une valeur à une variable se note :
  Variable := Valeur ;

Éléments de base du langage

Scalaires

- Ces types sont ordonnés et permettent, par exemple, la comparaison.
- Ils se divisent en plusieurs classes :
  - Les entiers.
  - Les réels.
  - Les énumérations.
  - Les types physiques.
- Les entiers
  - Le type entier (INTEGER) peut être qualifié dans un interval (range)
  - Exemple : Variable i : INTEGER RANGE 0 TO 16 ;
  - Le sous-type NATURAL représente les entiers non-positifs.
  - Le sous-type POSITIVE représente les entiers positifs.
  - Les entiers n'ont aucun rapport avec les vecteurs de bits.

Éléments de base du langage

- Les réels
  - Sont d'assez peu d'utilité pour la synthèse mais peuvent s'avérer utiles pour la simulation.
- Énumération
  - c'est une liste de valeurs possibles pour le type donné.
  - Syntaxe : TYPE nom_énumération IS (Valeur1, Valeur2,...,ValeurN) ;
- Types physiques
  - Servent à mesurer des grandeurs physiques, comme le temps dont les unités sont : fs, ns, us, ms, sec, min, hr.
  - Ils ne servent pas à la synthèse.

Éléments de base du langage

Tableaux

- Les seuls tableaux prédéfinis sont :
  - Les BIT_VECTOR.
  - Les STD_LOGIC_VECTOR.
  - Les STRING.
- On spécifie la déclaration de l'indice par
  - TO (croissant).
  - DOWNTO (décroissant).
- Syntaxe :

  TYPE nom_tableau IS ARRAY (Direction.indice) OF nature_éléments;

Éléments de base du langage

Enregistrements

Ø C'est la notion classique des langages de programmation.

Ø syntaxe :

```
TYPE nom_enregistrement IS RECORD
        champ1 : type_champ1 ;
        champ2 : type_champ2 ;
                .....
        champN : type_champN ;
END RECORD ;
```

Ø Utilisation de la variable :

```
Variable.champ1
Variable.champ2
```


Éléments de base du langage

Attributs

Ø Permettent de récupérer de l'information sur la nature des objets.

Ø Quelques attributs utiles :

- 'left : valeur de gauche.
- 'right : valeur de droite.
- 'high : valeur supérieure.
- 'low : valeur inférieure.
- 'length : longueur.
- 'range : portée.
- 'reverse_range : protée inverse.

Opérateurs logiques

- Classés par ordre croissant de précédences.
- Applicables sur booléens, bits et dérivés.

Opérateur	Description	Résultat
not	Complément logique unaire	même type
and	ET logique	même type
or	OU logique	même type
nand	NON ET logique	même type
nor	NON OU logique	même type
xor	OU Exclusif logique	même type
nxor	NON OU Exclusif logique	même type

197

Opérateurs relationnels

- Classés par ordre croissant de précédences.
- Applicables sur type scalaire.

Opérateur	Description	Résultat
=	égalité	Booléen
/=	Inégalité	Booléen
<	Inférieur	Booléen
<=	Inférieur ou égal	Booléen
>	Supérieur	Booléen
>=	Supérieur ou égal	Booléen

198

Opérateurs arithmétiques

- Classés par ordre croissant de précédences.
- Applicables sur les vecteurs de bits et types numériques.

Opérateur	Description	Résultat
+	Addition	même type
-	Soustraction	même type
abs	Valeur absolue	même type
&	Concaténation	même type, plus large

Opérateurs arithmétiques (non synthétisable par nature)

- Classés par ordre croissant de précédences.
- Applicables sur les types numériques (entiers signés et non signés, flottants).

Opérateur	Description	Résultat
*	Multiplication	dépend
/	Division	dépend
mod	Modulo sur entiers	entier
rem	Reste sur entiers	entier

Opérateurs de décalage

Ø Classés par ordre croissant de précédences.

Ø Applicables sur les tableaux d'éléments bits ou étendus.

Opérateur	Description	Résultat
sll	Décalage logique à gauche	même type
srl	Décalage logique à droite	même type
sla	Décalage arithmétique à gauche	même type
sra	Décalage arithmétique à droite	même type
rol	Rotation à gauche	même type
ror	Rotation à droite	même type

Modélisation en langage VHDL

Il y a trois moyens de modéliser un composant :

Ø Structurelle (structural) : description d'un montage sous forme textuelle. Ceci revient à décrire un schéma (composants, interconnexions entres les composants). Cette méthode revient pratiquement à écrire une *netlist*.

Ø Flot de données (*Dataflow*) : Utilisation d'instructions concurrentes décrivant le flux des données.

Ø Comportementale (*Behavioral*) : Utilisation de *process* à l'intérieur desquels les instructions se déroulent de manière séquentielle. Ce type de modélisation est aussi appelée niveau RTL (*Register Transfert Level*) pour qualifier une modélisation comportementale synthétisable.

IF

Ø Disponible uniquement à l'intérieur d'un process.

Ø Syntaxe :

```
 IF condition THEN
    instruction ;
   [ instruction ] ;
 [ ELSIF condition THEN
    instruction ;
   [ instruction ] ; ]
 [ ELSE condition THEN
    instruction ;
   [ instruction ] ; ]
  END IF ;
```

Ø Si plusieurs IF indépendants se suivent pour tester une même variable (ou signal), le dernier test sera prioritaire sur ceux qui le précédent.

Ø Il faut se méfier des clauses incomplètes qui peuvent engendrer des synthèses plus gourmandes en nombre de portes.

203

CASE

Ø Disponible uniquement à l'intérieur d'un process.

Ø Syntaxe :

```
 CASE expression IS
    WHEN valeur1              => instruction ; [ instruction ; ]
   [ WHEN valeur2             => instruction ; [ instruction ; ] ]
   [ WHEN valeur3 | valeur4   => instruction ; [ instruction ; ] ]
   [ WHEN OTHERS              => instruction ; [ instruction ; ] ]
ou
   [ WHEN OTHERS                        => NULL ; ]
 END CASE ;
```

204

FOR...LOOP

- Disponible uniquement à l'intérieur d'un process.

- Syntaxe :

[étiquette :] FOR indice IN intervalle LOOP
instruction ;
[instruction ;]
END LOOP [étiquette :] ;

- Indice est un identificateur implicite ou une variable déclarée.

- intervalle délimite un sous-ensemble fini exemple :
  - 31 DOWNTO 0
  - 1 TO 3

WHILE...LOOP

- Disponible uniquement à l'intérieur d'un process.

- Syntaxe :

[étiquette :] WHILE condition LOOP
instruction ;
[instruction ;]
END LOOP [étiquette :] ;

NEXT

- Disponible uniquement à l'intérieur d'un *process*.
- Elle s'utilise à l'intérieure d'une boucle.
- Elle permet de sauter les instructions qui suivent et de passer directement à l'instruction END LOOP.
- Syntaxe : NEXT [étiquette :] [WHEN condition] ;

EXIT

- Disponible uniquement à l'intérieur d'un *process*.
- Elle s'utilise à l'intérieure d'une boucle.
- Elle permet de terminer l'exécution de la boucle.
- Syntaxe : EXIT [étiquette :] [WHEN condition] ;

WAIT

- Disponible uniquement à l'intérieur d'un *process*.
- Elle suspend la simulation du *process*.
- Syntaxe :
  - WAIT ; -- suspension définitive.
  - WAIT ON signal ; -- Attend jusqu'à ce qu'un signal change d'état.
  - WAIT UNTIL condition ; -- Attend jusqu'à ce qu'une condition soit vrai.
  - WAIT FOR durée ; -- Attente pendant un certain temps.

Les process

Ø Un *process* est une région de code VHDL qui représente une tâche indépendante à l'intérieure de laquelle l'exécution des instructions est séquentielle, mais sans que le temps progresse.

Ø Si plusieurs *process* sont déclarés (y compris dans la même section «architecture») ils se dérouleront indépendamment et simultanément.

Ø Dans un *process*, peuvent intervenir des signaux mais aussi des variables qui sont locales au *process* (dans lequel elles sont obligatoirement déclarées).

Ø La déclaration d'un *process* est en général suivie d'une liste de signaux (*sensitivity list*) dont le changement d'état déclenchent l'exécution du *process*.

Les process

Ø Lorsque le *process* est activé, c'est à dire au moment où un signal de la liste de sensibilité changent d'état :

- q Le temps est « figé ».
- q Le *process* est analysé séquentiellement.
- q Lorsqu'une affectation de signaux est rencontrée elle est planifiée pour être exécutée une fois que l'intégralité du *process* a été déroulée.
- q Lorsqu'une affectation de variable est rencontrée elle prend effet immédiatement (là où elle apparaît).

=> Si plusieurs affectations à un même signal sont rencontrées en séquence dans un *process*, seule la dernière a un effet. Les autres sont ignorées.

Ø La liste de sensibilité n'est pas obligatoire. Si elle est absente :

- q Le *process* est activé au démarrage du simulateur.
- q Le *process* se réactive en permanence dès que le «END PROCESS» est atteint.
- q Il est donc indispensable qu'une instruction « WAIT » (au moins) soit rencontrée. Faute de quoi, le simulateur se bloque en « tournant en rond » sans échappatoire.

Construction d'un test bench

Ø Pour effectuer la simulation d'un module VHDL, il est en général nécessaire d'écrire un « *test Bench* ».

Ø Un « *test Bench* » est un fichier VHDL structurel qui décrit l'interconnexion du module testé et la génération des stimuli appliqués au module.

Ø Typiquement, un « *test Bench* » :
- q Commence par une déclaration ENTITY sans liste PORT MAP.
- q Continue par une déclaration de signaux (pour chacune des entrées/sorties du module à tester).
- q Instancie le module à tester.
- q Se termine par un ou plusieurs *process* décrivant les stimuli.

Ø Certain logiciel crée automatiquement le « corps » du fichier et vous laisse écrire les stimuli nécessaires au test.

211

SOLUTIONS

212

Exemple : additionneur

A		B		C	S	
A1	A0	B1	B0		S1	S0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

213

Simplification algébrique

Ø La simplification algébrique implique l'application des identités de base de l'algèbre de Boole pour réduire l'expression booléenne à une expression composée de moins d'éléments.

Exemple additionneur (Retenues)

$C = \overline{A1}.A0.B1.B0 + A1.\overline{A0}.B1.\overline{B0} + A1.\overline{A0}.B1.B0 +$
$A1.A0.\overline{B1}.B0 + A1.A0.B1.\overline{B0} + A1.A0.B1.B0$
$= \overline{A1}.A0.B1.B0 + A1.A0.\overline{B1}.\overline{B0} + A1.\overline{A0}.B1.\overline{B0} +$
$A1.\overline{A0}.B1.B0 + A1.A0.B1.\overline{B0} + A1.A0.B1.B0$
$= \overline{A1}.A0.B1.B0 + A1.A0.\overline{B1}.B0 +$
$A1.\overline{A0}.(B1.\overline{B0} + B1.B0) +$
$A1.A0.\ (B1.\overline{B0} + B1.B0)$
$= \overline{A1}.A0.B1.B0 + A1.A0.\overline{B1}.B0 +$
$A1.\overline{A0}.(B1.(\overline{B0} + B0)) +$
$A1.A0.\ (B1.(\overline{B0} + B0))$

214

Simplification algébrique

C = $\overline{A1}$.A0.B1.B0 + A1.A0.$\overline{B1}$.B0 +
A1.$\overline{A0}$.B1 + A1.A0.B1
= $\overline{A1}$.A0.B1.B0 + A1.A0.$\overline{B1}$.B0 +
A1.B1.($\overline{A0}$ + A0)
= $\overline{A1}$.A0.B1.B0 + A1.A0.$\overline{B1}$.B0 + A1.B1
= $\overline{A1}$.A0.B1.B0 + A1.B1 + A1.A0.$\overline{B1}$.B0 + A1.B1
= B1.($\overline{A1}$.A0.B0 + A1) + A1.($\overline{B1}$.A0.B0 + B1)

Voyons ce que donne $\overline{A}$.X + A

A	X	$\overline{A}$	$\overline{A}$.X	$\overline{A}$.X +A
0	0	1	0	0
0	1	1	1	1
1	0	0	0	1
1	1	0	0	1

Simplification algébrique

Donc $\overline{A}$.X + A = A + X

C = B1.(A1 + A0 . B0) + A1.(B1 + A0.B0)

C = A1.B1 + A0.B1.B0 + A1.A0.B0

Ø Avec un peu (beaucoup) d'entraînement on peut devenir très performant.

C'est un jeu comme un autre...

Autres Fonctions : NAND

$S = \overline{A}.\overline{B} + \overline{A}.B + A.\overline{B}$

$= \overline{A}.\overline{B} + \overline{A}.B + A.\overline{B} + \overline{A}.\overline{B}$

$= \overline{A}.(\overline{B} + B) + \overline{B}.(A + \overline{A})$

$S = \overline{A} + \overline{B}$

1

CQFD

Autres Fonctions : OR

$S = (A + \overline{B}).(\overline{A} + B).(\overline{A} + \overline{B})$

$= (A.\overline{A} + A.B + \overline{B}.\overline{A} + \overline{B}.B).(\overline{A} + \overline{B})$

$= (A.B + \overline{B}.\overline{A}).(\overline{A} + \overline{B})$

0

$= A.B.\overline{A} + A.B.\overline{B} + \overline{B}.\overline{A}.\overline{A} + \overline{B}.\overline{A}.\overline{B}$

$S = \overline{A}.\overline{B}$

0

CQFD

Autres Fonctions : XOR

$S = (A + B).(\overline{A} + \overline{B})$

$= A.\overline{A} + A.\overline{B} + B.\overline{A} + B.\overline{B}$

$S = A.\overline{B} + B.\overline{A}$

0

CQFD