



NOTE DE TD & COURS

FPGA

Segment 1



A. SEFOFO SOKPOR

Table des matières

I.	Représentation des nombres en informatique, base binaire, décimale et hexadécimale	3
A.	TD : Représenter les chiffres suivants en base 2 et 16	4
II.	La logique booléenne	5
a.	La porte « NON/NOT ».....	5
b.	La porte « ET/AND »	5
c.	La porte « OU/OR »	5
d.	La porte « OU exclusif/XOR »	6
e.	La porte « NAND »	6
f.	La porte « NOR »	6
g.	La porte « XNOR ».....	6
B.	TD : Réaliser des opérations basiques, les fondamentaux logique booléenne	7
C.	TD : La logique booléenne	7
D.	TD :.....	9
E.	TD : Réaliser un circuit combinant plusieurs full-adders pour additionner des chiffres sur 4 bits	14
F.	TD : Proposer un circuit réalisant l'opération de multiplication par 2 d'une entrée sur 3 bits, la sortie sera donnée sur 4 bits (faire un arrondi inférieur).	15
G.	Proposer un circuit réalisant l'opération de division par 2 d'une entrée sur 3 bits, la sortie sera donnée sur 2 bits (faire un arrondi inférieur)	15
III.	Base d'architecture d'un calculateur, mémoire et Process Unit	16
H.	TD :.....	18
I.	TD.....	18
J.	TD.....	20
K.	TD :.....	20
IV.	Electronique numérique fondamentale, le MOSFET	22
L.	TD :.....	24
M.	TD :.....	24
N.	TD :.....	33
O.	TD : Construire un full-substractor 4bits.....	35
V.	Logique numérique synchrone	39
h.	La bascule (le latch)	39
P.	TD :Bascule.....	40
i.	Le registre (flip flop)	42
Q.	TD :Bascule	42
j.	Le registre (flip flop) : la D-flipflop	44

k.	Le registre (flip flop) : La T-Fliflop.....	44
l.	Les compteurs.....	44
R.	TD :	47
S.	TD :	49

Notes de TD

Formation à l'électronique numérique FPGA

Segment 1 : Présentation du composant FPGA

Objectifs :

- Apprendre les bases du calcul numérique
 - Réaliser un premier circuit logique
 - Comprendre les architectures des calculateurs
 - Identifier les sous-composants d'un FPGA
 - Maitriser le comportement de sous-composants FPGA

I. Représentation des nombres en informatique, base binaire, décimale et hexadécimale

La forme adaptée pour permettre le traitement d'une information numérique par un circuit, est un système de numérotation de base B.

Les systèmes plus utilisés sont : décimal(base10), binaire(base2), octal(base8) et hexadécimal (base 16).

Base 16	Base 10	Base 2
0	0	0
1	1	01
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

A diagram showing the conversion of the binary number 100111 to its decimal equivalent. The binary digits are aligned with powers of 2 from right to left: 2^0 , 2^1 , 2^2 , 2^3 , 2^4 , and 2^5 . The value of each digit is calculated by multiplying it by its corresponding power of 2. The results are then summed up to get the decimal value.

Binary Digit	Power of 2	Value
1	2^0	$2^0 \times 1 = 1$
0	2^1	$2^1 \times 0 = 0$
0	2^2	$2^2 \times 0 = 0$
1	2^3	$2^3 \times 1 = 8$
1	2^4	$2^4 \times 1 = 16$
1	2^5	$2^5 \times 1 = 32$
		Decimal ← 39

De manière générale l'expression d'un nombre en base B est de la forme :

La forme polynomiale du nombre N donne :

$$(N)_B = a_n B^n + a_{n-1} B^{n-1} + \dots + a_1 B^1 + a_0 + a_{-1} B^{-1} + \dots + a_{-m} B^{-m} \dots$$

A. TD : Représenter les chiffres suivants en base 2 et 16

- 18
- 40
- 64

Conversion en base 2

Dans la base 2, tous les nombres sont exprimés à l'aide des chiffres 0 et 1, ces deux chiffres sont appelés bits (contraction de BIrary digiT).

- ✓ Le bit le plus à droite est le bit de poids le plus faible ou bien le moins significatif (LSB : Low Significant Bit).
- ✓ Le bit le plus à gauche est le bit de poids le plus fort ou bien le plus significatif (MSB : Most Significant Bit)

$$18 = 0001\ 0010$$

$$40 = 0010\ 1000$$

$$64 = 0100\ 0000$$

Conversion en base 16

Dans la base sont les dix chiffres de 0 à 9 complétés par les lettres A (pour 10), B (pour 11), C (pour 12), D (pour 13), E (pour 14) et F (pour 15).

$$18 = 0001\ 0010 = 0x12$$

$$40 = 0010\ 1000 = 0x28$$

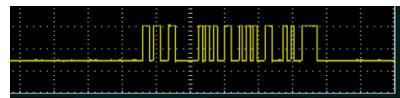
$$64 = 0100\ 0000 = 0x40$$

La représentation d'une donnée numérique lors des manipulations :

Chronogramme (simulation)



Mesure(vue physique)



II. La logique booléenne

La logique booléenne est une logique à deux états : 0 et 1. Dans la logique booléenne, les nombres sont les chiffres du système binaire (0 et 1) et les opérateurs de base sont le "OU", le "ET" et le "NON" auxquels on ajoute l'opérateur "OUI".

Chaque opérateur logique possède :

- Un symbole utilisé pour la **schématique RTL** (Register Transfer Level)
- Une **table de vérité** qui décrit le comportement de l'opérateur

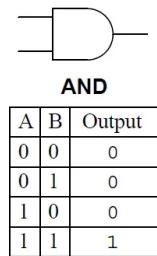
a. La porte « NON/NOT »

La porte "OUI" délivre le signal de sortie si l'information d'entrée est à l'état 1; la porte "NON" inverse le signal de sortie par rapport à celui de l'entrée.



b. La porte « ET/AND »

La porte "AND" produit une sortie à l'état logique 1 seulement si les deux entrées sont à l'état logique 1. Si l'une ou l'autre des deux entrées ou les deux à la fois sont à l'état logique 0, la sortie de cet opérateur est à l'état logique 0.



c. La porte « OU/OR »

La porte logique "OR" donne une sortie à l'état logique 1 quand l'une des entrées ou les deux à la fois sont à l'état logique 1. Elle présente une sortie à l'état logique 0 quand les deux entrées sont à l'état logique 0.



OR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

d. La porte « OU exclusif/XOR »

L'opérateur "OU Exclusif" représente le va et vient.



XOR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

e. La porte « NAND »

L'opérateur "NON ET" est l'inverse de l'opérateur "ET".



NAND

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

f. La porte « NOR »

L'opérateur "NON OU" est l'inverse de l'opérateur "OU".



NOR

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

g. La porte « XNOR »

L'opérateur "NON ET" est l'inverse de l'opérateur "ET".



XNOR

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

B. TD : Réaliser des opérations basiques, les fondamentaux logique booléenne

On suppose A = '0' et B = '1'

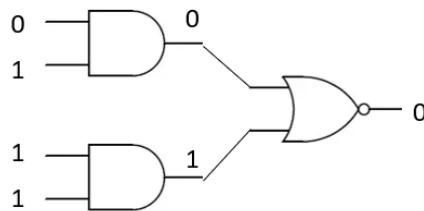
Quelle est la sortie des opérations suivantes ? Représenter le schéma logique de ces opérations.

- NOT((A AND B) OR (B AND B))

A = '0' et B = '1'.

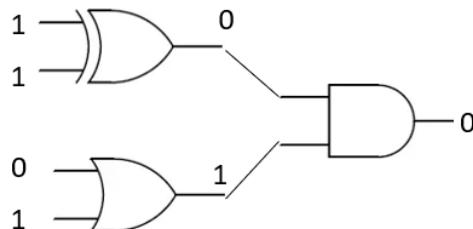
$$(A \text{ AND } B) = 0 ; (B \text{ AND } B) = 1 ; (A \text{ AND } B) \text{ OR } (B \text{ AND } B) = (0) \text{ OR } (1) = 1$$

$$\text{NOT}((A \text{ AND } B) \text{ OR } (B \text{ AND } B)) = 0$$



- (B XOR B) AND (A OR B)

$$(B \text{ XOR } B) = 0 ; (A \text{ OR } B) = 1 ; (B \text{ XOR } B) \text{ AND } (A \text{ OR } B) = 0$$



C. TD : La logique booléenne

Réaliser l'addition « A + B » en utilisant de la logique booléenne, A et B sont deux chiffres représentés sur un 1 bit.

Dans un tableau représenter par colonne

- Les tuples (combinaisons de valeurs) A et B en base binaire
- Le résultat attendu S en base 10 (décimale) pour chaque tuple
- Le résultat attendu S en base 2 (binaire)

Note : S sera représenté sur 2 bits

A	B	S(base10)	S(base2)
0	0	0	00
1	0	1	01
0	1	1	01
1	1	2	10

S(base2) représenté sur deux bits :

S(base2)	S1	S0
00	0	0
01	0	1
01	0	1
10	1	0

Simplifions ce tableau en conservant trois colonnes, A, B et S0

A	B	S0
0	0	0
1	0	1
0	1	1
1	1	0

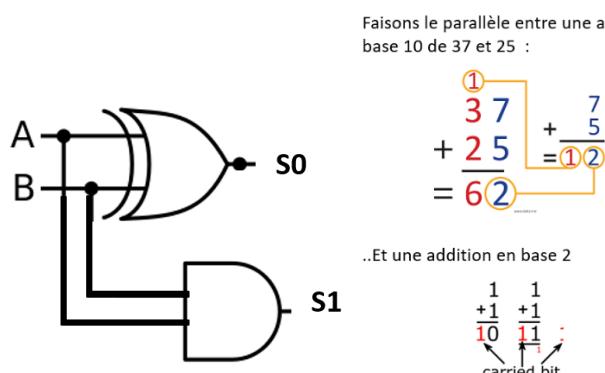
L'opérateur booléen qui permet d'obtenir S0 à partir de A et B est le OU Exclusif

Simplifions ce tableau en conservant trois colonnes, A, B et S1

A	B	S1
0	0	0
1	0	0
0	1	0
1	1	1

L'opérateur booléen permet d'obtenir S1 à partir de A et B est le AND

Représentons le schéma RTL de cette opération



S1 agit comme notre « Carried bit » ou notre retenu. La retenu comme pour l'opération en base 10 se transfert d'une addition vers une autre.

D. TD :

Améliorer le circuit précédemment élaboré afin de lui ajouter une entrée pour le « carried bit input » que vous noterez Cin, S1 sera noté Cout pour « carried bit output ».

Déterminons la table de vérité du circuit pour deux entrées A et B

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

On s'intéresse maintenant aux équations booléennes qui régissent « Cout » et « S ».

- Pour le signal « S », déterminons dans quels cas S est à 1 :

$$S = (\bar{A} \text{ and } \bar{B} \text{ and } C_{in}) \text{ or } (\bar{A} \text{ and } B \text{ and } \bar{C}_{in}) \text{ or } (A \text{ and } \bar{B} \text{ and } \bar{C}_{in}) \text{ or } (A \text{ and } B \text{ and } C_{in})$$

- Pour le signal « Cout », déterminons dans quels cas Cout est à 1

$$\begin{aligned} C_o \\ = (\bar{A} \text{ and } B \text{ and } C_{in}) \text{ or } (A \text{ and } \bar{B} \text{ and } C_{in}) \text{ or } (A \text{ and } B \text{ and } \bar{C}_{in}) \text{ or } (A \text{ and } B \text{ and } C_{in}) \end{aligned}$$

Pour plus de praticité, on utilise généralement les symboles mathématiques dans nos équations.

La syntaxe devient alors :

$$\begin{aligned} C_o &= (\bar{A} \cdot B \cdot C_{in}) + (A \cdot \bar{B} \cdot C_{in}) + (A \cdot B \cdot \bar{C}_{in}) + (A \cdot B \cdot C_{in}) \\ S &= (\bar{A} \cdot \bar{B} \cdot C_{in}) + (\bar{A} \cdot B \cdot \bar{C}_{in}) + (A \cdot \bar{B} \cdot \bar{C}_{in}) + (A \cdot B \cdot C_{in}) \end{aligned}$$

Boolean	NAME
$X = A \cdot B$	AND
$X = A + B$	OR
$X = \bar{A} \cdot \bar{B}$	NAND
$X = \bar{A} + \bar{B}$	NOR
$X = A \oplus B$	XOR
$X = \bar{A} \oplus \bar{B}$	XNOR
$X = \bar{A}$	NOT

Nous pourrions implémenter la totalité des portes logiques données dans les équations précédentes.

Cela dit, nous pouvons tenter de simplifier les équations en utilisant des **factorisations**

$$C_0 = (\bar{A} \cdot B \cdot \textcolor{red}{C_{in}}) + (A \cdot \bar{B} \cdot \textcolor{red}{C_{in}}) + (\textcolor{brown}{A} \cdot \textcolor{brown}{B} \cdot \bar{C}_{in}) + (\textcolor{brown}{A} \cdot \textcolor{brown}{B} \cdot C_{in})$$

$$C_0 = \textcolor{red}{C_{in}}(\bar{A} \cdot B + A \cdot \bar{B}) + \textcolor{brown}{A} \cdot \textcolor{brown}{B}(\bar{C}_{in} + C_{in})$$

$$C_0 = C_{in}(\bar{A} \cdot B + A \cdot \bar{B}) + A \cdot B(\bar{C}_{in} + C_{in})$$

L'expression $(\bar{A} \cdot B + A \cdot \bar{B})$ peut-elle « compressée » en une seule fonction booléenne ? Est-ce aussi le cas de l'expression $(\bar{C}_{in} + C_{in})$?

Pour le vérifier, établissons la table de vérité de l'expression $(\bar{A} \cdot B + A \cdot \bar{B})$

On procède par étape, les entrées A, B, Puis les expressions d'ordre supérieur, $\bar{A} \cdot B$ et $A \cdot \bar{B}$

A	B	\bar{A}	\bar{B}
0	0	1	1
0	1	1	0
1	0	0	1
1	1	0	0

\bar{A} and B	A and \bar{B}
0	0
1	0
0	1
0	0

Finalement la table de vérité complète

A	B	\bar{A}	\bar{B}	\bar{A} and B	A and \bar{B}	$(\bar{A} \text{ and } B) \text{ or } (A \text{ and } \bar{B})$
0	0	1	1	0	0	0
-	1	1	0	1	0	1
è0						
1	0	0	1	0	1	1
1	1	0	0	0	0	0

On remarque que l'expression finale correspond à la fonction « XOR », donc

$$(\bar{A} \cdot B + A \cdot \bar{B}) \Leftrightarrow \mathbf{A \text{ Xor } B}$$

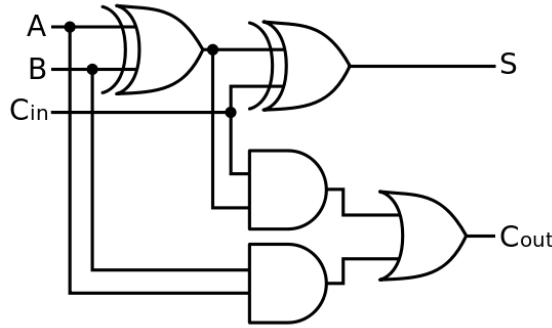
On opère de la même façon pour $(\bar{C}_{in} + C_{in})$:

Cin	\bar{C}_{in}	Cin or \bar{C}_{in}
0	1	1
1	0	1

Ce qui correspond à une fonction toujours vraie, de fait, elle peut être symbolisée par un « 1 »

$$C_0 = C_{in}(\bar{A} \cdot B + A \cdot \bar{B}) + A \cdot B(\bar{C}_{in} + C_{in}) \Leftrightarrow C_0 = C_{in}(A \oplus B) + A \cdot B \cdot 1$$

On obtient une première partie du circuit :



Pour le signal S, on applique la même méthode de factorisation

$$\begin{aligned} S &= (\bar{A} \cdot \bar{B} \cdot C_{in}) + (\bar{A} \cdot B \cdot \overline{C_{in}}) + (A \cdot \bar{B} \cdot \overline{C_{in}}) + (A \cdot B \cdot C_{in}) \\ S &= C_{in}(\bar{A} \cdot \bar{B} + A \cdot B) + \overline{C_{in}}(\bar{A} \cdot B + A \cdot \bar{B}) \end{aligned}$$

La table de vérité des expressions $(\bar{A} \cdot \bar{B} + A \cdot B)$ et $(\bar{A} \cdot B + A \cdot \bar{B})$ et simplifions les pour obtenir une seule porte logique comme précédemment. Nous obtenons alors

$$S = \overline{C_{in}}(A \oplus B) + C_{in}(\overline{A \oplus B})$$

On remarque un terme redondant $(A \oplus B)$

Appelons cette expression Y pour transformer l'équation en

$$S = \overline{C_{in}}(Y) + C_{in}(\bar{Y})$$

Réitérer la méthode de simplification précédemment utilisée, nous obtenons :

$$S = C_{in} \oplus Y \text{ soit } S = C_{in} \oplus A \oplus B$$

Plus la table de vérité est grande plus il devient long de résoudre les équations booléennes. Une solution est d'utiliser la table de karnaugh (K map)

On vient construire un tableau de taille 2^n avec n le nombre d'entrée du circuit. On donne les combinaisons possibles de a et b. Ceci permet de réduire la taille du tableau à lire.

b	a	X	x	Not(a)	a	x	Not(a)	a	x	Not(a)	a
0	0	1	Not(b)			Not(b)	1	1	Not(b)	1	1
0	1	1									
1	0	1									
1	1	0	b			b	1	0	b	1	0

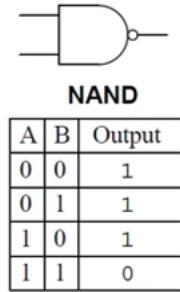
Pour chaque combinaison où X est vrai, inscrire « 1 » dans le tableau (2).

Vient alors la simplification, s'il existe des cases adjacentes à « 1 » alors une simplification est possible.

On observe les termes communs associés, ici Not(A) et Not(B) ; il s'agit d'un NAND (Tableau3).

b	a	X
0	0	1
0	1	1
1	0	1
1	1	0

X	Not(a)	a
Not(b)	1	1
b	1	0



On observe les termes communs associés, ici **Not(A)** et **Not(B)**; l'équation est alors **Not(B) OR Not(A) \Leftrightarrow NOT(A AND B) \Leftrightarrow A NAND B**

On observe les termes communs associés, ici **Not(A)** et **Not(B)**; l'équation est alors **Not(B) OR Not(A)**. Si on trace la table de vérité de cette expression, nous retrouvons un l'opérateur NAND.

Dans le cas où les cases ne sont pas adjacentes, et qu'aucun regroupement n'est possible, alors les opérations sont dites « exclusives »

X	Not(a)	a
Not(b)	0	1
b	1	0

L'équation associée est alors
X = (A and not(B)) or (B and not(A))

Équivalente à

X = A XOR B

L'équation associée est alors **X = A XOR B**, si on trace la table de vérité nous obtenons **XOR**

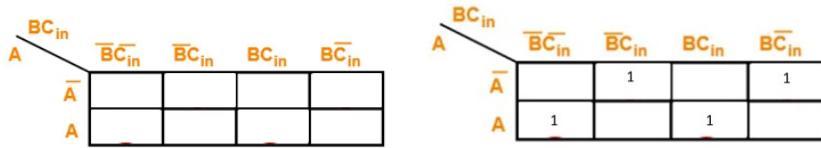
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Dans le cas de quatre entrées, nous regroupons les entrées par paires, H dépend de S1, S2, S3 et S4 dans ce cas. Nous regroupons S1 et S2 puis S3 et S4.

S1	S2	S3	S4	H
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

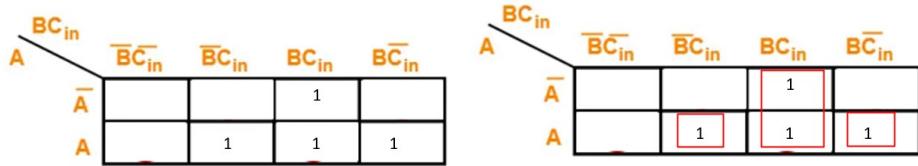
		H	
		S1.S2	S3.S4
		00	01
		1	1
00			
01			
11			
10			

Construisons la K-map pour la sortie S du circuit



$$S = A \cdot \text{not}(B) \cdot \text{not}(C) \text{ XOR } \text{not}(A) \cdot \text{not}(B) \cdot C \text{ XOR } A \cdot B \cdot C \text{ XOR } \text{not}(A) \cdot B \cdot \text{not}(C)$$

Construisons la K-map pour la sortie Cout du circuit



$$C_o = B \cdot C_{in} \text{ or } (A \text{ and } \overline{B} \text{ and } C_{in}) \text{ or } (A \text{ and } B \text{ and } \overline{C_{in}})$$

Simplifions l'équation on trouve : $S = C_{in} \oplus A \oplus B$

$$S = \text{A} \cdot \text{not}(B) \cdot \text{not}(C) \text{ XOR } \text{not}(A) \cdot \text{not}(B) \cdot C \text{ XOR } A \cdot B \cdot C \text{ XOR } \text{not}(A) \cdot B \cdot \text{not}(C)$$

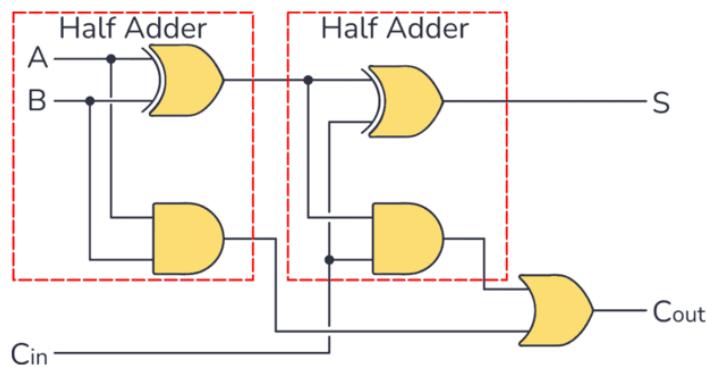
$$S = C \cdot (nA \cdot nB \text{ XOR } A \cdot B) \text{ XOR } nC \cdot (A \cdot nB \text{ XOR } nA \cdot B)$$

$$S = C \cdot (A \text{ XNOR } B) \text{ XOR } nC \cdot (A \text{ XOR } B)$$

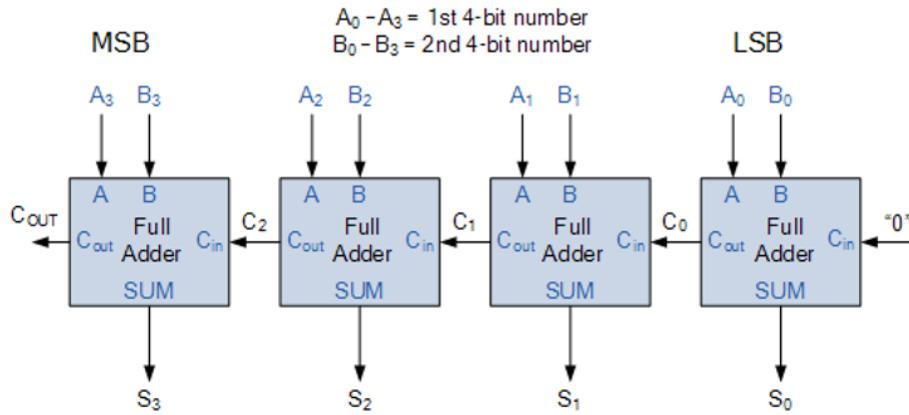
$$S = C \cdot (nY) \text{ XOR } nC \cdot (Y), \text{ sachant } Y = A \text{ XOR } B$$

$$\text{Donc } S = C_{in} \oplus A \oplus B$$

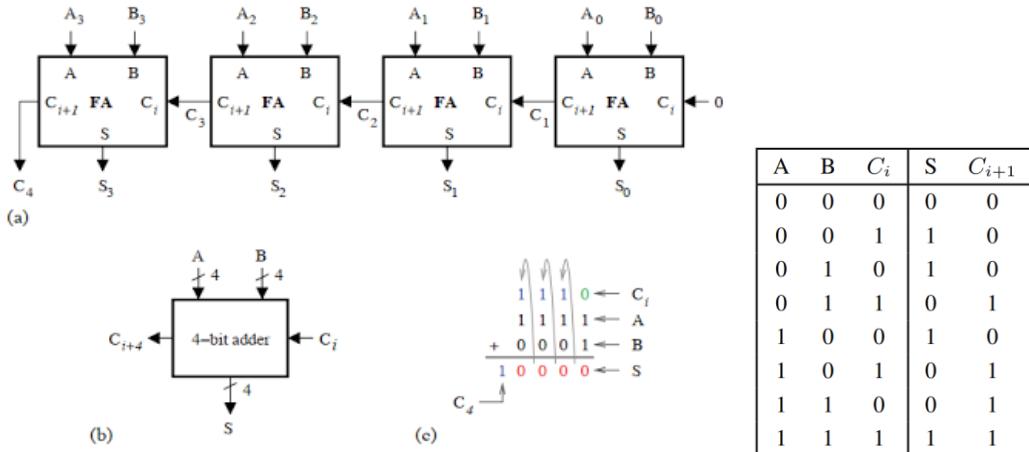
On observe que le principe est de cascader deux additionneurs



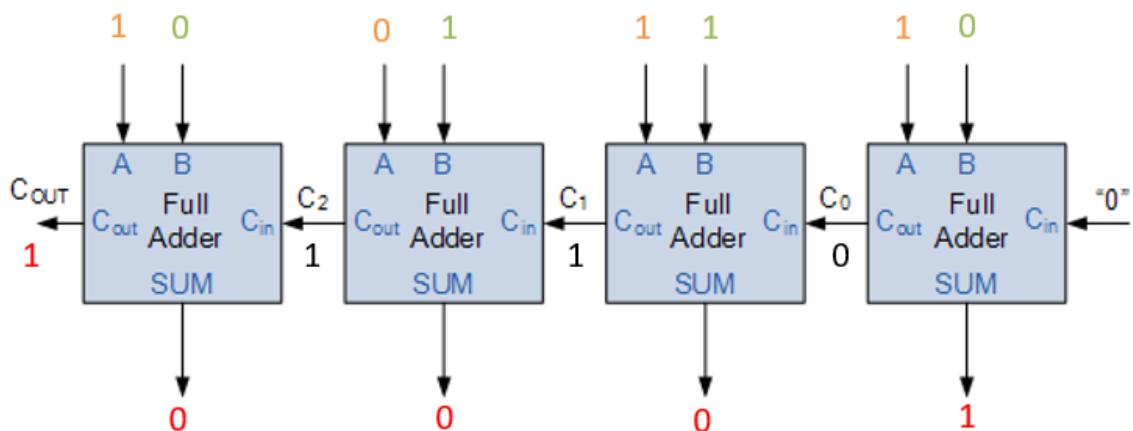
E. TD : Réaliser un circuit combinant plusieurs full-adders pour additionner des chiffres sur 4 bits



Présenter cette fonction sous la forme d'un paragraphe avec un exemple illustratif.



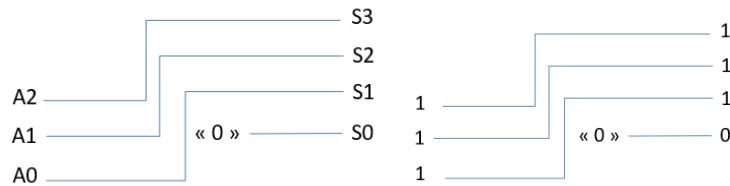
D'après la table de vérité donnée pour le full-adder, le comportement de la chaîne de full-adders est le suivant :



Nous obtenons $S = 1\ 0001_{(b2)} \Leftrightarrow 11_{(b16)} \Leftrightarrow 17_{(b10)}$

F. TD : Proposer un circuit réalisant l'opération de multiplication par 2 d'une entrée sur 3 bits, la sortie sera donnée sur 4 bits (faire un arrondi inférieur).

Multiplication par 2, 7*2



Base2			N	Multiplication base 2				N*2
A	B	C	Base(10)	P(4)	P(3)	P(2)	P(1)	Base(10)
0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	1	0	2
0	1	0	2	0	1	0	0	4
0	1	1	3	0	1	1	0	6
1	0	0	4	1	0	0	0	8
1	0	1	5	1	0	1	0	10
1	1	0	6	1	1	0	0	12
1	1	1	7	1	1	1	0	14

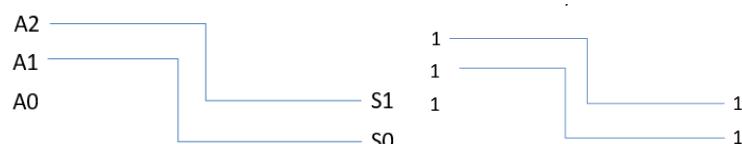
Lorsqu'on multiplie une entrée par une puissance de 2 (2, 4, 8, 16 ect..) il suffit d'opérer un décalage binaire respectivement à droite, le nombre de décalage dépend de la puissance de 2 utilisée.

$2 \times 2 = 4$ soit $10(b2) \times 10(b2) = 100(b2)$, on a ajouté un zéro à droite et décaler notre entrée de 1 bit

$5 \times 2 = 10$ soit $101(b2) \times 10(b2) = 1010(b2)$, on a ajouté un zéro à droite et décaler notre entrée de 1 bit

G. Proposer un circuit réalisant l'opération de division par 2 d'une entrée sur 3 bits, la sortie sera donnée sur 2 bits (faire un arrondi inférieur)

Division par 2, 7/2



Base2			N	Division base 2				N/2
A	B	C	Base(10)	D(4)	D(3)	D(2)	D(1)	Base(10)
0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0
0	1	0	2	0	0	0	1	1
0	1	1	3	0	0	0	1	1
1	0	0	4	0	0	1	0	2
1	0	1	5	0	0	1	0	2
1	1	0	6	0	0	1	1	3
1	1	1	7	0	0	1	1	3

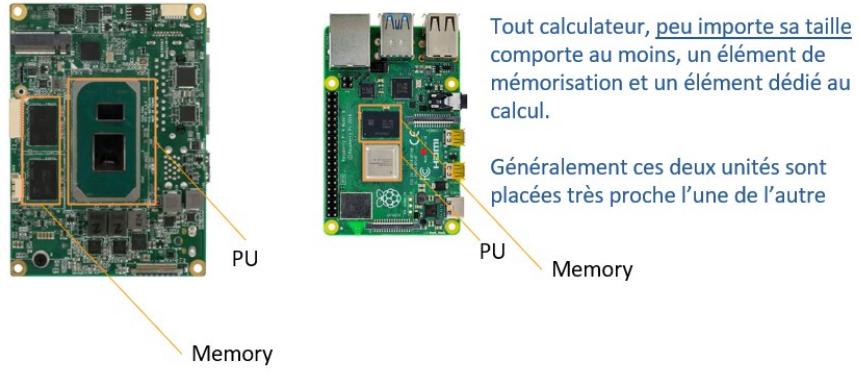
Lorsqu'on divise une entrée par une puissance de 2 (2, 4, 8, 16 etc..) il suffit d'opérer un décalage binaire respectivement à gauche, le nombre de décalage dépend de la puissance de 2 utilisée.

$6 / 2 = 3$ soit $110(b2) / 10(b2) = 011(b2)$, on a retiré un digit à gauche et décaler notre entrée de 1 bit à droite (suppression du **LSB**)

$7 / 2 = 3$ soit $111(b2) / 10(b2) = 011(b2)$, on a retiré un digit à gauche et décaler notre entrée de 1 bit à droite (suppression du **LSB**)

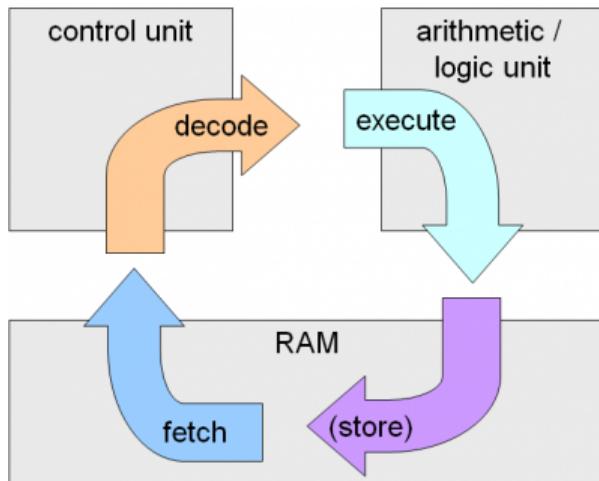
$7 / 4 = 1$ soit $111(b2) / 100(b2) = 001(b2)$, on a retiré deux digits à gauche et décaler notre entrée de 2 bits à droite (suppression des deux **LSB**)

III. Base d'architecture d'un calculateur, mémoire et Process Unit



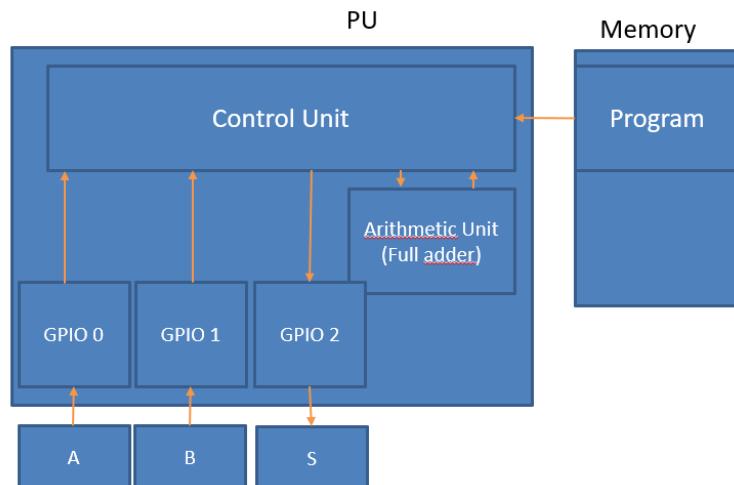
Lorsque les instructions sont enregistrées dans la mémoire ; c'est la programmation

Les instructions sont exécutées dans la PU ; c'est l'inférence.

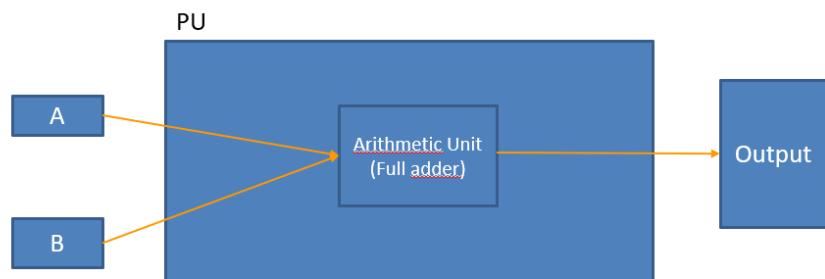


Ce mode d'exécution **est très versatile**, cependant, **chaque étape prend du temps** qui augmente la latence d'inférence (« ça RAM! ») par ailleurs chaque étape à un **coût en énergie non négligeable** lorsque répété plusieurs fois.

Etude de cas addition de deux chiffres par un CPU



Dans un composant FPGA, la logique est différente, nous ne sommes pas soumis à cette séquence d'étapes qui peut être longue, on peut faire des calculs dit « à la volée »



Ce mode d'exécution est peu versatile, en revanche, la consommation énergétique est basse car peu voire aucun accès externe. La latence est extrêmement courte car l'exécution ne dépend pas d'accès mémoire.

Pseudo Code :

Lire GPIO 0, Lire GPIO 1, Appliquer Full adder , Mettre à jour GPIO 2

H. TD :

Supposons qu'un cycle d'inférence d'instruction prends environ 9 us (donc chaque instruction du pseudo code mettra 9 us à s'exécuter). Calculer la **latence du système**, c'est-à-dire le temps écoulé entre le début de la première instruction du pseudo code et la fin de la dernière instruction. Calculer le temps qu'il faudrait pour répéter 500 fois le pseudo donné.

Latence : $9\text{us} \times 4 = 36\text{ us}$. Exécution de 500 itérations : $36\text{us} \times 500 = 18\text{ ms}$

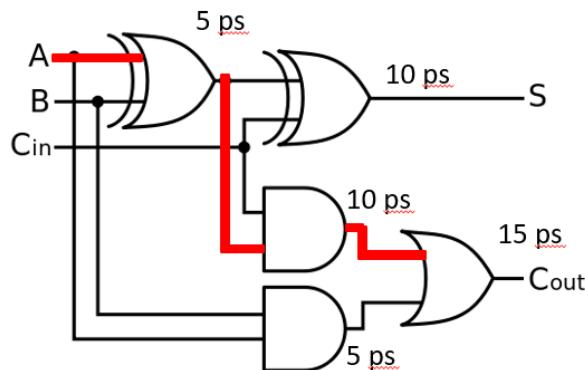
Dans le cas d'un CPU, plusieurs processus peuvent être exécutés simultanément ; les ressources sont alors partagées on admet qu'un autre pseudo code est ajouté dans notre programme, **l'utilisation du full adder est mutualisée**. Quelle incidence cela peut-elle avoir sur la latence système ? Sur le temps d'exécution à 500 cycles ?

Latence : une incertitude sur la latence qui peut être au minium de 36us, il peut y avoir du jitter.
Exécution de 500 itérations : même observation que pour la latence

I. TD

Supposons que A et B sont sur 4 bits chacun et que le temps de propagation d'une porte logique est de 5ps. Calculer la latence du système, c'est-à-dire le temps écoulé entre le début de la première instruction du pseudo code et la fin de la dernière instruction. Calculer le temps qu'il faudrait pour répéter 500 fois le pseudo donné.

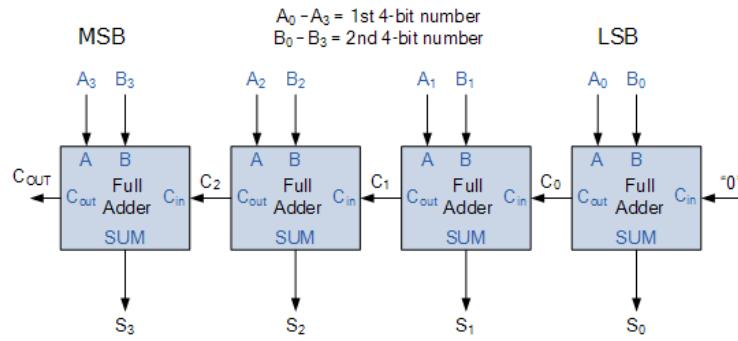
Pour chaque chemin on compte le nombre de porte logique que l'on traverse, on ajoute 5ps à chaque passage. S'il y a plusieurs entrées impliquées, on va observer le chemin le **plus long**. Par exemple, pour S, nous traversons 2 portes XOR (le chemin le plus long étant celui partant de A&B).



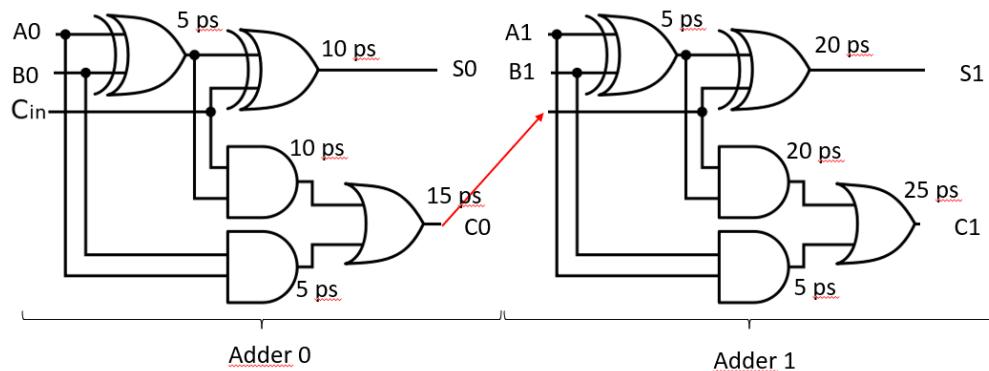
Le résultat en sortie du circuit est complet lorsque nous obtenons à la fois S et Cout. Ici on note la latence pour chaque point du circuit. La latence pour obtenir S est de 10ps et pour de 15 ps pour Cout.

Note : déterminer quel est le chemin qui passe par le plus de porte logique dans le circuit du full-adder.

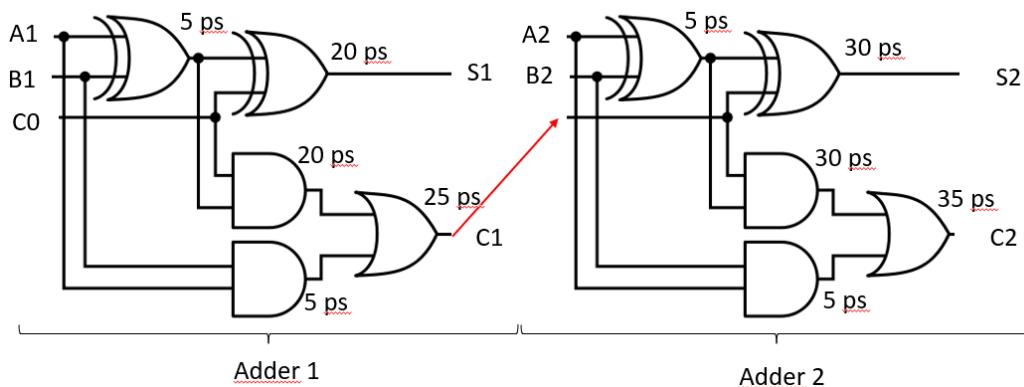
Dans notre cas, nous avons une **cascade** de full-adders, nous devons donc observer le **chemin critique** total.



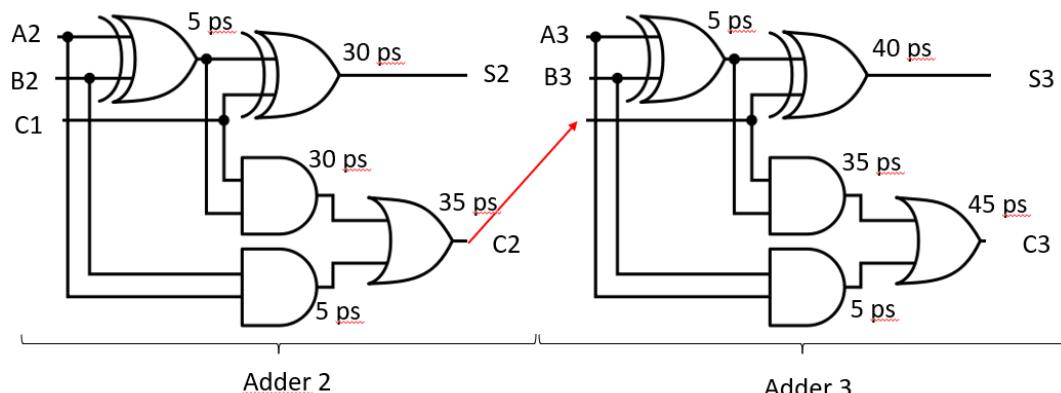
C0 se transfert de l'Adder 0 vers l'Adder 1, déterminons la latence pour obtenir **S1 et C1**



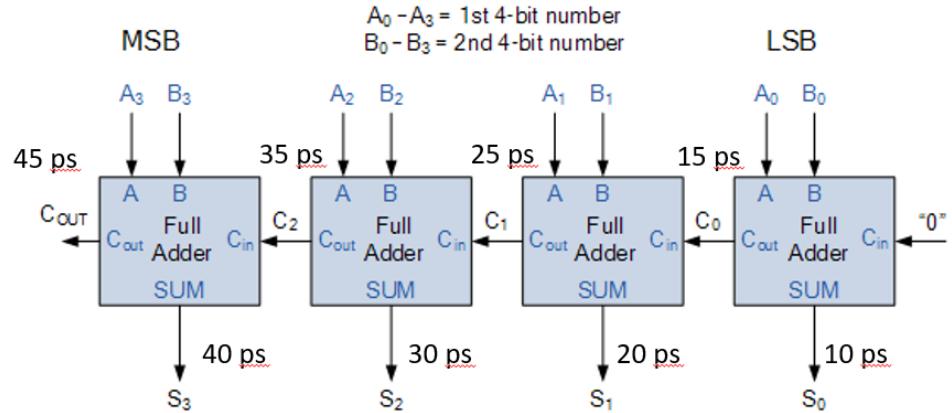
C1 se transfert de l'Adder 1 vers l'Adder 2, déterminons la latence pour obtenir **S2 et C2**



C2 se transfert de l'Adder 2 vers l'Adder 3, déterminons la latence pour obtenir **S3 et C3**



Dans notre cas, nous avons une cascade de full-adders, nous devons donc observer le chemin critique total.



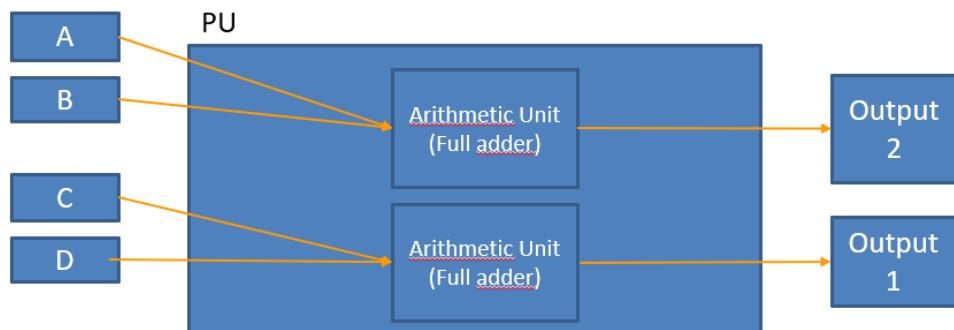
J. TD

Supposons que A et B sont sur 4 bits chacun et que **le temps de propagation** d'une porte logique est de 5ps. Calculer la **latence du système**, c'est-à-dire le temps écouté entre le début de la première instruction du pseudo code et la fin de la dernière instruction. Calculer le temps qu'il faudrait pour répéter 500 fois le pseudo donné

La latence est de 45ps, la répétitions du pseudo code serait alors de $500 * 45\text{ps} = 22.5\text{us}$.

Supposons le cas où nous devons exécuter un autre processus d'addition comme dans l'exemple du CPU.

Comment traiter ce cas ? Quelle incidence cela pourrait avoir sur notre latence pour obtenir nos deux résultats ? Quelle incidence sur nos consommations en **ressources logiques (Process Units)** ?

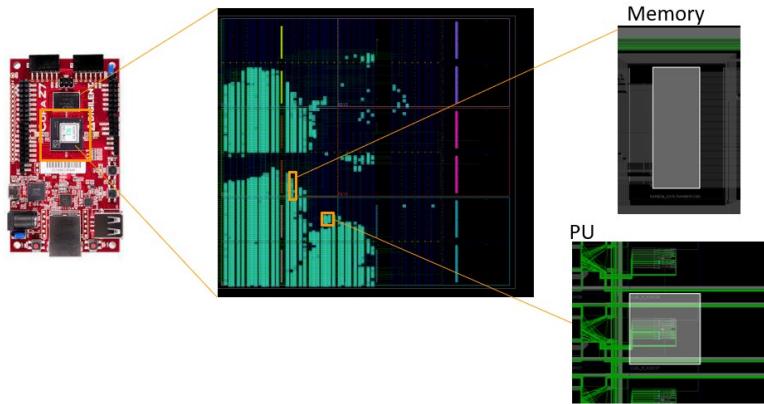


Pour traiter ce cas, il suffit de **paralléliser les traitements**.

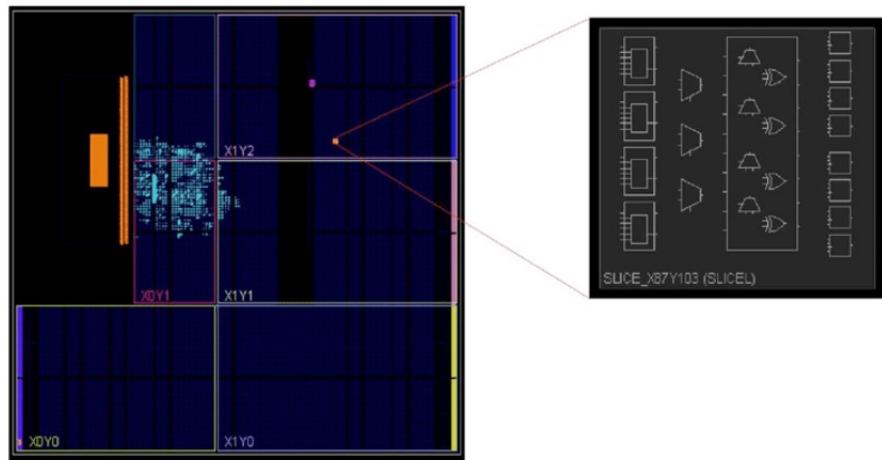
La latence **reste alors inchangée**. On utilisera deux fois plus de Process Units, aussi appelées **Logical Elements (LE)** dans le contexte d'un **FPGA**.

K. TD :

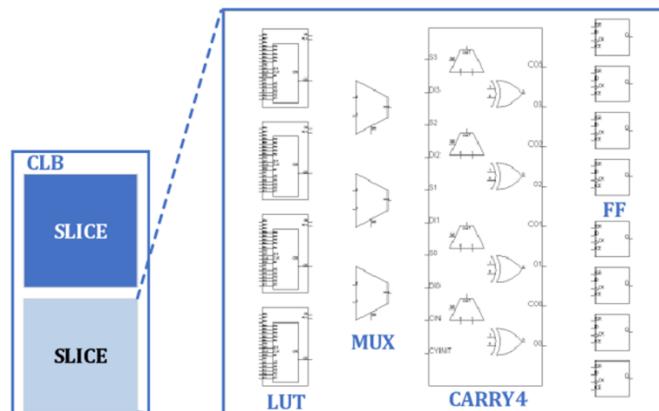
Exploration guidée, étude des composants internes du zynq7010, focus sur les sous-composants du FPGA ; les **Logical Elements** ainsi que la mémoire embarquée (**emBedded RAM ou BRAM**).



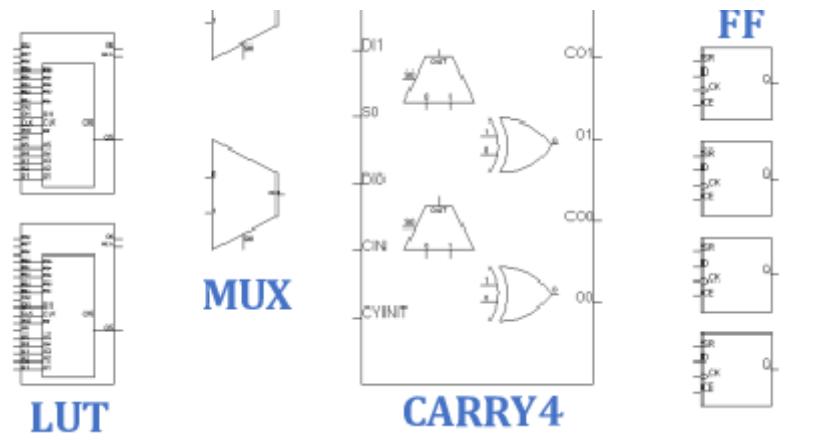
Pourquoi ne voit-on aucune porte logique OR XOR AND ? Qu'est qu'une slice ?



Une slice est un regroupement de **composants logiques bas niveau**. Chaque FPGA possède sa propre architecture de Slice, lui conférant des performances différentes des autres.



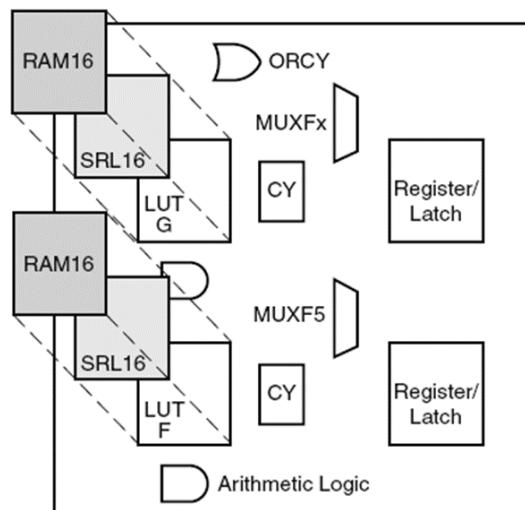
La LUT est une Look Up Table, elle permet de reproduire le comportement d'une porte logique. Le MUX permet de faire un aiguillage des signaux dans la Slice ou d'un Slice à une autre. La FF, Flip Flop est un registre et permet la mémorisation d'un signal. Chaque FF permet de mémoriser 1 bit, c'est un composant synchrone.



La **Carry4** est un composant ajouté par Xilinx dans la série zynq7010 et permet d'optimiser les fonctions de comptage très récurrentes dans les designs FPGA. Il s'agit d'un composant spécialement fait pour les séries zynq7.

Plus d'information sur ce lien <https://www.fpga4fun.com/Counters4.html>

D'autres FPGA vont regrouper d'autres fonctions dans leur Slice. Par exemple la série Virtex aura d'autres fonctions. Cela dit, on retrouve toujours des LUTs, des MUX et des registres.



Ce qu'il faut retenir, c'est qu'une Slice parfois appelée aussi « Logic Cell » va toujours contenir : Des LUTs, Des multiplexeurs (MUX), Des registres (FF).

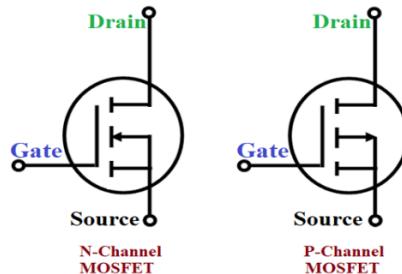
Selon la série, ces éléments sont présents en plus ou moins grande quantité.

IV. Electronique numérique fondamentale, le MOSFET

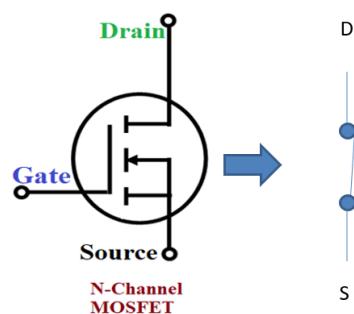
Le transistor MOS est le composant de base de tout circuit numérique. Une étude en détail de ce composant est disponible en référence ; dans le cadre de cette formation, nous nous limiterons à une compréhension simplifiée de ce dernier. Les MOSFETs possèdent 3 broches notées ci-après. Il en existe deux types, le MOSFET N et le MOSFET P.

Selon la tension appliquée entre la broche **Gate** et la broche **Source** (V_{GS}) les broches **Drain** et **Source** peuvent être reliées entre elles par un **canal**.

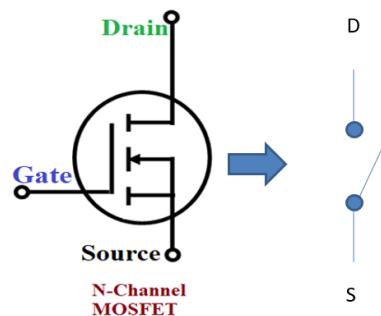
Exemple avec V_{GS} dans une configuration qui relie Drain et Source.



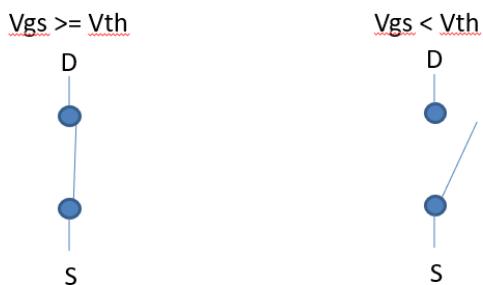
Exemple avec V_{GS} dans une configuration qui relie Drain et Source.



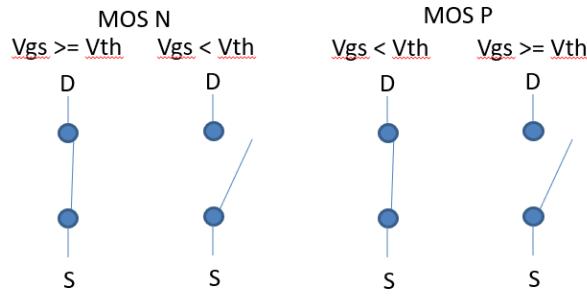
Exemple avec V_{GS} dans une configuration qui sépare Drain et Source.



Avec un peu plus de précision, si V_{GS} est supérieur à une valeur notée V_{th} (Threshold) alors Drain et Source sont reliées, autrement Drain et Source sont séparées. V_{th} est fixe et dépend du composant MOS utilisé.



Le MOS N et le MOS P ont des comportements opposés ; la règle ouverture / fermeture de l'interrupteur dans le cas du MOS P est la contraposée de la règle du MOS N.



L. TD :

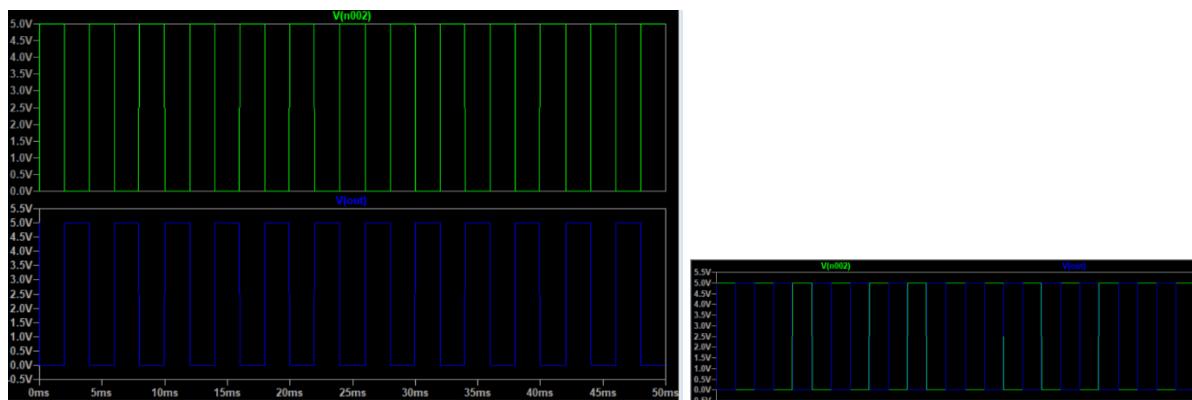
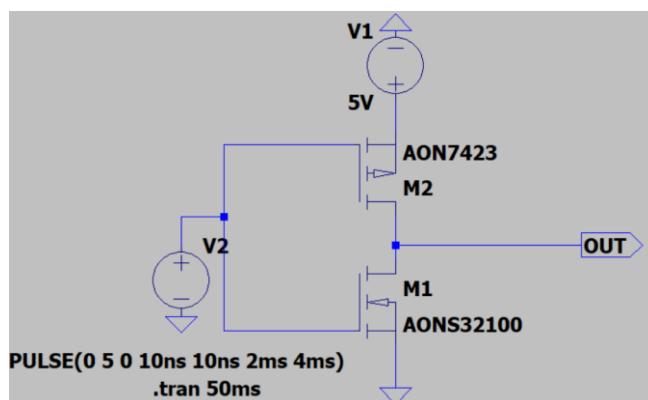
Modélisation de portes logiques et simulation sous Ltspice.

Reproduire le schéma ci-dessous sous Ltspice, utilisez « F2 » pour accéder à la librairie de composants et chercher « nmos », « Voltage » et « res ». Pour la masse utiliser le menu dans la barre supérieure

M. TD :

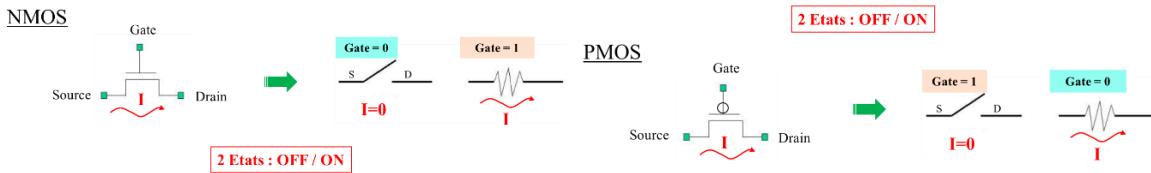
Porte Inverseuse, NOT

Réaliser un commentaire sur le fonctionnement de ce circuit, que se passe t'il lorsque $V_2 = 0V$? Lorsque $V_2 = 5V$? Le circuit a-t-il une **latence strictement parfaite** ? C'est-à-dire aucun délai entre l'entrée du circuit et la sortie.



Le circuit se comporte en inverseuse, la porte Not.

L'inverseur est constitué de deux transistors, un **NMOS** et un **PMOS**, connectés en série entre l'**alimentation** qui constitue le 1 logique et la **masse** qui représente le 0 logique.

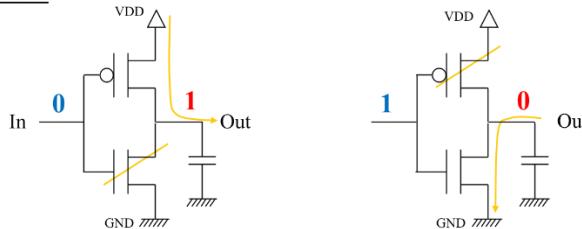


Lorsque, dans l'inverseur réalisé plus haut, l'entrée est basse, aucun courant ne passe. Lorsque l'entrée est haute, le NMOS s'active et « tire » la sortie vers l'état bas, on parle de « *pull-down* ». De manière symétrique, on peut concevoir un circuit avec un PMOS qui « tire » la sortie vers l'état haut, à savoir un « *pull-up* », lorsque l'entrée est basse.

En associant les deux circuits, *pull-up* et *pull-down*, on fabrique un inverseur CMOS. Voici comment il se comporte :

- Lorsque l'entrée est 0 ou basse ($V_2=0V$, $V_{GS} < V_{TH}$ et $V_s=5V$), le PMOS est passant donc conduit, le NMOS bloque. La sortie est directement reliée à l'alimentation, donc à l'état haut.
- Lorsque l'entrée est 1 ou haute ($V_2=5V$; $V_{GS} > V_{TH}$ et $V_{DS} > (V_{GS} - V_{TH})$), le NMOS est passant, donc conduit et le PMOS bloque. La sortie est directement reliée à la masse, donc à l'état bas.

Fonctionnement :



Le transistor NMOS est passant si sa Grille est à 1. Le transistor PMOS est passant si sa Grille est à 0. La Grille du PMOS et celle du NMOS, reliées, constituent l'**entrée** de l'inverseur. Le Drain du PMOS et celui du NMOS, reliés, constituent la **sortie** de l'inverseur.

En résumé un 0 en entrée de l'inverseur donne un 1 en sortie et un 1 en entrée de l'inverseur donne un 0 en sortie.

Symbole IEEE :

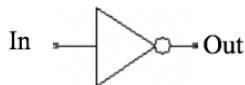
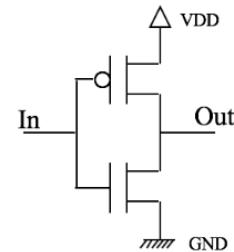


Table de vérité

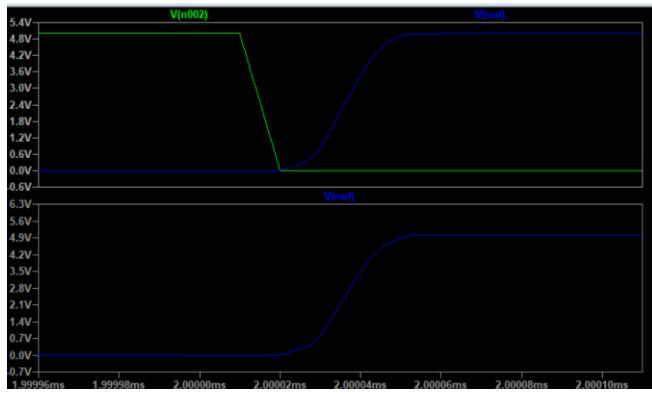
In	Out
0	1
1	0
X	X

Implémentation



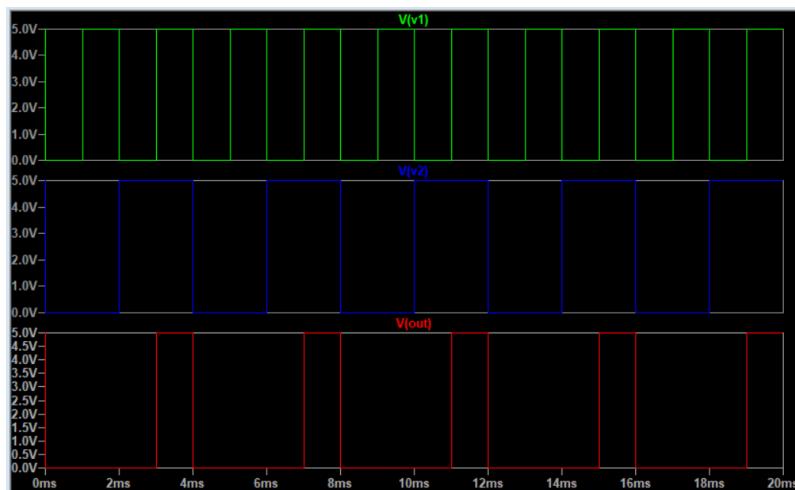
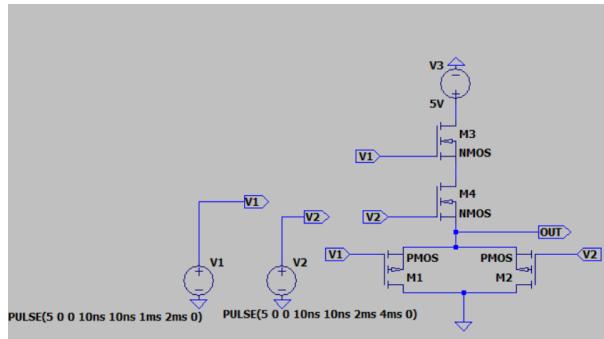
Le circuit a-t-il une **latence strictement parfaite** ? C'est-à-dire aucun délai entre l'entrée du circuit et la sortie.

Non, il y a un délai entre l'entrée et la sortie du circuit.



Porte AND

Réaliser un commentaire sur le fonctionnement de ce circuit, que se passe t'il lorsque $V_2 = 0V$? Lorsque $V_2 = 5V$? Le circuit a-t-il une latence strictement parfaite? C'est-à-dire aucun délai entre l'entrée du circuit et la sortie.



En associant deux circuits *pull-up* et deux circuits *pull-down*, soit 4 transistors, on fabrique une porte AND CMOS. Voici comment il se comporte :

- $V_1=0, V_2=0$

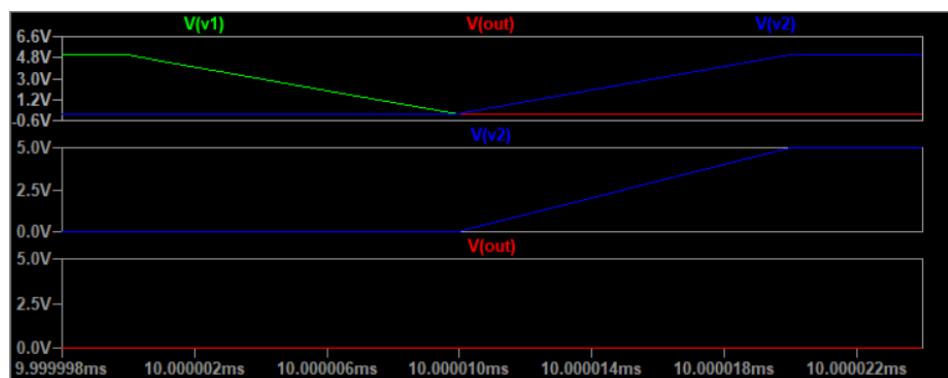
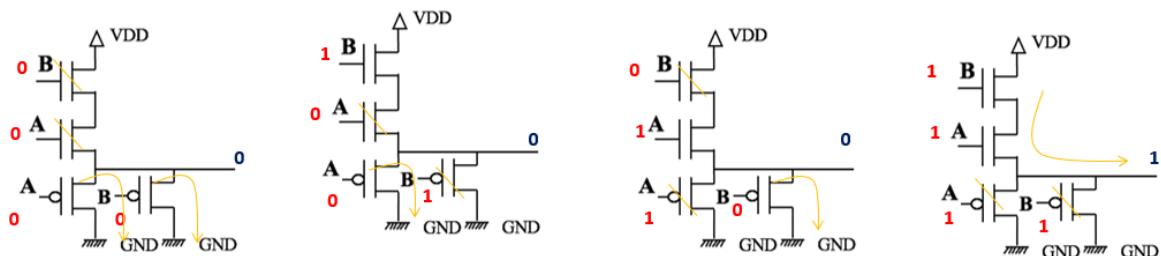
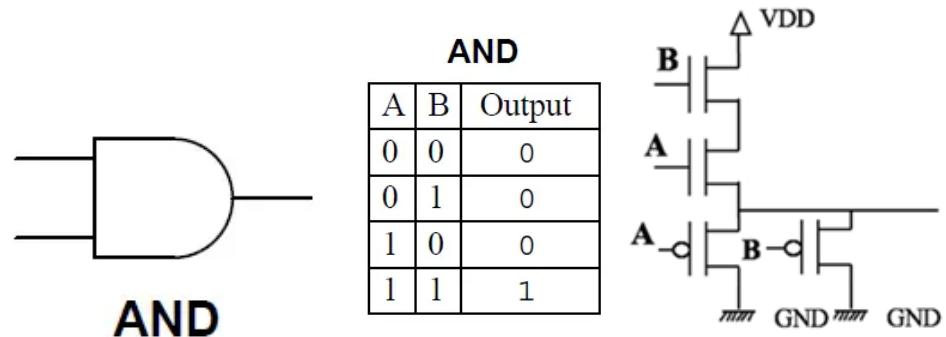
Lorsque les entrées sont à 0 ou basse ($V_2=0V$, $V_{GS} < V_{TH}$ et $V_s=5V$), les PMOS sont passants donc conduits. Les grilles des deux transistors NMOS étant à 0, ils sont bloqués. La sortie est directement reliée à l'alimentation à travers les deux PMOS, donc passe alors à l'état bas.

- $V1=0, V2=1$

Lorsqu'une entrée est à 1 et la deuxième à 0, le NMOS dont la grille est 0 est ouvert, le deuxième est passant et le PMOS dont la grille est à 1 est ouvert, le deuxième dont la grille est à 0 est passant. Ceci permet le passage du courant de se décharger d'une capacité vers la masse. La tension de sortie passe alors à l'état bas.

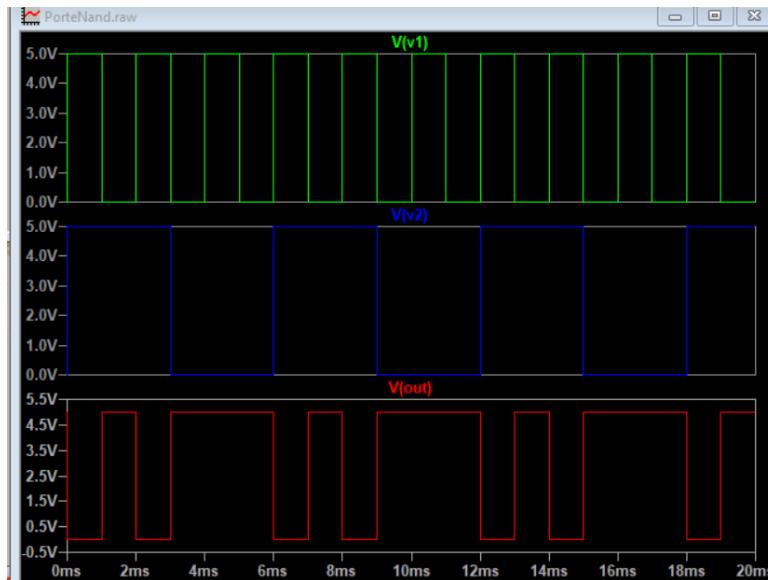
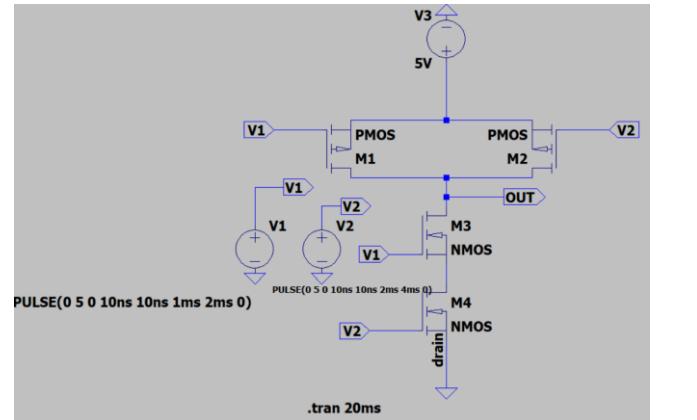
- $V1=1, V2=1$

Dans ce cas les deux transistors PMOS sont ouverts et les deux transistors NMOS sont passants. Le courant se charge la capacité de sortie à travers les deux transistors NMOS, et la sortie passe à l'état haut.



Porte NAND

Réaliser un commentaire sur le fonctionnement de ce circuit, que se passe t'il lorsque $V_2 = 0V$? Lorsque $V_2 = 5V$? Le circuit a-t-il une latence strictement parfaite? C'est-à-dire aucun délai entre l'entrée du circuit et la sortie.



En associant deux circuits *pull-up* et deux circuits *pull-down*, soit 4 transistors, on fabrique une porte NAND CMOS. Voici comment il se comporte :

- $V_1=0, V_2=0$

Lorsque les entrées sont à 0 ou basse ($V_2=0V$, $V_{GS} < V_{TH}$ et $V_s=5V$), les PMOS sont passants donc conduits. Les grilles des deux transistors NMOS étant à 0, ils sont bloqués. La sortie est directement reliée à l'alimentation à travers les deux PMOS, donc passe alors à l'état haut.

- $V_1=0, V_2=1$

Lorsqu'une entrée est à 1 et la deuxième à 0, le NMOS dont la grille est 0 est ouvert, le deuxième est passant et le PMOS dont la grille est à 1 est ouvert, le deuxième dont la grille est à 0 est passant. Ceci permet le passage du courant vers la sortie. Elle passe alors à l'état haut.

- $V_1=1, V_2=1$

Dans ce cas les deux transistors PMOS sont ouverts et les deux transistors NMOS sont passants. Le courant se décharge à travers les deux transistors, et la sortie passe à l'état bas.

LA PORTE NAND

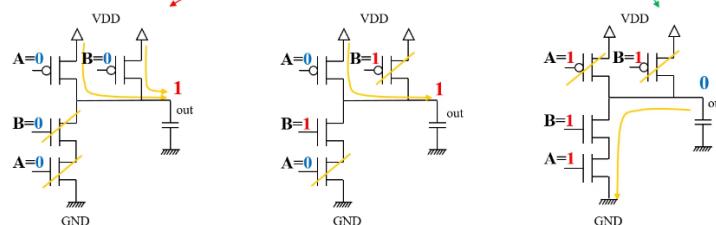
Symbole IEEE :



Table de vérité

A	B	out
0	0	1
0	1	1
1	0	1
1	1	0

Implémentation



On vérifie ainsi les fonctions de la porte NAND.

Symbole IEEE :

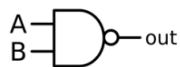
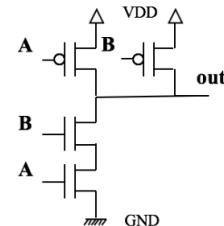


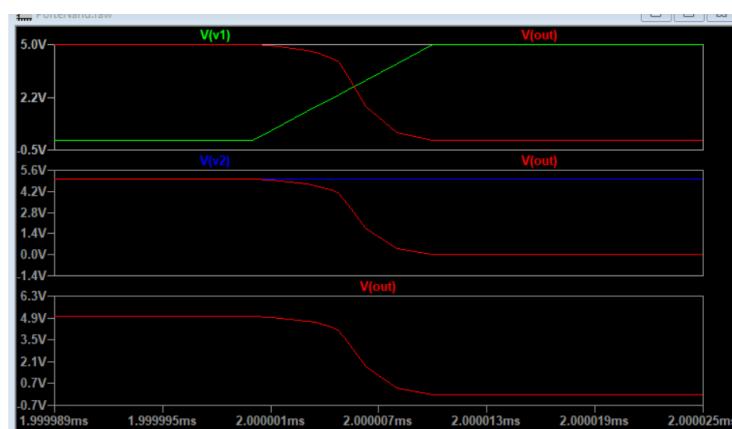
Table de vérité

A	B	out
0	0	1
0	1	1
1	0	1
1	1	0

Implémentation

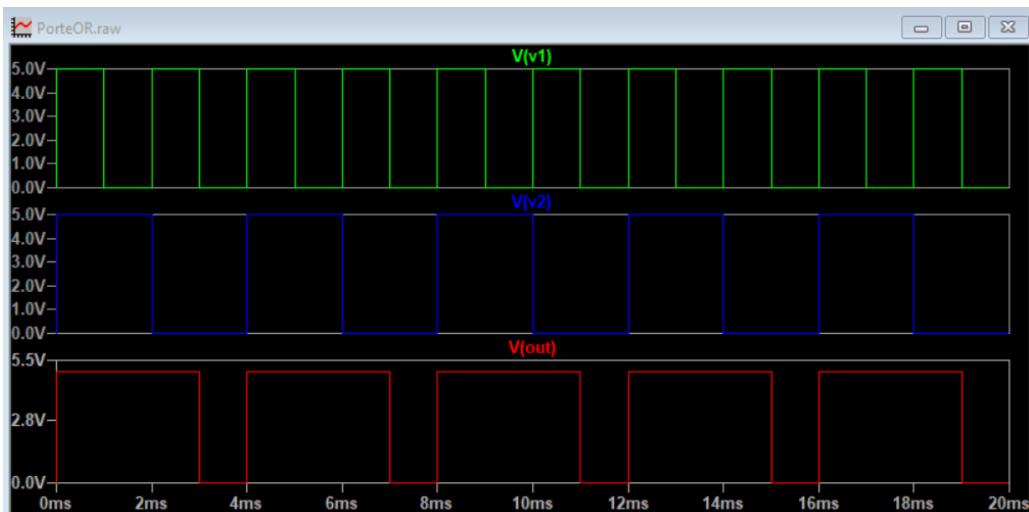
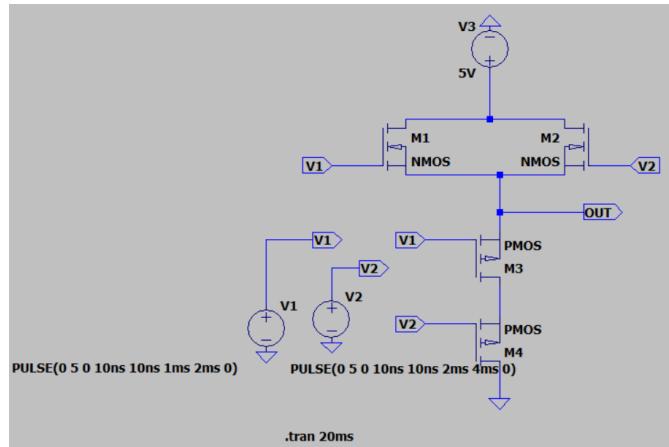


Non, il y a un délai entre l'entrée et la sortie du circuit.



Porte OR

Réaliser un commentaire sur le fonctionnement de ce circuit, que se passe t'il lorsque $V_2 = 0V$? Lorsque $V_2 = 5V$? Le circuit a-t-il une **latence strictement parfaite**? C'est-à-dire aucun délai entre l'entrée du circuit et la sortie.



On peut observer une porte OU. Le schéma fonctionne de la manière suivante.

- Pour $V_1=0$ et $V_2=0$

Les PMOS ont une tension de grille $V_{gs}=0V$ ($V_{gs}>-0,5$) sont passants. Pour nos NMOS, on a $V_{gs}=0V$ donc $V_{gs}<1,1$ donc sont aussi bloqués. La sortie V_{out} est à son état bas et vaut $0V$.

- Pour $V_1=1$ et $V_2=0$.

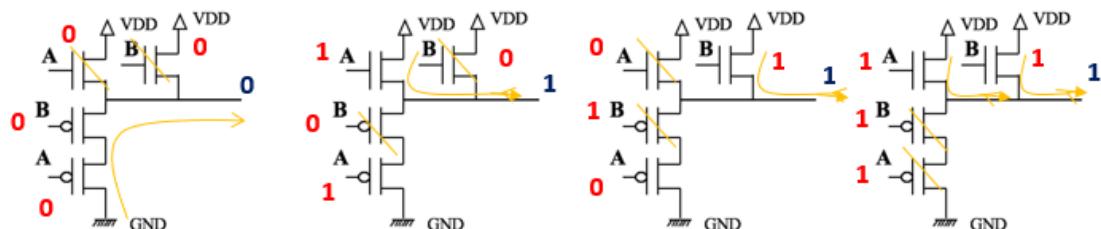
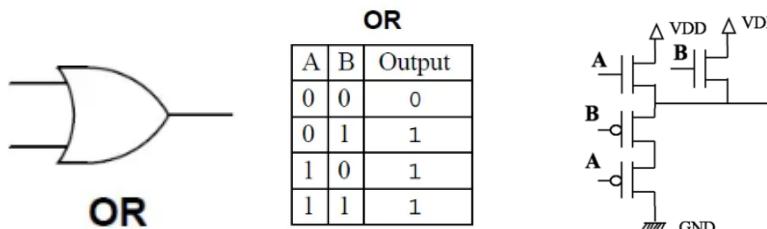
Pour le premier PMOS, on a $V_{gs}= 5V$ donc $V_{gs}>-0,5$ donc le PMOS est bloqué. Pour le deuxième PMOS, est passant. Pour le premier NMOS, on a $V_{gs}=5V$ donc $V_{gs}>1,1$ donc le NMOS est passant. Pour le deuxième NMOS, on a $V_{gs}=0V$ donc $V_{gs}<1,1$ donc le NMOS est bloqué d'où $V_{out}=5V$; La sortie V_{out} est à son état haut et vaut 1 .

- Pour $V_1=0$ et $V_2=1$

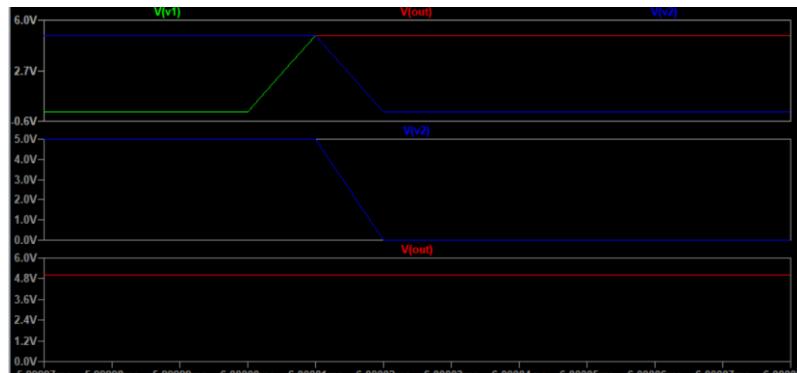
Pour nos PMOS. Pour le premier PMOS, O sur la grille, donc le PMOS est passant. Pour le deuxième PMOS est bloqué. Pour le premier NMOS, on a $V_{gs}=0V$ donc $V_{gs}<1,1$ donc le NMOS est bloqué. Pour le deuxième NMOS, on a $V_{gs}=5V$ donc $V_{gs}>1,1$ donc le NMOS est passant d'où $V_{out}=5V$; La sortie V_{out} est à son état haut et vaut 1 .

- Pour $V1=1$ et $V2=1$

pour nos PMOS, on a $Vgs = 5V$ donc $Vgs > -0,5$ donc nos deux PMOS sont bloqués. Pour nos NMOS, on a $Vgs = 5 - 0 = 0V$ donc $Vgs > 1,1$ donc nos NMOS sont passants donc $Vout = 5V$, La sortie $Vout$ est à son état haut et vaut 1.

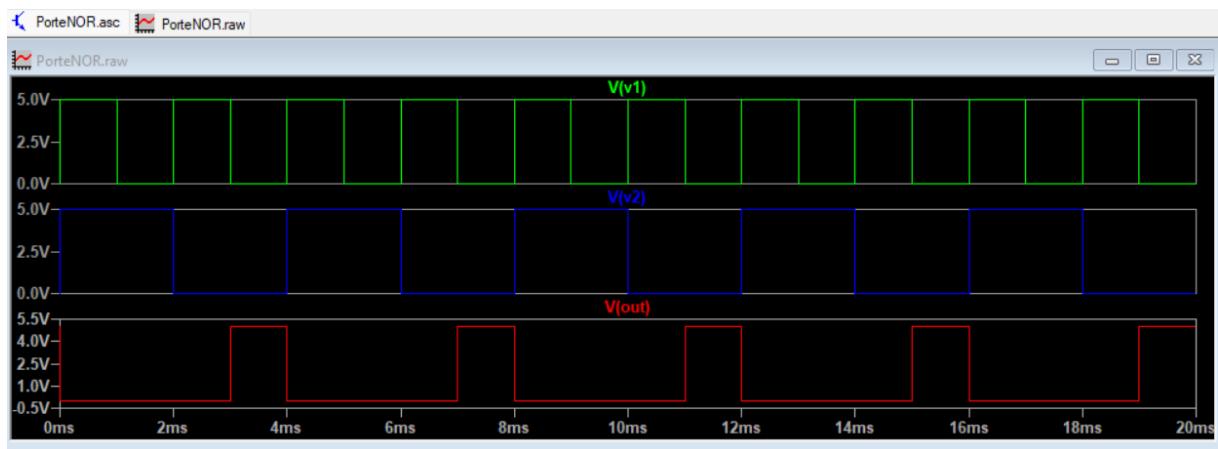
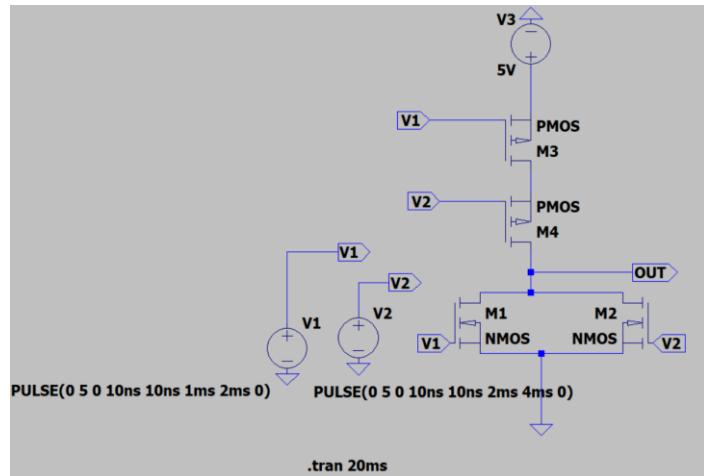


Non, il y a un délai entre l'entrée et la sortie du circuit.



Porte NOR

Réaliser un commentaire sur le fonctionnement de ce circuit, que se passe t'il lorsque $V2 = 0V$? Lorsque $V2 = 5V$? Le circuit a-t-il une **latence strictement parfaite**? C'est-à-dire aucun délai entre l'entrée du circuit et la sortie.



Pour une porte NOR

- $V1=0, V2=0$

Lorsque les entrées sont à 0 ou basse ($V2=0V$, $V_{GS} < V_{TH}$ et $V_s=5V$), les PMOS sont passants donc conduits. Les grilles des deux transistors NMOS étant à 0, ils sont bloqués. La sortie est directement reliée à l'alimentation à travers les deux PMOS, et charge la capacité, donc passe alors à l'état haut.

- $V1=0, V2=1$

Lorsqu'une entrée est à 1 et la deuxième à 0, le NMOS dont la grille est 0 est ouvert, le deuxième est passant et le PMOS dont la grille est à 1 est ouvert, le deuxième dont la grille est à 0 est passant. Ceci permet le passage du courant de se décharger d'une capacité vers la masse. La tension de sortie passe alors à l'état bas.

- $V1=1, V2=1$

Dans ce cas les deux transistors PMOS sont ouverts et les deux transistors NMOS sont passants. Le courant se décharge de la capacité de sortie à travers les deux transistors NMOS, et la sortie passe à l'état bas.

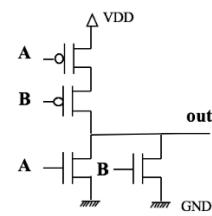
Symbol IEEE :



Table de vérité

A	B	out
0	0	1
0	1	0
1	0	0
1	1	0

Implémentation



LA PORTE NOR

Symbol IEEE :

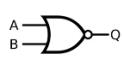
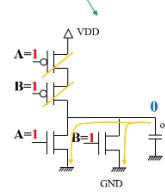
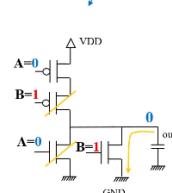
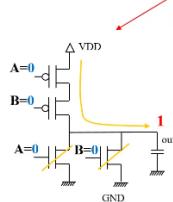


Table de vérité

A	B	out
0	0	1
0	1	0
1	0	0
1	1	0

Implémentation



En conclusion :

Lorsqu'on a 0 sur la grille, les PMOS sont passants (fermés) donc conduits.

Lorsqu'on a 0 sur la grille les NMOS sont ouverts donc bloqués.

Lorsqu'on a 1 sur la grille les PMOS sont ouverts donc bloqués.

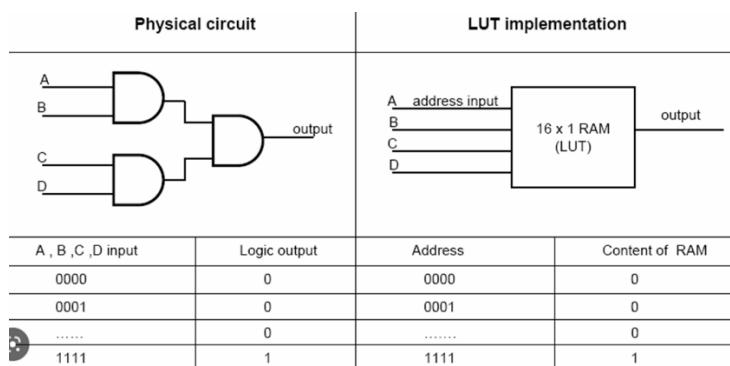
Lorsqu'on a 1 sur la grille, les NMOS sont passants (fermés) donc conduits.

N. TD :

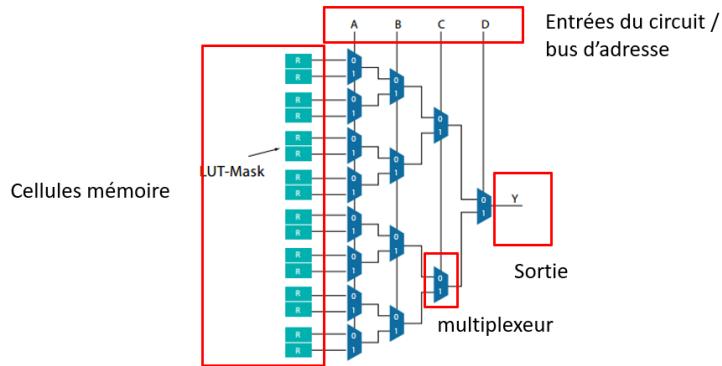
Si les portes logiques sont assemblées avec des transistors **gravés dans le silicium**, pourquoi dans un FPGA toutes les portes logiques sont **reconfigurables** ?

Ce sont les LUTs qui permettent cette reconfiguration du composant, physiquement parlant, une **LUT est une nano unité de mémoire** qui contient la **table de vérité d'un circuit**

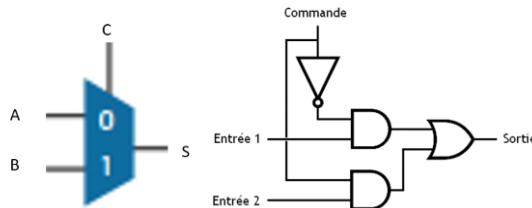
La LUT étant une mémoire, possède un bus d'adresse sur lequel on connecte nos entrées



Ci-dessous un schéma représentant cette nano unité de mémoire



Comment réaliser un multiplexeur à deux entrées ? A, B et C sont nos entrées du circuit, S est notre sortie.



Comment réaliser un multiplexeur à deux entrées?

A	B	C	S
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

Comment réaliser un multiplexeur à deux entrées?

A	B	nA	nB	AnB	A.B	nA.B	AnB+A.B	nA.B+A.B
0	0	1	1	0	0	0	0	0
0	1	1	0	0	0	0	1	0
1	0	0	1	1	0	0	1	0
1	1	0	0	0	0	1	0	1

On remarque que

AnB+A.B	A
0	0
0	=
1	1
1	1

nA.B+A.B	B
0	0
1	=
0	0
1	1

Donc, $S = A \cdot nB \cdot nC + A \cdot B \cdot nC + nA \cdot B \cdot C + A \cdot B \cdot C$

Donc, $S = nC \cdot (A \cdot nB + A \cdot B) + C \cdot (nA \cdot B + A \cdot B)$

Donc, $S = nC \cdot (A + A \cdot B + nB) + C \cdot (B + B \cdot nA + A)$

Comment réaliser un multiplexeur à deux entrées? Méthode avec la K-map

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

Autrement,
on peut rassembler les cases adjacentes, par exemple comme ceci :

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

Nous aurions alors $S = AB + C \cdot nA \cdot B + nC \cdot A \cdot nB$

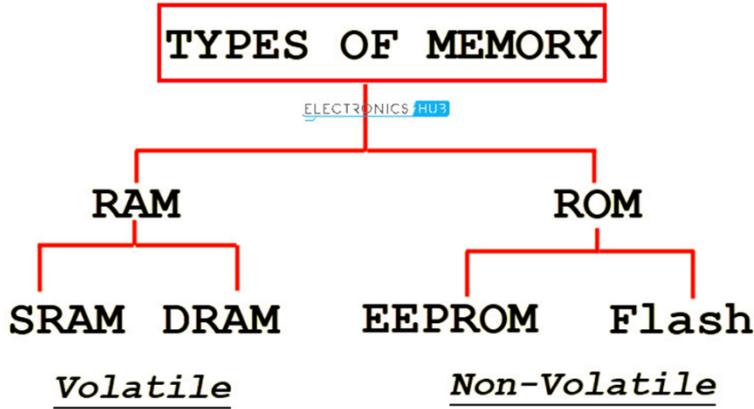
$S = cb + nca + ab$. On arrive à la porte logique

Différence entre fpga flash et room. Définir l'accès au composant. Fpga non volatile et fpga volatile ; utilisation de fusible.

Quel type de mémoire est utilisée pour constituer une LUT ? Doit-on toujours utiliser de la RAM ?

Pas toujours, la plupart du temps on utilise de la Static Random Acces Memorie (SRAM). Mais sur certaines applications on peut employer de la flash notamment pour que le temps de programmation du FPGA soit le plus court possible et donc qu'il soit **opérationnel le plus tôt possible**.

Les types de mémoires modernes



Pour enregistrer le bitstream et donc pour programmer les LUTs, tout les types de mémoire peuvent être utilisé ; avec leur avantages et inconvénients de chacun.

O. TD : Construire un full-substractor 4bits.

- Construire la table de vérité

Un soustracteur complet est un circuit combinatoire qui effectue la soustraction de deux bits, l'un est minuend et l'autre est soustrait, en tenant compte de l'emprunt du bit minuend inférieur adjacent précédent. Ce circuit a trois entrées et deux sorties. Les trois entrées A, B et Bin désignent respectivement la diminution de la fin, la soustraction et l'emprunt précédent. Les deux sorties, D et Bout représentent respectivement la différence et l'emprunt de sortie. Bien que la soustraction soit généralement obtenue en ajoutant le complément de soustraire à la minuend, il est d'un intérêt académique d'élaborer la table de vérité et la réalisation logique d'un soustracteur complet ; x est le diminutif ; y est le sous-traitant ; z est l'emprunt d'entrée ; D est la différence ; et B désigne l'emprunt de sortie. Les cartes correspondantes pour les fonctions logiques pour les sorties du soustracteur complet, à savoir la différence et l'emprunt.

Pour calculer la différence A - B de deux nombre signés A et B, on utilise un circuit qui calcule d'abord l'opposé -B de B puis effectue la somme de A avec -B grâce à un additionneur. Le calcul de -B est réalisé en prenant la négation de B bit à bit puis en ajoutant 1 au résultat obtenu. Ce dernier 1 est en fait ajouté directement à la somme de A et -B en l'injectant comme retenue C0 à l'additionneur.

TD : Construire un full-substractor 4bits

On représente un chiffre négatif par le code suivant :

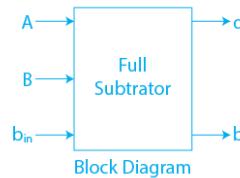
$$\text{Pour 8 bits, } 0xFF = 1111\ 1111 = -128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 \\ = -128 + 127 = 1$$

TD : Construire un full-substractor 4bits

On représente un chiffre négatif par le code suivant :

$$\text{Pour 2 bits, } 0x3 = 11 = -2 + 1 = -1$$

Le circuit ci-dessous effectue une somme ou une différence suivant la valeur de la commande Cmd. Si Cmd vaut 0, le circuit calcule la somme A + B. Si, au contraire, Cmd vaut 1, le circuit calcule la différence A - B. En effet, chacune des portes XOR effectue la négation ou non d'une entrée Bin suivant la valeur de Cmd.



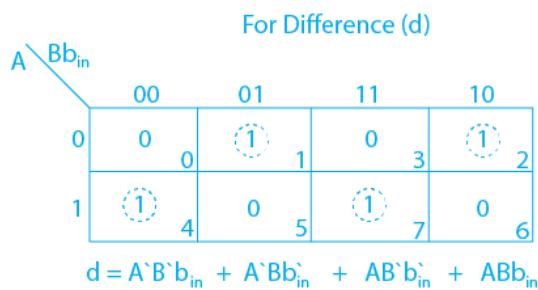
Pour rappel les entrées sont A, B, Bin et les sorties sont D, la différence et Bout, l'emprunt.

A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

- Réaliser le schéma électronique

La table de Karnaugh pour le soustracteur. Elle permet d'arriver à une simplification directe de l'expression.

Pour la différence (D).



$$D = (\bar{A} \cdot \bar{B} \cdot B_{in}) + (\bar{A} \cdot B \cdot \bar{B}_{in}) + (A \cdot \bar{B} \cdot \bar{B}_{in}) + (A \cdot B \cdot B_{in})$$

Pour l'emprunt (Bout)

For Borrow (b)

	00	01	11	10
0	0 0	1 1	1 3	1 2
1	0 4	0 5	1 7	0 6

$b = A' \cdot b_{in} + A \cdot B + B \cdot b_{in}$

$$Bout = (\bar{A} \cdot B_{in}) + (\bar{A} \cdot B) + (B \cdot B_{in})$$

Construction du full subtractor avec des portes logiques.

TD : Construire un full-subtractor 4bits

$$\begin{aligned} D &= nA \cdot nB \cdot Bin + nA \cdot B \cdot nBin + A \cdot nB \cdot nBin + A \cdot B \cdot Bin \\ &= Bin(nA \cdot nB + A \cdot B) + nBin(A \cdot nB + nA \cdot B) \end{aligned}$$

A	B	nA	nB	nA.nB	A.B	A.nB	nA.B + A.B	A.nB + nA.B
0	0	1	1	1	0	0	1	0
0	1	1	0	0	0	1	0	1
1	0	0	1	0	0	1	0	1
1	1	0	0	0	1	0	1	0

$$\begin{aligned} D &= Bin(A \cdot XNOR B) + nBin(A \cdot XOR B) \\ D &= (A \cdot XOR B) \cdot XOR BIN \end{aligned}$$

TD : Construire un full-subtractor 4bits

$$\begin{aligned} Bout &= nA \cdot nB \cdot Bin + nA \cdot B \cdot nBin + nA \cdot B \cdot Bin + A \cdot B \cdot Bin \\ &= Bin(A \cdot B + nA \cdot nB) + nA \cdot B \cdot (Bin + nBin) \end{aligned}$$

A	B	nA	nB	nA.nB	A.B	nA.nB + A.B
0	0	1	1	1	0	1
0	1	1	0	0	0	0
1	0	0	1	0	1	0
1	1	0	0	0	0	1

$$\begin{aligned} Bout &= Bin(A \cdot XNOR B) + nA \cdot B \\ Bout &= Bin.(Not(A \cdot XOR B)) + nA \cdot B \end{aligned}$$

Pour la différence

$$D = (\bar{A} \cdot \bar{B} \cdot B_{in}) + (\bar{A} \cdot B \cdot \overline{B_{in}}) + (A \cdot \bar{B} \cdot \overline{B_{in}}) + (A \cdot B \cdot B_{in})$$

$$D = \bar{A}((\bar{B} \cdot B_{in}) + (B \cdot \overline{B_{in}})) + A((\bar{B} \cdot \overline{B_{in}}) + (B \cdot B_{in}))$$

$$D = A(not(B \oplus B_{in}) + \bar{A}(B \oplus B_{in}))$$

$$D = A \oplus (B \oplus B_{in})$$

$$D = A \oplus B \oplus B_{in}$$

Pour l'emprunt

$$Bout = (\bar{A} \cdot \bar{B} \cdot B_{in}) + (\bar{A} \cdot B \cdot \overline{B_{in}}) + (\bar{A} \cdot B \cdot \overline{B_{in}}) + (A \cdot B \cdot B_{in})$$

$$Bout = B_{in}(A \cdot B + \bar{A} \cdot \bar{B}) + \bar{A} \cdot B (B_{in} + \overline{B_{in}})$$

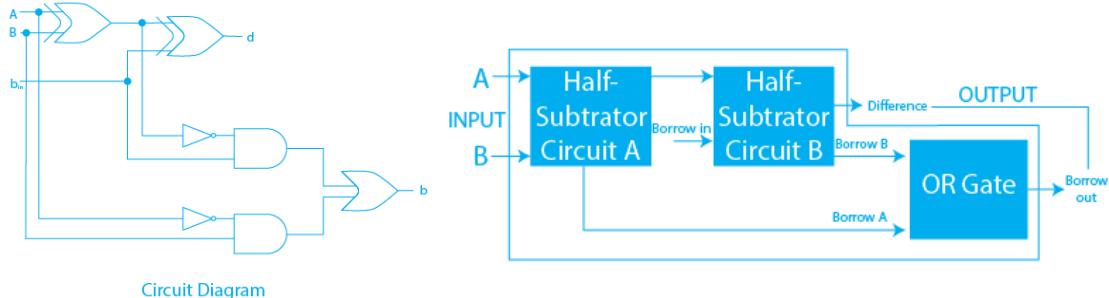
$$Bout = B_{in}(\overline{A \oplus B}) + \bar{A} \cdot B$$

Ou

$$Bout = (\bar{A} \cdot \bar{B} \cdot B_{in}) + (\bar{A} \cdot B \cdot \overline{B_{in}}) + (\bar{A} \cdot B \cdot \overline{B_{in}}) + (A \cdot B \cdot B_{in})$$

$$Bout = (\bar{A} \cdot B_{in})(B + \bar{B}) + (\bar{A} \cdot B)(B_{in} + \overline{B_{in}}) + (A + \bar{A})(B \cdot B_{in})$$

$$Bout = (\bar{A} \cdot B_{in}) + (\bar{A} \cdot B) + (B \cdot B_{in})$$



Circuit Diagram

Implémentation d'un soustracteur complet à l'aide de demi-soustracteurs – 2 demi-soustracteurs et une porte OU sont nécessaires pour implémenter un soustracteur complet.

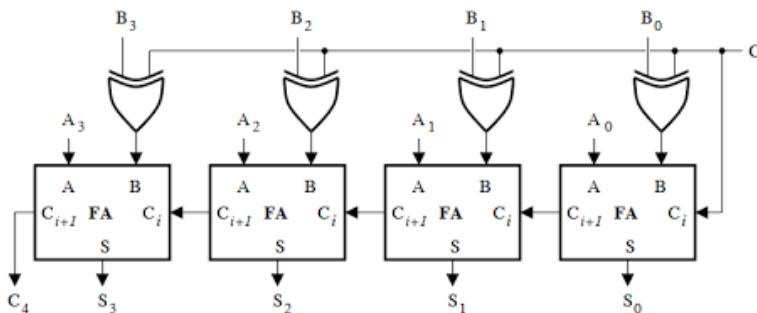
- Réaliser l'analyse de chemin critique

L'analyse du chemin critique donne :

Pour D, on trouve 10ps soit 5ps sur la 1^{ère} et 5ps sur la 2^e porte.

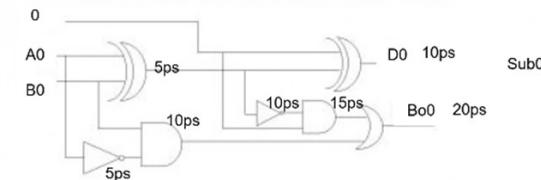
Pour le Bout, on trouve 20ps qui représente le chemin critique.

La différence est ici calculée avec un simple additionneur par propagation de retenue mais tout autre additionneur plus sophistiqué aurait pu aussi être utilisé.

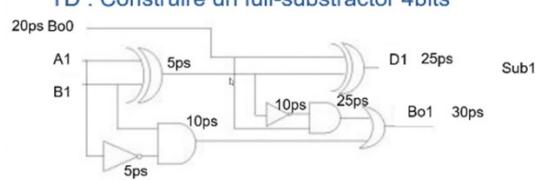


Soustracteur 4 bits

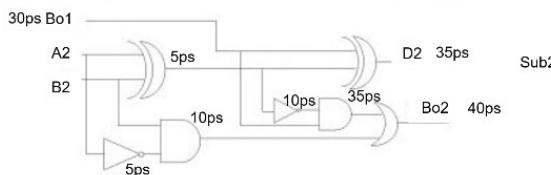
TD : Construire un full-substractor 4bits



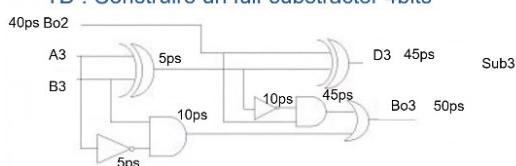
TD : Construire un full-substractor 4bits



TD : Construire un full-substractor 4bits



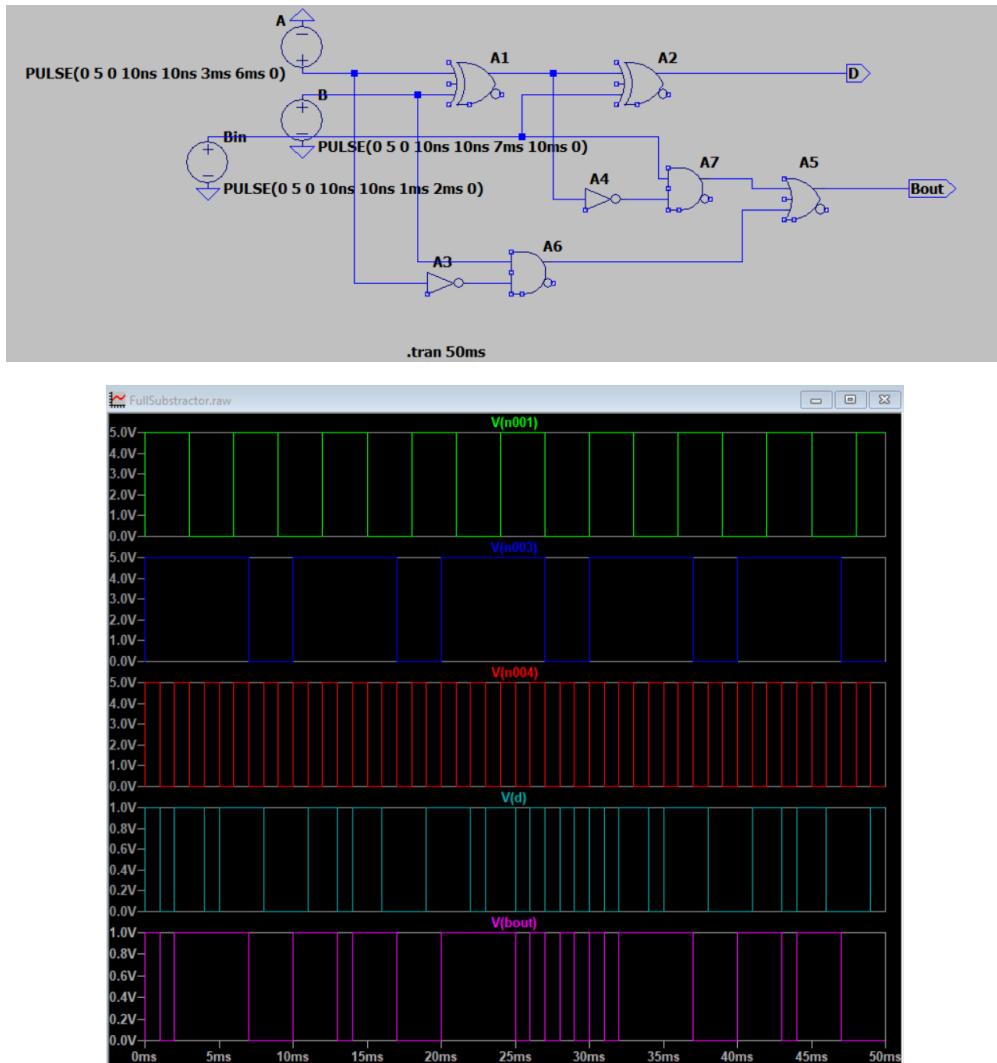
TD : Construire un full-substractor 4bits



Pour 4 soustracteur :

Pour D, on trouve 45ps et pour le Bout, on trouve 50ps qui représente le chemin critique.

- Simuler le circuit sous LT-spice, prouver son fonctionnement



V. Logique numérique synchrone

h. La bascule (le latch)

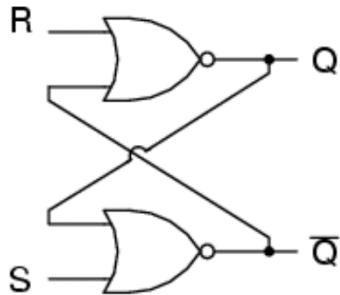
Un latch est un élément de circuit numérique qui a deux états stables et peut être utilisé pour stocker des informations.

Les deux **états stables** d'un latch sont communément appelés états set et reset.

-Lorsqu'on entre dans l'état latch ($S = 0$ et $R = 0$) alors la sortie Q conserver sa valeur d'état précédent (**état de mémoire**).

Lorsque $S = 1$ et $R = 0$ puis de transiter S à l'état bas ($S=0$). On observerait alors la sortie Q conserver son état haut ($Q = 1$).

$S = 0$ et $R = 1$, nous aurions observer que Q serait à l'état bas ($Q = 0$).



S	R	Q	\bar{Q}
0	0	latch	latch
0	1	0	1
1	0	1	0
1	1	0	0

/!\ On considère que le cas $R = 1$ et $S = 1$ est illégal (**combinaison indésirable**)

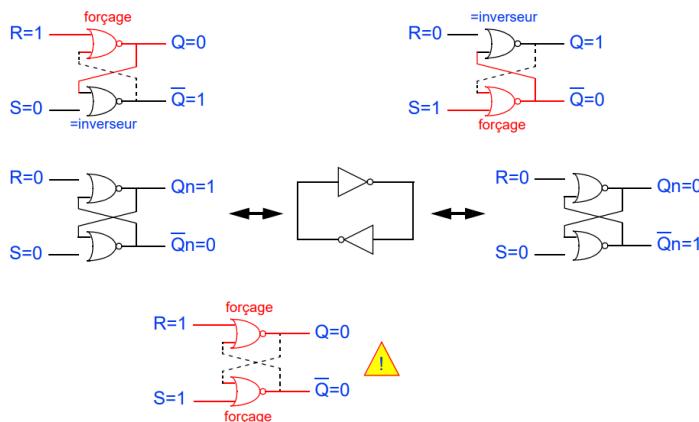


Table de vérité :

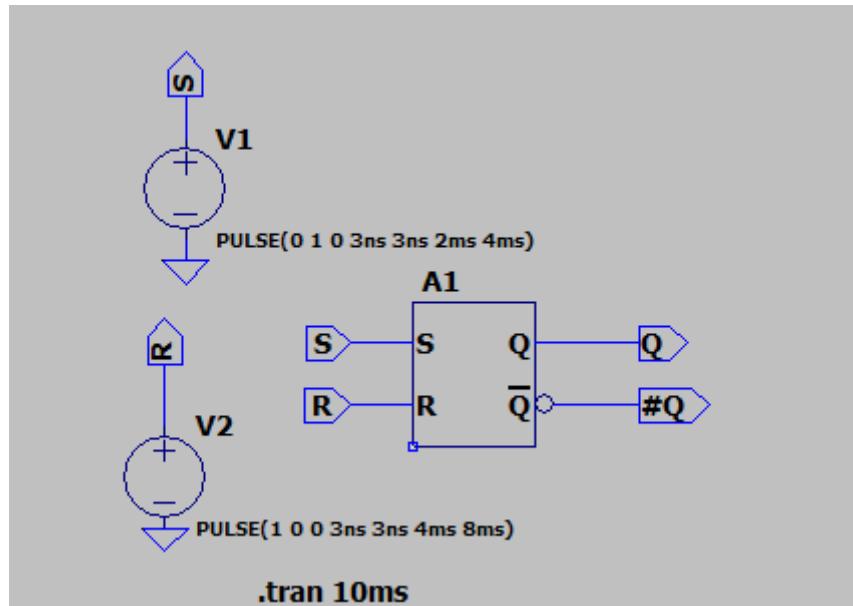
S	R	Q	\bar{Q}	remarque
0	0	q	\bar{q}	mémorisation
0	1	0	1	mise à 0
1	0	1	0	mise à 1
1	1	0	0	cas particulier

La bascule est un élément de mémorisation. On peut créer un circuit qui entre dans un état de conservation de son état. Etat de lock, état de latch, si $q=1$, il reste à 1.

Pour deux situations données, Q peut donner deux sorties différentes, ce qui donne la mémorisation de la bascule.

P. TD :Bascule

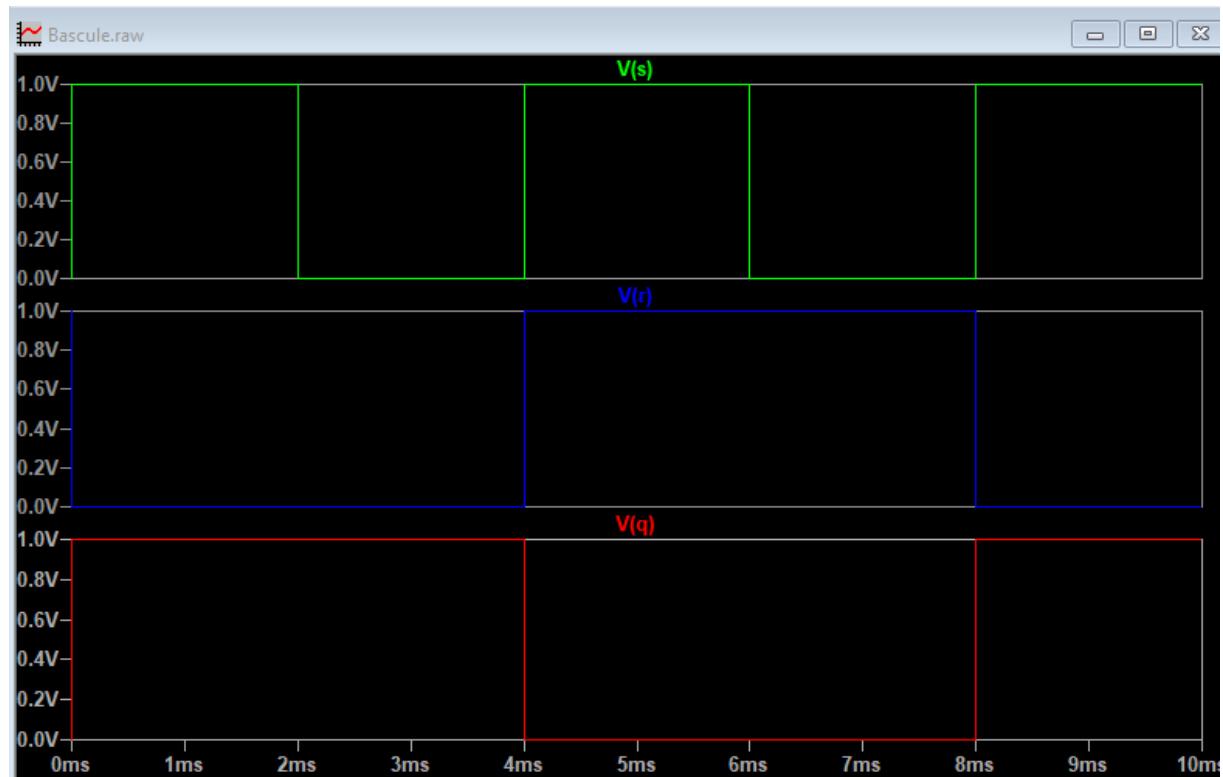
Sous Ltspice, utiliser le symbole de « srflop » de la librairie digitale



Réaliser les deux cas suivants :

- $S = 0$ et $R = 1$ puis $S = 0$ et $R = 0$
- $S = 1$ et $R = 0$ puis $S = 0$ et $R = 0$

Observer la sortie Q dans chaque cas. Pas besoin d'observer la latence du circuit.



On observe que lorsque $S = 1$ et $R = 0$, $Q=1$. $S = 0$ et $R = 0$, Q conserve sa valeur de l'état précédent qui est 1.

Lorsque $S = 0$ et $R = 1$, $Q=0$ et $S = 1$ et $R = 1$, Q reste à 0.

On vérifie bien la table de vérité de la bascule latch.

Le latch se symbolise de la façon suivante et permet de réaliser la fonction de mémorisation (de 1bit).

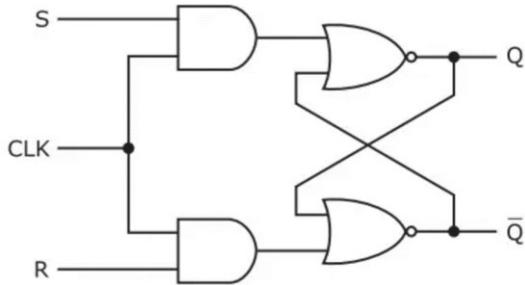
Néanmoins une fonction très recherchée est celle de la **synchronisation** c'est-à-dire de **déclencher la mémorisation** lorsqu'on le souhaite.

On va alors employer un composant bien connu qui rassemble ces deux fonctions : **le registre**

i. Le registre (flip flop)

Le circuit est assez similaire à celui de la bascule. On ajoute un étage de portes AND et un nouveau signal **CLK** qui correspond à notre **horloge**.

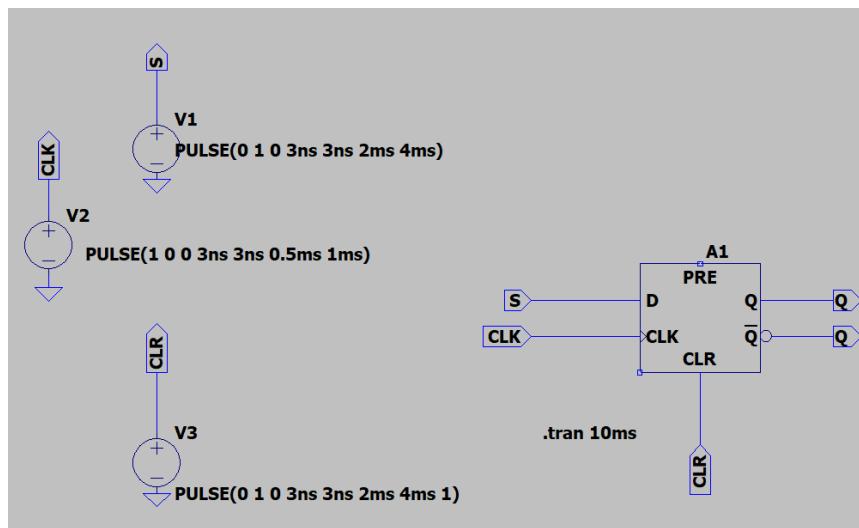
- CLK inactif : la bascule est isolée (il faut s'assurer qu'elle se trouve en configuration mémoire).
- CLK actif : la bascule fonctionne normalement.



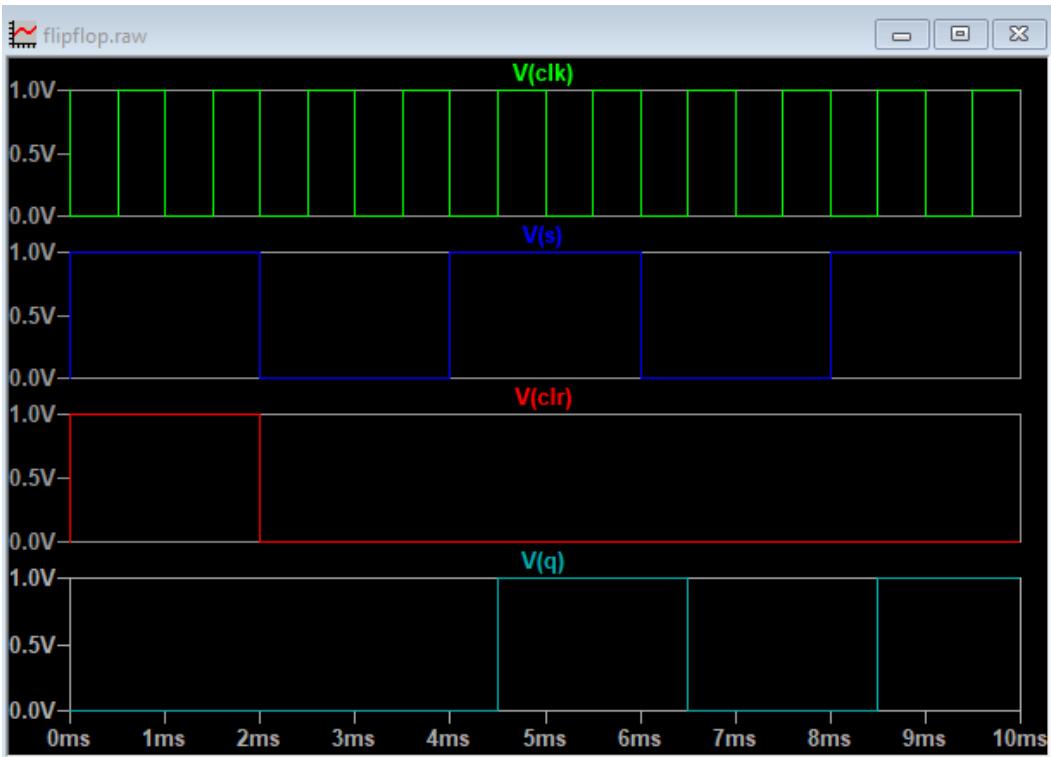
RESET	D	CK	Q	\bar{Q}
1	-	-	1	0
0	-	-	0	1
0	-	-	1	1
1	1	1	1	0
1	0	1	0	1

Q. TD :Bascule

Sous Ltspice, utiliser le symbole de « dflop » de la librairie digitale



Réaliser les deux cas suivants :



Observer la sortie Q dans chaque cas. Pas besoin d'observer la latence du circuit.

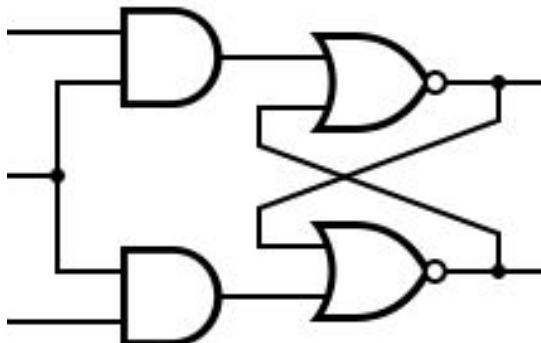


Table de vérité :

E	S	R	Q	\bar{Q}	remarque
0	x	x	q	\bar{q}	mémorisation
1	0	0	q	\bar{q}	mémorisation
1	0	1	0	1	mise à 0
1	1	0	1	0	mise à 1
1	1	1	0	0	cas particulier

CLK = 0, R et S indifférents car A1 et A2 imposent un état R = S = 0 et par conséquent : Qn = Qn-1 (état mémoire).

CLK = 1, R=0 et S =0 car A1 et A2 imposent un état R = S et par conséquent : Qn = Qn-1 (état mémoire)

CLK = 1, R=0 et S =1, Q=1

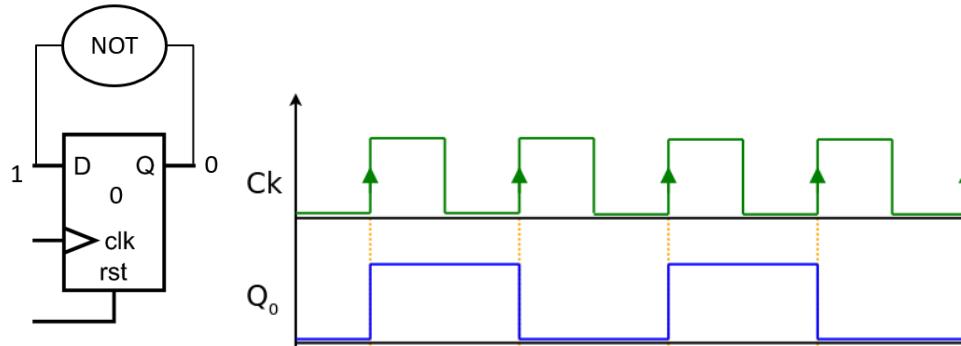
CLK = 1, R=1 et S =0, Q=0

En conclusion, CLK = 1: fonctionnement classique de la bascule car A1 et A2 se comportent comme s'ils sont transparents. La bascule est transparente : la sortie réagit "immédiatement" à l'entrée lorsque CLK=1.

j. Le registre (flip flop) : la D-flipflop

Dans le cas présenté, le registre va alterner entre la valeur 0 et 1 (porte not dans la contre-réaction), **sur front d'horloge**. On peut décider que la mise à jour soit faite sur front montant, descendant ou les deux.

La plupart du temps, on travaille sur front montant par convention. Ici nous avons un exemple de compteur sur 1bit.



T	D	J	K	Q _{n+1}	fonction
0	X	X	X	Q _n	Mémo
1	1	1	0	1	Set
1	0	0	1	0	Reset

→

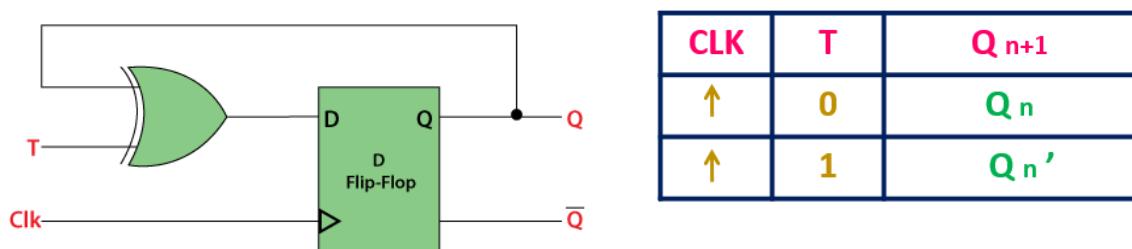
T	D	Q _{n+1}
0	X	Maintenir Q _n
1	D	Écrire D

Quand le signal est actif, la sortie recopie l'entrée D : Fonction Ecriture.

Signal inactif : Fonction Mémoire

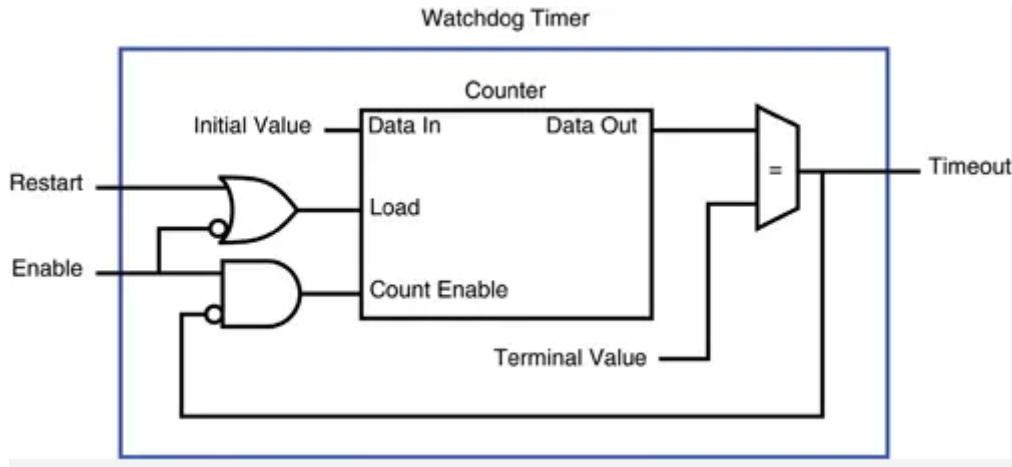
k. Le registre (flip flop) : La T-Fliflop

Sur front montant d'horloge, si l'entrée du registre est à 1, alors la valeur courante du registre est inversée. Sinon, la valeur du registre est inchangée.

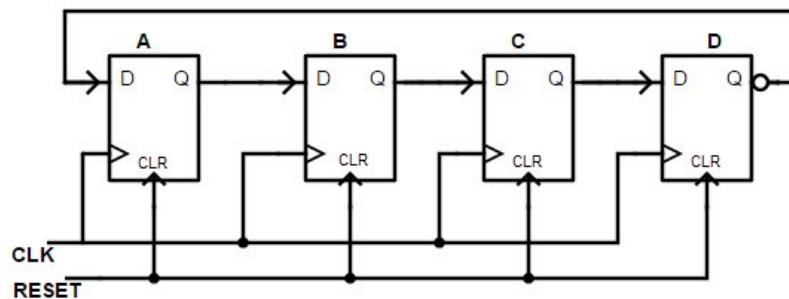


1. Les compteurs

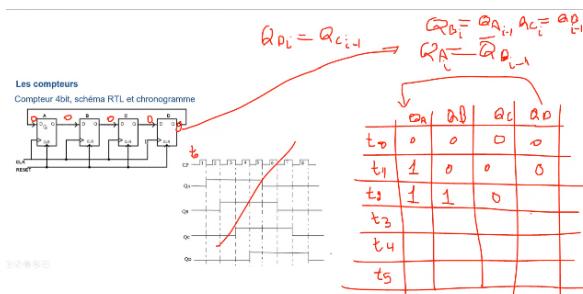
On croise de façon très récurrente des compteurs dans les designs numériques. Ils permettent de réaliser par exemple des watchdogs ; très connu en informatique, il permet d'interrompre un processus lorsque ce dernier ne répond pas au bout d'un certain temps.



Compteur 4bit, schéma RTL et chronogramme

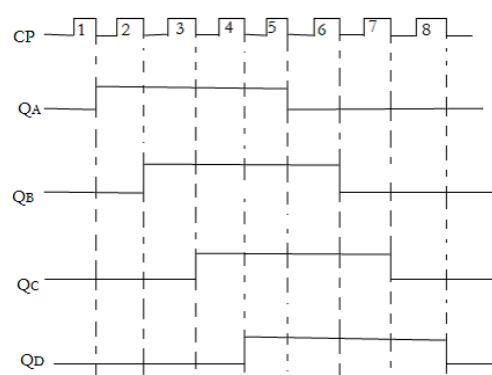


Réaliser le chronogramme de ce circuit, on suppose que les registres sont initialisés à zéro



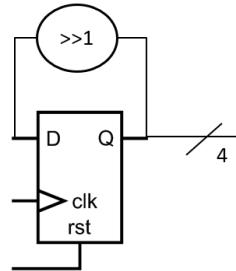
$$Q_{Ai} = \overline{Q_{Di-1}}, \quad Q_{Bi} = Q_{Ai}, \quad Q_{Ci} = Q_{Bi-1}; \quad Q_{Di-1} = Q_{Ci-1}.$$

Q_A	Q_B	Q_C	Q_D
0	0	0	0
1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1
0	1	1	1
0	0	1	1
0	0	0	1
repeat			

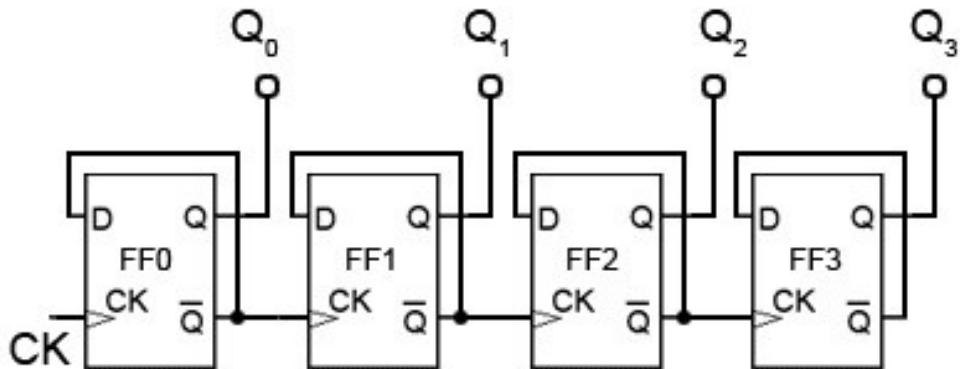


Le plus souvent, on rassemble logique combinatoire et logique synchrone pour réaliser des compteurs. Une autre façon de représenter le schéma RTL d'un compteur Johnson est donné ici.

On part du principe que les registre sont à 0001 et le rst est à 0(non actif)



Compteur 4bit, base de D-flip-flops schéma RTL, « Ripple counter »



R. TD :

Réaliser le chronogramme, (les transitions se font lorsque le port « CK » fait une transition montante) ;

Les 4 compteurs sont des registres d flipflop. Le registre va alterner entre la valeur 0 et 1 **sur front d'horloge**. On peut décider que la mise à jour soit faite sur front montant.

T	D	J	K	Q _{n+1}	fonction
0	X	X	X	Q _n	Mémo
1	1	1	0	1	Set
1	0	0	1	0	Reset

→

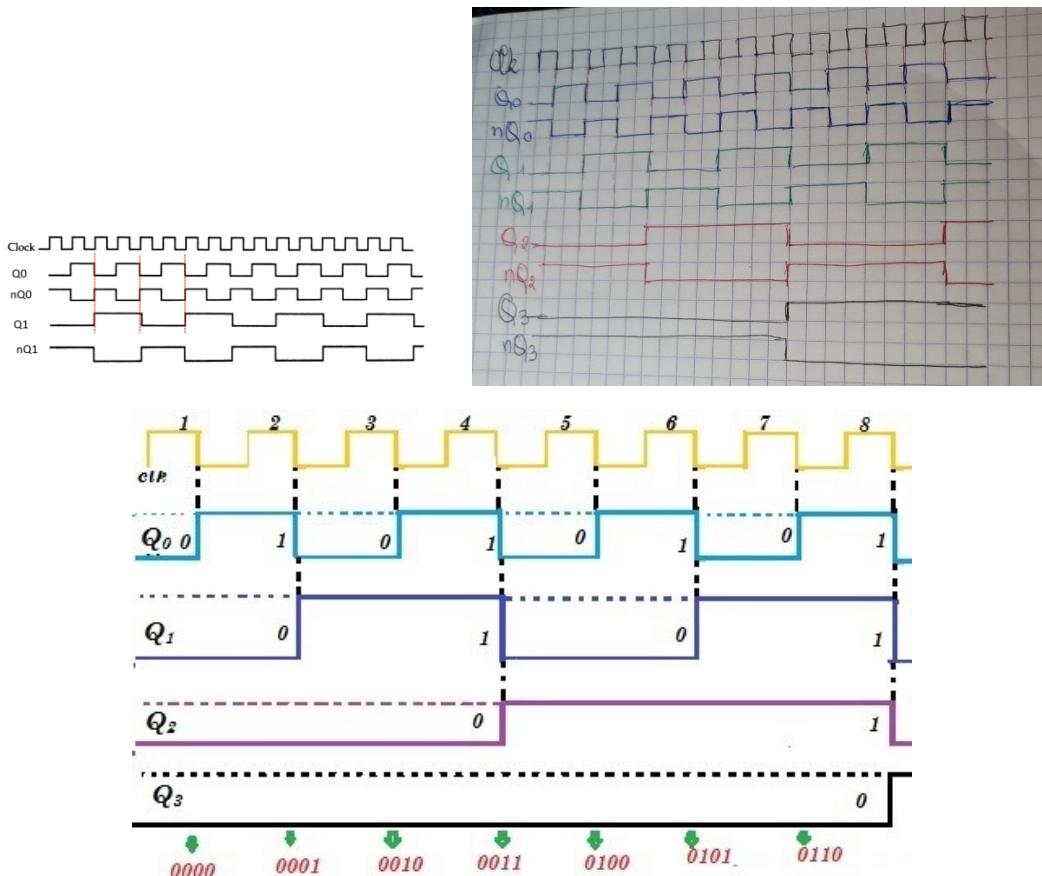
T	D	Q _{n+1}
0	X	Maintenir Q _n
1	D	Écrire D

Les sorties Q0 à Q3 sont respectivement les bits LSB et MSB. La table de vérité de la bascule JK nous aide à comprendre le fonctionnement du compteur.

L'état de la sortie Q0 change lorsque le front d'horloge positif passe à la bascule. Initialement, toutes les bascules sont mises à 0. Ces bascules changent d'état lorsque l'horloge passe de 0 à 1. La bascule JK bascule lorsque les entrées des bascules sont à un, puis la bascule change d'état de 0 à 1. Pour toutes les impulsions d'horloge, le processus reste le même.

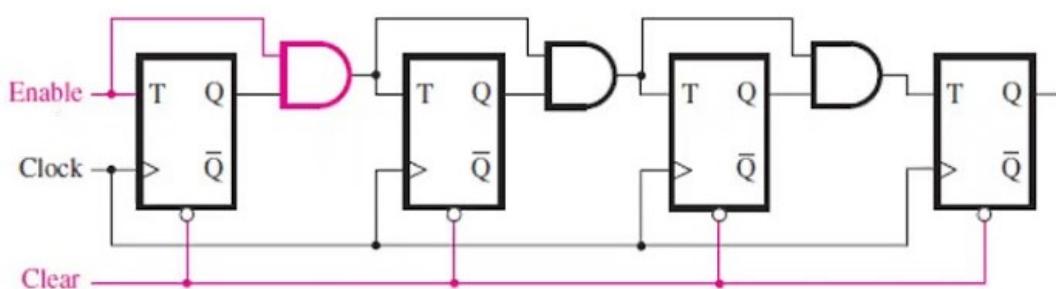
Pour le cycle suivant, $\text{clk} = \text{not } Q_0$. On répète le même raisonnement pour tracer Q_1 (L'état de la sortie Q_1 change lorsque le front d'horloge positif passe à la bascule).

Pour Q_2 , $\text{clk} = \text{not } Q_1$. Pour Q_2 , $\text{clk} = \text{not } Q_2$.



On remarque sur le schéma qu'on compte.

Compteur 4bit, schéma RTL

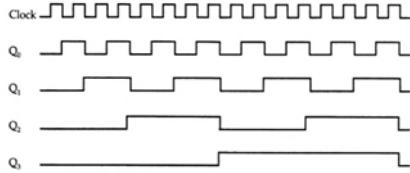
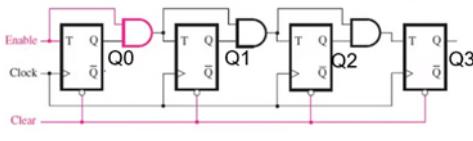


S. TD :

Réaliser le chronogramme de ce circuit pour $Q_0..Q_3$. On suppose que les registres sont initialisés à zéro. On suppose l'entrée « enable » à 1.

Les compteurs

Compteur 4bit, schéma RTL et chronogramme



T0 = 1
T1 = Q0 & 1 => T1 = Q0
T2 = Q1 & Q0 =
T3 = Q2 & Q1 & Q0

Q0 = E => Q0 = 0
Q1 = Q0 & AND E => Q1 = Q0...
Q2 = Q1 & AND Q0... AND E => Q2 = Q1 AND Q0
Q3 = Q2 & AND Q1 & AND Q0... AND E => Q3 = Q2 AND Q1 AND Q0

clk	Q0	Q1	Q2	Q3
↑	0	0	0	0
↑	1	0	0	0
↑	0	1	0	0
↑	1	1	0	0
↑	0	0	1	0
↑				
↑				
↑				
↑				
↑				

T0 = 1
T1 = Q0 & 1 => T1 = Q0
T2 = Q1 & T1
T3 = T2 & Q1 & Q0

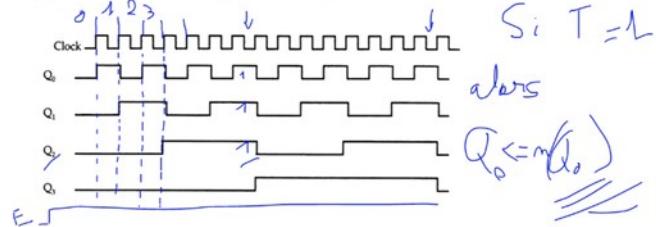
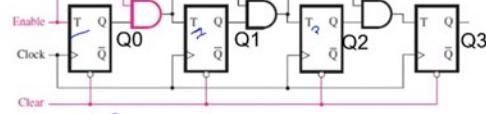
$$T_3 = Q_2 \text{ AND } T_2$$

$$= Q_2 \text{ AND } Q_1 \text{ AND } Q_0 \quad T_2 = Q_1 \text{ AND } T_1$$

$$= Q_1 \text{ AND } Q_0 \quad T_1 = Q_0$$

Les compteurs

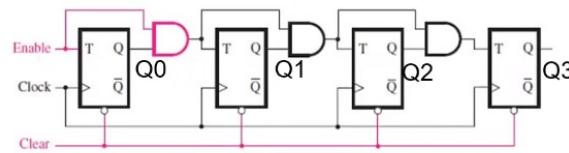
Compteur 4bit, schéma RTL et chronogramme



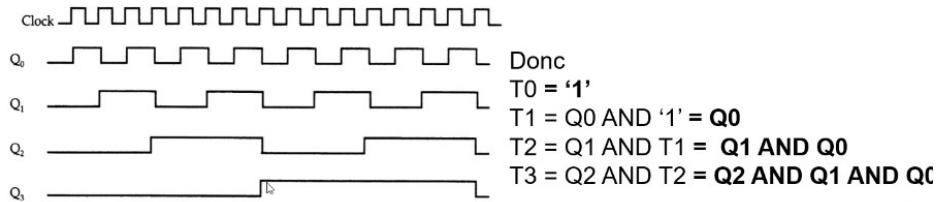
clk	Q0	Q1	Q2	Q3
↑	0	0	0	0
↑	1	0	0	0
↑	0	1	0	0
↑	1	1	0	0
↑	0	0	1	0
↑				
↑				
↑				
↑				

Q0 = E
Q1 = Q0 & E
Q2 = Q1 & Q0 & E
Q3 = Q2 & Q1 & Q0 & E

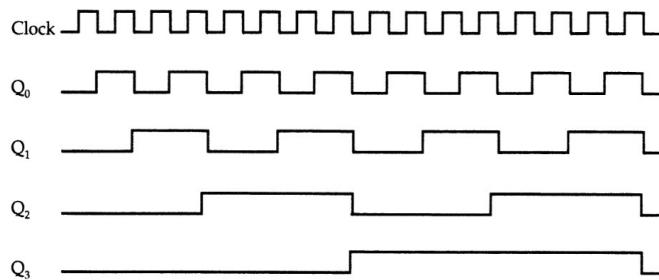
Q0 = E
Q1 = Q0 & E



T0 = Enable = '1'
T1 = Q0 AND T0
T2 = Q1 AND T1
T3 = Q2 AND T2

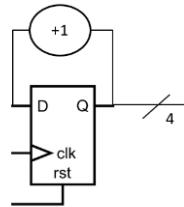


Rappel : Sur une T flipflop, si T est actif alors $Q \leq \text{not}(Q)$



Le plus souvent, on rassemble logique combinatoire et logique synchrone pour réaliser des compteurs. Une autre façon de représenter le schéma RTL d'un compteur naturel est donné ici; la fonction d'incrément n'est pas aussi détaillée

On part du principe que les registres sont à 0 et rst est activé durant un cycle d'horloge puis devient inactif

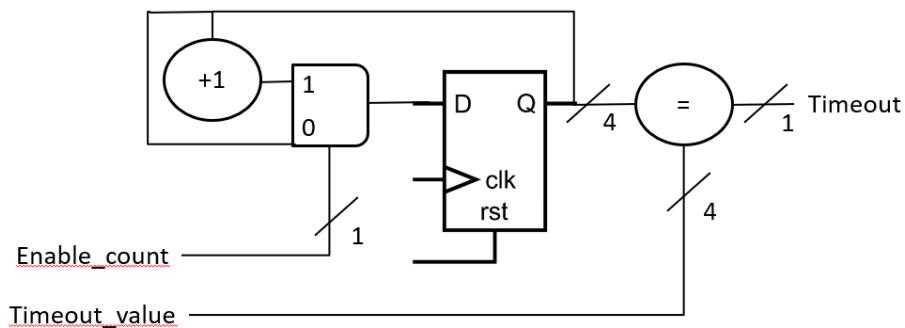


T. TD :

Ajouter les fonctionnalités suivantes au compteur naturel 4bits :

- Mise à l'état haut d'un signal de sortie « Timeout » lorsque la valeur du compteur est de 6 (b0110)
- Mise en pause du compteur lorsqu'une entrée « enable_count » est à l'état haut

Reprendre le schéma RTL associé (utiliser des multiplexeurs et un comparateur, les symboles sont donnés ci-dessous).

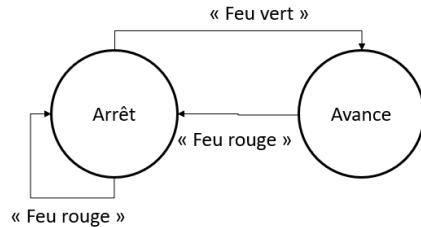


VI. Les machines à état

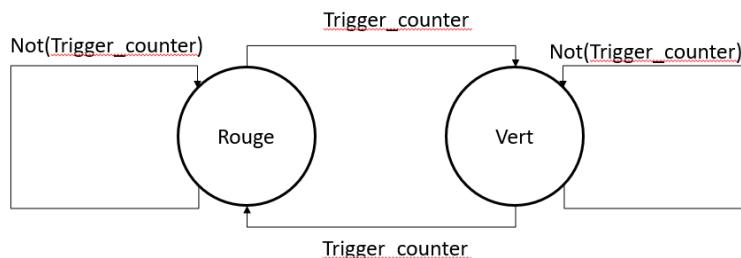
Une machine à état est généralement définie par un ensemble fini d'états possibles, ainsi que par les événements ou les conditions qui déclenchent la transition d'un état à un autre. Chaque état peut avoir un ensemble de sorties associées, qui représentent l'action ou le comportement du système à un moment donné.

Un exemple sur la route : « je reste à l'arrêt au feu tant que ce dernier est rouge, s'il passe au vert j'avance ». Nous avons deux états, « être à l'arrêt » et « avancer » ainsi qu'une condition de passage d'un état vers l'autre, « passage du feu au vert ».

On représente les machines à état par des diagrammes d'état comme dans l'exemple ci-dessous :

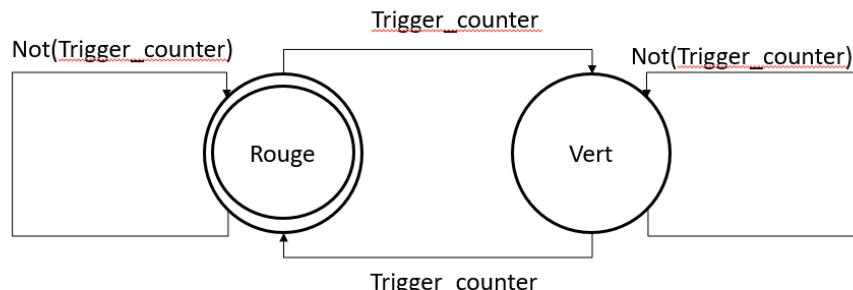


La machine à état du côté du feu rouge ressemblerait au diagramme ci-dessous :



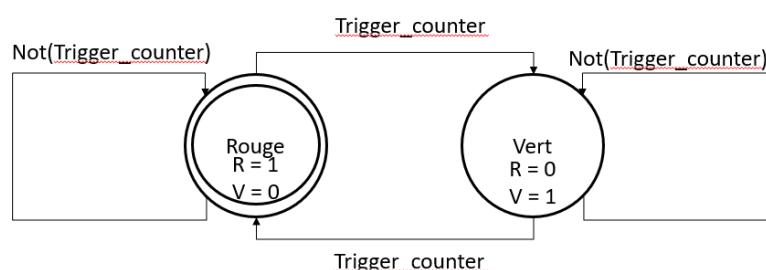
On indique un signal qui engendre les passages d'un état vers un autre, ici il est donné par un compteur de temporisation

On indique généralement l'état initial à l'aide d'un double cercle



Dans certains cas, on définit un état « idle » spécifiquement pour l'attente

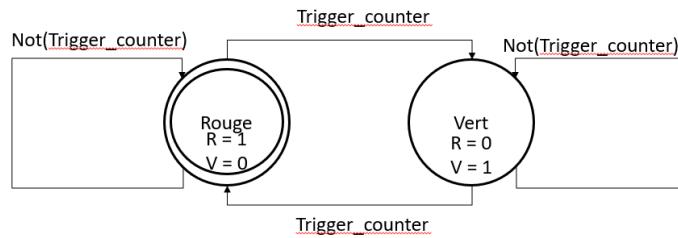
Finalement, on inscrit la valeur des signaux de sortie pour chaque état



Ici, R et V sont des signaux sur un 1 bit chacun qui viendront piloter des LEDs

Les machines à état

Lorsqu'un établit la FSM, les parties opératives ne sont pas représentées



Les parties opératives sont par exemple, le compteur binaire qui génère trigger_counter ou la partie pilotant l'intensité de lumière des deux LEDs rouge et verte

Lorsque la FSM est établie et que les parties opératives sont déterminées, on peut générer un synoptique de notre système .



Il existe deux types de machine à état, les machines de Mealy et les machines de moore.

FSM Moore : l'état suivant dépend uniquement de l'état courant

FSM Mealy : l'état suivant dépend de l'état courant et aussi de signaux d'entrées

U. TD

Consignes :

- Déterminer le nombre d'états nécessaires pour réaliser la machine à états.
- Analyser les transitions entre états en fonction des entrées et des sorties.
- Réaliser le diagramme de la machine à états sous forme de graphe ou de table d'états.
- Si vous avez le temps, réaliser le synoptique système en définissant les parties opératives

Cahier des charges :

- La machine à états doit être capable de contrôler l'ouverture et la fermeture d'une porte de garage.
- Les entrées sont constituées d'un bouton poussoir pour l'ouverture (B0), un bouton poussoir pour la fermeture (B1), et un détecteur de fin de course pour indiquer la position de la porte. On notera S0 et S1 les signaux représentant respectivement « porte ouverte » et « porte fermée ».
- Les sorties sont constituées d'un moteur pour actionner la porte, A0 et A1 pour respectivement ouvrir et fermer la porte

- Le fonctionnement de la machine à états doit respecter les règles suivantes :
 - Si la porte est fermée et qu'on appuie sur le bouton d'ouverture, la porte doit s'ouvrir.
 - Si la porte est ouverte et qu'on appuie sur le bouton de fermeture, la porte doit se fermer.
 - Si la porte est en train de s'ouvrir et qu'on appuie sur le bouton de fermeture, la porte doit s'arrêter.
 - Si la porte est en train de se fermer et qu'on appuie sur le bouton d'ouverture, la porte doit s'arrêter.

Correction :

Le nombre d'états nécessaires est de quatre : "fermée", "en train de s'ouvrir", "en train de se fermer" et "ouverte".

Les transitions entre les états sont les suivantes :

De l'état "fermée" à l'état "en train de s'ouvrir" si le bouton d'ouverture est enfoncé et que la porte n'est pas déjà en train de s'ouvrir.

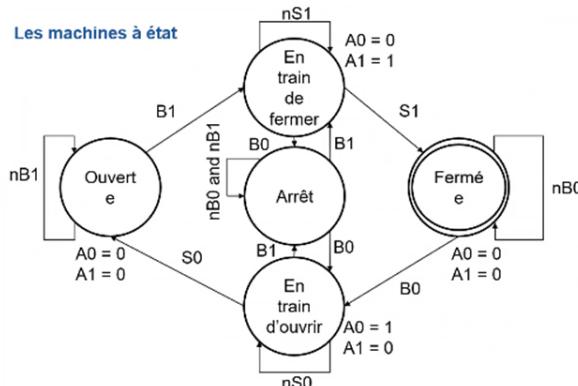
De l'état "en train de s'ouvrir" à l'état "ouverte" lorsque le détecteur de fin de course indique que la porte est complètement ouverte.

De l'état "ouverte" à l'état "en train de se fermer" si le bouton de fermeture est enfoncé et que la porte n'est pas déjà en train de se fermer.

De l'état "en train de se fermer" à l'état "fermée" lorsque le détecteur de fin de course indique que la porte est complètement fermée.

De l'état "fermée" à l'état "en train de se fermer" si le bouton de fermeture est enfoncé et que la porte est déjà en train de s'ouvrir.

De l'état "ouverte" à l'état "en train de s'ouvrir" si le bouton d'ouverture est enfoncé et que la porte est déjà en train de se fermer.



V. TD :

Consignes :

Considérons un système de verrouillage à code. Le système possède un pavé numérique composé de 10 boutons, chacun étiqueté de 0 à 9, pour entrer un code de verrouillage composé de 4 digits.

Lorsque l'utilisateur appuie sur un bouton deux signaux sont générés, un signal de validité ainsi qu'un signal sur 4bits correspondant au digit, si quatre chiffres ont été entrés, le système doit vérifier si le code est correct.

Si le code est correct, le système doit déverrouiller la porte pendant 10 secondes. La porte est par défaut verrouillée.

Note : vous disposez d'une horloge cadencée à 100Mhz

Définir

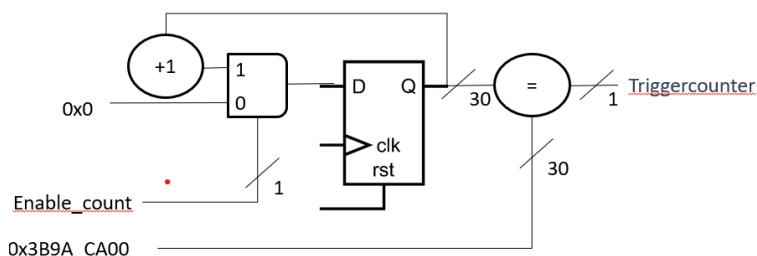
- les états
- Les sorties et les entrées
- Les parties opératives du système pilotant / générant les entrées si besoin
- Les transitions entre état

Un jeu d'état possible pour ce problème est :

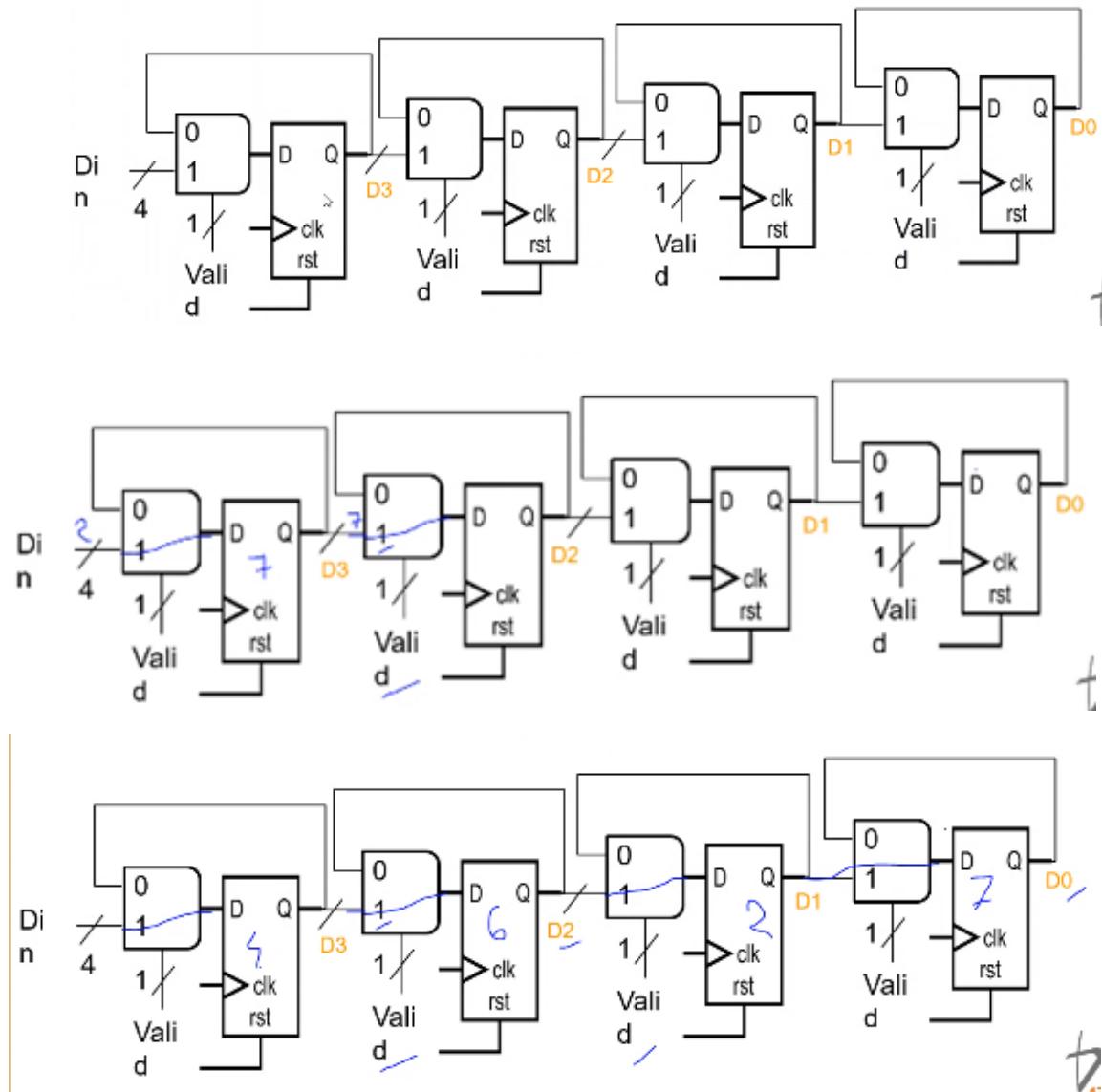
- Attente de la saisie du premier chiffre (état initial)
- Attente de la saisie du deuxième chiffre
- Attente de la saisie du troisième chiffre
- Attente de la saisie du quatrième chiffre
- Comparaison du code
- Ouverture porte
- Fermeture porte

La Sortie du système est noté « O » et pilotera l'ouverture de la porte. Nous utiliserons l'entrée « Valid » générée par le tableau de commande pour signifier le passage d'un état de saisie vers un autre. Nous générerons un signal « Triggercounter » au moyen d'un compteur de temporisation sur 29bits, ce dernier devra compter 1 000 000 000 cycles d'horloges (10 secondes d'attente); nous utiliserons un comparateur pour déterminer la fin de comptage.

Nous générerons un signal « Triggercounter » au moyen d'un compteur de temporisation sur 30bits, ce dernier devra compter 1 000 000 000 cycles d'horloges (10 secondes d'attente); nous utiliserons un comparateur pour déterminer la fin de comptage. Un signal « Enable_count » permettra le démarrage du compteur et sa remise à zero.

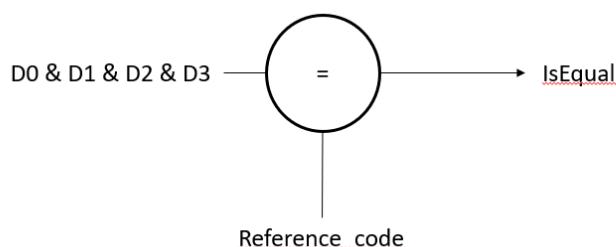


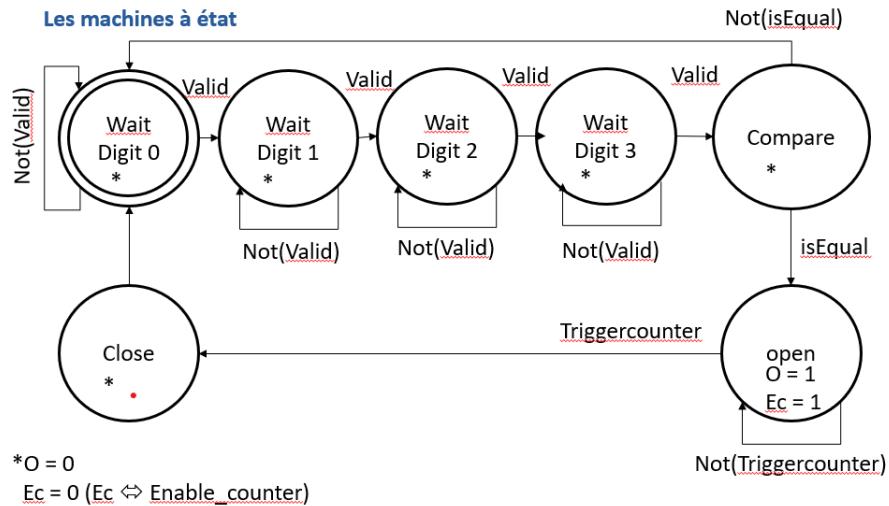
Nous générerons un signal « IsEqual » pour vérifier que le code entré est bien celui demandé. Pour cela, nous utiliserons un registre à décalage pour stocker la valeur de la séquence de 4 digits. Le décalage aura lieu si et seulement si le signal « Valid » est actif.



« Is Equal » sera généré si le code correspond à une valeur donnée (non spécifiée ici), on placera un comparateur entre les digits du registre à décalage et les digits du code attendu.

Note : Le comparateur prendra la forme d'une cascade de AND et de NOT, ici nous simplifions l'écriture

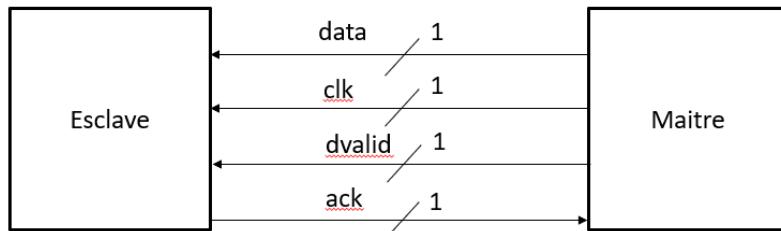




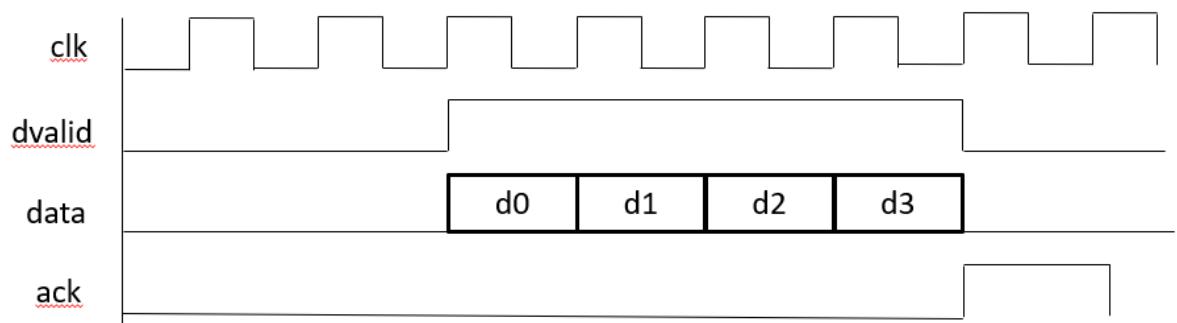
W. TD :

Supposons un protocole de communication entre deux composants. Ce dernier utilise :

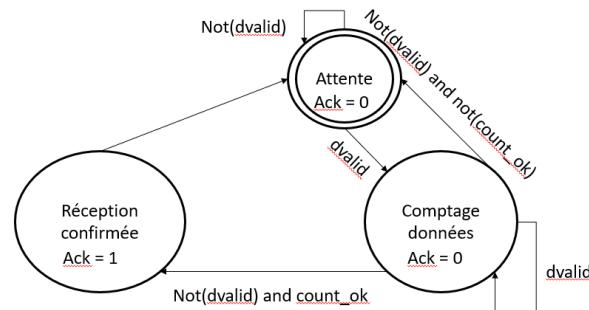
- Une ligne de transmission de données du maître vers l'esclave
- Une ligne pour la transmission d'horloge du maître vers l'esclave
- Une ligne pour la transmission d'un signal de confirmation de l'esclave vers le maître
- Une ligne pour la transmission d'un signal de validité de la donnée



Lors de la transmission, un mot de 4 bit est transmis sur la ligne data. Une horloge est générée telle que sur le chronogramme. Une pulsation ack est générée afin de signifier que la transmission du mot est complète.

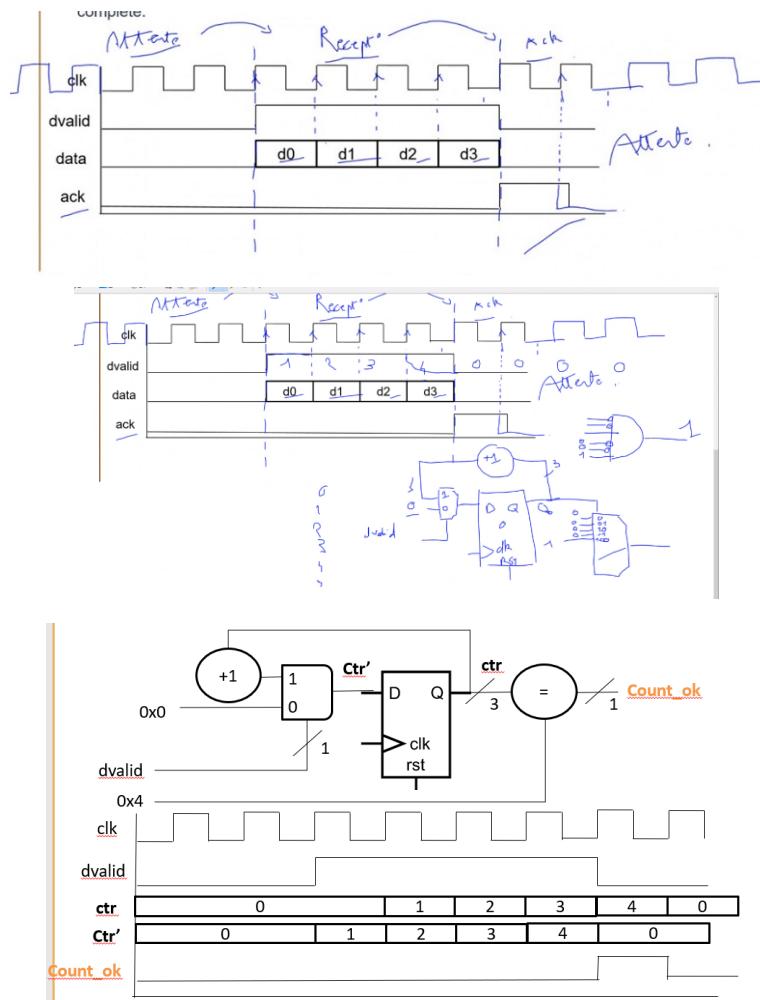


Réaliser un design FPGA qui permet la réception des données de ce protocole. Déterminer la machine à état du système ainsi que les parties opératives. La donnée doit être disponible.

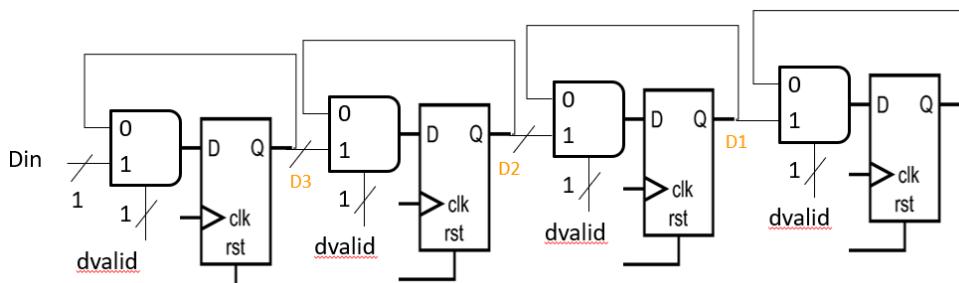


Lorsque dvalid=1, au 3^e front montant, on stock data d0,d1, d2, d3.

Clk montant, dvalid=0, ack=1 et on récupère toutes les valeurs.



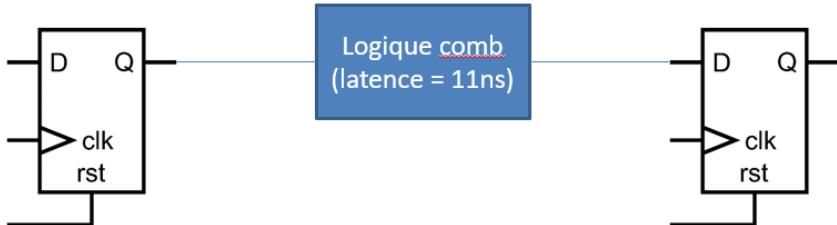
Un registre à décalage permet de récupérer la donnée.



m. Gestion de timing

Jusqu'ici, nous avons vu que tous les circuits logiques induisent une latence. Souvent la logique combinatoire est associée avec la logique synchrone. Que se passe-t-il si la latence de la logique combinatoire est très grande ?

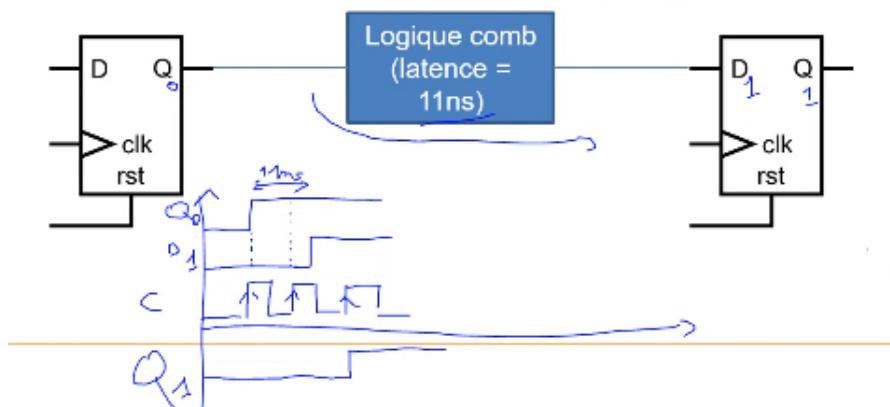
Soit le cas suivant, avec CLK = 100 Mhz, le circuit est-il fonctionnel ?



Soit le cas suivant, avec CLK = 100 Mhz, le circuit est-il fonctionnel ?

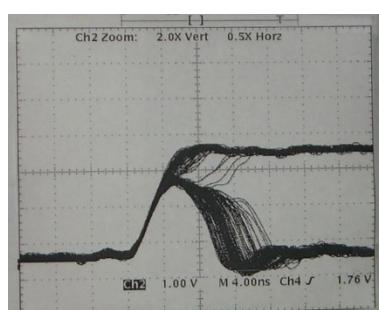
Soit le cas suivant, avec CLK = 100 Mhz, le circuit est-il fonctionnel ?

$$T = 10 \text{ ns}$$



La période de notre horloge est de 10ns. Entre deux fronts d'échantillonnage, le signal en sortie du premier registre n'a pas le temps de parcourir les LUTs et en entrée du second registre, le signal n'est pas prêt et potentiellement non stable au moment de l'échantillonnage.

C'est une problématique courante en micro-électronique. On résout cela en faisant appel à l'**Analyse de timing**.



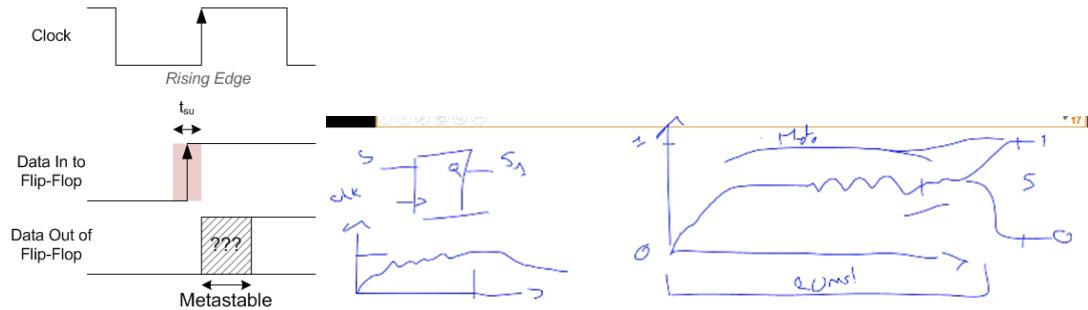
Si nous ignorons ce phénomène, il est possible que le signal en sortie de la logique combinatoire soit en état transitoire au moment où nous effectuons l'**échantillonnage**. La conséquence de cela est un signal **non stable** dans le registre de sortie. On dit qu'il est **métastable**.

Dans le cas de la métastabilité, les transistors de portes logiques sont « à moitié » ouverts ou fermés, le signal de sortie est alors dans un état non logique (à peu près vers 1 ou 0).

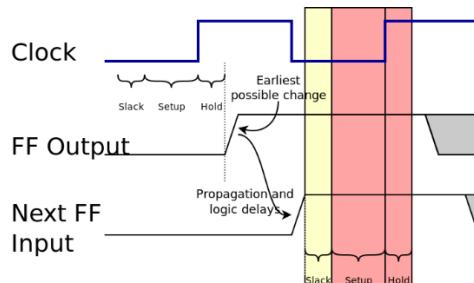
Il est alors impossible de poursuivre les calculs avec ce type de signaux **au risque de propager des erreurs**.

Pour éviter ceci, il existe des spécifications données par le constructeur FPGA.

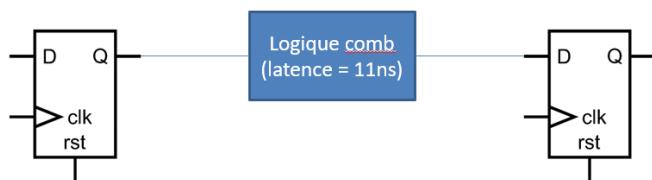
« Tout signal, en entrée d'un registre, doit être dans un état stable durant une plage de temps s'étalant avant et après le front montant d'horloge »



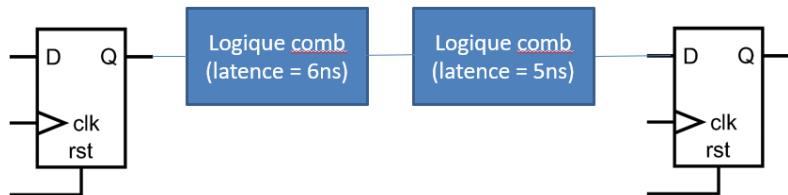
Durant cette plage de timing, le signal ne doit pas changer d'état et être établi à un état donné (0 ou 1). La plage de temps située avant le front montant s'appelle la plage de « setup », celle après est la plage de « hold »

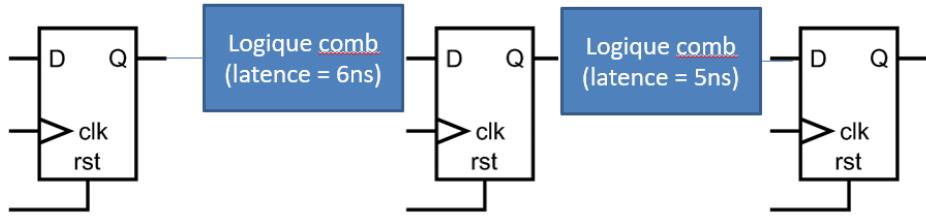


Reprenons le design précédent, quel serait la solution pour satisfaire les conditions de timing ?



Prenons le cas ci-dessous, quel pourrait être la méthode à employer dans ce cas ?



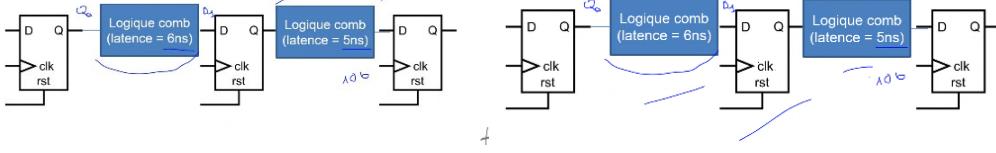


Gestion de timing

Prenons le cas ci-dessous, quel pourrait être la méthode à employer

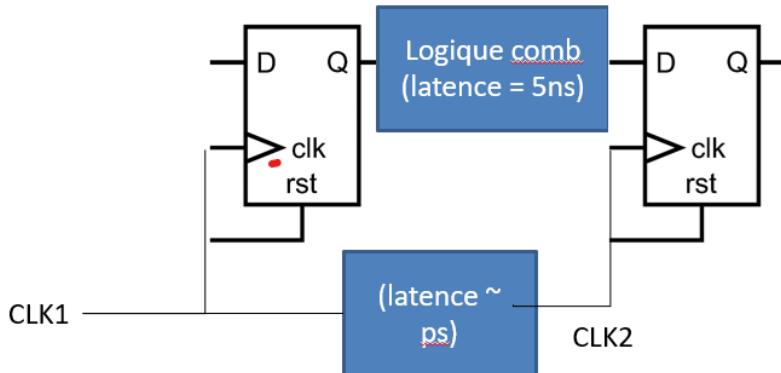
Gestion de timing

Prenons le cas ci-dessous, quel pourrait être la méthode à employer dans ce cas ?

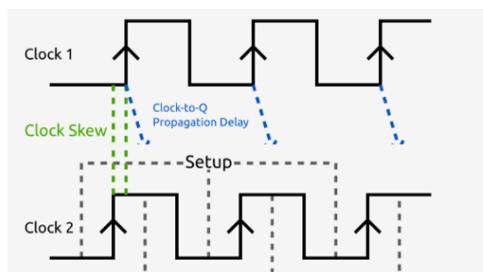


Gestion timing

Le clock skew, horloge désynchronisée. Cela se produit lorsque le signal d'horloge n'arrive pas en même temps aux registres

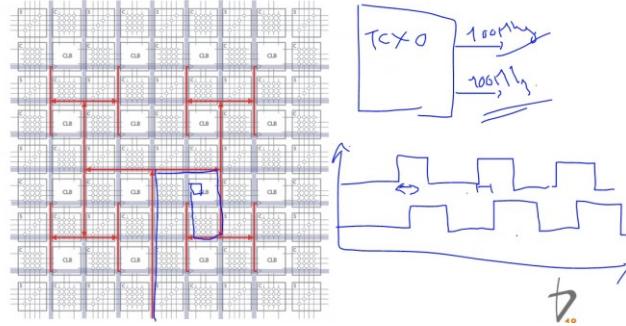


Le clock skew, horloge désynchronisée. Cela se produit lorsque le signal d'horlogé n'arrive pas en même temps aux registres



Gestion de timing

Pour limiter le clock skew, les horloges sont routées sur un réseau particulier spécialement conçu pour de la faible latence, il s'appelle l'**arbre d'horloge (clock tree)**



Une règle d'or est de ne jamais installer de logique combinatoire sur le porte d'horloge. Autrement les désynchronisations vont apparaître dans le circuit.

Globalement, qu'est ce qui augmente la résilience d'un système aux erreurs de timing ?

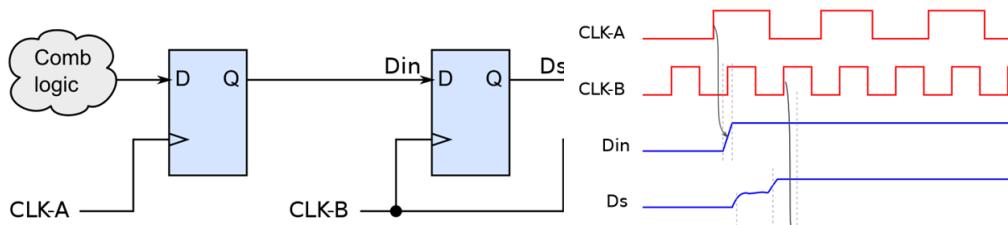
Un chemin critique le plus court possible

Une fréquence de fonctionnement basse

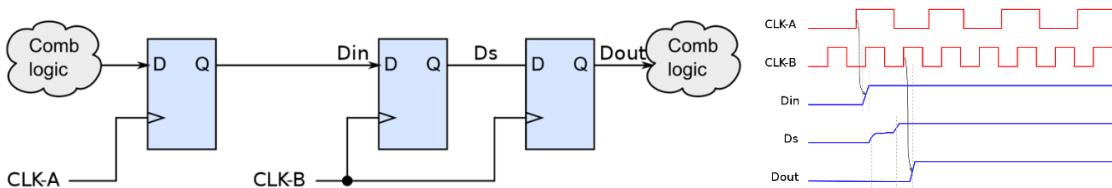
Un silicium de bonne facture (-4C ou -1C)

n. Gestion de timing, cas particuliers et résolution

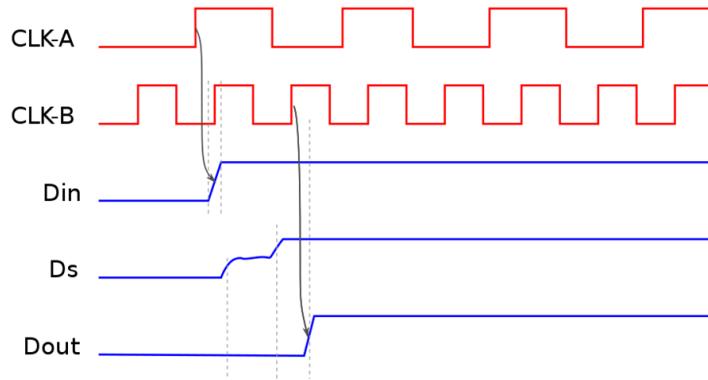
Utilisation de deux horloges dans un système. Très fréquent lorsque vous concevez une interface, souvent le protocole est basé sur une horloge avec une fréquence moins élevée que celle de votre système



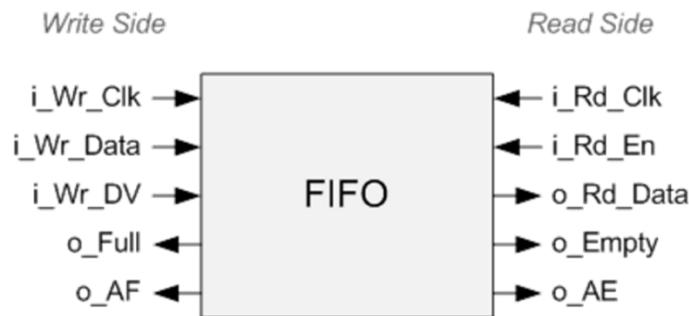
Dans ce cas, on place le plus souvent un « double rideau » afin d'éviter la propagation de la métastabilité. Un registre est tout suite ajouté à la sortie Ds



Plus le signal métastable est échantillonné tôt moins on risque la propagation de la métastabilité dans le reste du circuit. On ne place donc pas de portes logiques entre les deux registres



Dans le cas d'un CCD (Cross Clock Domain) au niveau d'un bus de données. On peut aussi faire appel à une Fifo. Attention à l'équilibre des flux de données



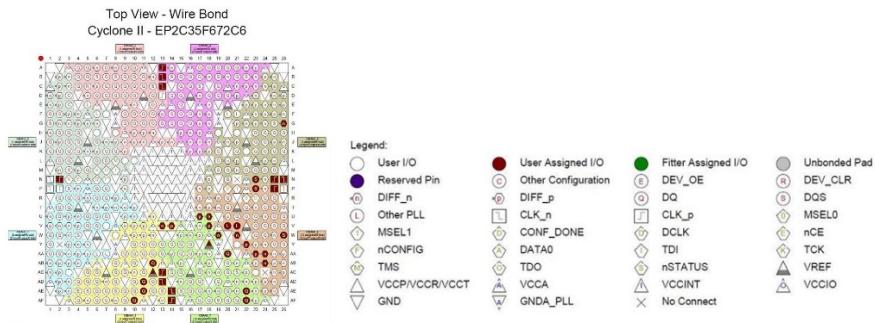
FIFO : mémoire tampon.

Permet d'écrire, une séquence de donnée dans la mémoire et de lire les autres à cotés. Faire attention à l'équilibre des données.

VII. Les I/Os dans un FPGA

Les FPGA offrent une grande flexibilité au niveau de la configuration de ses I/O. On peut sélectionner un niveau de tension et le type de liaison que nous souhaitons dans la limite des capacités de notre composant. Certains pins restent dédiés à des applications d'alimentation ou de gestion d'horloges. Le plus souvent, les I/O sont groupées en **bank**. Chaque bank possède une configuration pour l'alimentation des **buffers** situés dans les **PADs**.

Avec un seul composant, on peut obtenir une multitude d'interfaces qui seront pour certaines en 1,8V, 2,5V et 3,3V par exemple.



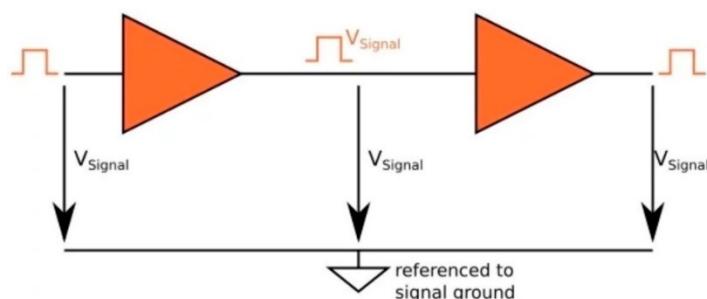
Un cas de design typique exploitant cet versatilité : le « composant proximité ». Il rassemblent toutes informations provenant d'une grande quantités de capteurs et transfert les informations sur une liaison unique plus « user friendly » .

Les FPGA vous offrent la possibilité de configurer vos pins d'entrée ou de sortie en fonction de plusieurs standards comme par exemple SSTL ou LVDS avec différents niveaux de tension.

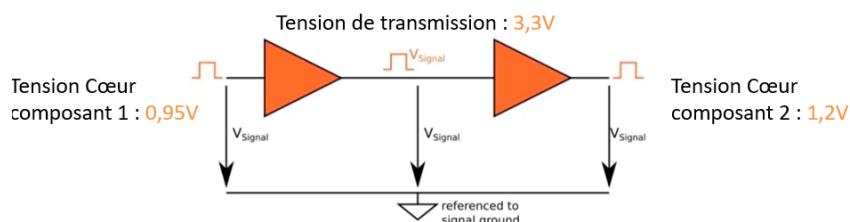
D'autres standards sont disponible selon le type de FPGA utilisé.

o. Les interfaces de transmission, Single Ended

C'est le standard le plus simple, il utilise une seule pin et permet deux états logiques GND ou VCCIO. Globalement, c'est celui utilisé pour des transmissions « lente » comme par exemple pour relié le FPGA à une LED ou un bouton poussoir. Il est aussi utilisé pour communiquer avec des composants comme des périphériques ou des capteurs. Par exemple, il est souvent emprunté dans l'interfaçage Arduino. Les protocoles de transmission I2C, Spi, QSPI sont construits, le plus souvent, avec sur des bus de donnée en single ended.



On retrouve des **buffers** de tension au niveau des PADs des composants. Leur rôle est de convertir la tension du signal depuis le niveau extérieur vers le niveau du cœur, souvent plus bas.



p. Les interfaces de transmission, Différentielle

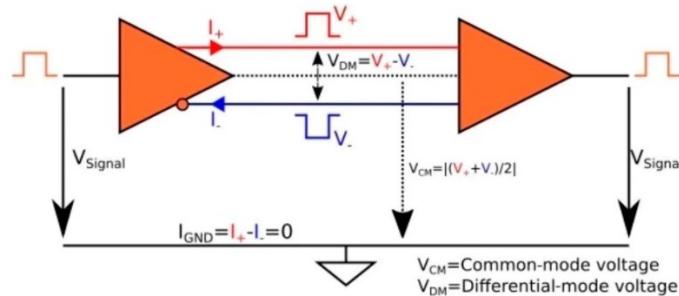
C'est un standard qui fait appel à deux PADs pour émettre un signal différentiel. Il y a donc deux câbles souvent appelés « S_p » et « S_n ».

On a $S_n = -S_p$

Au niveau du buffer de réception, on obtient le signal S en faisant l'opération :

$$S = (S_p - S_n)/2$$

$$\text{Donc } S = (S_p - (-S_p))/2 = 2*S_p/2 = S_p$$



Le principe est que si une perturbation (IEM par exemple) survient. Alors la sortie reste **inchangée** les deux pistes sont affectées.

On a alors

- $S_n' = S_n + \text{iem}$
- $S_p' = S_p + \text{iem}$

Comme $S = (S_p' - S_n')/2$ On a alors

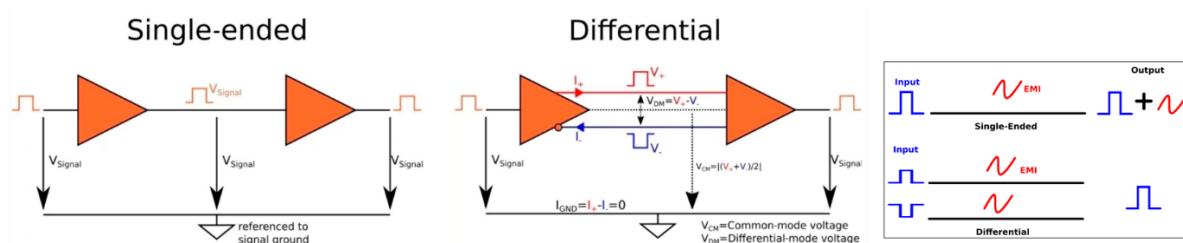
$$S = (S_p + \text{iem} - (S_n + \text{iem}))/2$$

$$S = (S_p + \text{iem} - S_n - \text{iem})/2$$

$$S = (S_p - S_n)/2$$

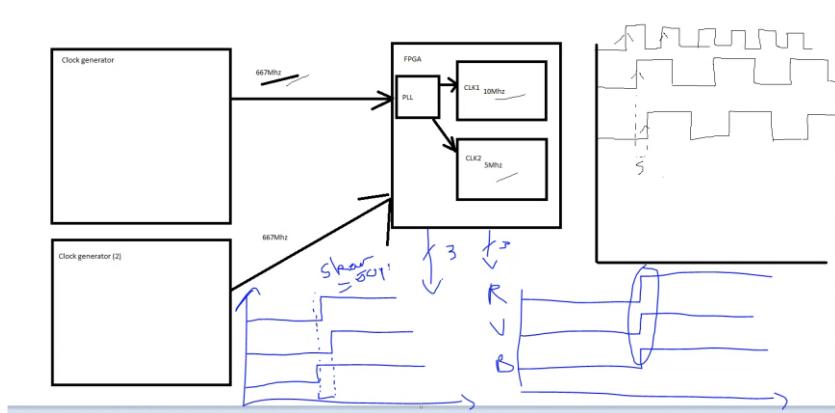
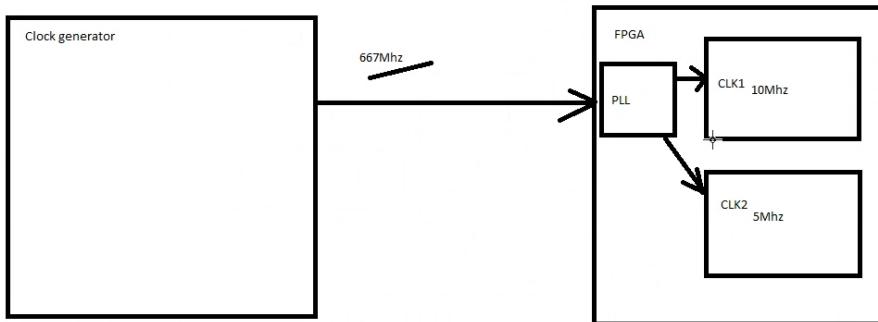
$$S = (S_p + S_p)/2$$

$$S = S_p$$



Exploration datasheet composant efinix et vision microscope carte coraZ7.

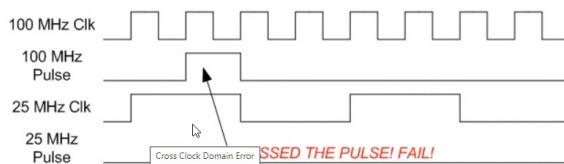
Annexe



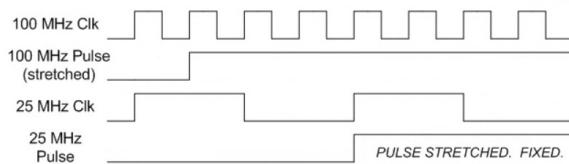
NANDLAND

THE GO BOARD

that you will never see this pulse if you simply sample the data with your 25 MHz clock! In order to transfer a signal from a fast clock domain to a slow clock domain you must stretch out your signal. Refer to the figures below for a visual representation of this.



100 MHz Pulse is not stretched. 25 MHz pulse never triggers.



100 MHz Pulse is stretched. 25 MHz pulse will now trigger.

Stretching your data occurs when you stretch out a pulse or data long enough to guarantee that the slow clock domain has a chance to sample it. In the example above, you would need to stretch out your data to ensure that setup and hold times are met no matter when the data is sampled. To properly guarantee that setup and hold times are met, I recommend stretching out your pulses such that they take at least 2 clock cycles in the slow