Structure des ordinateurs

Pierre Wolper

Email: pw@montefiore.ulg.ac.be

URL: http: //www.montefiore.ulg.ac.be/~pw/
http: //www.montefiore.ulg.ac.be/
~pw/cours/struct-candi.html

Références

Stephen A. Ward and Robert H. Halstead, *Computation Structures*, MIT Press, 1990.

http://6004.lcs.mit.edu.

Objectifs du cours

Comprendre l'informatique *par le bas* :

1. L'information et sa représentation numérique (digitale);

2. L'électronique numérique (notions de base);

3. La construction d'un processeur (sur base d'un exemple simple, mais donné de façon détaillée);

4. La programmation en langage machine et en assembleur;

5. Les concepts utilisés pour permettre l'implémentation efficace des systèmes d'exploitation et des langages de haut niveau (gestion d'interruptions et mode privilégié, mémoire virtuelle, . . .).

Organisation du cours

- Exposés présentant la matière;
- Séances d'exercices;
- Travaux de programmation limités à réaliser (micro-code, assembleur).

De l'information symbolique à l'électronique numérique

L'information et sa représentation symbolique

- Une représentation symbolique est une représentation de l'information à l'aide de séquences de symboles (signes reconnaissables) pris dans un ensemble fini fixé.

- Un texte écrit, ou un nombre représenté à l'aide des chiffres $\{0, 1 \ldots, 9\}$, sont des représentations symboliques.

- L'informatique étudie le "traitement" d'informations représentées symboliquement. De façon très générale, le but d'un traitement informatique est de dériver à partir de l'information fournie une information sous une forme plus pertinente et plus facilement exploitable.

L'information symbolique : Au delà des nombres et des textes

Une information qui n'est pas au départ symbolique, par exemple un signal sonore, peut être représentée sous forme d'une suite de nombres (et donc de symboles) à l'aide de deux techniques complémentaires : l'échantillonnage et la discrétisation.

1. L'**échantillonnage** (*sampling*) consiste à ne considérer la valeur du signal qu'à une série discrète (en général périodique) d'instants. Si l'échantillonnage est suffisamment rapide par rapport à la vitesse de variation du signal toute l'information présente dans le signal est conservée.

2. La **discrétisation** (*discretisation*, *quantization*) consiste à représenter des valeurs prises dans un intervalle continu par un nombre fini de valeurs discrètes choisies dans cet intervalle.

   La discrétisation introduit une erreur mais celle-ci est sans conséquence si elle est faible par rapport à l'imprécision aléatoire (bruit) qui existe dans tout signal.

Ces techniques sont notamment utilisées dans les CD et DVD qui utilisent un codage digital (et donc symbolique) de l'information.

Le choix de l'ensemble de symboles

- La seule différence fondamentale entre les ensembles de symboles est leur taille; la représentation utilisée par les symboles n'est qu'une convention.

- L'ensemble de symboles utilisé en informatique est celui de taille 2; on parle alors de représentation **binaire** (*binary*). On représente en général l'ensemble binaire par $\{0, 1\}$ ou encore $\{\mathbf{T(rue)}, \mathbf{F(alse)}\}$.

- Le binaire facilite la représentation physique des symboles et n'est pas limitatif car un ensemble plus large de symboles peut se représenter par des groupes de symboles binaires.

Calculer avec de l'information binaire

- Traiter de l'information binaire c'est, à partir d'un certain nombre de *bits* (symboles binaires), calculer d'autres bits.

- Si le nombre de bits de données est fixé, on calcule une **fonction booléenne** (*Boolean function*) qui à partir des bits de données fournit un bit de résultat — pour plusieurs bits de résultat, on calcule plusieurs fonctions booléennes.

- Nous allons donc maintenant étudier les fonctions booléennes et leur représentation dans l'**algèbre booléenne** (*Boolean algebra*) qui est la théorie des opérations sur l'ensemble binaire.

Les fonctions booléennes

Définition : Une *fonction booléenne* d'*arité* n est une fonction de n variables booléennes d'entrée vers une variable booléenne de sortie ($n \geq 0$).

Pour définir une fonction booléenne, il suffit de donner sa valeur de sortie pour toutes les combinaisons possibles de valeurs de ses arguments.

La table associant ces valeurs de sortie aux combinaisons de valeurs d'entrée est appelée **table de vérité** (*truth table*).

La table de vérité d'une fonction d'arité n possède 2^n lignes.

Exemple

Table de vérité de la fonction f à trois arguments qui est vraie si et seulement si exactement deux de ses arguments sont vrais :

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Les fonctions booléennes de base

Combien y a-t-il de fonctions booléennes distinctes d'arité n ?

Réponse :

- La table de vérité d'une fonction d'arité n possède 2^n lignes, et
- Chaque ligne d'une table de vérité peut prendre la valeur 0 ou la valeur 1,
- Donc il y a 2^{2^n} tables de vérité distinctes.

Il existe donc $2^{2^2} = 16$ fonctions booléennes d'arité 2.

Nous allons étudier certaines de ces fonctions qui présentent un intérêt particulier et qui, à l'aide du mécanisme de composition de fonctions, permettent de définir toutes les fonctions booléennes.

La fonction AND

Cette fonction possède la table de vérité suivante :

x_1	x_2	$AND(x_1, x_2)$
0	0	0
0	1	0
1	0	0
1	1	1

Elle se dénote par l'opérateur binaire "$\cdot$" :

$$AND(x_1, x_2) = x_1 \cdot x_2$$

La valeur $x_1 \cdot x_2$ est vraie si et seulement si x_1 *et* x_2 sont vrais.

La fonction OR

Cette fonction possède la table de vérité suivante :

x_1	x_2	$OR(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	1

Elle se dénote par l'opérateur binaire "+" :

$$OR(x_1, x_2) = x_1 + x_2$$

La valeur $x_1 + x_2$ est vraie si et seulement si x_1 *ou* x_2 sont vrais.

La fonction XOR

Cette fonction (aussi appelée *ou exclusif*) possède la table de vérité suivante :

x_1	x_2	$XOR(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

Elle se dénote par l'opérateur binaire "$\oplus$" :

$$XOR(x_1, x_2) = x_1 \oplus x_2$$

La valeur $x_1 \oplus x_2$ est vraie si et seulement si x_1 *ou bien* x_2 est vrai.

La fonction NAND

Cette fonction possède la table de vérité suivante :

x_1	x_2	$NAND(x_1, x_2)$
0	0	1
0	1	1
1	0	1
1	1	0

Remarque : Pour les mêmes valeurs d'arguments, cette fonction renvoie toujours une valeur opposée à celle de la fonction AND. On a donc

$$NAND(x_1, x_2) = NOT(AND(x_1, x_2)),$$

où NOT est une fonction unaire renvoyant la valeur inverse de celle de son argument.

La fonction NOT

La table de vérité de cette fonction est par conséquent la suivante :

x_1	$NOT(x_1)$
0	1
1	0

Elle se dénote par une barre horizontale au dessus de son argument, ou bien par l'opérateur unaire $\neg$:

$$NOT(x_1) = \overline{x_1} = \neg x_1.$$

On a donc

$$NAND(x_1, x_2) = \overline{x_1 \cdot x_2}.$$

La fonction NOR

Cette fonction possède la table de vérité suivante :

x_1	x_2	$NOR(x_1, x_2)$
0	0	1
0	1	0
1	0	0
1	1	0

Remarque : Pour les mêmes valeurs d'arguments, cette fonction renvoie toujours une valeur opposée à celle de la fonction OR. On a donc

$$NOR(x_1, x_2) = \overline{x_1 + x_2}.$$

Les expressions booléennes

Les fonctions booléennes à deux arguments (opérateurs booléens) que nous venons de voir peuvent être combinées pour définir d'autres fonctions. Pour représenter la composition d'opérateurs booléens, on utilise des expressions booléennes (*Boolean expressions*) similaires aux expressions arithmétiques représentant la composition d'opérations arithmétiques.

Les opérateurs de base que nous utiliserons sont : $\cdot, +, \oplus$ et $^-$.

Par convention, l'opérateur $\cdot$ a une priorité plus élevée que $+$ et $\oplus$: $x_1 + x_2 \cdot x_3$ est équivalent à $x_1 + (x_2 \cdot x_3)$.

De même, $x_1 \oplus x_2 \cdot x_3$ est équivalent à $x_1 \oplus (x_2 \cdot x_3)$.

L'algèbre booléenne

L'algèbre booléenne étudie les propriétés des opérateurs booléens et des fonctions qu'ils permettent de définir. Quelques propriétés particulièrement intéressantes sont données ci-dessous.

La commutativité

Les opérateurs $\cdot$, $+$ et $\oplus$ sont commutatifs :

$$\begin{aligned} x_1 \cdot x_2 &= x_2 \cdot x_1, \\ x_1 + x_2 &= x_2 + x_1, \\ x_1 \oplus x_2 &= x_2 \oplus x_1. \end{aligned}$$

L'associativité

Les opérateurs $\cdot$, $+$ et $\oplus$ sont associatifs :

$$\begin{aligned} x_1 \cdot (x_2 \cdot x_3) &= (x_1 \cdot x_2) \cdot x_3, \\ x_1 + (x_2 + x_3) &= (x_1 + x_2) + x_3, \\ x_1 \oplus (x_2 \oplus x_3) &= (x_1 \oplus x_2) \oplus x_3. \end{aligned}$$

Remarque : L'associativité permet d'éliminer les parenthèses des expressions précédentes.

La distributivité

L'opérateur $\cdot$ est distributif sur les opérateurs $+$ et $\oplus$:

$$\begin{aligned} x_1 \cdot (x_2 + x_3) &= (x_1 \cdot x_2) + (x_1 \cdot x_3), \\ x_1 \cdot (x_2 \oplus x_3) &= (x_1 \cdot x_2) \oplus (x_1 \cdot x_3). \end{aligned}$$

L'opérateur $+$ est distributif sur l'opérateur $\cdot$:

$$x_1 + (x_2 \cdot x_3) = (x_1 + x_2) \cdot (x_1 + x_3).$$

Remarque : Cette dernière propriété n'est pas valide en arithmétique !

Les règles de DeMorgan

Ces règles permettent d'exprimer chacun des opérateurs $+$ et $\cdot$ en fonction de l'autre et du complément $^-$:

$$\begin{aligned} \overline{x_1 + x_2 + \cdots + x_n} &= \overline{x_1} \cdot \overline{x_2} \cdot \cdots \cdot \overline{x_n}, \\ \overline{x_1 \cdot x_2 \cdot \cdots \cdot x_n} &= \overline{x_1} + \overline{x_2} + \cdots + \overline{x_n}. \end{aligned}$$

Il est aussi possible d'exprimer l'opérateur $\oplus$ en fonction des autres opérateurs :

$$x_1 \oplus x_2 = \overline{x_1} \cdot x_2 + x_1 \cdot \overline{x_2}.$$

Les règles d'absorption

Ces règles permettent de simplifier certaines expressions :

$$x_1 + (x_1 \cdot x_2) = x_1 \qquad x_1 \cdot (x_1 + x_2) = x_1$$
$$x_1 + (\overline{x_1} \cdot x_2) = x_1 + x_2 \qquad x_1 \cdot (\overline{x_1} + x_2) = x_1 \cdot x_2$$
$$x_1 \oplus (x_1 \cdot x_2) = x_1 \cdot \overline{x_2} \qquad x_1 \cdot (x_1 \oplus x_2) = x_1 \cdot \overline{x_2}$$

Autres règles

$$x_1 + 1 = 1 \qquad x_1 \cdot 0 = 0 \qquad x_1 \oplus 0 = x_1$$
$$x_1 + 0 = x_1 \qquad x_1 \cdot 1 = x_1 \qquad x_1 \oplus 1 = \overline{x_1}$$
$$x_1 + x_1 = x_1 \qquad x_1 \cdot x_1 = x_1 \qquad x_1 \oplus x_1 = 0$$
$$x_1 + \overline{x_1} = 1 \qquad x_1 \cdot \overline{x_1} = 0 \qquad x_1 \oplus \overline{x_1} = 1$$

Des fonctions booléennes aux expressions booléennes

Toute fonction booléenne peut être représentée par une expression booléenne ne faisant intervenir que les opérateurs $\cdot$, $+$ et $\bar{}$. Par exemple, construisons une expression booléenne dénotant la fonction possédant la table de vérité

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Cette fonction n'est vraie que pour trois valeurs de ses arguments :

$$(0, 1, 1),$$
$$(1, 0, 1),$$
$$(1, 1, 0).$$

Pour chaque valeur des arguments, on peut écrire une expression qui est vraie pour cette valeur, et fausse pour toutes les autres :

$$\begin{aligned} f_{(0,1,1)}(x_1, x_2, x_3) &= \overline{x_1} \cdot x_2 \cdot x_3, \\ f_{(1,0,1)}(x_1, x_2, x_3) &= x_1 \cdot \overline{x_2} \cdot x_3 \\ f_{(1,1,0)}(x_1, x_2, x_3) &= x_1 \cdot x_2 \cdot \overline{x_3}. \end{aligned}$$

La fonction f est vraie si au moins une des trois fonctions précédentes est vraie :

$$\begin{aligned} f(x_1, x_2, x_3) &= f_{(0,1,1)}(x_1, x_2, x_3) \\ &\quad + f_{(1,0,1)}(x_1, x_2, x_3) \\ &\quad + f_{(1,1,0)}(x_1, x_2, x_3) \\ &= \overline{x_1} \cdot x_2 \cdot x_3 + x_1 \cdot \overline{x_2} \cdot x_3 \\ &\quad + x_1 \cdot x_2 \cdot \overline{x_3}. \end{aligned}$$

En appliquant les règles de l'algèbre booléenne :

$$\begin{aligned} f(x_1, x_2, x_3) &= (\overline{x_1} \cdot x_2 + x_1 \cdot \overline{x_2}) \cdot x_3 \\ &\quad + x_1 \cdot x_2 \cdot \overline{x_3} \\ &= (x_1 \oplus x_2) \cdot x_3 + x_1 \cdot x_2 \cdot \overline{x_3}. \end{aligned}$$

La représentation physique des valeurs booléennes

- Pour construire une machine qui manipule des symboles binaires, il faut représenter ceux-ci physiquement. Il y a un grand nombre de représentations physiques possibles, mais celle très largement utilisée actuellement est la tension électrique.

- Toutefois, une tension électrique n'est pas limitée à des valeurs discrètes. Il faut donc construire des circuits dans lesquels les valeurs discrètes sont représentées par des intervalles de valeurs.

Les niveaux de tension

On associe à chacune des valeurs booléenne un niveau de tension :

- La tension nulle pour la valeur 0;
- La tension d'alimentation pour la valeur 1.

Il est cependant impossible de construire des circuits qui produisent exactement ces tensions. On définit donc des *intervalles de validité* plutôt que des niveaux de tension ponctuels.

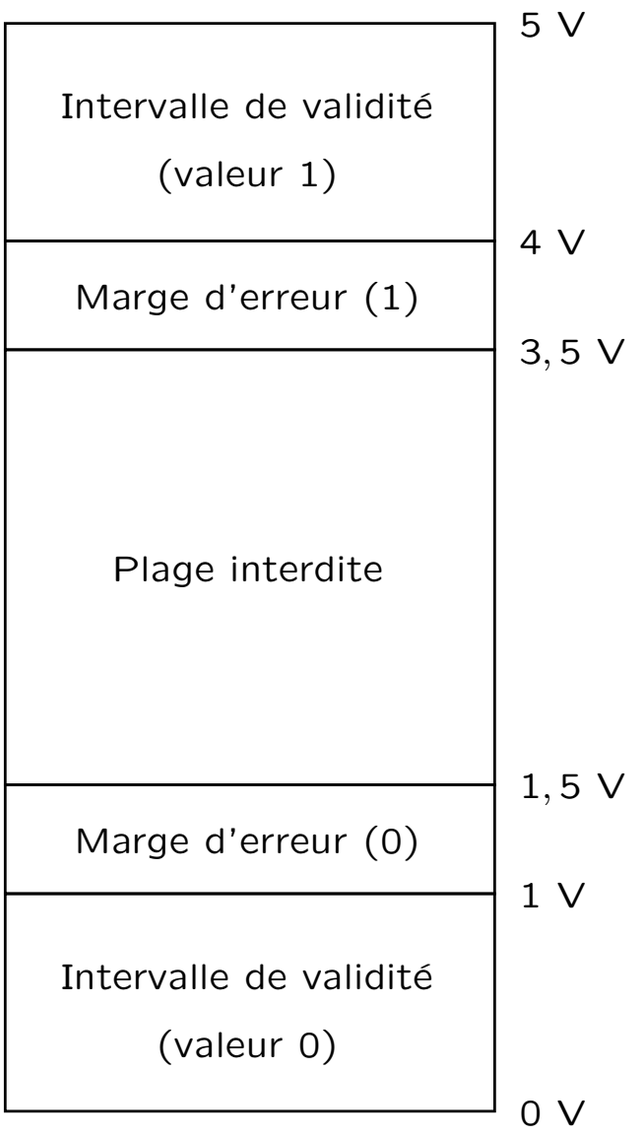
Les circuits sont construits de façon à ne générer que des signaux valides.

Les marges d'erreur

Bien que les signaux soient valides à la sortie des composants qui les génèrent, ils sont corrompus par une certaine quantité de bruit avant d'arriver à l'entrée d'autres composants.

On tient compte de ce bruit en dotant chaque intervalle de validité d'une *marge d'erreur* qui lui est adjacente. Les signaux situés dans les marges d'erreur peuvent être fiablement décodés en valeurs booléennes.

Les marges d'erreur : exemple



La discipline statique et l'abstraction digitale

Un circuit ne peut être connecté à d'autres circuits que s'il satisfait aux règles suivantes :

- Si on fournit des signaux valides constants aux entrées du circuit, alors ce dernier finira par générer des signaux de sortie valides après un certain *délai de propagation*.
- Un signal d'entrée est considéré valide si sa tension se situe dans un des deux intervalles de validité ou dans la marge d'erreur correspondante (mais pas dans la plage interdite).
- Un signal de sortie est considéré valide si sa tension appartient à un des deux intervalles de validité (mais pas à une marge d'erreur ni à la plage interdite).

La discipline statique et l'abstraction digitale (suite)

L'ensemble de ces règles constitue une *discipline statique*. Celle-ci garantit l'absence de signaux invalides en dehors des périodes de propagation et de transition des valeurs.

Grâce à cette discipline, on peut raisonner à propos des circuits réalisés comme s'ils manipulaient effectivement des valeurs discrètes. C'est ce que l'on appelle **l'abstraction digitale** (*digital abstraction*).

Les familles logiques

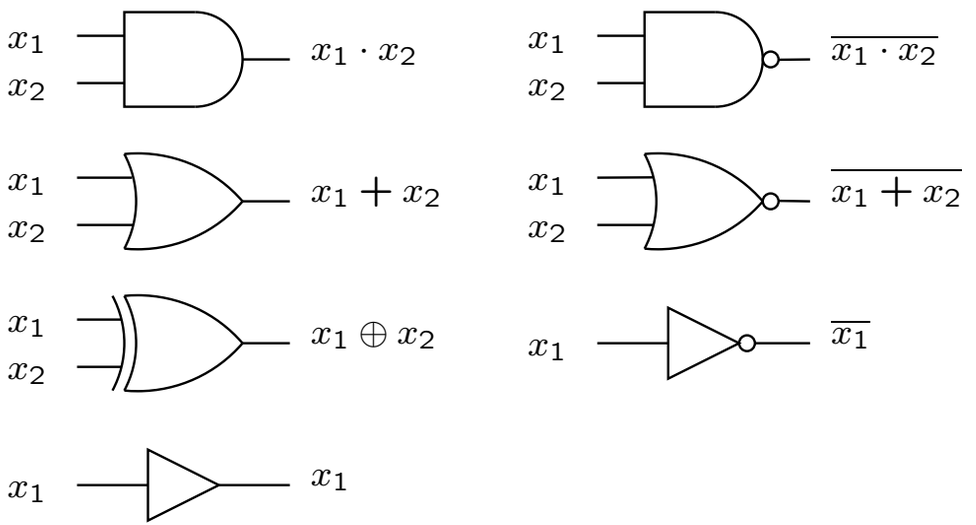
Il existe plusieurs types de circuits capables de traiter des signaux digitaux. Une *famille logique* est une norme définissant

- des niveaux de tension (intervalles de validité, marges d'erreur);
- des circuits de base;
- un ensemble de contraintes à respecter (sur la forme des connexions permises, les temps de propagation, . . .);
- un procédé de fabrication des composants;
- . . .

Les principales familles actuellement utilisées sont TTL, CMOS et ECL.

Les portes logiques

Les *portes logiques* sont des circuits digitaux élémentaires réalisant les fonctions booléennes de base. Les symboles conventionnels attribués aux portes sont les suivants :



Remarques :

- Ces symboles peuvent aussi décrire des portes possédant plus de deux entrées;
- Bien sûr, les portes respectent la discipline statique !

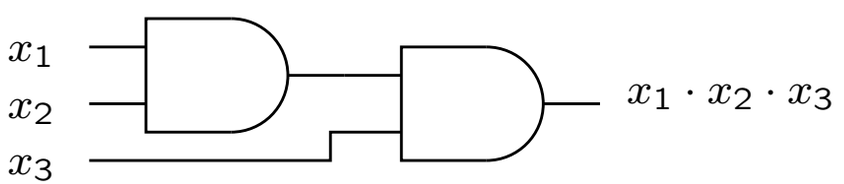
L'interconnexion des portes

L'interconnexion de plusieurs portes logiques permet de construire des circuits plus complexes.

Les règles d'interconnexion sont les suivantes :

- On ne peut pas connecter entre elles les sorties de plusieurs portes;
- Il est permis de connecter la sortie d'une porte aux entrées d'autres portes.

Exemple : Porte AND à trois entrées construite à partir de deux portes à deux entrées :



Les circuits combinatoires

Un *chemin* est un parcours d'un circuit, à partir d'un de ses points, effectué en suivant les fils et en franchissant les portes d'une entrée vers la sortie.

Un *cycle* est un chemin dont le point d'origine et le point de destination sont identiques, et qui franchit au moins une porte.

Définition : Un *circuit combinatoire* est un circuit digital dont aucun chemin n'est un cycle.

Les circuits combinatoires (suite)

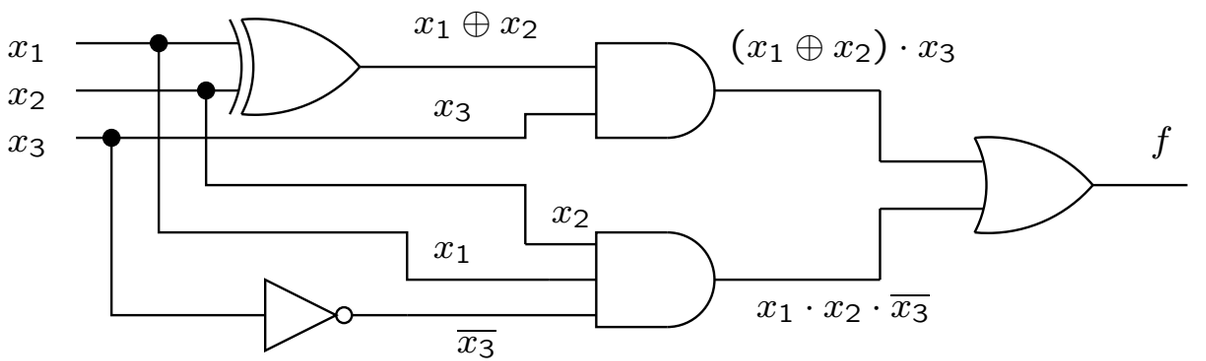
Propriété : Les valeurs booléennes générées aux sorties d'un circuit combinatoire dépendent uniquement des valeurs fournies aux entrées de celui-ci.

En d'autres termes, un circuit combinatoire à n entrées et m sorties peut être défini par m fonctions booléennes d'arité n.

Remarque : Sous l'hypothèse où toutes les portes possèdent le même délai de propagation, le délai de propagation maximal d'un circuit combinatoire correspond au chemin le plus long d'une entrée vers une sortie.

Exemple 1

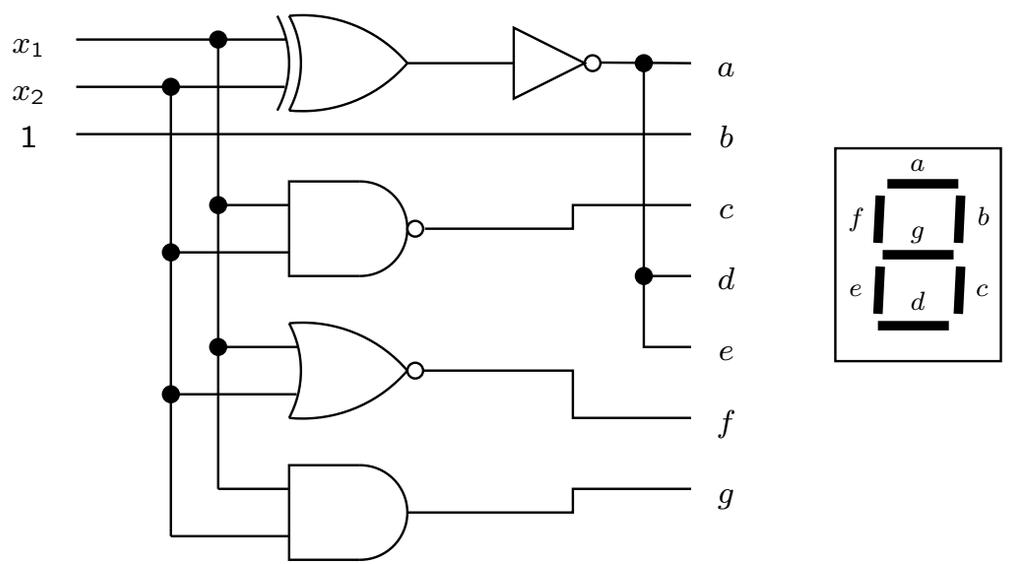
Circuit combinatoire réalisant la fonction d'arité 3 vraie si et seulement si deux de ses entrées sont vraies :



Temps de propagation maximal : 3τ, où τ est le temps de propagation maximal d'une porte.

Exemple 2

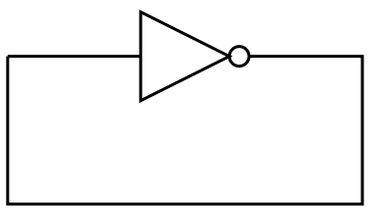
Circuit à deux entrées contrôlant un affichage à sept segments. Le chiffre affiché totalise le nombre d'entrées vraies. Un segment est allumé lorsque la sortie correspondante est vraie :



Temps de propagation maximal : 2τ.

L'instabilité et les circuits non combinatoires

Il est facile de construire des circuits qui ne sont pas combinatoires :



Ce circuit possède une particularité : Il est impossible d'affecter une valeur booléenne fixe à sa seule connexion ! Ce circuit est *instable*.

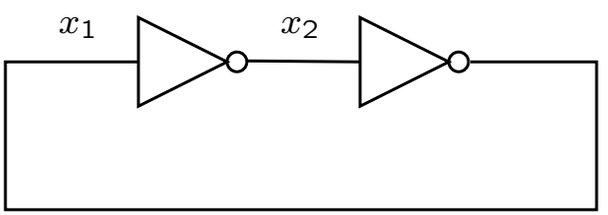
Définition : Un circuit est *stable* s'il est possible d'affecter une valeur booléenne persistante à chacune de ses connexions.

Les circuits instables sont à proscrire ! En pratique, de tels circuits génèrent

- des signaux invalides, ou
- des oscillations.

Les circuits non combinatoires stables

Tous les circuits non combinatoires ne sont pas instables :



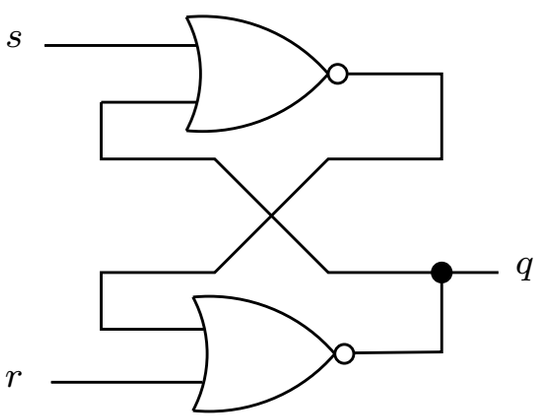
L'attribution des valeurs $x_1 = 0$ et $x_2 = 1$ est persistante, et donc le circuit est stable. Le choix des valeurs $x_1 = 1$ et $x_2 = 0$ est également persistant.

Remarque : La stabilité d'un circuit ne garantit pas l'absence de signaux invalides ou d'oscillations dans une réalisation pratique de ce circuit !

Les verrous

Le circuit précédent possède deux points de stabilité, et peut donc se trouver dans deux *états* distincts. Il est donc capable de *mémoriser* un bit d'information.

Ce circuit ne permet cependant pas de choisir la valeur booléenne mémorisée. Pour pallier à cet inconvénient, on lui ajoute des entrées permettant de contrôler la valeur circulant dans le cycle :



Le circuit obtenu porte le nom de **verrou** (*latch*).

Le fonctionnement d'un verrou

- **Si** $\mathbf{s} = 0$ **et** $\mathbf{r} = 0$ **:** Le circuit est équivalent à deux inverseurs en boucle, et mémorise donc un bit d'information.

  La valeur mémorisée peut être vue comme celle présente à la sortie q.

- **Si** $\mathbf{s} = 1$ **et** $\mathbf{r} = 0$ **:** La valeur mémorisée devient égale à 1 (*set*).

- **Si** $\mathbf{s} = 0$ **et** $\mathbf{r} = 1$ **:** La valeur mémorisée devient égale à 0 (*reset*).

  Le verrou est donc capable de retenir laquelle des entrées s ou r a été activée en dernier lieu.

- **Si** $\mathbf{s} = 1$ **et** $\mathbf{r} = 1$ **:** La valeur de mémorisée devient égale à 0, mais peut ensuite basculer vers n'importe quelle valeur lorsque s et r reprennent la valeur 0. Une telle **condition de course** (*race condition*) est à éviter !

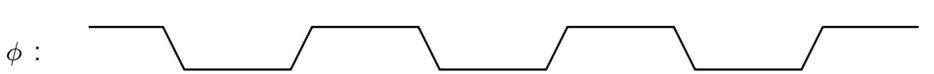
Le signal d'horloge

L'utilisation des verrous pose plusieurs problèmes :

- Le verrou charge une nouvelle valeur dès le moment où une de ses entrées prend la valeur 1. Ce moment peut dépendre des délais de propagation d'autres portes.
- Il faut garantir l'absence de conditions de course.

On souhaite que les données mémorisées par un circuit ne soient modifiées qu'à des instants ponctuels, bien déterminés.

La solution consiste à fournir au circuit un **signal d'horloge** (*clock signal*). Ce signal est généré par un composant spécial, et est constitué d'une alternance périodique de valeurs 0 et 1.



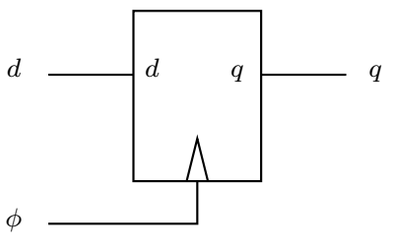
Les valeurs mémorisées par ce circuit ne sont alors modifiées qu'aux instants où l'horloge effectue une transition de la valeur 0 à la valeur 1, c'est-à-dire lors de ses **flancs montants** (*raising edges*).

Note : Une autre convention consiste à considérer les transitions de la valeur 1 à la valeur 0, les **flancs descendants** (*falling edges*).

Le flip-flop

Le composant de mémorisation élémentaire présent dans les circuits basés sur une horloge est le *flip-flop*.

Symbole :



Fonctionnement :

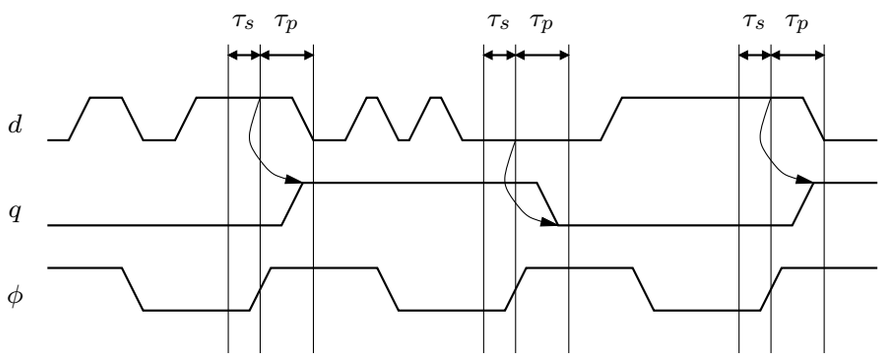
- Un flip-flop est capable de retenir un bit. La valeur retenue est disponible à la sortie q;
- Lors d'un flanc montant de l'horloge, le flip-flop charge la valeur présente à l'entrée d. On dit que le flip-flop est **déclenché par le flanc** (*edge triggered*).

Les délais d'un flip-flop

Le fonctionnement d'un flip-flop n'est pas instantané :

- La valeur mémorisée n'est disponible à la sortie qu'un certain temps après avoir été chargée. Ce délai est le *délai de propagation* τ_p du flip-flop;
- Pour qu'une valeur d'entrée puisse être chargée, il faut qu'elle reste constante un certain laps temps avant le coup d'horloge. Ce délai est le *délai de stabilisation* τ_s du flip-flop.

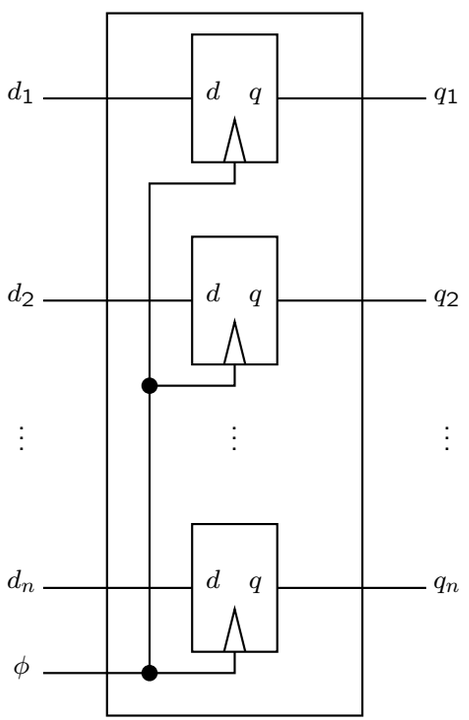
Exemple :



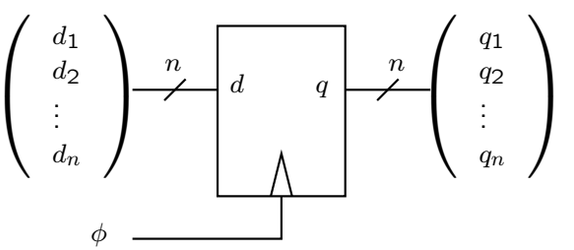
Les registres

En général, un circuit mémorise plus d'un bit d'information. Un *registre* est un composant obtenu en regroupant plusieurs flip-flops partageant la même horloge.

Circuit équivalent :



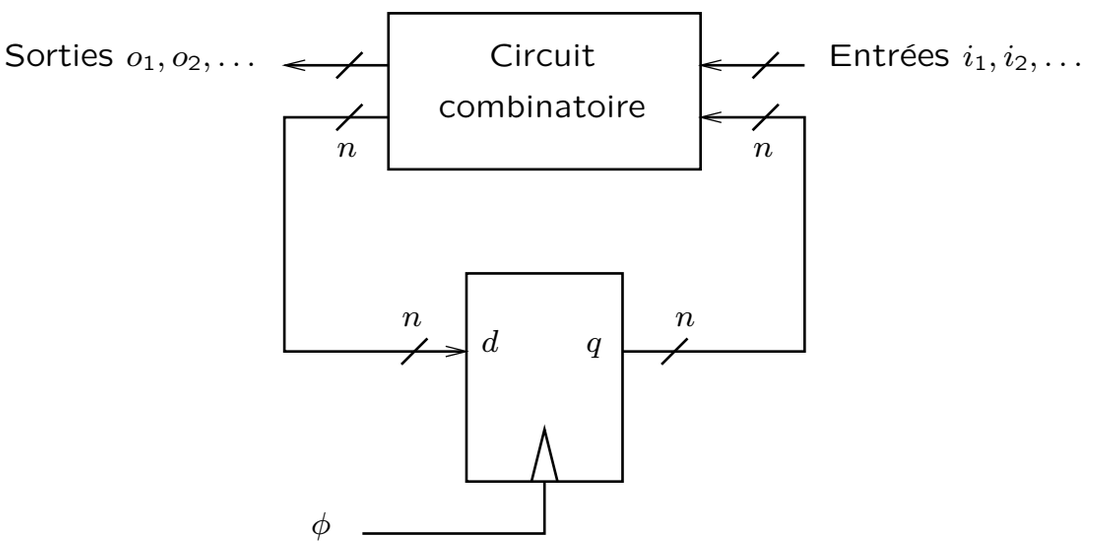
Symbole :



Les circuits séquentiels

Dans un circuit digital, on peut séparer les composants mémorisant les données de ceux dédiés à leur traitement.

Un *circuit séquentiel* est un circuit possédant la forme générale suivante :



Les valeurs retenues par le registre déterminent l'*état* du circuit. La capacité du registre étant de n bits, le circuit peut potentiellement se trouver dans 2^n états.

Fonctionnement d'un circuit séquentiel

Soient

- T la période de l'horloge;
- τ_c le temps de propagation du circuit combinatoire;
- τ_p le temps de propagation du registre;
- τ_s le temps de stabilisation du registre.

Si $T > \tau_c + \tau_p + \tau_s$, le circuit change d'état à chaque coup d'horloge. Lors d'un changement d'état, le nouvel état s_{t+1} est déterminé par le circuit combinatoire à partir de

- l'état précédent s_t, et
- la valeur i_t des entrées du circuit.

On a donc $s_{t+1} = f(s_t, i_t)$, où f est une *fonction de transition* réalisée par le circuit combinatoire.

La discipline dynamique

Pour que le changement d'état s'effectue correctement, les entrées du circuit doivent rester stables pendant une durée au moins égale à $\tau_c + \tau_s$ avant chaque coup d'horloge.

Un circuit séquentiel respectant cette condition obéit à la règle de *discipline dynamique*.

Un circuit séquentiel peut également posséder des sorties. Leur valeur o_t est déterminée par le circuit combinatoire à partir de

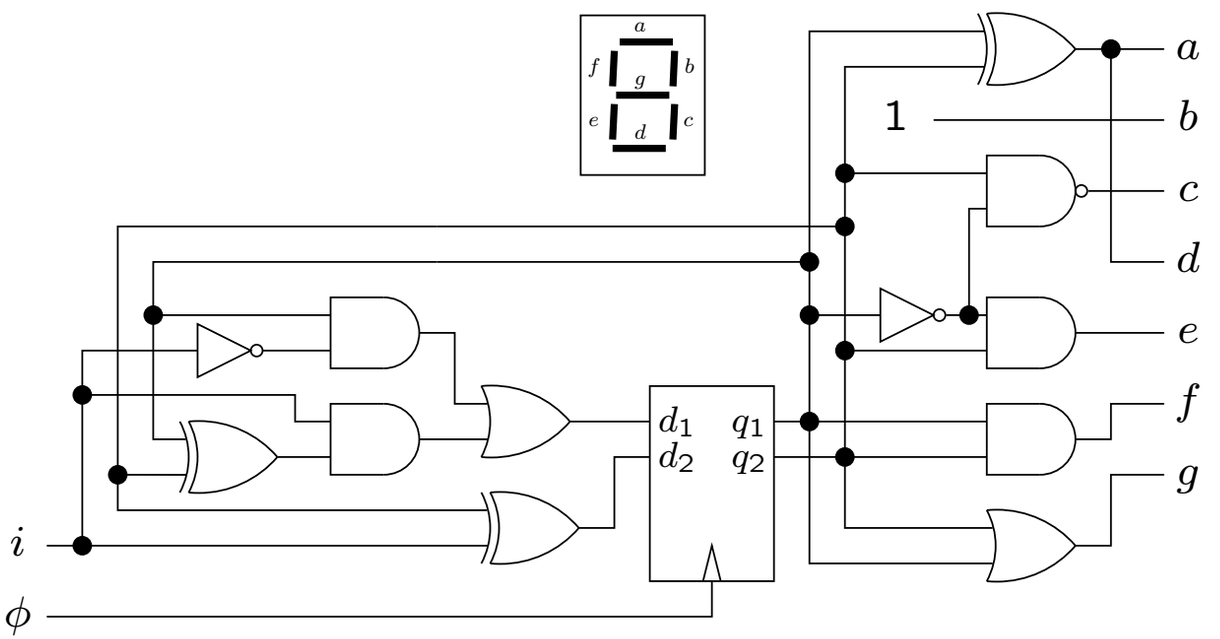
- l'état courant s_t, et
- la valeur i_t des entrées du circuit.

On a donc $o_t = f'(s_t, i_t)$, où f' est une *fonction de sortie* réalisée par le circuit combinatoire.

Remarque : La stabilité des sorties n'est garantie que pendant un certain intervalle précédant chaque coup d'horloge.

Exemple

Circuit séquentiel d'un compteur pilotant un affichage à sept segments :



Fonction de transition :

Etat courant		Entrée	Etat suivant	
q_1	q_2	i	q_1	q_2
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

Fonction de sortie :

Etat		Sorties						
q_1	q_2	a	b	c	d	e	f	g
0	0	0	1	1	0	0	0	0
0	1	1	1	0	1	1	0	1
1	0	1	1	1	1	0	0	1
1	1	0	1	1	0	0	1	1

Pourquoi des circuits séquentiels ?

- Ils permettent de traiter des données qui ne sont pas toutes disponibles simultanément.

- Ils permettent de traiter un ensemble de données dont la taille n'est pas fixée à priori.

- Ils permettent de simplifier la logique combinatoire nécessaire en permettant la décomposition d'un calcul de fonction booléenne en plusieurs étapes.