ELECINF 102 : Processeurs et Architectures Numériques

De la logique combinatoire à l'arithmétique

Tarik Graba
tarik.graba@telecom-paris.fr

Objectifs du cours

Le cours d'ELECINF102 présente :

- les grands principes de la logique combinatoire et séquentielle synchrone ;
- les architectures d' opérateurs de traitement numériques ;
- comment les mettre en œuvre pour concevoir un micro-processeur ;
- certains enjeux de l'industrie micro-électronique.

Informations utiles

Site pédagogique (et le polycopié)

- `http://sen.enst.fr/elecinf102`

Les transparents du cours :

- `http://perso.telecom-paris.fr/~graba/`

Plan

Logique Booléenne

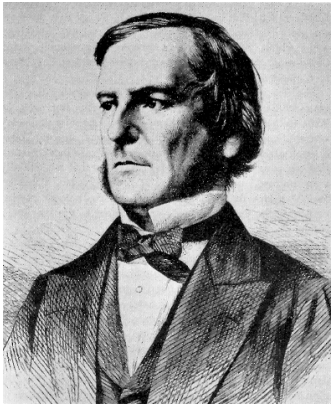
Représentation des nombres

Opérateurs Arithmétiques

Implémetation materielle

Algèbre de Boole
Formalisme de la logique

On le doit à George Boole



Crédits image : wikipedia (`http://fr.wikipedia.org/wiki/George_Boole`)

Variables et fonctions logiques

Représenter mathématiquement la logique

Variables logiques

- Une variable logique est un élément qui appartient à l'ensemble $E = \{0, 1\}$
- Ne possède que deux états possibles : 0 ou 1

Fonctions logiques

- Fonction d'une ou plusieurs variables logiques.

$$\begin{cases} E \times E \ldots \times E \to E \\ e_0, e_1, \ldots, e_n \to s = F(e_0, e_1, \ldots, e_n) \end{cases}$$

Bit : Binary Digit

L'élément binaire qui permet de représenter :

- un état logique (`vrai`/`faux`),
- mais aussi une valeur (`1`/`0`)

Fonctions logiques
Implémentation matérielle

Un support physique :

- 2 niveaux de tension ($0/5V$ ou $-12/12V$ ou ...)
- 2 niveaux de courant électrique
- Absence/présence de lumière sur une fibre

Une interprétation :

- logique (`vrai`/`faux`)
- arithmétique (`1`/`0`)

Fonctions logiques
Représentations

Plusieurs représentations :

Table de vérité : En donnant toutes les valeurs possibles pour toutes les entrées possibles.

Analytique : En donnant l'équation analytique

Graphique : En utilisant les symboles de fonctions de base

HDL :[1] Langage "informatique" de description du matériel (Hardware Description Language)

1. Pour ce cours *SystemVerilog*

Fonctions logiques
Deux catégories

Fonctions combinatoires

La sortie ne dépend que de l'état actuel des entrées

$$\forall t, s(t) = F(e_0(t), e_1(t), \ldots, e_n(t))$$

Fonctions séquentielles

La sortie dépend de l'état des entrées et du passé

$$s(t) = F(e_0(t), e_1(t), \ldots, e_n(t), e_0(t - t_1), e_1(t - t_1) \ldots)$$

Fonctions logiques
Construction

- Assemblage simples :
  - Portes logiques
- Assemblage en opérateurs
  - Arithmétique, contrôle . . .
- Circuits électroniques exécutant des fonctions complexes
  - Microprocesseur
  - ASIC [2] (Circuits spécifiques à une application)
  - FPGA [3] (Circuits logiques programmables)
  - SOC [4] (Système sur puce)

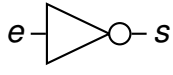
2. *Application Specific Integrated Circuit*
3. *Field Programmable Gate Array*
4. *System On Chip*

Plan

Fonctions élémentaires
L'inverseur (Not)

- La sortie est le complément de l'entrée
- La sortie vaut 1 si et seulement si l'entrée vaut 0

Symbole



Équation

$s = \overline{e}$

Table de vérité

e	s
0	1
1	0

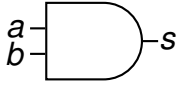
En *SystemVerilog* :

```
logic s, e;

always@(*)
     s <=  ~e;
```

Fonctions élémentaires
Le “ET” (And)

- La sortie vaut 1 si et seulement si les deux entrées valent 1
- Si l’une des entrées vaut 0 alors la sortie vaut 0

Symbole



Équation

$$s = a \cdot b$$

Table de vérité

a	b	s
0	0	0
0	1	0
1	0	0
1	1	1

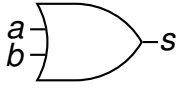
En *SystemVerilog* :

```
logic s, a, b;

always@(*)
    s <=  a & b;
```

Fonctions élémentaires
Le “OU” (Or)

- Si l’une des entrées vaut 1 alors la sortie vaut 1
- La sortie vaut 0 si et seulement si les deux entrées valent 0

Symbole



Équation

$$s = a + b$$

Table de vérité

a	b	s
0	0	0
0	1	1
1	0	1
1	1	1

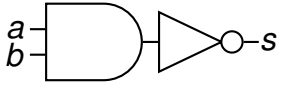
En *SystemVerilog* :

```
logic s, a, b;

always@(*)
    s <=  a | b;
```

Fonctions de base
Le “NON ET” (Nand)

- La fonction complémentaire du And
- La sortie vaut 1 si l’une des entrées est à 0

Symbole



Équation

$$s = \overline{a \cdot b}$$

Table de vérité

a	b	s
0	0	1
0	1	1
1	0	1
1	1	0

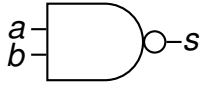
En *SystemVerilog* :

```
logic s, a, b;

always@(*)
    s <=  ~(a & b);
```

Fonctions de base
Le “NON ET” (Nand)

- La fonction complémentaire du And
- La sortie vaut 1 si l’une des entrées est à 0

Symbole



Équation

$$s = \overline{a \cdot b}$$

Table de vérité

a	b	s
0	0	1
0	1	1
1	0	1
1	1	0

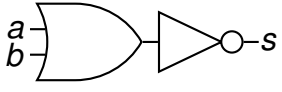
En *SystemVerilog* :

```
logic s, a, b;

always@(*)
    s <=  ~(a & b);
```

Fonctions de base
Le “NON OU” (Nor)

- La fonction complémentaire du Or
- La sortie vaut 0 si l’une des entrées est à 1

Symbole



Équation

$$s = \overline{a + b}$$

Table de vérité

a	b	s
0	0	1
0	1	0
1	0	0
1	1	0

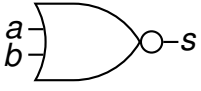
En *SystemVerilog* :

```
logic s, a, b;

always@(*)
    s <=  ~(a | b);
```

Fonctions de base
Le “NON OU” (Nor)

- La fonction complémentaire du Or
- La sortie vaut 0 si l’une des entrées est à 1

Symbole



Équation

$$s = \overline{a + b}$$

Table de vérité

a	b	s
0	0	1
0	1	0
1	0	0
1	1	0

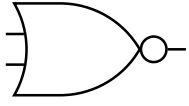
En *SystemVerilog* :

```
logic s, a, b;

always@(*)
    s <=  ~(a | b);
```

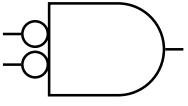
Équivalence And/Or

Théorème de De Morgan

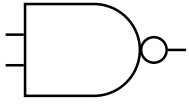
$$\overline{a+b} = \overline{a} \cdot \overline{b}$$



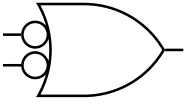
$\equiv$



$$\overline{a \cdot b} = \overline{a} + \overline{b}$$



$\equiv$



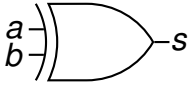
Exercice

- Comment réaliser une porte à deux entrées, dont la sortie vaut ’1’ si et seulement si les deux entrées sont différentes ?
- Comment réaliser une porte à deux entrées, dont la sortie vaut ’1’ si et seulement si les deux entrées sont identiques ?

Fonctions de base
Le “OU Exclusif” (Xor)

- La sortie vaut 1 si une seule entrée est à 1
- La sortie vaut 1 si les deux entrées sont différentes

Symbole



Équation

$$s = a \oplus b$$
$$s = a \cdot \overline{b} + \overline{a} \cdot b$$

Table de vérité

a	b	s
0	0	0
0	1	1
1	0	1
1	1	0

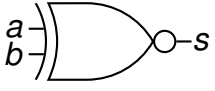
En *SystemVerilog* :

```
logic s, a, b;

always@(*)
    s <=  a ^ b;
```

Fonctions de base
Le “Non Ou exclusif” (Xnor)

- La sortie vaut 1 si les deux entrées sont identiques
- C’est la porte égalité et la fonction complémentaire du xor

Symbole



Équation

$$s = \overline{a \oplus b}$$
$$s = a \cdot b + \overline{a} \cdot \overline{b}$$

Table de vérité

a	b	s
0	0	1
0	1	0
1	0	0
1	1	1

En *SystemVerilog* :

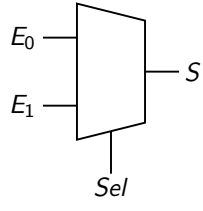
```
logic s, a, b;

always@(*)
    s <=  ~(a ^ b);
```

Plan

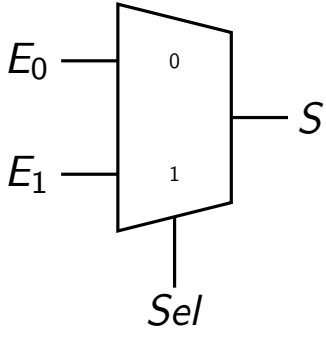
Exercices ?

Multiplexeur

- On veut réaliser une fonction d'aiguillage $2 \rightarrow 1$.
- Cette porte permet de sélectionner l'une des deux entrées en fonction d'une troisième entrée de sélection



Le multiplexeur
La fonction d'aiguillage



Sel	E_1	E_0	S
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$$S = Sel \cdot E_1 + \overline{Sel} \cdot E_0$$

Le multiplexeur
en SystemVerilog

Expression booléenne

```
logic S, E0, E1, Sel;

always@(*)
    S <= (Sel & E1) | (~Sel & E1);
```

Le multiplexeur
en SystemVerilog

Expression comportementale

```
logic S, E0, E1, Sel;

always@(*)
   if (Sel)
       S <= E1;
   else
       S <= E0;
```

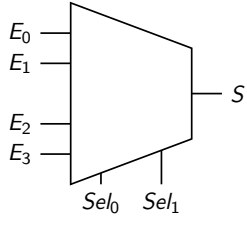
Le multiplexeur
en SystemVerilog

Expression comportementale

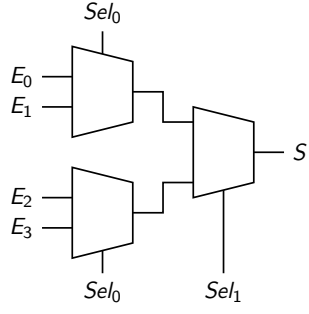
```
logic S, E0, E1, Sel;

always@(*)
   case(Sel)
     1'b0: S <= E0;
     1'b1: S <= E1;
  endcase
```

Le multiplexeur

Fonction d'aiguillage à quatre entrées



4 entrées $\Rightarrow$ 2 entrées de sélection



Peut être réalisé à partir de 3 mux 2 vers 1

$$S = \overline{Sel_1} \cdot \overline{Sel_0} \cdot E_0 + \overline{Sel_1} \cdot Sel_0 \cdot E_1 + Sel_1 \cdot \overline{Sel_0} \cdot E_2 + Sel_0 \cdot Sel_1 \cdot E_3;$$

Le multiplexeur
en SystemVerilog

Expression comportementale

```
logic S, Sel0, Sel1;
logic E0, E1, E2, E3;

always@(*)
   case({Sel1, Sel0}) // 2bits concaténés
     2'b00: S <= E0;
     2'b01: S <= E1;
     2'b10: S <= E2;
     2'b11: S <= E3;
  endcase
```

Le multiplexeur
Généralisation

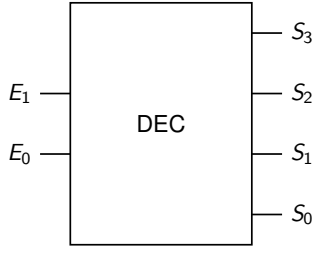
Pour un multiplexeur à n entrées ($n = 2^p$ une puissance de 2) :

- Il faut $p = \log_2(n)$ entrées de sélection
- Il peut être réalisé avec $n - 1$ multiplexeurs à 2 entrées organisés en p couches

Aussi, connaissant la table de vérité d'une fonction combinatoire de p entrées, nous pouvons l'implémenter en positionnant n constantes (1 ou 0) aux entrées d'un multiplexeur.

Le décodeur

- n entrées 2^n sorties.
- Une seule sortie à 1 toutes les autres à 0.



E_0	E_1	S_3	S_2	S_1	S_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

- La valeur de l'entrée E représente l'indice de la sortie active.

Le décodeur
en SystemVerilog

Expression comportementale

```
logic S0, S1, S2, S3;
logic E0, E1;

always@(*)
  begin
     S0 <= ~E1 & ~E0;
     S1 <= ~E1 &  E0;
     S2 <=  E1 & ~E0;
     S3 <=  E1 &  E0;
  end
```

Le décodeur
en SystemVerilog

Expression comportementale

```
logic S0, S1, S2, S3;
logic [1:0]E; // bus de 2bits

always@(*)
  begin
     S0 <= ~E[1] & ~E[0];
     S1 <= ~E[1] &  E[0];
     S2 <=  E[1] & ~E[0];
     S3 <=  E[1] &  E[0];
  end
```

Le décodeur
en SystemVerilog

Expression comportementale

```
logic S0, S1, S2, S3;
logic [1:0]E; // bus de 2bits

always@(*)
  begin
     S0 <= (E == 2'b00);
     S1 <= (E == 2'b01);
     S2 <= (E == 2'b10);
     S3 <= (E == 2'b11);
  end
```

Le décodeur
Généralisation

On peut exprimer toute fonction combinatoire sous la forme de somme de mintermes (OUs de ETs) :

$$F(e_0, e_1, \ldots e_n) = \sum \prod e_i$$

Par exemple : $F(E) = E_0 \cdot \overline{E_1} + \overline{E_0} \cdot E_1$ (le OU Exclusif)

Il suffit alors de faire le ou entre les sorties du décodeur correspondantes.

Plan

Logique Booléenne

Représentation des nombres

Opérateurs Arithmétiques

Implémetation materielle

Plan

Un entier positif N dans une base b se représente par un vecteur $(a_{n-1}, a_{n-2}, \ldots, a_1, a_0)$ tel que :

$$N = a_{n-1} \cdot b^{n-1} + a_{n-2} \cdot b^{n-2} + \ldots + a_1 \cdot b^1 + a_0 \cdot b^0$$

Où :

- a_{n-1} est le chiffre le plus significatif
- a_0 est le chiffre le moins significatif
- a_i appartient à un ensemble de b symboles valant de 0 à $b-1$

Bases souvent utilisées

- $b = 10$
  - Représentation Décimale
  - $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $b = 16$
  - Représentation Hexadécimale
  - $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$
- $b = 8$
  - Représentation Octale
  - $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7\}$
- $b = 2$
  - Représentation Binaire
  - $a_i \in \{0, 1\}$ est un bit (binary digit)

Représentation binaire
Les nombres naturels

Un entier positif N en base 2 se représente par un vecteur $(a_{n-1}, a_{n-2}, \ldots, a_1, a_0)$ tel que :

$$N = a_{n-1} \cdot 2^{n-1} + a_{n-2} \cdot 2^{n-2} + \ldots + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

Où :

- a_i est un bit
- a_i appartient à un ensemble de 2 symboles valant 0 ou 1

Exemples

- Représentez 2, 3, 4 en binaire
- Représentez 54 en binaire

binaire $\leftrightarrow$ hexadécimal

hexadécimal = base 16

$$N = \sum_{i=0}^{n-1} a_i \cdot 2^i$$

Profitons du fait que $2^4 = 16$. [1]

$$N = \sum_{k=0}^{n/4-1} (a_{4k+3} \cdot 8 + a_{4k+2} \cdot 4 + a_{4k+1} \cdot 2 + a_{4k}) \cdot 16^k$$

Passer à une représentation hexadécimale permet d'avoir une représentation compacte !

1. *Pour simplifier l'expression le nombre de bit n est supposé être multiple de 4*

Exemples

- Représentez 54 en hexadécimal
- Représentez 254 en hexadécimal
  - En déduire la représentation en binaire

Exemples

- Représentez 13 et 29 en binaire :
  - en utilisant 4 bits
  - en utilisant 5 bits

Représentation binaire
Taille finie

Physiquement, le nombre de bits ne peut être infini (*par exemple dans un processeur*)

Pour n **bits** :

- Il y a 2^n valeurs représentables
- On ne peut représenter que les nombres dans l'intervalle $[0, 2^n - 1]$.
- L'arithmétique sur ces représentations est faite modulo 2^n

Plan

Représentation binaire
Exemple sur 4 bits

- Avec 4 bits on peut représenter les nombres allant de 0 à $15 = 2^4 - 1$
- L'arithmétique est alors modulo $2^4 = 16$
- Par exemple :
  - $15 + 1 = 0$
  - $0 - 1 = 15$

Décimal	Binaire
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Représentation binaire
Exemple sur 4 bits

- Avec 4 bits on peut représenter les nombres allant de 0 à $15 = 2^4 - 1$
- L'arithmétique est alors modulo $2^4 = 16$
- Par exemple :
  - $15 + 1 = 0$
  - $0 - 1 = 15$
- Comment conserver le même comportement et représenter des nombres signés ?

Décimal	Binaire
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Représentation binaire
Exemple sur 4 bits

- Avec 4 bits on peut représenter les nombres allant de 0 à $15 = 2^4 - 1$
- L'arithmétique est alors modulo $2^4 = 16$
- Par exemple :
  - $15 + 1 = 0$
  - $0 - 1 = 15$
- On interprète 1111 comme -1 au lieu de 15 !

Signé	Binaire
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
-1	1111

Représentation binaire
Exemple sur 4 bits

- Avec 4 bits on peut représenter les nombres allant de 0 à $15 = 2^4 - 1$
- L'arithmétique est alors modulo $2^4 = 16$
- Par exemple :
  - $15 + 1 = 0$
  - $0 - 1 = 15$
- On convient que la moitié des nombres sont interprétés comme négatifs !

Signé	Binaire
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
−8	1000
−7	1001
−6	1010
−5	1011
−4	1100
−3	1101
−2	1110
−1	1111

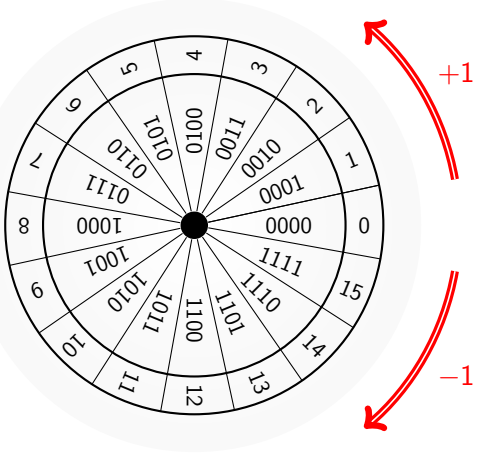
Représentation binaire
Exemple sur 4 bits

- Avec 4 bits on peut représenter les nombres allant de 0 à $15 = 2^4 - 1$
- L'arithmétique est alors modulo $2^4 = 16$
- Par exemple :
  - $15 + 1 = 0$
  - $0 - 1 = 15$
- On convient que la moitié des nombres sont interprétés comme négatifs !

Signé	Binaire
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
$-8 = 8 - 16$	1000
$-7 = 9 - 16$	1001
$-6 = 10 - 16$	1010
$-5 = 11 - 16$	1011
$-4 = 12 - 16$	1100
$-3 = 13 - 16$	1101
$-2 = 14 - 16$	1110
$-1 = 15 - 16$	1111

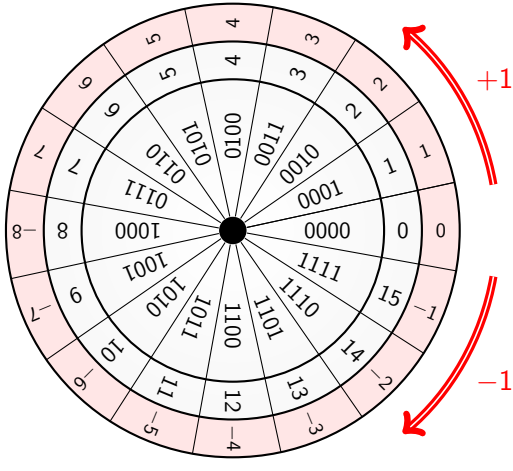
Représentation binaire des nombres

Les nombres entiers signés : exemple sur 4 bits



Représentation binaire des nombres

Les nombres entiers signés : exemple sur 4 bits



La valeur d'un nombre en CA2 Sur *n* bits

$$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

- C'est une interprétation du mot binaire.
- Le bit de poids fort représente le signe
  - $0 \rightarrow$ nombre positif $= \sum_{i=0}^{n-2} a_i 2^i$
  - $1 \rightarrow$ nombre négatif $= -2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$
- Permet de représenter les nombres dans l'intervalle $\mathbf{[-2^{n-1}, 2^{n-1} - 1]}$

Représentation binaire

Entiers relatifs sur 4 bits

- Avec 4 bits on peut représenter les nombres allant de -8 à 7
- càd $[-(2^3), 2^3 - 1]$

Décimal	Binaire	Signé
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	-8
9	1001	-7
10	1010	-6
11	1011	-5
12	1100	-4
13	1101	-3
14	1110	-2
15	1111	-1

Exemples

- Représentez -8 et +8
  - en utilisant 4 bits
  - en utilisant 5 bits

Exemples

- Représentez -8 et +8
  - en utilisant 4 bits
  - en utilisant 5 bits
- Représentez -1
  - en utilisant 1 bit
  - en utilisant 2 bits
  - en utilisant 3 bits ...

Représentation binaire des nombres
Extension de signe pour le CA2

Soit $N = (a_{n-1}, a_{n-2}, \ldots, a_0)_{\text{CA2}/n}$ un entier relatif représenté en CA2 sur n bits.

Comment représenter N sur un $n+1$ bits.

$$N = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i = -a_{n-1}2^{n-1} \cdot (2-1) + \sum_{i=0}^{n-2} a_i 2^i$$

$$= -a_{n-1}2^n + \sum_{i=0}^{n-1} a_i 2^i$$

D'où $N = (a_{n-1}, a_{n-1}, a_{n-2}, \ldots, a_0)_{\text{CA2}/n+1}$
On a dupliqué le bit de poids fort, on parle d'extension de signe.

Représentation binaire des nombres

Les nombres décimaux en virgule fixe

Un nombre décimal D en base 2 peut être approximé un par vecteur $(a_{n-1}, a_{n-2}, \ldots, a_1, a_0, a_{-1} \ldots a_{-m})$ tel que :

$$D = a_{n-1} \cdot 2^{n-1} + a_{n-2} \cdot 2^{n-2} + \ldots + a_1 \cdot 2^1 + a_0 \cdot 2^0 + a_{-1} \cdot 2^{-1} + \ldots a_{-m} \cdot 2^{-m}$$

Où :

- Division implicite par 2^m
- $(a_{n-1}, \ldots a_0)$ est la partie entière
- $(a_{-1}, \ldots a_{-m})$ est la partie fractionnaire
- 2^{-m} représente la précision de cette approximation

Exemples

- Représentez 0.5, 3.625

Exemples

- Représentez 0.5, 3.625
- Représentez 0.6
  - en utilisant 1 bit
  - en utilisant 3 bits
  - en utilisant 5 bits

Plan

Logique Booléenne

Représentation des nombres

Opérateurs Arithmétiques

Implémetation materielle

Plan

Addition

Exemple

Faire une addition en binaire sur 4 bits...

Addition

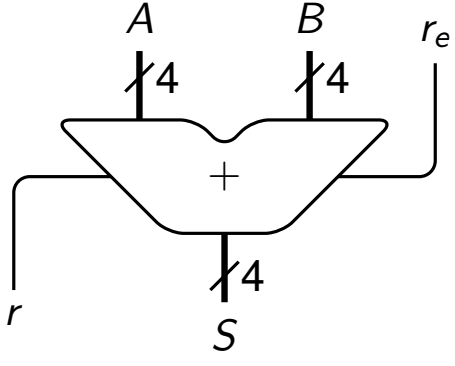
Exemple

Faire une addition en binaire sur 4 bits...

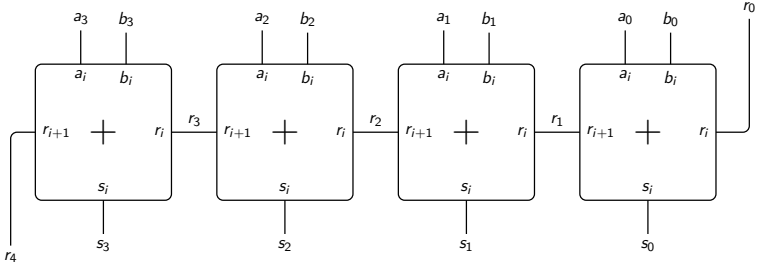
Décomposition de l'addition

L'addition peut être décomposée en plusieurs additions élémentaires sur 1 bit

Additionneur à propagation de retenue



Additionneur à propagation de retenue



Additionneur complet sur 1 bit

Arithmétiquement

$$a_i + b_i + r_i = 2 \cdot r_{i+1} + s_i$$

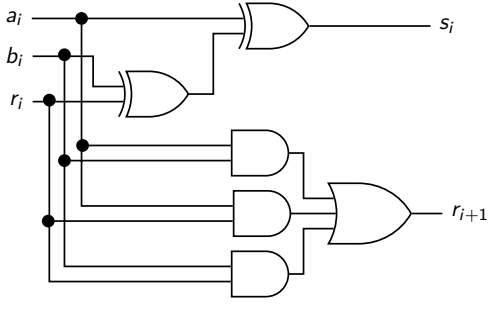
Table de vérité

a_i	b_i	r_i	r_{i+1}	s_i	Décimal
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	1	0	2
1	0	0	0	1	1
1	0	1	1	0	2
1	1	0	1	0	2
1	1	1	1	1	3

Additionneur complet sur 1 bit

$$s_i = a_i \oplus b_i \oplus r_i$$

$$r_{i+1} = a_i \cdot b_i + a_i \cdot r_i + b_i \cdot r_i$$



Plan

Soustracteur complet sur 1 bit

Arithmétiquement

$$a_i - b_i - r_i = -2 \cdot r_{i+1} + s_i$$

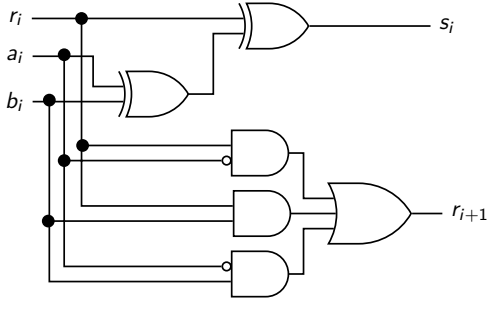
Table de vérité

a_i	b_i	r_i	r_{i+1}	s_i	Décimal
0	0	0	0	0	0
0	0	1	1	1	-1
0	1	0	1	1	-1
0	1	1	1	0	-2
1	0	0	0	1	1
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	1	1	-1

Soustracteur complet sur 1 bit

$$s_i = a_i \oplus b_i \oplus r_i$$

$$r_{i+1} = \overline{a_i} \cdot b_i + \overline{a_i} \cdot r_i + b_i \cdot r_i$$



Exercice

- Pour un bit b exprimez “logiquement” en fonction de b le résultat du calcul arithmétique $1 - b$.
- A est un nombre signé codé en CA2 sur n bits. Montrez que $-A = \overline{A} + 1$ (ici $+$ est l'addition arithmétique)
- Proposez alors l'architecture d'un soustracteur utilisant un additionneur à propagation de retenue et des portes logique supplémentaires.
- Proposez l'architecture d'un additionneur/soustracteur commandé.

Plan

Nombres positifs

Pour deux nombres A et B représentés sur n bits nous avons :

$$\begin{aligned} A &\leq 2^n - 1 \\ B &\leq 2^n - 1 \\ A + B &\leq 2^{n+1} - 2 < 2^{n+1} \end{aligned}$$

$A + B$ est toujours représentable sur $n + 1$ bits.

exemple :

$$\begin{array}{cccccc} & & 1 & 1 & 1 & (7) \\ + & & 0 & 0 & 1 & (1) \\ \hline = & 1 & 0 & 0 & 0 & (8) \end{array}$$

Dynamique de l'addition
en SystemVerilog

Les opérateurs arithmétiques existent !

```
logic [3:0] A, B;
logic [3:0] S;

always@(*)
   S <= A + B; // résultat sur tronqué sur 4bits
```

Dynamique de l'addition
en SystemVerilog

Les opérateurs arithmétiques existent !

```
logic [3:0] A, B;
logic [4:0] S;

always@(*)
   S <= A + B; // résultat sur 5bits
```

Dynamique de l'addition
en SystemVerilog

Les opérateurs arithmétiques existent !

```
logic [3:0] A, B;
logic [3:0] S;
logic r;

always@(*)
   {r,S} <= A + B; // résultat sur (4+1)bits
```

CA2

Pour deux nombres A et B représentés en CA2 sur n bits nous avons :

$$\begin{array}{rcccl} -2^{n-1} & \leq & A & \leq & 2^{n-1}-1 \\ -2^{n-1} & \leq & B & \leq & 2^{n-1}-1 \\ -2^{n} & \leq & A+B & \leq & 2^{n}-2 < 2^{n} \end{array}$$

$A + B$ est toujours représentable sur $n + 1$ bits.

Dynamique de l'addition

Dépassement de capacité pour l'addition

Exemples en CA2

					non signé	*CA2*
		1	1	1	7	-1
$+$		0	0	1	1	1
$=$	1	0	0	0	8	0 ou -8?

				non signé	*CA2*
	0	1	1	3	3
$+$	0	0	1	1	1
$=$	1	0	0	4	-4

				non signé	*CA2*
	1	1	1	7	-1
$+$	1	0	0	4	-4
$=$ 1	0	1	1	11	$+3$ + retenue ?

En CA2 l'interprétation de la retenue n'est pas la même que pour les nombres non signés.

Dynamique de l'addition

Dépassement de capacité pour l'addition

CA2

Une solution simple pour toujours avoir le bon résultat est de d'abord étendre les nombres sur un bit de plus puis faire la somme.
La retenue produite au-delà du bit ajouté n'est pas prise en compte.

		1	1	1	1	−1
+		1	1	0	0	−4
=	1	1	0	1	1	−5

		1	1	1	1	−1
+		0	0	0	1	1
=	1	0	0	0	0	0

Plan

Logique Booléenne

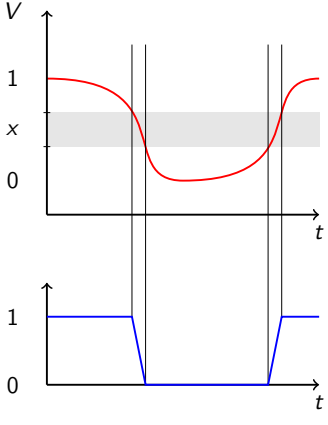
Représentation des nombres

Opérateurs Arithmétiques

Implémetation materielle

Plan

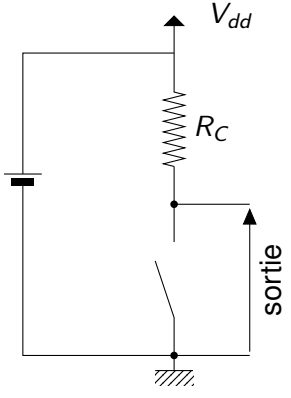
signal numérique



- Utilisation de la tension comme support.
  - 2 intervalles $\Rightarrow$
  - 2 valeurs logiques.
- Qui évolue dans le temps.
  - Des temps de transition non nuls.
  - On ne l'interprète que quand il est stable.

Génération d'un signal électrique binaire

Par exemple :

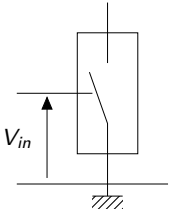


- Interrupteur fermé
  - $\rightarrow 0V$ en sortie
- Interrupteur ouvert
  - $\rightarrow V_{dd}$ en sortie

tension	niveau logique
$0V$	0
V_{dd}	1

Génération d'un signal électrique binaire

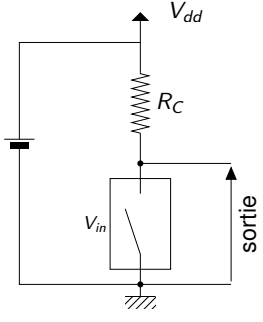
à partir d'un signal électrique...



- Interrupteur commandé :
  - Si $V_{in} < V_{ref}$ alors l'interrupteur est ouvert
  - Si $V_{in} > V_{ref}$ alors l'interrupteur est fermé
- Cet interrupteur peut être :
  - Un relais électromagnétique
  - Un tube à vide
  - Un transistor

Plan

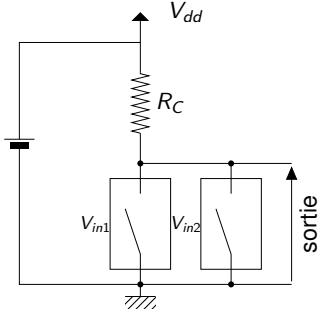
Implémentation des portes logiques
Fonction Non



V_{in}	V_s	In	Sortie
$< V_{ref}$	V_{dd}	0	1
$> V_{ref}$	$0V$	1	0

- La fonction Non
- La sortie vaut 0 ssi l'entrée vaut 1

Implémentation des portes logiques
Fonction Non–Ou



V_{in1}	V_{in2}	V_s	In_1	In_1	Sortie
$< V_{ref}$	$< V_{ref}$	V_{dd}	0	0	1
$< V_{ref}$	$> V_{ref}$	$0V$	0	1	0
$> V_{ref}$	$< V_{ref}$	$0V$	1	0	0
$> V_{ref}$	$> V_{ref}$	$0V$	1	1	0

- Fonction Non–Ou
- la sortie vaut 0 si l'une des entrées vaut 1

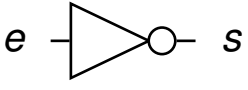
Plan

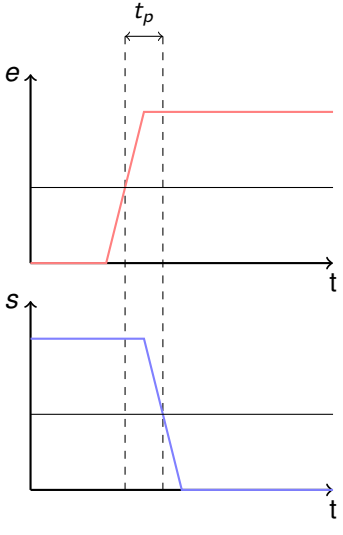
Temps de propagation d'une porte

- Les portes logiques respectent les lois de la physique. Les changements d'état ne peuvent pas être instantanés.
- Le **temps de propagation** est le temps entre le moment où les entrées d'une porte changent d'état et la stabilisation de la valeur de sa sortie.
- La valeur de la sortie n'est valide qu'après ce temps.

Temps de propagation d'une porte simple

Exemple simple : Un inverseur



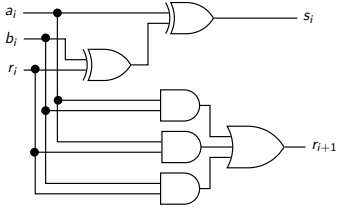


Temps de propagation d'une porte
Règles pour les portes complexes

- Les portes logiques de base sont pré-caractérisées.
- Pour une technologie particulière, on connait le temps de propagation des portes de base.
- À partir de ces tables on calcule le temps de propagation des portes complexes en faisant la somme des temps individuels des portes qui se suivent.
- Le temps de propagation d'une porte complexe est le temps de propagation du chemin le plus long.

Temps de propagation d'une porte

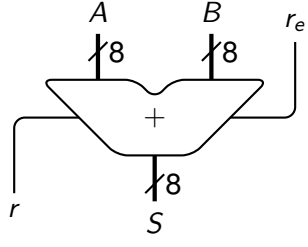
Exemple : Full adder



- Considérons que le temps de propagation est de :
  - 1ns pour les portes and et or
  - 2ns pour les portes xor
- Quel est le temps de propagation des entrées vers chaque sortie ?

Temps de propagation d'une porte

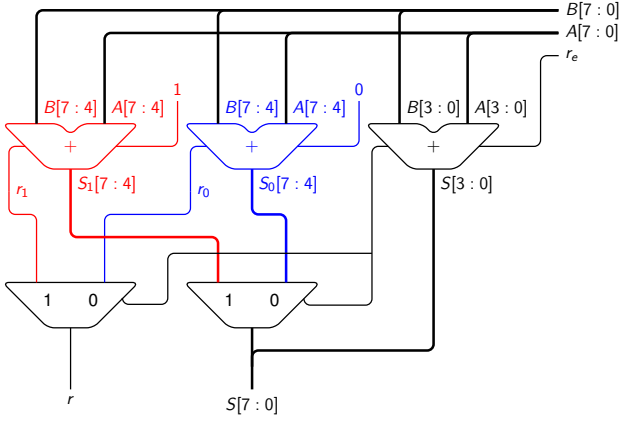
Exemple : Carry adder



- Quel est le temps de calcul maximum d'un additionneur 8bits à propagation de retenue ?

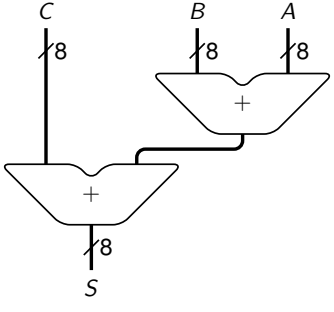
Temps de propagation d'une porte
Exemple 2

- Quelle est la fonction de ce montage ?
- Quels sont ses avantages et inconvénients ?



Temps de propagation d'une porte
Exemple 3

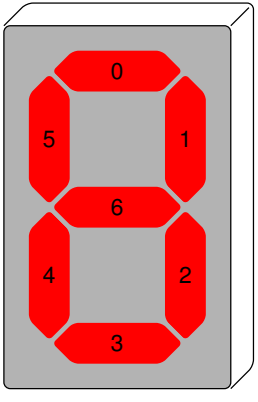
- Quel est le temps de propagation de cette addition ?



Exercice

- Proposez 2 solutions pour construire un opérateur qui, pour 2 nombres non signés représentés sur 8 bits, renvoie le maximum.
  - L'une utilisant un opérateur arithmétique
  - L'autre comparant bit à bit les 2 nombres (en commençant par le poids fort)
- Quel est le temps de propagation dans chaque cas.
- Proposez dans chaque cas une architecture permettant de comparer 4 nombres. Quel est alors temps de propagation.

À préparer pour la prochaine séance !



Le décodeur 7 segments

- Permet d'afficher de l'hexadécimal (0,1,...,9,A,B...,F)
- L'entrée est sur 4 bits ($0 \rightarrow F$)
- La sortie est sur 7 bits
  - Chaque bit contrôle un segment
  - Si le bit est à 0 le segment est allumé
- Donnez les équations de chaque sortie