

SN360/SN361 Introduction à la conception des circuits numériques

Module « Systèmes matériels et logiciels »

Crédit SN360 : 4/4 / SN361 : 2/6

Vincent Beroulle

Bureau : D202

vincent.beroulle@esisar.grenoble-inp.fr

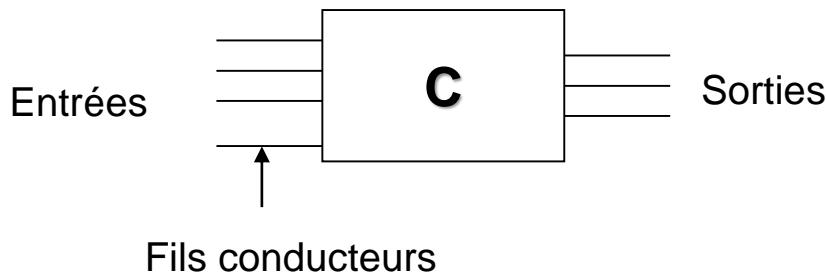
Plan

- I Introduction
- II Représentation des nombres en binaire
- III Circuits combinatoires**
- IV Circuits séquentiels
- V Machine à états finis
- VI Complément sur les HDL
- VII Circuits reconfigurables (optionnel)

III Circuits combinatoires

Définitions

- Algèbre Booléenne **F** \Leftrightarrow Circuit combinatoire **C**
 - mathématique/logique
 - électronique
- Les sorties d'un circuit combinatoire peuvent être déterminées à partir de la connaissance de ses entrées

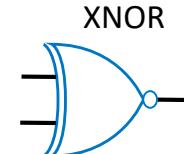
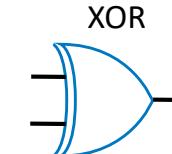
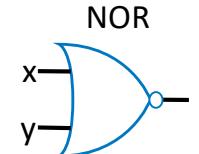
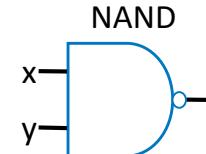
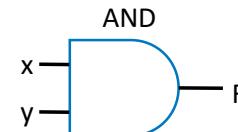
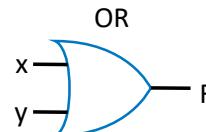
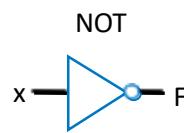


- Les circuits combinatoires sont composés uniquement de portes logiques combinatoires (ET, OU, inverseurs, ...)
- Les boucles dans les circuits combinatoires sont interdites

III Circuits combinatoires

Portes logiques combinatoires

- Portes logiques de base :



x	F
0	1
1	0

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

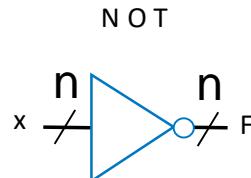
x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

Pour réaliser des opérations sur chaque bit d'un vecteur ne n bits on utiliser les n mêmes portes:



III Circuits combinatoires

Codes VHDL&Verilog - opérateurs logiques

VHDL

-- avec a et b de simples bits ou vecteurs

s <= not(a); --inv

--or, nor

s1 <= a or b; **s2 <= a nor b;**

--and, nand

s1 <= a and b; **s2 <= a nand b;**

--xor, xnor

s1 <= a xor b; **s2 <= a xnor b;**

-- **sll** décalage à gauche de 1 bit

-- **srl** décalage à droite de 1 bit

Verilog

// avec a et b de simples bits ou vecteurs

assign s = ~a; // not(a)

assign s1 = a | b; // a or b

assign s2 = ~(a | b); // a nor b

assign s3 = a & b; // a and b

assign s4 = ~(a & b); // a nand b

assign s5 = a ^ b; // a xor b

assign s6 = a ~^ b; // a xnor b

// << décalage à gauche de 1 bit

// >> décalage à droite de 1 bit

III Circuits combinatoires

Codes VHDL&Verilog - portes logiques

VHDL

```
-- entité = vue externe  
entity INV is  
  port(e : in bit;  
        s : out bit);  
end INV;  
  
-- architecture = vue interne  
architecture RTL of INV is  
  -- zone de déclaration de  
  l'architecture  
  begin  
    s<=not(e);  
  end RTL;
```

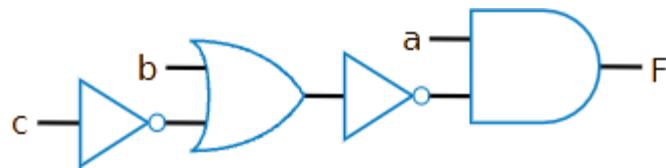
Verilog

```
module INV (  
  input wire e,  
  output wire s );  
  assign s = ~e;  
endmodule
```

III Circuits combinatoires

Portes logiques combinatoires - exercice

- $F = a \text{ AND NOT } (b \text{ OR NOT } c) = a.(b+c')'$



Représentez l'équation suivante avec les symboles $(\cdot; +; ')$ et avec un circuit logique :

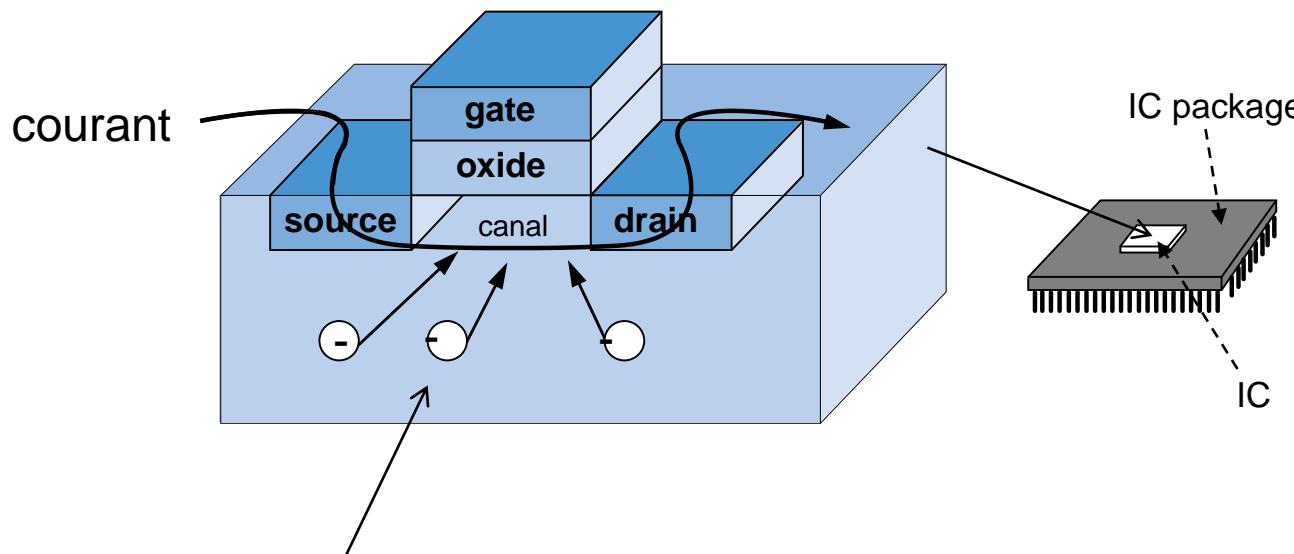
$$F = (a \text{ AND NOT}(b)) \text{ OR } (b \text{ AND NOT}(c))$$

III Circuits combinatoires

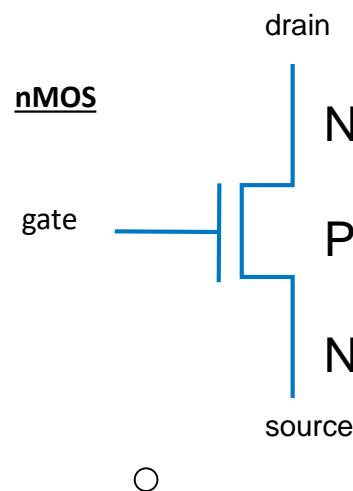
Portes logiques au niveau transistor

Technologie CMOS : Complementary Metal Oxyde Semiconductor

Une tension positive sur la grille attire les e- dans le canal

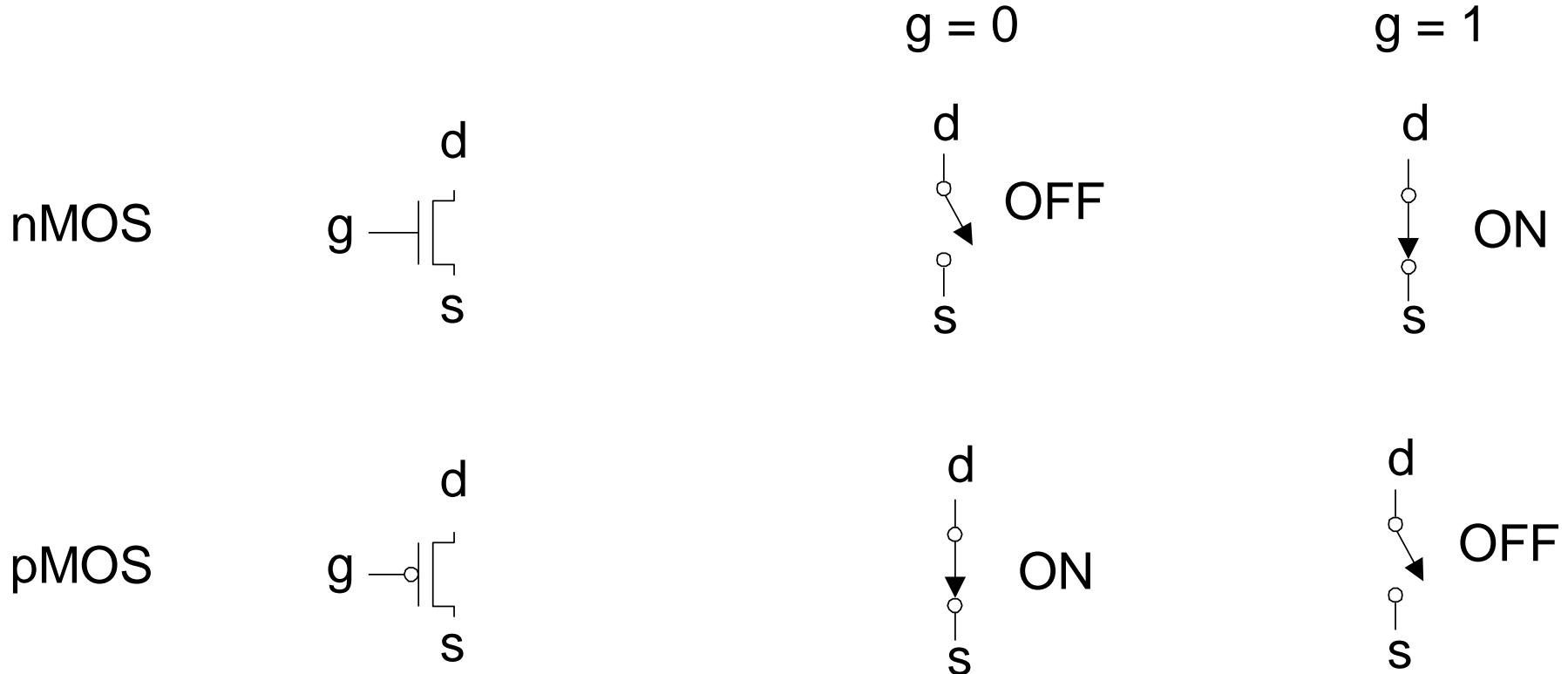


Silicium (dopé P, pour un nMOS)



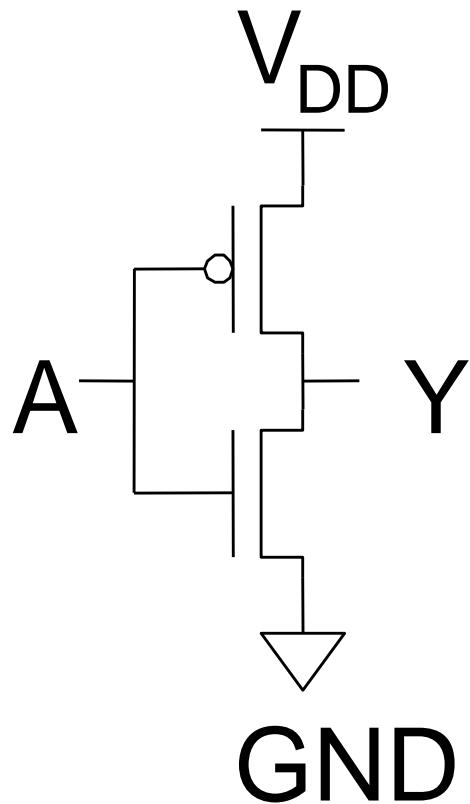
III Circuits combinatoires

Portes logiques - modèle interrupteur



III Circuits combinatoires

Portes logiques - inverseur



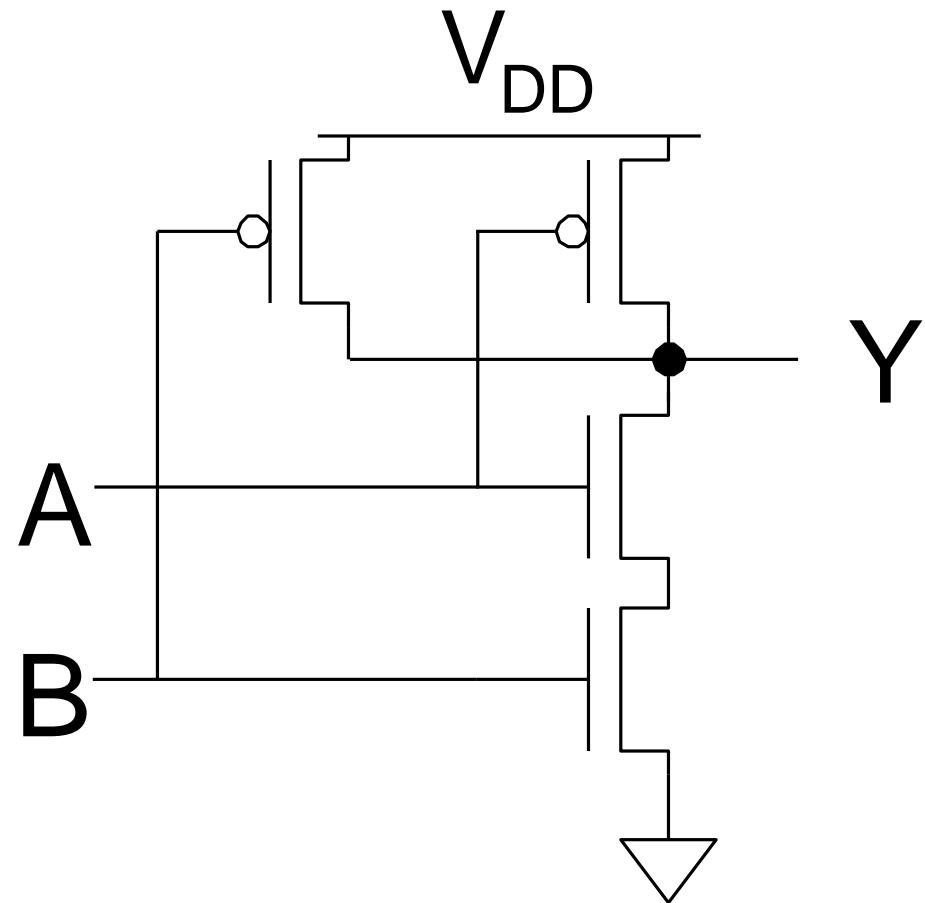
Complétez la table de vérité

A	Y
0	
1	

III Circuits combinatoires

Portes logiques - NAND

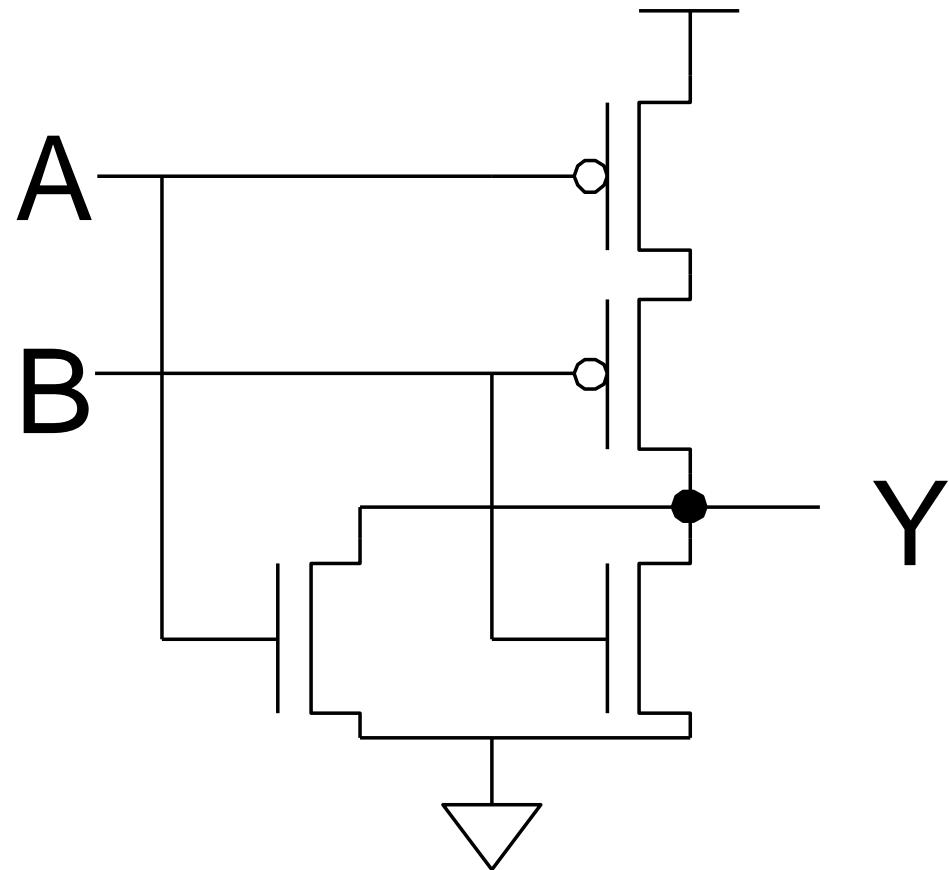
A	B	Y
0	0	
0	1	
1	0	
1	1	



III Circuits combinatoires

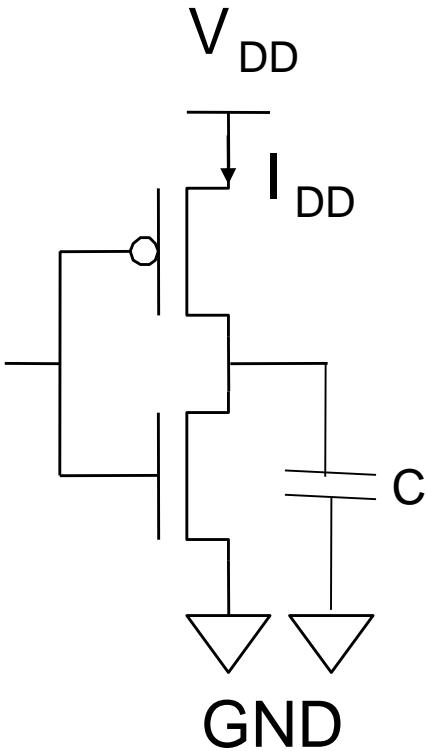
Portes logiques - NOR

A	B	Y
0	0	
0	1	
1	0	
1	1	



II Circuits combinatoires

Portes logiques - puissance dissipée



Puissance dissipée :

- Charge et décharge des capacités parasites

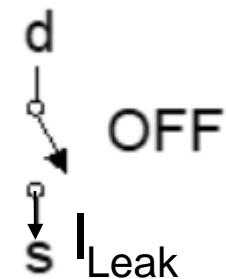
$$P_{\text{dynamique}} = C \cdot V_{DD}^2 \cdot F$$

Avec C la capacité parasite en sortie

Avec F la fréquence de charge/décharge de C

- Courants de fuite dans les transistors
(normalement OFF)

$$P_{\text{statique}} = I_{\text{Leak}} \cdot V_{DD}$$



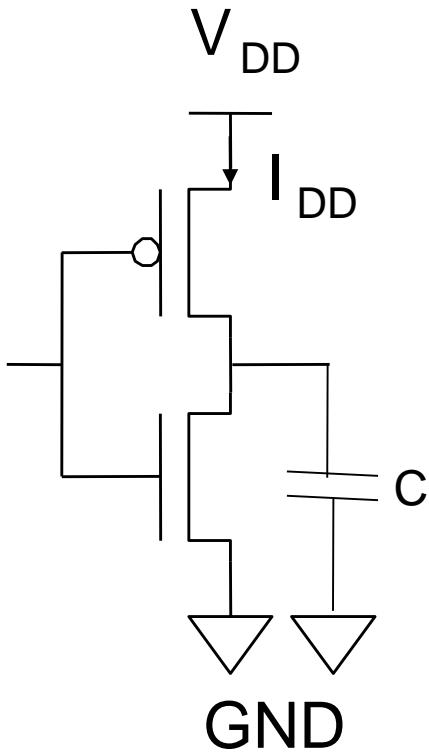
Plus les transistors sont petits plus les capacités parasitent C diminuent
Mais plus leur densité et les courants de fuite I_{Leak} augmentent

III Circuits combinatoires

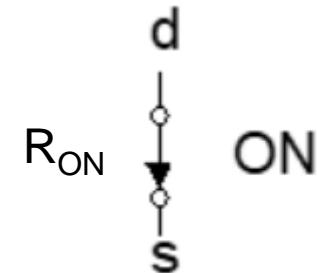
Portes logiques - Temps de propagation

Temps de propagation :

- Temps de charge et décharge des capacités parasites



$$T_{\text{propagation}} \propto R_{\text{on}} \cdot C$$



Plus les transistors sont petits plus R_{ON} augmente mais plus C diminue

Les temps de propagations des portes logiques des FPGA (LUT) est de l'ordre de la **centaine de ps**

Les temps de propagations des interconnexions deviennent non négligeables

III Circuits combinatoires

Codes VHDL&Verilog - temps de propagation

VHDL

```
-- sans processus avec une  
instruction séquentielle  
s <= not(a) after 10 ns;  
-- avec un processus avec une  
liste de sensibilité contenant a  
process(a)  
begin  
wait for 10 ns;  
s <= not(a);  
end process;
```

Verilog

```
// unité de temps/précision  
`timescale 1ns/1ps  
  
assign #10 s = ~a;  
//délai de 10ns à chaque  
évènement  
//#10 dans un always est  
équivalent à un wait for 10  
ns
```

**Toutes les instructions avec des délais
sont simulables mais pas synthétisables**

III Circuits combinatoires

Table de vérité de $F=A \cdot B + C$

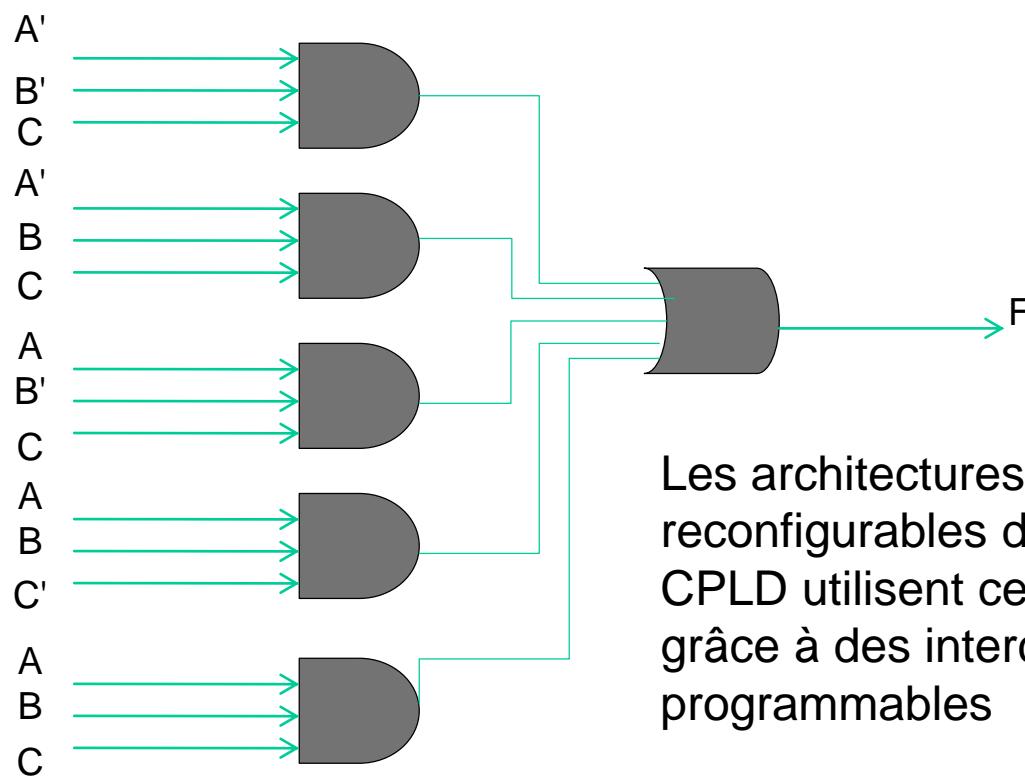
A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Toutes les fonctions combinatoires peuvent être représentées par une table de vérité

III Circuits combinatoires

Forme canonique de $F = A \cdot B + C$

- ▶ On peut choisir d'implémenter toutes les possibilités en entrée pour avoir '1' en sortie
 - ▶ $F = A'B'C + A'BC + AB'C + ABC' + ABC$
- } 5 '1' en sortie dans la table de vérité
=> 5 termes produits



Les architectures reconfigurables de type CPLD utilisent ce principe grâce à des interconnexions programmables

III Circuits combinatoires

Codes VHDL&Verilog - entity & architecture & module

VHDL

-- entité = vue externe

entity Fonction **is**

port(a, b, c : **in** bit;

 F : **out** bit);

end Fonction;

-- architecture = vue interne

architecture comb of Fonction **is**

begin

 F<=(a **and** b) **or** c;

end comb;

Verilog

// les circuits combinatoires utilisent des **affectations bloquantes** =

module Fonction (

input wire a, b, c,

output wire F);

assign F = (a & b) | c;

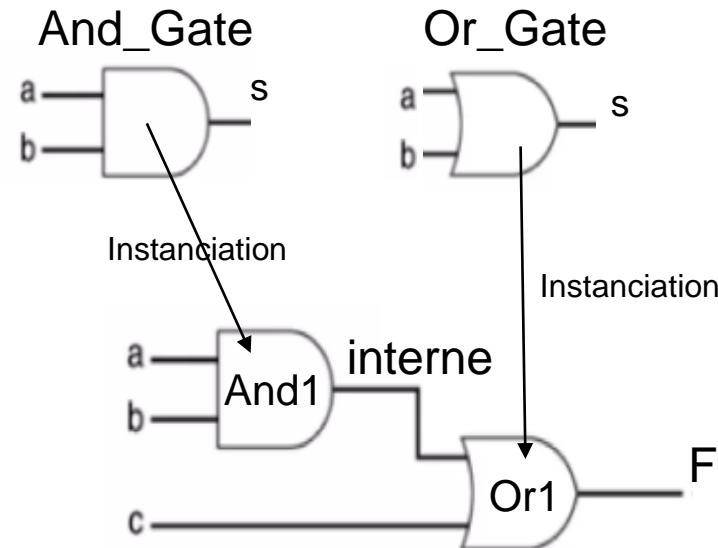
endmodule

III Circuits combinatoires

Code VHDL - Description structurelle (port map)

VHDL

```
component And_Gate
port(a, b:in bit;s : out bit);
end component;
component Or_Gate
port(a, b:in bit;s : out bit);
end component;
signal interne : bit;
begin
And1: And_Gate
port map(a=>a, b=>b,
s=>interne);
Or1: Or_Gate
port map(interne,c, F);
```



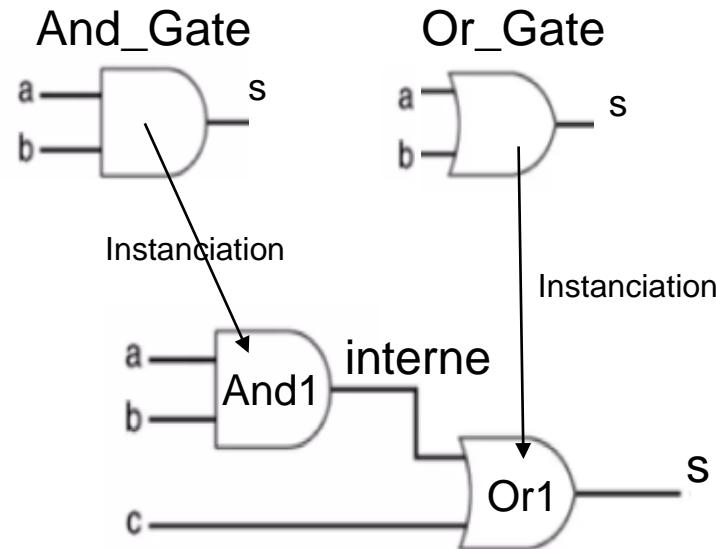
Instanciation par dénomination
ou
Instanciation par position

III Circuits combinatoires

Code Verilog - Description structurelle (port map)

Verilog

```
module Fonction (
    input wire a, b, c,
    output wire F );
    wire interne;
    And_Gate And1 (.s(interne), .a(a),
    .b(b));
    Or_Gate  Or1  (.s(F),   .a(interne),
    .b(c));
endmodule
```



III Circuits combinatoires

Tableau de Karnaugh

- ▶ Simplification (manuelle) des fonctions logiques :

The diagram illustrates the manual simplification of a logic function using a Karnaugh map. It shows two tables and a process of grouping.

Initial Table:

C \ AB	00	01	11	10
0	0	0	1	0
1	1	1	1	1

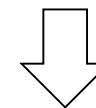
A triangle with an exclamation mark (!) has an arrow pointing to the cell for AB=11, C=0, which contains a 1.

Resulting Table after Grouping:

C \ AB	00	01	11	10
0	0	0	1	0
1	1	1	1	1

The cell for AB=11, C=0 is highlighted with a red box. The cells for AB=00, C=1 and AB=10, C=1 are also highlighted with red boxes, indicating they are part of the same group.

Text below the tables: Regroupement par les 1



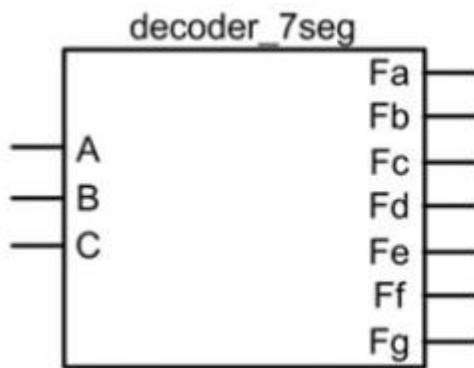
$$F = A \cdot B + C$$

III Circuits combinatoires

Tableau de Karnaugh - exercices

Q		a b			
		0 0	0 1	1 1	1 0
c d	0 0	1	0	0	1
	0 1	1	1	1	1
	1 1	1	1	0	0
	1 0	0	0	0	0

Donnez la fonction $Q(a,b,c,d)$ simplifiée.



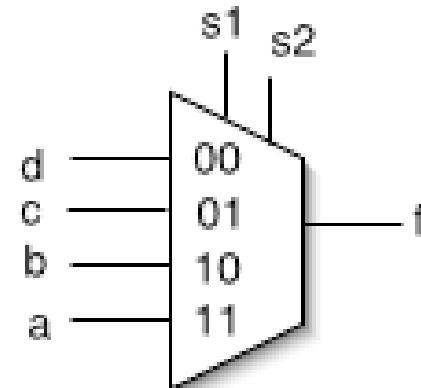
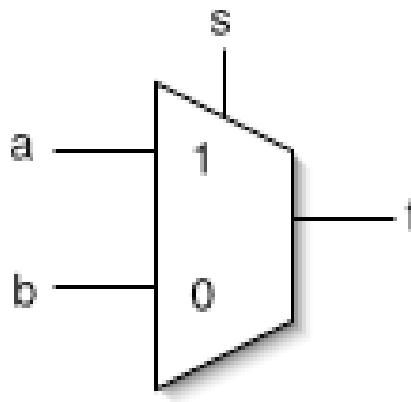
A	B	C	Fa	Fb	Fc	Fd	Fe	Ff	Fg
0	0	0	1	1	1	1	1	1	0
0	0	1	0	1	1	0	0	0	0
0	1	0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	0	0	1
1	0	0	0	1	1	0	0	1	1
1	0	1	1	0	1	1	0	1	1
1	1	0	1	0	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0

Donnez les fonctions F_x simplifiée correspondant à chaque segment d'un afficheur 7 segments

III Circuits combinatoires

Circuits usuels : Multiplexeur

- Multiplexeur : dirige une des entrées vers la sortie
 - 2 vers 1 : $f(a,b,s) = a.s + b.s'$
 - 4 vers 1 : $f(a,b,c,d,s_1,s_2) = a.s_1.s_2 + b.s_1.s_2' + c.s_1'.s_2 + d.s_1'.s_2'$



III Circuits combinatoires

Codes VHDL&Verilog - Multiplexeur

VHDL

-- avec un processus

```
process(a, b, s)
begin
case s is
    when '0' => f<=a;
    when '1' => f<=b;
    when others => f<=a;
end case;
end process;
-- sans processus
with s select
    f<=a when '0';
    f<=b when '1';
    f<=a when others;
end;
```

Verilog

```
wire a, b, s;
reg f; //pour affectation dans
always
    begin
        case (s)
            1'b0: f = a;
            1'b1: f = b;
        endcase
    end
```

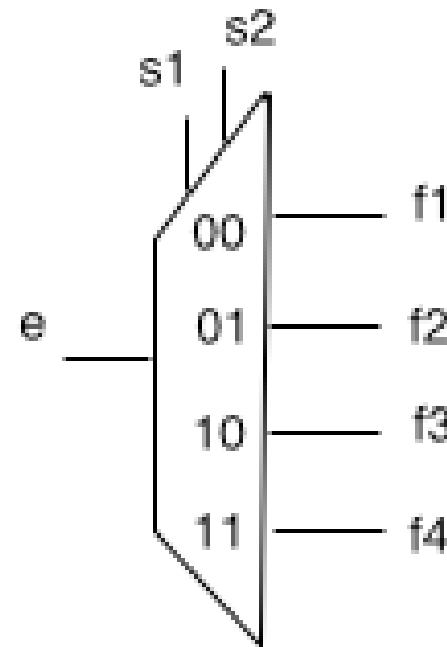
III Circuits combinatoires

Circuits usuels : Démultiplexeur

- Démultiplexeur : dirige l'entrée vers une des sorties

- 1 vers 4 :

- $f_1(e,s_1,s_2) = e \cdot \bar{s}_1 \cdot \bar{s}_2$
 - $f_2(e,s_1,s_2) = e \cdot \bar{s}_1 \cdot s_2$
 - $f_3(e,s_1,s_2) = e \cdot s_1 \cdot \bar{s}_2$
 - $f_4(e,s_1,s_2) = e \cdot s_1 \cdot s_2$



III Circuits combinatoires

Codes VHDL&Verilog - Démultiplexeur

VHDL

-- avec un processus avec une instruction séquentielle case... when

```
process(e, s)
begin
f1<='0'; f2<='0'; f3<='0'; f4<='0';
case s is
when "00" => f1<=e;
when "01" => f2<=e;
when "10" => f3<=e;
when "11" => f4<=e;
when others =>
end case;
end process;
-- sans processus avec une instruction concurrente with .. select
with s select f1<=e when "00" else '0';
...
...
```

Verilog

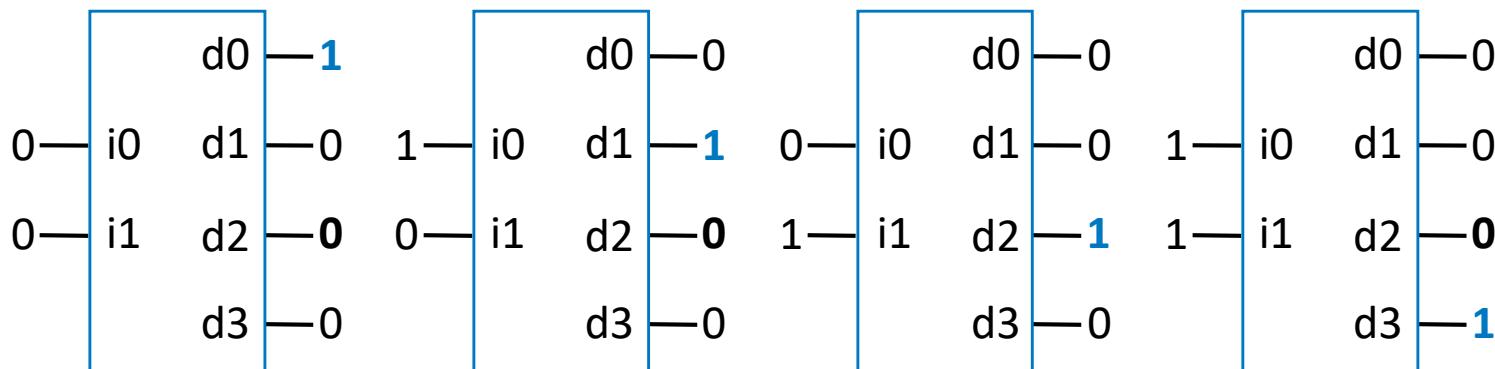
```
wire e, s;
reg f1, f2, f3, f4;
```

```
always @(e or s) begin
f1=0;f2=0;f3=0;f4=0;
case (s)
2'b00: f1 = e;
2'b01: f2 = e;
2'b10: f3 = e;
2'b11: f4 = b;
endcase
end
```

III Circuits combinatoires

Circuits usuels : Décodeur

- Décodeur : convertit un nombre binaire en entrée vers une unique sortie haute



Donnez les équations de d_0, d_1, d_2 et d_3

III Circuits combinatoires

Codes VHDL&Verilog - Décodeur

VHDL

-- avec un processus

```
process(i, d)
begin
d<="0000";
case i is
when "00" => d(0)<='1';
when "01" => d(1)<='1';
when "10" => d(2)<='1';
when "11" => d(3)<='1';
when others =>
end case;
end process;
```

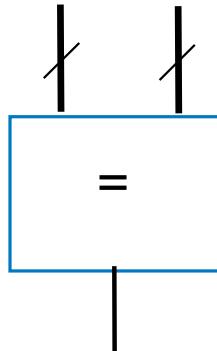
Verilog

```
wire i;
reg [3:0] d;
```

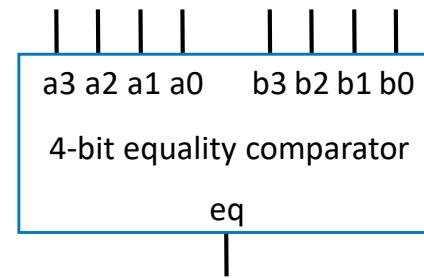
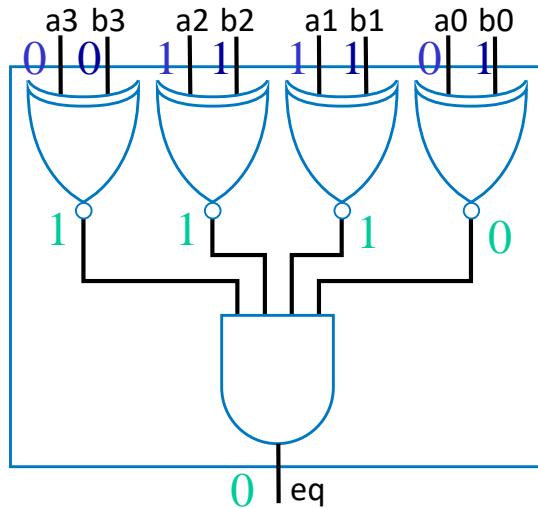
```
always @(*) begin
d=4'b0000;
case (i)
2'b00: d[0] = 1;
2'b01: d[1] = 1;
2'b10: d[2] = 1;
2'b11: d[3] = 1;
endcase
end
```

III Circuits combinatoires

Circuits usuels : Comparateur



$0110 = 0111 ?$



III Circuits combinatoires

Codes VHDL&Verilog - instructions de comparaison

VHDL

```
process(a,b)
begin
if a=b then
    s<='1';
else
    s<='0';
end if;
end process;
-- ou avec une instruction
concurrente
s<='1' when a=b else '0';
```

Verilog

```
wire [3:0] a,b;
reg s;
always @(*) begin
    if (a==b) begin
        s=1;
    end else begin
        s=0;
    end
end
// ou en utilisant des assignations
continues
wire s;
assign s = (a == b) ? 1'b1 : 1'b0;
```

III Circuits combinatoires

Codes VHDL&Verilog - opérateurs de comparaison

VHDL

-égalité

$a=b$

--différence

$a \neq b$

--inférieur ou égal

$a \leq b$

--supérieur ou égal

$a \geq b$

Verilog

-égalité

$a==b$

--différence

$a!=b$

--inférieur ou égal

$a \leq b$

--supérieur ou égal

$a \geq b$

III Circuits combinatoires

Additionneur

$1+1 = 0$ avec retenue positive sur le bit suivant

$1+0 = 1$

$0+0 = 0$

- Exemple : $13 + 5 = 1101 + 0101$

$$\begin{array}{r} 1 \ 1 \ 1 \\ & & \\ 1101 & & \\ & & \end{array} = 10010$$

$$\begin{array}{r} 1101 \\ +0101 \\ \hline 10010 \end{array}$$

L'addition de deux nombres de 4 bits
donne un résultat sur 5 bits.
Le MSB est un indicateur de retenue.

III Circuits combinatoires

Additionneur

VHDL

```
library IEEE;  
use IEEE.std_logic_1164.all;  
use IEEE.numeric_std.all;  
signal s1: signed(3 downto 0);  
signal s2: signed(3 downto 0);  
signal s3 : signed(4 downto 0);
```

s3 <= s1(3)&s1 + s2;
-- n bit + k bit crée max(n, k)
-- & = opérateur de concaténation

Verilog

```
wire signed [3:0] s1, s2;  
wire signed [4:0] s3;  
assign s3=s1+s2;
```

III Circuits combinatoires

Soustracteur

- La soustraction **en complément à 2**
 - Principe : $A - B$ revient à faire $A + (-B)$
On code A et $-B$ en complément à 2
On additionne A et $-B$

– Exemple : $13 - 6$

$13_{c_2} = 01101$ on ajoute un bit de signe 0

$-6_{c_2} = /0110+0001=1001+0001=11010$

$$\begin{array}{r} 11 \\ 001101 \\ + 11010 \\ \hline 000111 \end{array}$$

5 bits + 5 bits
= 6 bits

$\rightarrow +7$

Exemple : $6-13$

$6_{c_2}=00110$

$-13_{c_2}=1101+0001=0010+0001=10011$

$$\begin{array}{r} /111001+000001 \\ 000110 000110+000001 \\ +110011 (000111)_2=(7)_{10} \\ \hline 111001 \end{array} \rightarrow -7$$

III Circuits combinatoire

Codes VHDL&Verilog - soustraction

VHDL

```
library IEEE;  
use IEEE.std_logic_1164.all;  
use IEEE.numeric_std.all;  
signal s1: signed(3 downto 0);  
signal s2: signed(3 downto 0);  
signal s3 : signed(4 downto 0);
```

s3 <= s1(3)&s1 - s2;
-- n bit - k bit crée max(n, k)
-- & = opérateur de concaténation

Verilog

```
wire signed [3:0] s1, s2;  
wire signed [4:0] s3;  
assign s3=s1-s2;
```

III Circuits combinatoires

Exercices

- Additionnez les nombres suivants en binaire :
 1. $25 + 5 =$
 2. $42 + 13 =$
- puis en complément à deux.
 1. $-25 + 5 =$
 2. $-43 + 13 =$
 3. $13 + (-13) =$

III Circuits combinatoires

Multiplieur (nombres entiers positifs)

Multiplicand M (14)

1 1 1 0

Multiplier Q (13)

× 1 1 0 1

0s are implicitly
assumed

0 0 0 0 1 1 1 0

0 0 0 0 0 0 0

0 0 1 1 1 0

0 1 1 1 0

Product P

(182)

1 0 1 1 0 1 1 0

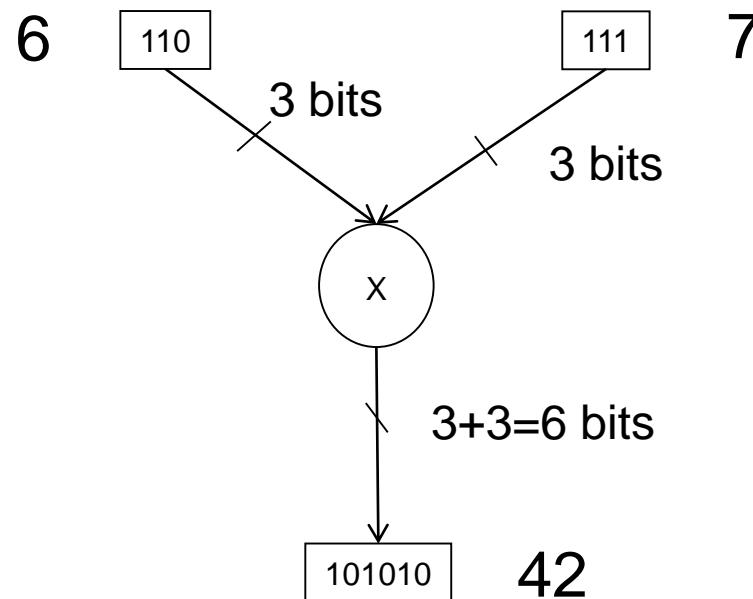
Le produit de 2 nombres de n bits positifs génère $2n$ bits

III Circuits combinatoires

Multiplieur (nombres entiers négatifs)

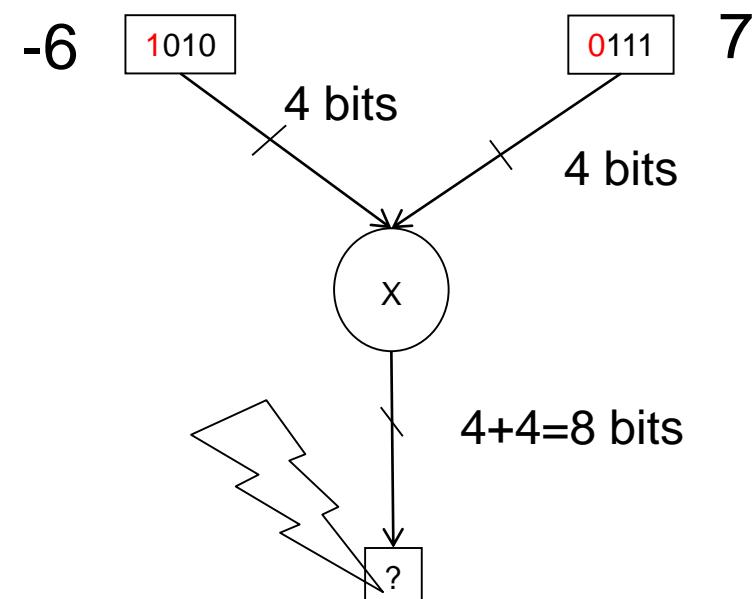
Exemple : 6×7

Nombres entiers positifs



Exemple : -6×7

Nombres entiers négatifs
(complément à 2)

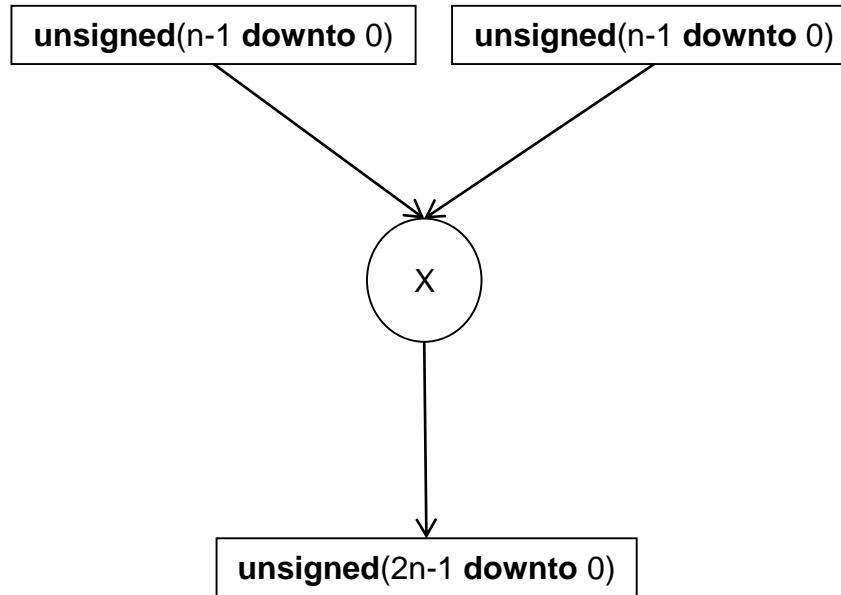


Le produit de nombres négatifs
avec un multiplieur classique ne
donne pas le bon résultat

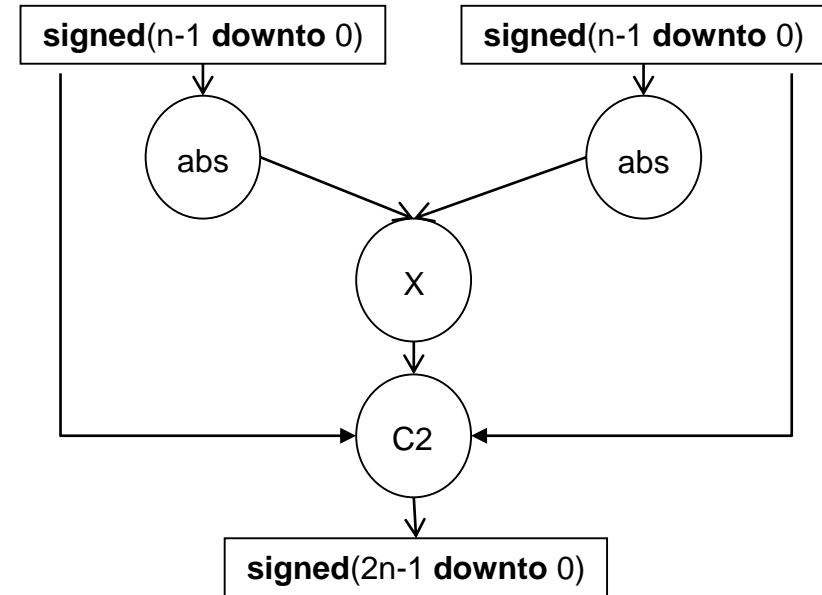
III Circuits combinatoires

Multiplieur (nombres négatifs)

Nombres entiers positifs



Nombres entiers négatifs



C2: Complément à 2 (activé ou non activé)

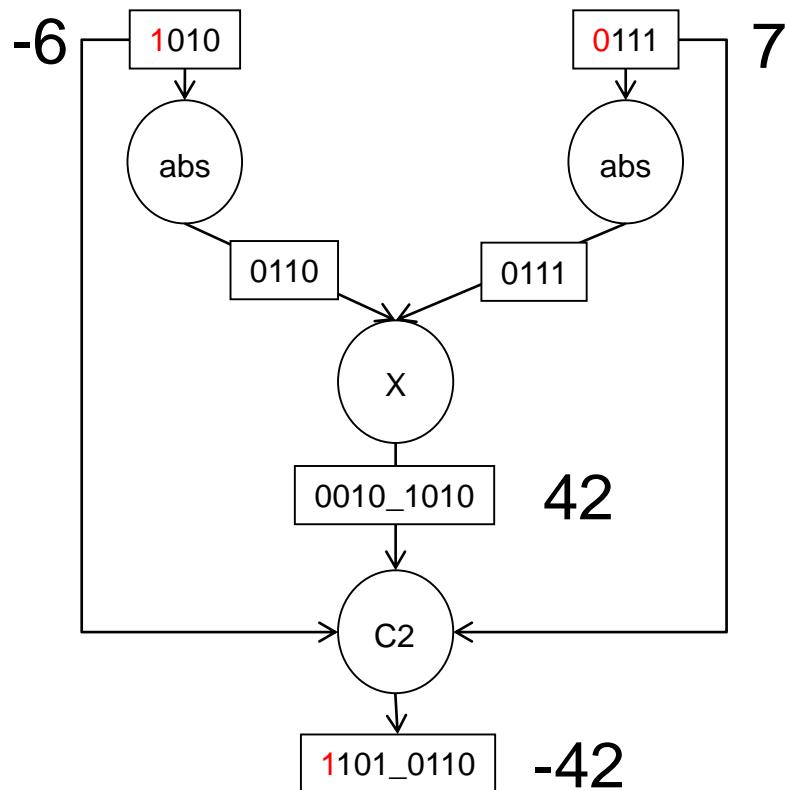
La « nature non signée ou signée des arguments» a une importance sur la structure matérielle du multiplicateur à utiliser

III Circuits combinatoires

Multiplieur (nombres entiers négatifs)

Exemple : -6×7

Nombres entiers négatifs
(complément à 2)



Il faut convertir les nombres négatifs en nombre positifs (abs) avant de les multiplier puis éventuellement réappliquer le signe ($C2$)

III Circuits combinatoires

Codage binaire - Complément à 2

- La division (/) n'est synthétisable que pour des divisions par des puissances de 2 : une division par 2^n équivaut à un décalage vers la droite de n bits

III Circuits combinatoires

Codes VHDL&Verilog - multiplicateur

VHDL

```
library IEEE;  
use IEEE.std_logic_1164.all;  
use IEEE.numeric_std.all;  
signal s1: signed(3 downto 0);  
signal s2: signed(3 downto 0);  
signal s4: signed(7 downto 0);
```

-- n bit * k bit créé n+k bit

```
s4 <= s1 * s2;
```

Verilog

```
wire signed [3:0] s1, s2;  
wire signed [7:0] s4;  
assign s2=s1*s2;
```

III Circuits combinatoires

Exercices

- Multipliez les nombres suivants

- $12 * 3 =$

- $5 * 6 =$

Faire le QCM : QCM3 SN360/SN361 Circuits combinatoires