2E200 : Electronique Numérique, Combinatoire et Séquentielle

Bertrand Granado

Licence E^2A

Hiver 2019

Plan

SCIENCES SORBONNE UNIVERSITÉ

Plan

Plan

Plan

SCIENCES SORBONNE UNIVERSITÉ

Plan

SCIENCES SORBONNE UNIVERSITÉ

Logique Combinatoire

- Transposition de l'algèbre de Boole à *l'électronique*

Logique Combinatoire

- Transposition de l'algèbre de Boole à *l'électronique*
- Rendu possible grâce au composant tel que le transistor commandé en tension

Logique Combinatoire

- Transposition de l'algèbre de Boole à *l'électronique*
- Rendu possible grâce au composant tel que le transistor commandé en tension
- Le domaine de validité de ce qui suit est l'électronique numérique

Logique Combinatoire

Définition :

Un circuit électronique est dit combinatoire si ses

Logique Combinatoire

Définition :
Un circuit électronique est dit combinatoire si ses sorties sont déterminées par la

Logique Combinatoire

Définition :

Un circuit électronique est dit combinatoire si ses sorties sont déterminées par la combinaison de ses

Logique Combinatoire

Définition :

Un circuit électronique est dit combinatoire si ses sorties sont déterminées par la combinaison de ses variables d'entrées et ceci après

Logique Combinatoire

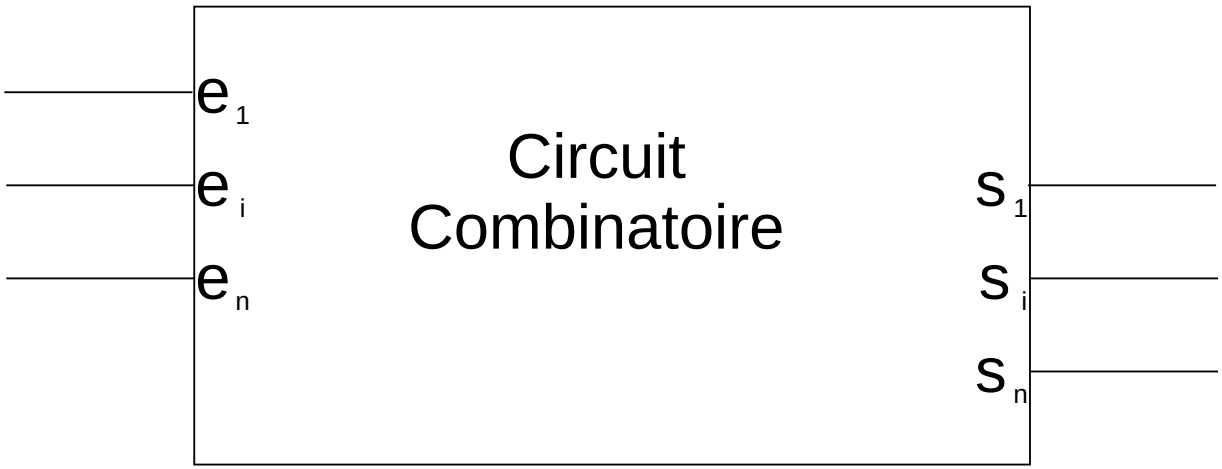
Définition :

Un circuit électronique est dit combinatoire si ses sorties sont déterminées par la combinaison de ses variables d'entrées et ceci après un temps fini.

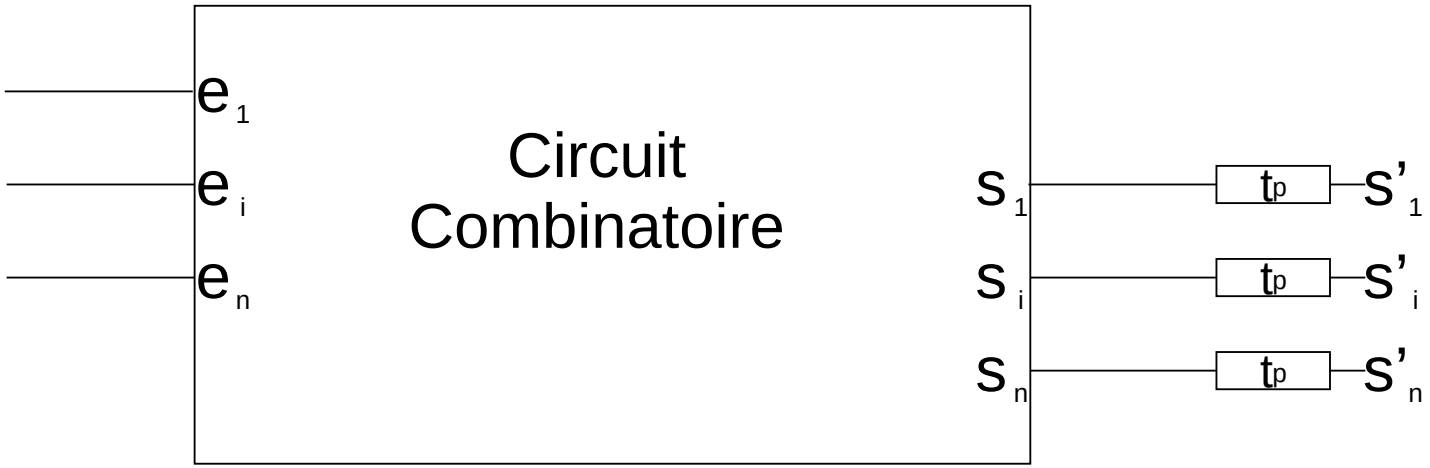
Logique Combinatoire

Définition :
Un circuit électronique est dit combinatoire si ses sorties sont déterminées par la combinaison de ses variables d'entrées et ceci après un temps fini. L'état d'un système est donc défini par la combinaison des variables $e_1, \ldots, e_i, \ldots, e_n$.

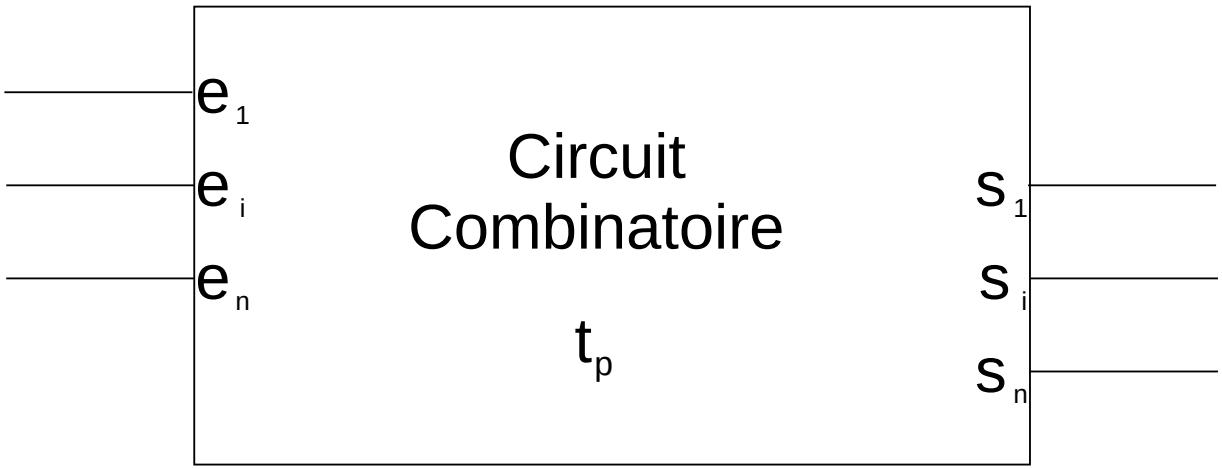
Logique Combinatoire



Logique Combinatoire



Logique Combinatoire



Les Aléas Temporels

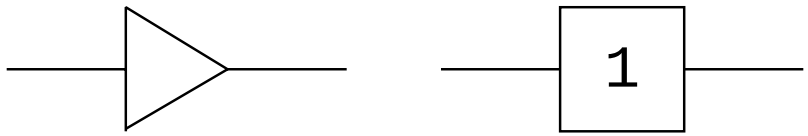
- $a + \overline{a} = 0$?

Logique Combinatoire - Opérateurs de base

- Fonctions à une variable

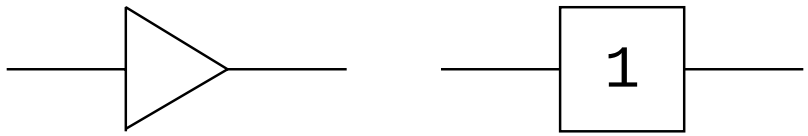
Logique Combinatoire - Opérateurs de base

- Fonctions à une variable
- Buffer (identité) : $s = a$

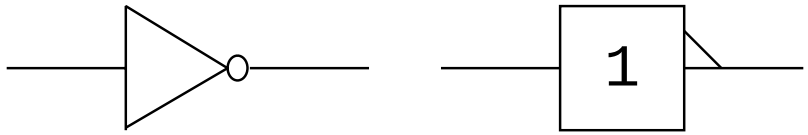


Logique Combinatoire - Opérateurs de base

- Fonctions à une variable
- Buffer (identité) : $s = a$



- Inverseur : $s = \overline{a}$

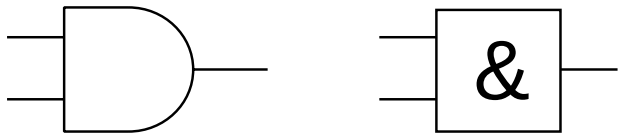


Logique Combinatoire - Opérateurs de base

- Fonctions à deux variables

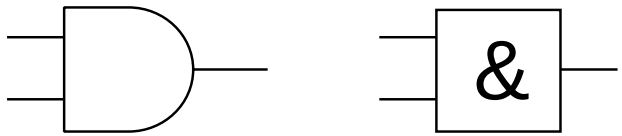
Logique Combinatoire - Opérateurs de base

- Fonctions à deux variables
- ET (AND) : $s = a.b$

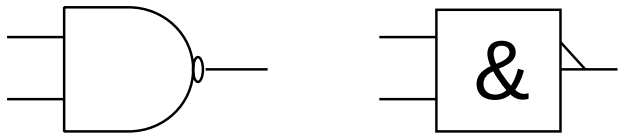


Logique Combinatoire - Opérateurs de base

- Fonctions à deux variables
- ET (AND) : $s = a.b$



- NON-ET (NAND) : $s = \overline{a.b}$

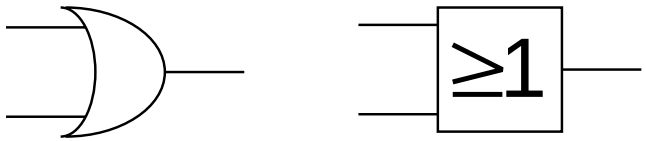


Logique Combinatoire - Opérateurs de base

- Fonctions à deux variables

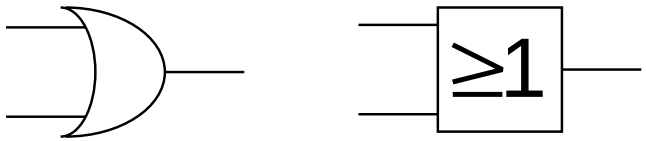
Logique Combinatoire - Opérateurs de base

- Fonctions à deux variables
- OU (OR) : $s = a + b$

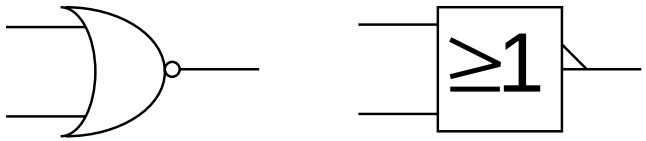


Logique Combinatoire - Opérateurs de base

- Fonctions à deux variables
- OU (OR) : $s = a + b$



- NON-OU (NOR) : $s = \overline{a + b}$

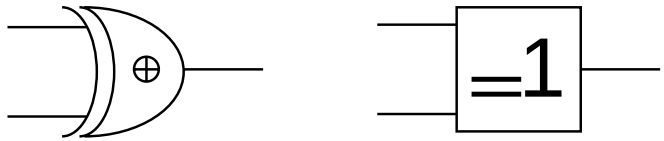


Logique Combinatoire - Opérateurs de base

- Fonctions à deux variables

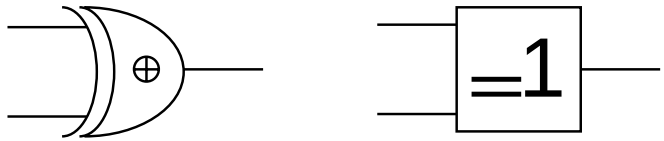
Logique Combinatoire - Opérateurs de base

- Fonctions à deux variables
- OU-EXCLUSIF : $s = a \oplus b = a\bar{b} + \bar{a}b$

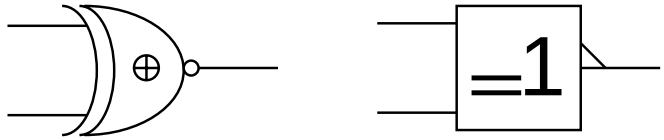


Logique Combinatoire - Opérateurs de base

- Fonctions à deux variables
- OU-EXCLUSIF : $s = a \oplus b = a\bar{b} + \bar{a}b$



- NON-OU-EXCLUSIF : $s = \overline{a \oplus b} = \overline{a\bar{b} + \bar{a}b} = ab + \bar{a}\bar{b}$



Plan

introduction

- Composées à partir des opérateurs de base

introduction

- Composées à partir des opérateurs de base
- Conditionnement de données

introduction

- Composées à partir des opérateurs de base
- Conditionnement de données
- Contrôle de données

introduction

- Composées à partir des opérateurs de base
- Conditionnement de données
- Contrôle de données
- Définies par leur table de vérité

Fonction Egalité

- Egalité 2 bits

Fonction Egalité

- Egalité 2 bits

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

Fonction Egalité

- Egalité 2 bits

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

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

Egalité - VHDL

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity egalite is
port( a,b : in std_logic;
        s : out std_logic);
end entity egalite;

architecture flot of egalite is
begin
    s <= not(a xor b);
end architecture flot;
```

Fonction Egalité

- Egalité 2 mots de 2 bits
- $a = a_1, a_0$ et $b = b_1, b_0$

Fonction Egalité

- Egalité 2 mots de 2 bits
- $a = a_1, a_0$ et $b = b_1, b_0$

a_1	a_0	b_1	b_0	s
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0

a_1	a_0	b_1	b_0	s
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Fonction Egalité

- Egalité 2 mots de 2 bits
- $a = a_1, a_0$ et $b = b_1, b_0$

b_1b_0 \ a_1a_0	00	01	11	10
00	1			
01		1		
11			1	
10				1

Fonction Egalité

- Egalité 2 mots de 2 bits
- $a = a_1, a_0$ et $b = b_1, b_0$

b_1b_0 \ a_1a_0	00	01	11	10
00	1			
01		1		
11			1	
10				1

Fonction Egalité

- Egalité 2 mots de 2 bits
- $a = a_1, a_0$ et $b = b_1, b_0$

b_1b_0 \ a_1a_0	00	01	11	10
00	1			
01		1		
11			1	
10				1

Fonction Egalité

- Egalité 2 mots de 2 bits
- $a = a_1, a_0$ et $b = b_1, b_0$

- $s = \overline{a_1}.\overline{a_0}.\overline{b_1}.\overline{b_0} + a_1\overline{a_0}.b_1.\overline{b_0} + \overline{a_1}.a_0.\overline{b_1}.b_0 + a_1.a_0.b_1.b_0$

Fonction Egalité

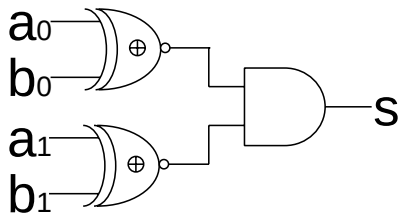
- Egalité 2 mots de 2 bits
- $a = a_1, a_0$ et $b = b_1, b_0$

- $s = \overline{a_1}.\overline{a_0}.\overline{b_1}.\overline{b_0} + a_1\overline{a_0}.b_1.\overline{b_0} + \overline{a_1}.a_0.\overline{b_1}.b_0 + a_1.a_0.b_1.b_0$
- $s = (\overline{a_1 \oplus b_1})(\overline{a_0 \oplus b_0})$

Fonction Egalité

- Egalité 2 mots de 2 bits
- $a = a_1, a_0$ et $b = b_1, b_0$

- $s = \overline{a_1}.\overline{a_0}.\overline{b_1}.\overline{b_0} + a_1\overline{a_0}.b_1.\overline{b_0} + \overline{a_1}.a_0.\overline{b_1}.b_0 + a_1.a_0.b_1.b_0$
- $s = (\overline{a_1 \oplus b_1})(\overline{a_0 \oplus b_0})$



Fonction Egalité

- Egalité 2 mots de 2 bits
- $a = a_1, a_0$ et $b = b_1, b_0$

- $s = \overline{a_1}.\overline{a_0}.\overline{b_1}.\overline{b_0} + a_1\overline{a_0}.b_1.\overline{b_0} + \overline{a_1}.a_0.\overline{b_1}.b_0 + a_1.a_0.b_1.b_0$
- $s = (\overline{a_1 \oplus b_1})(\overline{a_0 \oplus b_0})$
- Egalité de 2 mots de n bits :

Fonction Egalité

- Egalité 2 mots de 2 bits
- $a = a_1, a_0$ et $b = b_1, b_0$

- $s = \overline{a_1}.\overline{a_0}.\overline{b_1}.\overline{b_0} + a_1\overline{a_0}.b_1.\overline{b_0} + \overline{a_1}.a_0.\overline{b_1}.b_0 + a_1.a_0.b_1.b_0$
- $s = (\overline{a_1 \oplus b_1})(\overline{a_0 \oplus b_0})$
- Egalité de 2 mots de n bits :
  $s = (\overline{a_{n-1} \oplus b_{n-1}})(\overline{a_{n-2} \oplus b_{n-2}})(...)(\overline{a_1 \oplus b_1})(\overline{a_0 \oplus b_0})$

Egalité - VHDL

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity egalite_2bits is
port( a,b : in std_logic_vector(1 downto 0);
        s : out std_logic);
end entity egalite_2bits;

architecture flot of egalite_2bits is
begin
 s <= not(a(1) xor b(1)) and not(a(0) xor b(0));
end architecture flot;
```

Egalité - VHDL

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity egalite_2bits is
port( a,b : in std_logic_vector(1 downto 0);
        s : out std_logic);
end entity egalite_2bits;

architecture comp of egalite_2bits is
begin
    process(a,b) is
    begin
      if a= b then
         s<='1';
      else
         s<='0';
     end if;
    end process;
end architecture comp;
```

Multiplexeurs

- Multiplexeur = Aiguillage

Multiplexeurs

- Multiplexeur = Aiguillage
- Une commande choisie l'entrée

Multiplexeurs

- Multiplexeur = Aiguillage
- Une commande choisie l'entrée
- Entrée choisie recopiée sur la sortie

Multiplexeurs

- Multiplexeur = Aiguillage
- Une commande choisie l'entrée
- Entrée choisie recopiée sur la sortie
- Partie Commande : p bits

Multiplexeurs

- Multiplexeur = Aiguillage
- Une commande choisie l'entrée
- Entrée choisie recopiée sur la sortie
- Partie Commande : p bits
- Partie Donnée : $2^p = n$ entrées, 1 sortie

Multiplexeurs 2 vers 1 - Table de vérité

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

Multiplexeurs 2 vers 1 - Table de vérité

Multiplexeurs 2 vers 1 - Table de vérité

sel \ a b	0 0	0 1	1 1	1 0
0			1	1
1		1	1	

Multiplexeurs 2 vers 1 - Table de vérité

sel \ a b	0 0	0 1	1 1	1 0
0			1	1
1		1	1	

Multiplexeurs 2 vers 1 - Table de vérité

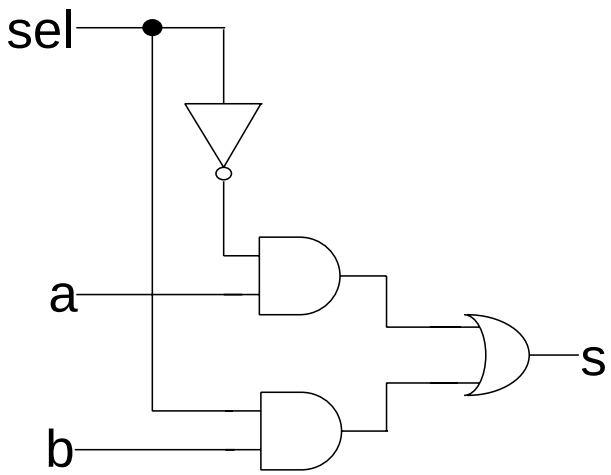
- $s = \overline{sel}.a + sel.b$

Multiplexeurs 2 vers 1

- Schéma

Multiplexeurs 2 vers 1

- Schéma



VHDL - mux2v1

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity m2v1 is
port(a,b,sel : in std_logic;
          s: out std_logic);
end entity m2v1;

architecture flot of m2v1 is
begin
    s <= (a and not(sel)) or (b and sel);
end architecture flot;
```

VHDL - mux2v1

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity m2v1 is
port(a,b,sel : in std_logic;
          s: out std_logic);
end entity m2v1;

architecture flot of m2v1 is
begin
    s <= a when sel='0' else b; -- s prend la valeur de a si sel = '0'
                                -- sinon si sel <> '0' s prend la valeur de b
end architecture flot;
```

Multiplexeurs 4 vers 1

- 4 données et 2 commandes

Multiplexeurs 4 vers 1

- 4 données et 2 commandes
- $2^6 = 64$ lignes dans la table de vérité

Multiplexeurs 4 vers 1

- 4 données et 2 commandes
- $2^6 = 64$ lignes dans la table de vérité
- Toutes les lignes ne sont pas intéressantes

Multiplexeurs 4 vers 1

- 4 données et 2 commandes
- $2^6 = 64$ lignes dans la table de vérité
- Toutes les lignes ne sont pas intéressantes
- Une commande $\Rightarrow$ Une variable pertinente

Multiplexeurs 4 vers 1

sel1	sel0	a	b	c	d	s
0	0	0	X	X	X	0
0	0	1	X	X	X	1
0	1	X	0	X	X	0
0	1	X	1	X	X	1
1	0	X	X	0	X	0
1	0	X	X	1	X	1
1	1	X	X	X	0	0
1	1	X	X	X	1	1

Multiplexeurs 4 vers 1

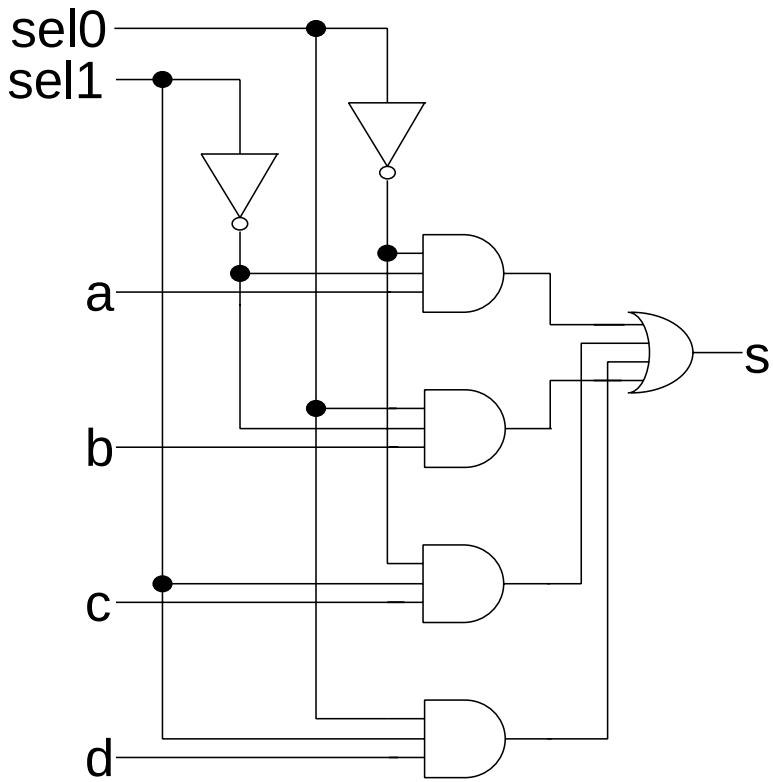
sel1	sel0	s
0	0	a
0	1	b
1	0	c
1	1	d

Multiplexeurs 4 vers 1

sel1	sel0	s
0	0	a
0	1	b
1	0	c
1	1	d

$$s = \overline{sel1}.\overline{sel0}.a + \overline{sel1}.sel0.b + sel1.\overline{sel0}.c + sel1.sel0.d$$

Multiplexeurs 4 vers 1



Multiplexeurs

- Permet de réaliser des fonctions logiques

Multiplexeurs

- Permet de réaliser des fonctions logiques
- Un Multiplexeur n vers 1 réalise 2^n fonctions

Multiplexeurs

- Permet de réaliser des fonctions logiques
- Un Multiplexeur n vers 1 réalise 2^n fonctions
- Valeurs des entrées = valeurs de la fonction

Multiplexeurs

- Permet de réaliser des fonctions logiques
- Un Multiplexeur n vers 1 réalise 2^n fonctions
- Valeurs des entrées = valeurs de la fonction
- Un Multiplexeur 4 vers 1

Multiplexeurs

- Permet de réaliser des fonctions logiques
- Un Multiplexeur n vers 1 réalise 2^n fonctions
- Valeurs des entrées = valeurs de la fonction
- Un Multiplexeur 4 vers 1

x	y	s	entrée mux
0	0	0	a = 0
0	1	0	b = 0
1	0	0	c = 0
1	1	1	d = 1

Multiplexeurs

- Permet de réaliser des fonctions logiques
- Un Multiplexeur n vers 1 réalise 2^n fonctions
- Valeurs des entrées = valeurs de la fonction
- Un Multiplexeur 4 vers 1

x	y	s	entrée mux
0	0	0	a = 0
0	1	0	b = 0
1	0	0	c = 0
1	1	1	d = 1

- x et y commandes du multiplexeur

Démultiplexeurs

- Inverse du Multiplexeurs

Démultiplexeurs

- Inverse du Multiplexeurs
- 1 données, p commandes, $2^p = n$ sorties

Démultiplexeurs

- Inverse du Multiplexeurs
- 1 données, p commandes, $2^p = n$ sorties
- Démultiplexeur 1 vers 2

Démultiplexeurs

- Inverse du Multiplexeurs
- 1 données, p commandes, $2^p = n$ sorties
- Démultiplexeur 1 vers 2

sel	a	s1	s0
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

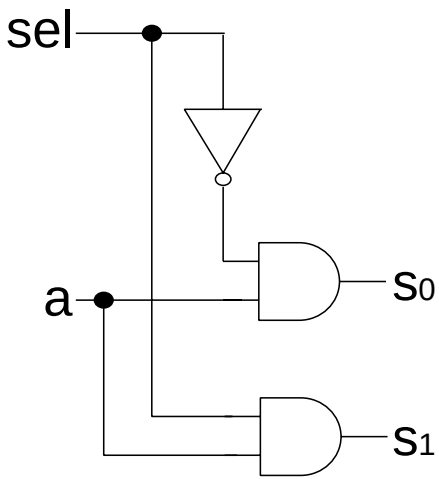
Démultiplexeurs

sel	a	s1	s0
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

Démultiplexeurs

- $s0 = \overline{sel}.a$ et $s1 = sel.a$

Démultiplexeurs



Démultiplexeurs - VHDL

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity demux is
port ( sel,a : in std_logic;
       s0,s1 : out std_logic);
end entity demux;

architecture flot of demux is
begin
   s0 <= a when sel = '0' else '0';
   s1 <= a when sel = '1' else '0';
end architecture flot;
```

Décodeurs

- Décodage Binaire $\rightarrow$ Codage 1 parmi n

Décodeurs

- Décodage Binaire $\rightarrow$ Codage 1 parmi n
- n entrées, 2^n sorties

Décodeurs

- Décodage Binaire $\rightarrow$ Codage 1 parmi n
- n entrées, 2^n sorties

a	b	s3	s2	s1	s0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Décodeurs

- Décodage Binaire $\rightarrow$ Codage 1 parmi n
- n entrées, 2^n sorties

a	b	s3	s2	s1	s0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

- Trivial :

Décodeurs

- Décodage Binaire $\rightarrow$ Codage 1 parmi n
- n entrées, 2^n sorties

a	b	s3	s2	s1	s0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

- Trivial :
  - $s0 = \overline{a}.\overline{b}$,

Décodeurs

- Décodage Binaire $\rightarrow$ Codage 1 parmi n
- n entrées, 2^n sorties

a	b	s3	s2	s1	s0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

- Trivial :
  - $s0 = \overline{a}.\overline{b}$,
  - $s1 = \overline{a}.b$,

Décodeurs

- Décodage Binaire $\rightarrow$ Codage 1 parmi n
- n entrées, 2^n sorties

a	b	s3	s2	s1	s0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

- Trivial :
  - $s0 = \overline{a}.\overline{b}$,
  - $s1 = \overline{a}.b$,
  - $s2 = a.\overline{b}$,

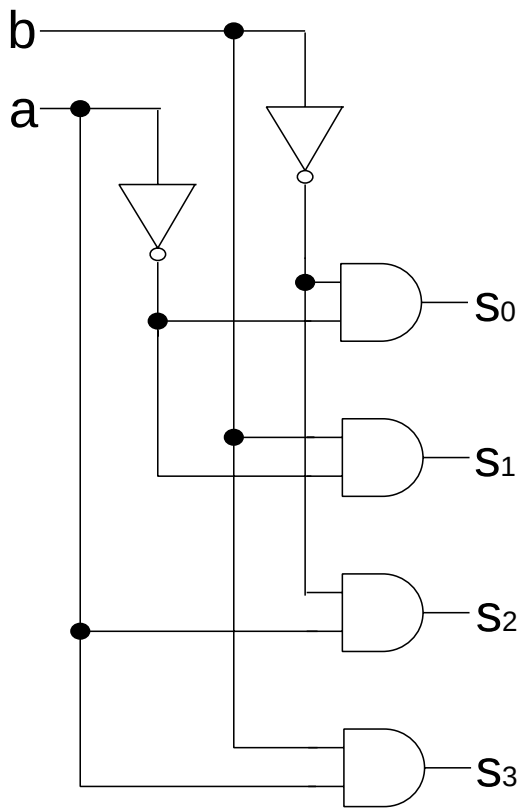
Décodeurs

- Décodage Binaire $\rightarrow$ Codage 1 parmi n
- n entrées, 2^n sorties

a	b	s3	s2	s1	s0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

- Trivial :
  - $s0 = \overline{a}.\overline{b}$,
  - $s1 = \overline{a}.b$,
  - $s2 = a.\overline{b}$,
  - $s3 = a.b$

Décodeur

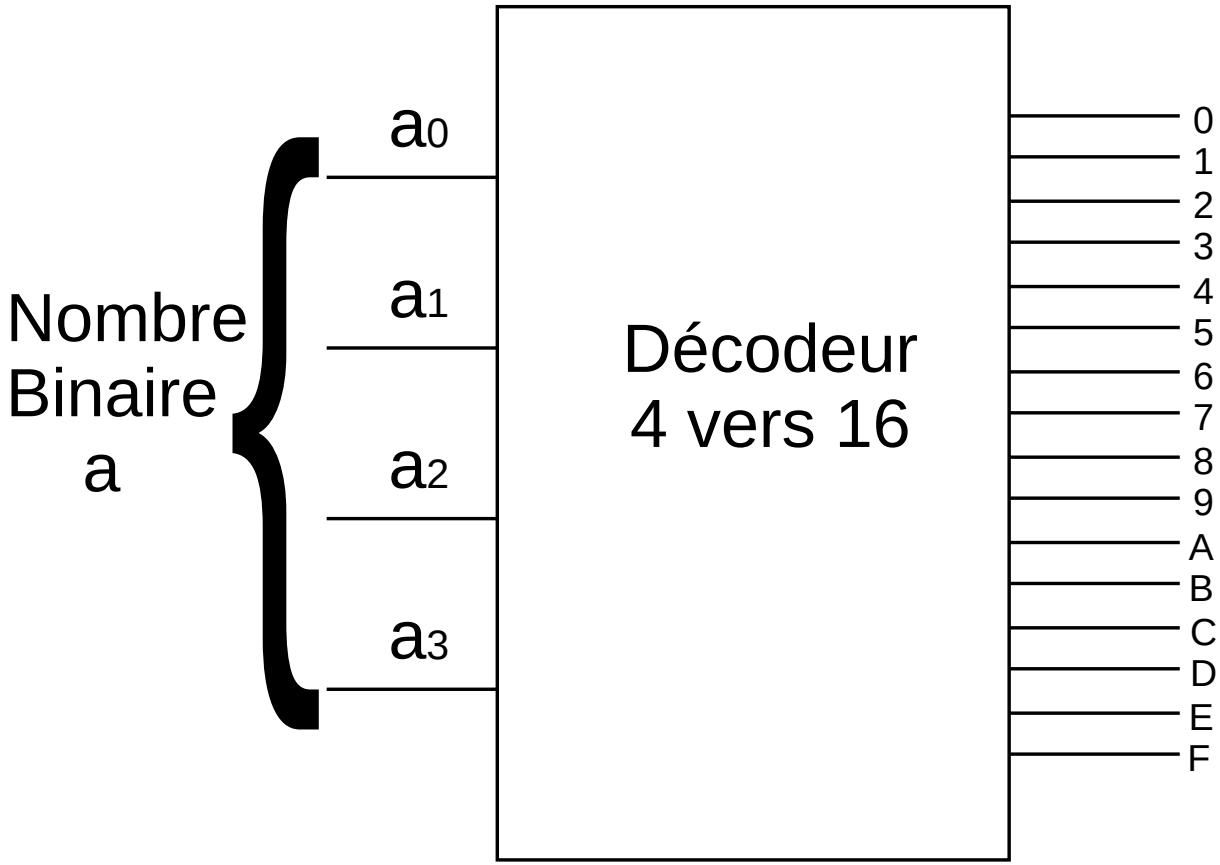


Décodeur

- Décodeur Binaire Base n

Décodeur

- Décodeur Binaire Base n



Encodeurs

- inverseur des décodeurs :codeurs

Encodeurs

- inverseur des décodeurs :codeurs
- 2^n entrées, n sorties

Encodeurs

- inverseur des décodeurs :codeurs
- 2^n entrées, n sorties

s3	s2	s1	s0	a	b
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Encodeurs

- inverseur des décodeurs :codeurs
- 2^n entrées, n sorties

s3	s2	s1	s0	a	b
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

- $b = \overline{s3}.\overline{s2}.s1.\overline{s0} + s3.\overline{s2}.\overline{s1}.\overline{s0} = \overline{s2}.\overline{s0}(s3 \oplus s1)$

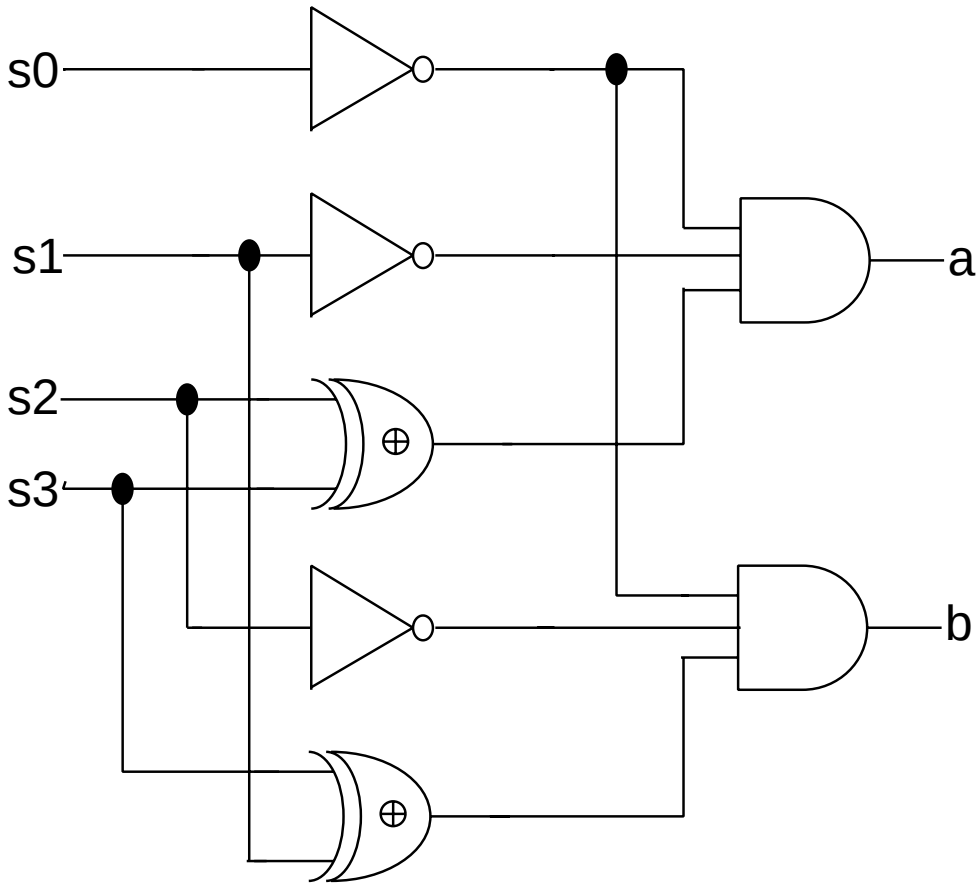
Encodeurs

- inverseur des décodeurs :codeurs
- 2^n entrées, n sorties

s3	s2	s1	s0	a	b
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

- $b = \overline{s3}.\overline{s2}.s1.\overline{s0} + s3.\overline{s2}.\overline{s1}.\overline{s0} = \overline{s2}.\overline{s0}(s3 \oplus s1)$
- $a = \overline{s3}.s2.\overline{s1}.\overline{s0} + s3.\overline{s2}.\overline{s1}.\overline{s0} = \overline{s1}.\overline{s0}.(s3 \oplus s2)$

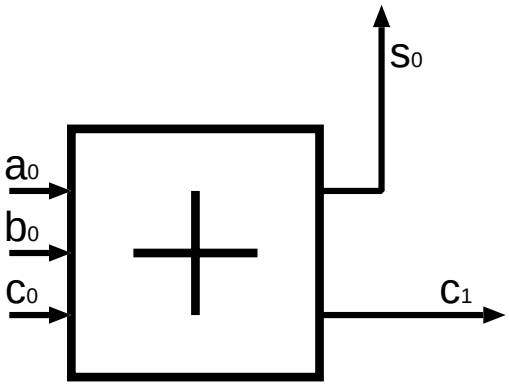
Encodeurs



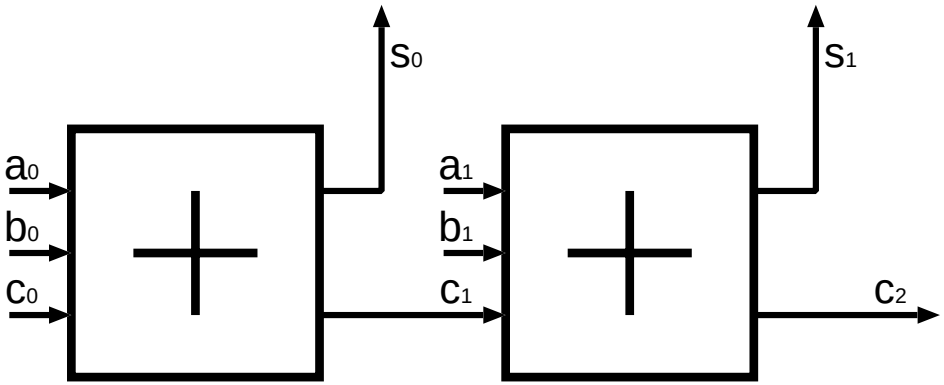
Fonctions Combinatoires Arithmétiques

Additionneur 4 bits

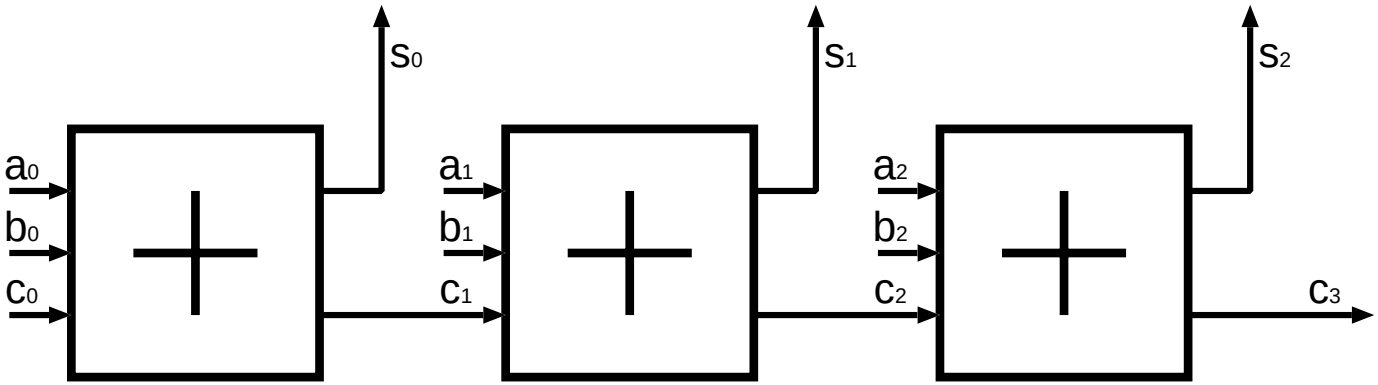
Additionneur 4 bits



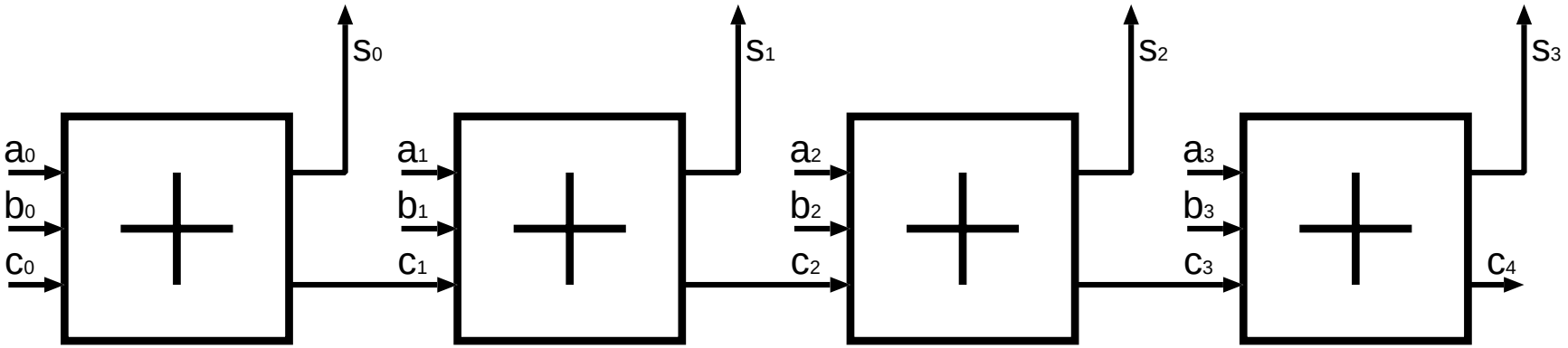
Additionneur 4 bits



Additionneur 4 bits



Additionneur 4 bits



Additionneur 4 bits : Entité

```
library ieee;
use ieee.std_logic_1164.all;

entity add4 IS
  port (a,b : in std_logic_vector(3 downto 0);
        cin : in std_logic;
        s : out std_logic_vector(3 downto 0);
        cout : out std_logic);
end entity add4;
```

‘

```
architecture struct_simple of add4 IS
signal c : std_logic_vector(4 downto 0);
begin

  c(0) <= cin;
  cout <= c(4);

  add1_0 : entity work.add1(flot)
           port map (a(0),b(0),c(0),s(0),c(1));

  add1_1 : entity work.add1(flot)
           port map (a(1),b(1),c(1),s(1),c(2));

  add1_2 : entity work.add1(flot)
           port map (a(2),b(2),c(2),s(2),c(3));

  add1_3 : entity work.add1(flot)
           port map (a(3),b(3),c(3),s(3),c(4));
end architecture struct_simple;
```

SCIENCES SORBONNE UNIVERSITÉ

Additionneur 4 bits : Architecture avec Génération

```
library ieee;
use ieee.std_logic_1164.all;

entity add4 IS
  port (a,b : in std_logic_vector(3 downto 0);
        cin : in std_logic;
        s : out std_logic_vector(3 downto 0);
        cout : out std_logic);
end entity add4;
```

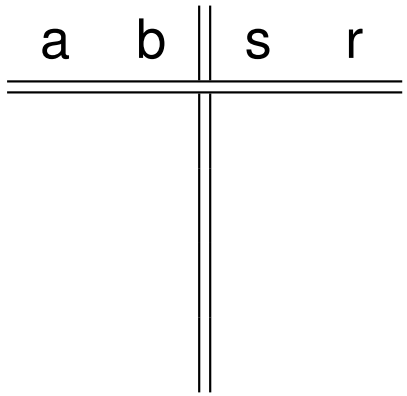
```
ARCHITECTURE struct_generate OF add4 IS
signal c : std_logic_vector(4 downto 0);
BEGIN

  c(0) <= cin;
  cout <= c(4);

  instance : for i in 0 to 3 generate
  add1_i : entity work.add1(flot)
           port map (a(i),b(i),c(i),s(i),c(i+1));
  end generate;

END ARCHITECTURE struct_generate;
```

Additionneur 1 bit : tentative

- Réalisation d'un additionneur 1 bit

Additionneur 1 bit : tentative

- Réalisation d'un additionneur 1 bit



Additionneur 1 bit : tentative

- Réalisation d'un additionneur 1 bit

a	b	s	r
0	0		

Additionneur 1 bit : tentative

- Réalisation d'un additionneur 1 bit

a	b	s	r
0	0	0	

Additionneur 1 bit : tentative

- Réalisation d'un additionneur 1 bit

a	b	s	r
0	0	0	0

Additionneur 1 bit : tentative

- Réalisation d'un additionneur 1 bit

a	b	s	r
0	0	0	0
0	1		

Additionneur 1 bit : tentative

- Réalisation d'un additionneur 1 bit

a	b	s	r
0	0	0	0
0	1	1	

Additionneur 1 bit : tentative

- Réalisation d'un additionneur 1 bit

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

Additionneur 1 bit : tentative

- Réalisation d'un additionneur 1 bit

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

Additionneur 1 bit : tentative

- Réalisation d'un additionneur 1 bit

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

Additionneur 1 bit : tentative

- Réalisation d'un additionneur 1 bit

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

Additionneur 1 bit : tentative

- Réalisation d'un additionneur 1 bit

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

Additionneur 1 bit : tentative

- Réalisation d'un additionneur 1 bit

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

Additionneur 1 bit : tentative

- Réalisation d'un additionneur 1 bit

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

Additionneur 1 bit : tentative

- Réalisation d'un additionneur 1 bit

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

- $s = a \oplus b$

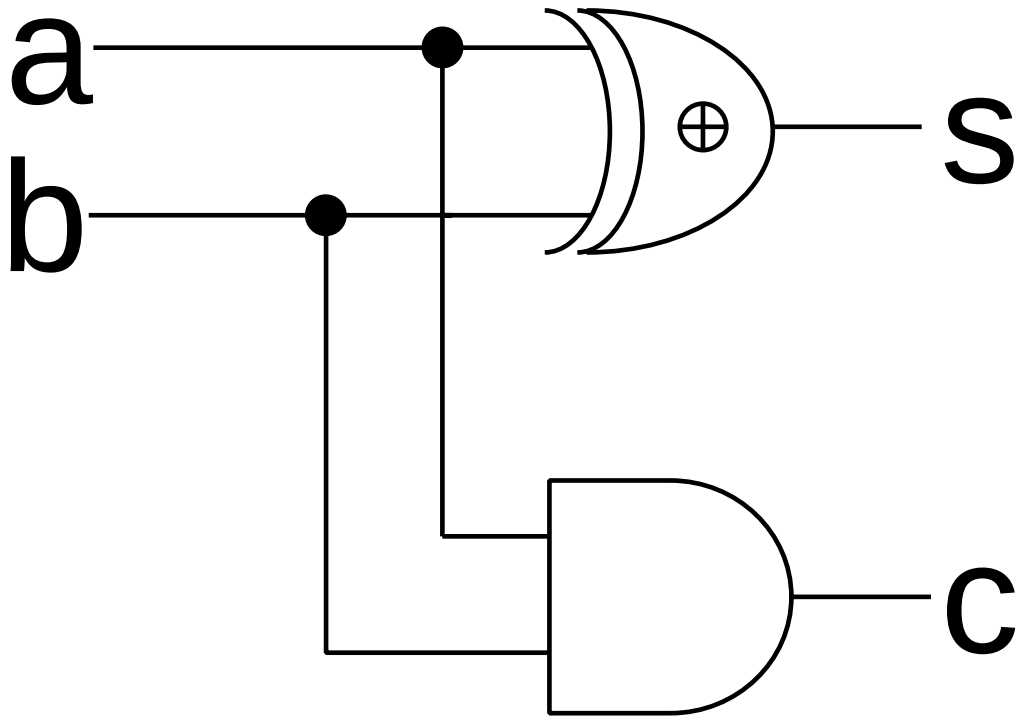
Additionneur 1 bit : tentative

- Réalisation d'un additionneur 1 bit

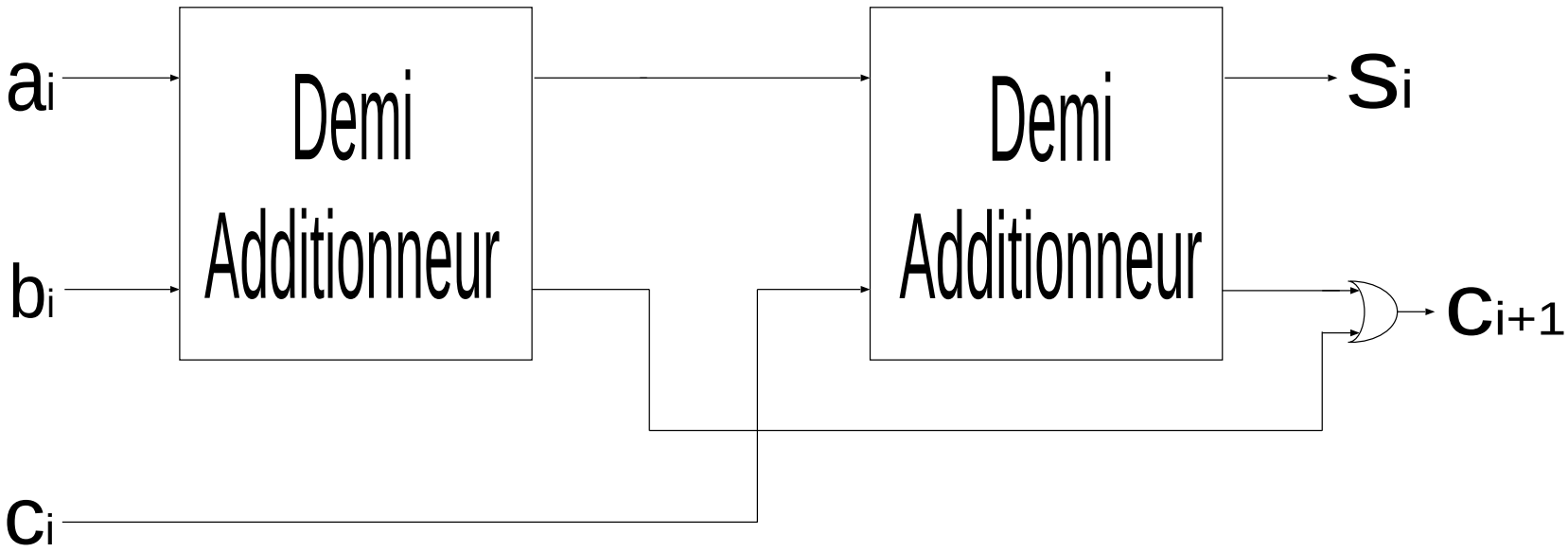
a	b	s	r
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- $s = a \oplus b$
- $r = a.b$

Additionneur 1 bit : tentative



Additionneur 1 bit



Additionneur 1 bit

```
library ieee;
use ieee.std_logic_1164.all;

entity add1 is
port( a,b,cin : in std_logic;
      s,cout : out std_logic);
end entity add1;
architecture struct of add1 is
signal stemp,ctemp1,ctemp2 : std_logic;
begin
 demi-add1 : entity work.demi-add(flot)
             port map(a,b,stemp,ctemp1);
 demi-add2 : entity work.demi-add(flot)
             port map(stemp,cin,s,ctemp2);
 cout <= ctemp1 or ctemp2
end architecture struct;
```

Additionneur 1 bit

- Introduction d'une retenue d'entrée

Additionneur 1 bit

- Introduction d'une retenue d'entrée
- Trois variables d'entrées, deux de sorties

Additionneur 1 bit

- Introduction d'une retenue d'entrée
- Trois variables d'entrées, deux de sorties
- a_i,b_i,c_i et s_i,c_{i+1}

Additionneur 1 bit

- Introduction d'une retenue d'entrée
- Trois variables d'entrées, deux de sorties
- a_i,b_i,c_i et s_i,c_{i+1}
- $s_i = a_i \oplus b_i \oplus c_i$

Additionneur 1 bit

- Introduction d'une retenue d'entrée
- Trois variables d'entrées, deux de sorties
- a_i,b_i,c_i et s_i,c_{i+1}
- $s_i = a_i \oplus b_i \oplus c_i$
- $c_{i+1} = a_i.b_i + a_i.c_i + b_i.c_i$

Additionneur 1 bit

```
entity add1 is
port( a,b,cin : in std_logic;
      s,cout : out std_logic);
end entity add1;
architecture flot of add1 is
begin
    s<= a xor b xor cin;
    cout<= (a and b) or (a and cin)
           or (b and cin);
end architecture flot;
```

Additionneur 1 bit

```
entity add1 is
port( a,b  : in std_logic;
      s    : out std_logic);
end entity add1;
architecture comport of add1 is
begin
    s<= a + b;
end architecture comport;
```

Multiplieur

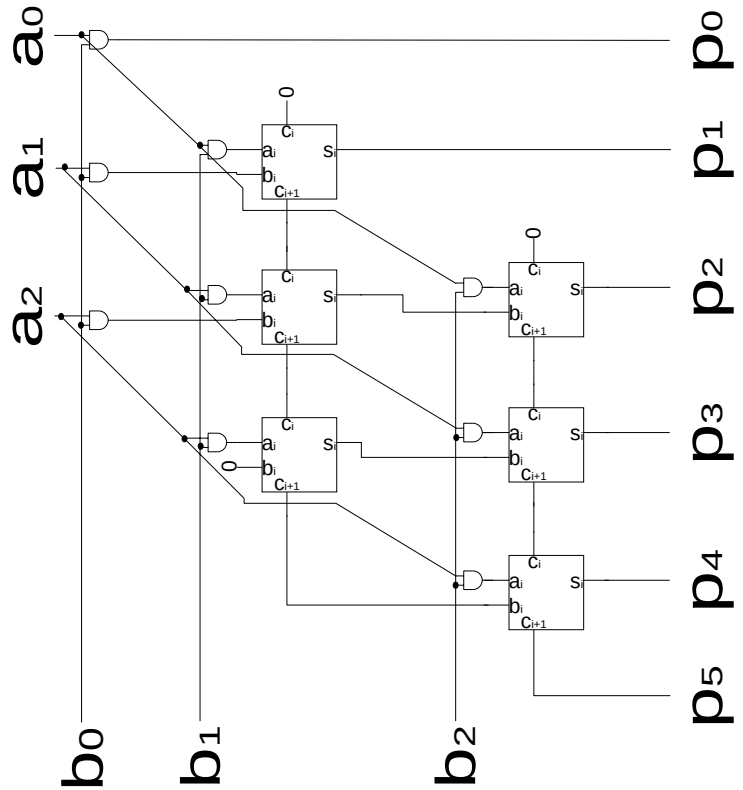
- Utilisation de l'algorithme de Multiplication

Multiplieur

- Utilisation de l'algorithme de Multiplication
- $n * m$ m additions de n

Multiplieur

- Utilisation de l'algorithme de Multiplication



Complément VHDL : Modélisation du temps

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity mon-et is
port(a,b : in std_logic;
     s : out std_logic);
end entity mon-et;

architecture flot of mon-et is
begin
       s <= a and b after 25 ns;
end architecture flot;
```

Complément VHDL : Décalage et mise à l'échelle

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity conversion is
port(a : in std_logic_vector(5 downto 0);
     s,s2,s3 : out std_logic_vector(11 downto 0));
end entity conversion;

architecture flot of conversion is
begin
        -- soit a = "010101";
        s <= "0000" & a & "00"; -- s = "000001010100"
        s2 <= "00000" & a & '0'; -- s2 = "000000101010"
        s3 <= a & "000000"; -- s3 = "010101000000"
end architecture flot;
```

Plan

Plan

Plan

Plan

SCIENCES SORBONNE UNIVERSITÉ