

PRESENTATION DU COMPOSANT FPGA

Notes - cours - segment # 0 #1 de la formation FPGA
(56h/8 jours /24/04/2023 - 04/05/2023)



04 /05/ 2023

NOTES COURS FPGA- SEGMENT 1 - EVE CHAR

Table des matières

1Segment 0 : Introduction générale	3
1.1Programme de la formation.....	3
2Segment 1 : Présentation du composant FPGA	3
2.1Objectifs	3
2.2Rappel / rafraîchissement sur les bases.....	3
2.2.1.1Exercice 1	4
2.2.1.2Exercice 2.....	5
2.2.1.3Exercice 3.....	5
2.2.1.4Exercice 4.....	6
2.2.1.5Exercice 5.....	7
2.2.1.6Exercice 6.....	9
2.2.1.7Exercice 7.....	10
2.2.2Travaux pratiques - TP1 – Full adder.....	12
2.3Base d'architecture d'un calculateur.....	12
2.3.1Rappel sur les Process unit.....	12
2.3.1.1Jargon à retenir :.....	12
2.3.1.2Exercice 8.....	14
2.3.2Rappel sur les MOSFET (Electronique numérique fondamentale)	15
2.3.3Travaux pratiques – TP2 – portes logiques et simulation sous Ltspice	15
2.3.3.1Exercice 9	16
2.3.4Rappel sur les Mémoires.....	18
2.3.4.1Exercice 10.....	18
2.4Rappel sur la Logique numérique synchrone.....	19
2.4.1La bascule (le latch)	20
2.4.1.1Exercice 11	20
2.4.2Les registres	22
2.4.2.1Différence entre les bascules D et T	23
2.4.2.2Exercice 12.....	24
2.4.3Les Compteurs :	24
2.4.3.1Compteur binaire.....	25
2.4.3.2Exercice 13.....	25
2.4.3.3Exercice 14.....	26

2.4.3.4Exercice 15.....	27
2.4.4Les Machines d'états.....	28
2.4.4.1Exercice 16.....	28
2.4.4.2Exercice 17.....	30
2.4.4.3Exercice 18.....	31
2.4.5Gestion de timing.....	32
2.4.5.1Exemple de Métastabilité.....	33
2.4.5.2Notions de Gestion de timing.....	33
2.4.5.3Jargon à retenir :.....	34
3ANNEXE	36

1 Segment 0 : Introduction générale

1.1 Programme de la formation

- ✓ Présentation du composant FPGA
- ✓ Méthode de développement d'un système logique FPGA
- ✓ Projet encadré interface vidéo VGA
- ✓ Création d'un filtre FIR sur le flux vidéo
- ✓ Etude d'un système FPGA complexe SoC
- ✓ Projet Final & soutenance

2 Segment 1 : Présentation du composant FPGA

2.1 Objectifs

- ✓ Apprendre les bases du calcul numérique
- ✓ Réaliser un premier circuit logique
- ✓ Comprendre les architectures des calculateurs
- ✓ Identifier les sous-composants d'un FPGA
- ✓ Maîtriser le comportement de sous-composants FPGA
- Nombre de journées : 8
- Nombre de TDs : 15

2.2 Rappel / rafraîchissement sur les bases

Représentation des nombres en informatique, base binaire, décimale et hexadécimale :

$$\begin{array}{r}
 100111 \\
 \hline
 \begin{array}{l}
 \rightarrow 2^0 \times 1 = 1 \\
 \rightarrow 2^1 \times 1 = 2 \\
 \rightarrow 2^2 \times 1 = 4 \\
 \rightarrow 2^3 \times 0 = 0 \\
 \rightarrow 2^4 \times 0 = 0 \\
 \rightarrow 2^5 \times 1 = 32 \\
 \hline
 \text{Decimal} \leftarrow 39
 \end{array}
 \end{array}$$

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

2.2.1.1 Exercice 1

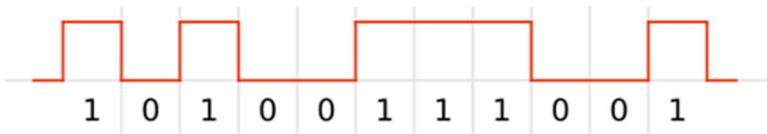
Représenter les chiffres suivants en base 2 et 16

décimale	base 2	base 16 (HEX)
18	0001 0010	0x12
40	0010 1000	0x28
64	0100 0000	0x40

- La représentation d'une donnée numérique lors des manipulations

Le chronogramme

Vue analytique de la donnée liée à une simulation



La mesure

Vue physique de la donnée liée à une mesure en labo



A retenir :

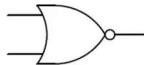
Chaque opérateur logique possède

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



NAND

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



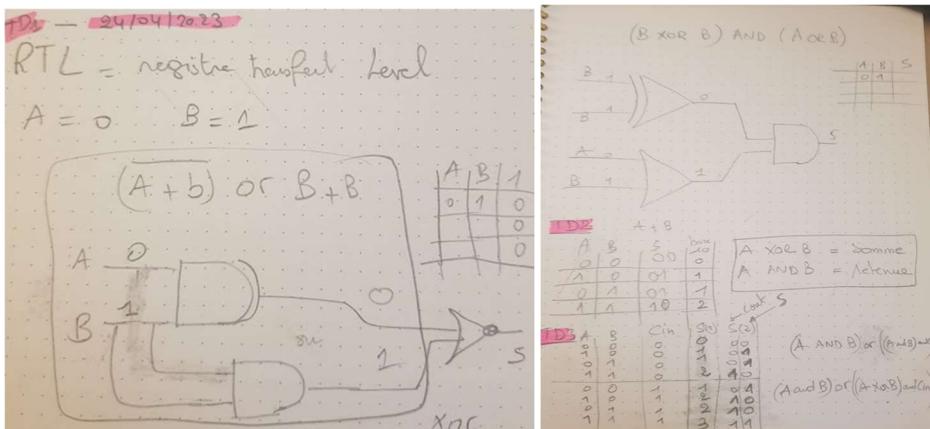
NOR

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



XNOR

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



2.2.1.2 Exercice 2

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

- Quelle est la sortie des opérations suivantes ?
- Représenter le schéma logique de ces opérations
- $\text{NOT}((A \text{ AND } B) \text{ OR } (B \text{ AND } B))$
- $(B \text{ XOR } B) \text{ AND } (A \text{ OR } B)$

NOT((A AND B) OR (B AND B))	<pre> graph LR A1[A] --> AND1[AND] B1[B] --> AND1 AND1 --> OR1[OR] B2[B] --> AND2[AND] B2 --> OR1 AND2 --> OR1 OR1 --> S1[S] </pre>	<pre> graph LR A1[0] --> AND1[AND] B1[1] --> AND1 AND1 --> OR1[OR] B2[1] --> AND2[AND] B2 --> OR1 AND2 --> OR1 OR1 --> S1[0] </pre>
$(B \text{ XOR } B) \text{ AND } (A \text{ OR } B)$	<pre> graph LR B1[B] --> NOTB[NOT] NOTB --> AND1[AND] A1[A] --> OR1[OR] AND1 --> S1[S] </pre>	<pre> graph LR B1[1] --> NOTB[NOT] NOTB --> AND1[AND] A1[1] --> OR1[OR] AND1 --> S1[0] </pre>

2.2.1.3 Exercice 3

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

1) 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 (base 10)	S (base 2)
0	0	0	00
1	0	1	01
0	1	1	01
1	1	2	10

2) Simplifier ce tableau en conservant trois colonnes, A, B et S0

3) Quel opérateur booléen permet d'obtenir S0 à partir de A et B ?

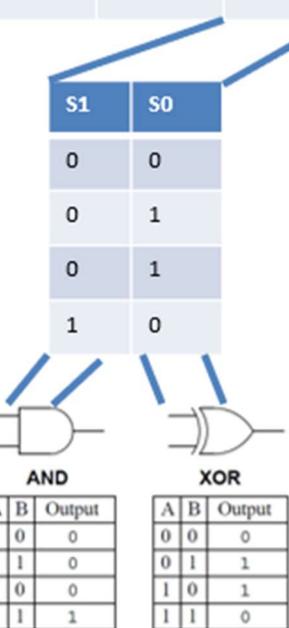
4) Même question pour obtenir S1.

S1	S0
0	0
0	1
0	1
1	0

Réponse : ma compréhension

A	B	S base 10)	S base 2)
0	0	0	00
1	0	1	01
0	1	1	01
1	1	2	10

S1 agit comme notre « Carried bit » ou notre retenue



AND

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



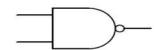
OR

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



XOR

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



NAND

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



NOR

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

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

2.2.1.4 Exercice 4

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 » déterminer 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 »

L'équation 'S' obtenue est alors :

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

L'équation 'Cout' obtenue est alors :

$$\begin{aligned}
 C_o &= (\bar{A} \text{ and } B \text{ and } C_{in}) \text{ or } (A \text{ and } \bar{B} \text{ and } C_{in}) \text{ or } (A \text{ and } B \text{ and } \overline{C_{in}}) \text{ or } (A \text{ and } B \text{ and } C_{in}) \\
 C_o &= (\bar{A} \cdot B \cdot C_{in}) + (A \cdot \bar{B} \cdot C_{in}) + (A \cdot B \cdot \overline{C_{in}}) + (A \cdot B \cdot C_{in}) \\
 C_0 &= \textcolor{red}{C_{in}}(\bar{A} \cdot B + A \cdot \bar{B}) + \textcolor{green}{A \cdot B}(\overline{C_{in}} + C_{in}) \\
 \Leftrightarrow C_0 &= C_{in}(A \oplus B) + A \cdot B \cdot \textcolor{brown}{1}
 \end{aligned}$$

2.2.1.5 Exercice 5

Etablissez la table de vérité des expressions

$$(\bar{A} \cdot \bar{B} + A \cdot B) \text{ et } (\quad \quad \quad \bar{A} \cdot B + A \cdot \bar{B}) \quad \text{et}$$

simplifier les pour obtenir une seule porte logique comme précédemment.

On va essayer de réduire et simplifier l'équation suivante et de justifier comment elle se réduit en :

A	B	not(A)	not(B)	not(A) and not(B)	\oplus	A and B	Output (XNOR)
0	0	1	1	1	0	0	1
0	1	1	0	0	0	0	0
1	0	0	1	0	0	0	0
1	1	0	0	0	0	1	1

A	B	not(A)	not(B)	A and not(B)	not(A) and B	Output (XOR)
0	0	1	1	1	0	0
0	1	1	0	0	0	1
1	0	0	1	1	1	0
1	1	0	0	0	0	0

Réduction de l'équation :

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

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

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

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

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

Remarque : 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)

** la K-map pour la sortie **S** du circuit

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
.				

** la K-map pour la sortie Cout du circuit

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	0
1	1	0	0	1
1	1	1	1	1

On va essayer de réduire et simplifier l'équation suivante et de justifier comment elle se réduit en :

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

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

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

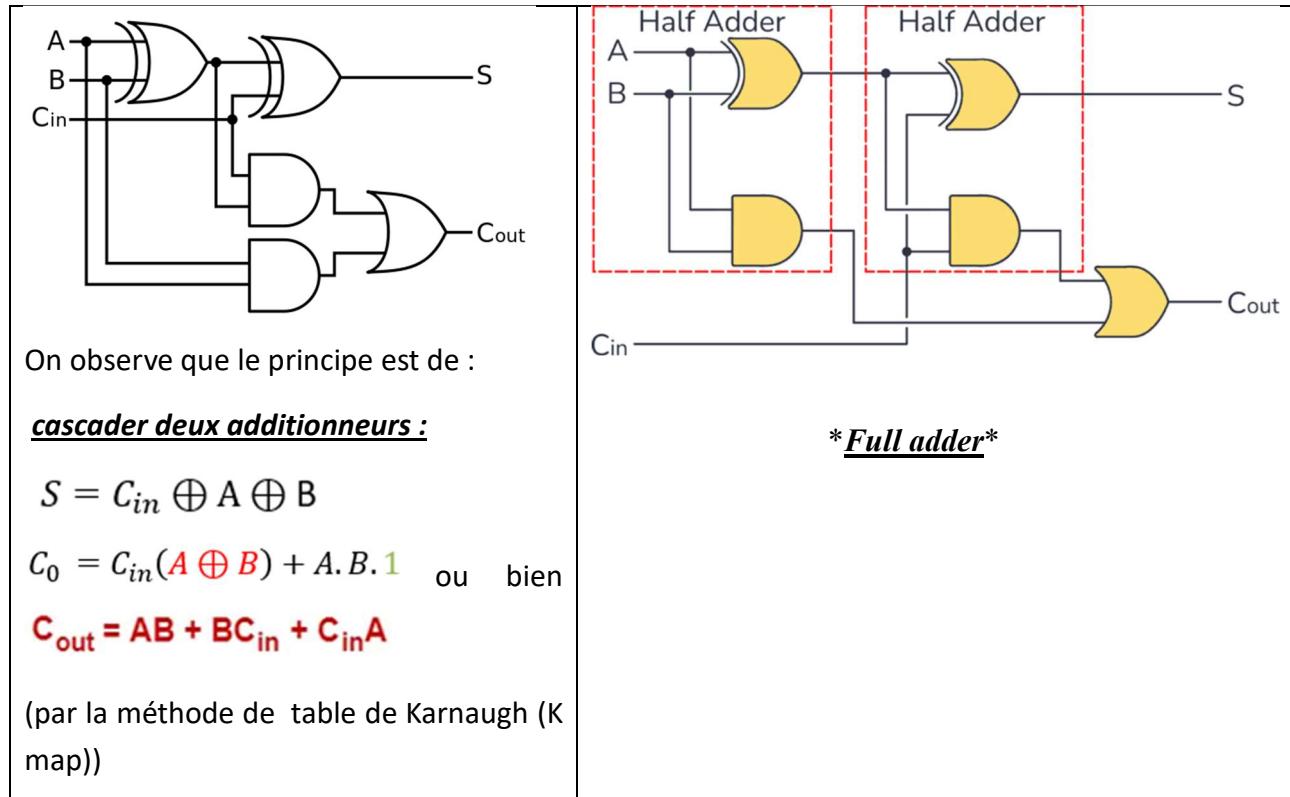
Cin	not(Cin)	Cin or not(Cin)
0	1	1
0	1	1

L'expression peut être « compressée » en une seule fonction booléenne :

Sachant : $(\overline{C_{in}} + C_{in})$: Ce qui correspond à une fonction toujours vraie, de fait, elle peut être symbolisée par un « 1 »

Donc : $\Leftrightarrow C_0 = C_{in}(A \oplus B) + A \cdot B \cdot 1$

On obtient le circuit entier suivant :



2.2.1.6 Exercice 6

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). 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).

Pour la multiplication : on remarque un décalage à gauche d'un bit

Multiplication x 2

DEC	A	B	C	résultat				
				en dec	out [3]	out [2]	out [1]	out [0]
0	0	0	0	0	0	0	0	0
1	0	0	1	1	2	0	0	1
2	0	1	0	0	4	0	1	0
3	0	1	1	1	6	0	1	1
4	1	0	0	0	8	1	0	0
5	1	0	1	1	10	1	0	1
6	1	1	0	0	12	1	1	0
7	1	1	1	1	14	1	1	1

Pour la division : on remarque un décalage d'un bit à droite

Division par 2

DEC	A	B	C	résultat en dec	out [3]	out [2]	out [1]	out [0]
0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1
2	0	1	0	1	0	0	1	0
3	0	1	1	1	0	0	1	1
4	1	0	0	2	0	1	0	0
5	1	0	1	2	0	1	0	1
6	1	1	0	3	0	1	1	0
7	1	1	1	3	0	1	1	1

Conclusion :

Lorsqu'on multiplie/divise une entrée par une puissance de 2 (2, 4, 8, 16 etc..) il suffit d'opérer un décalage binaire respectivement à gauche ou à droite, le nombre de décalage dépend de la puissance de 2 utilisée (1 bit (2), 2bit (2^2), 3bit (2^3), 4bit (2^4)))

- $2 \times 2 = 4$ soit $10(\text{b}2) \times 10(\text{b}2) = 100(\text{b}2)$, on a ajouté un zéro à droite et décaler notre entrée de 1 bit
- $5 \times 2 = 10$ soit $101(\text{b}2) \times 10(\text{b}2) = 1010(\text{b}2)$, on a ajouté un zéro à droite et décaler notre entrée de 1 bit
- $6 / 2 = 3$ soit $110(\text{b}2) / 10(\text{b}2) = 011(\text{b}2)$, on a retiré un digit à gauche et décaler notre entrée de 1 bit à droite (suppression du **LSB**)
- $7 / 2 = 3$ soit $111(\text{b}2) / 10(\text{b}2) = 011(\text{b}2)$, on a retiré un digit à gauche et décaler notre entrée de 1 bit à droite (suppression du **LSB**)
- $7 / 4 = 1$ soit $111(\text{b}2) / 100(\text{b}2) = 001(\text{b}2)$, on a retiré deux digits à gauche et décaler notre entrée de 2 bits à droite (suppression des deux **LSB**)

(**MSB** : Most Significant Bit & **LSB** : Less Significant Bit)

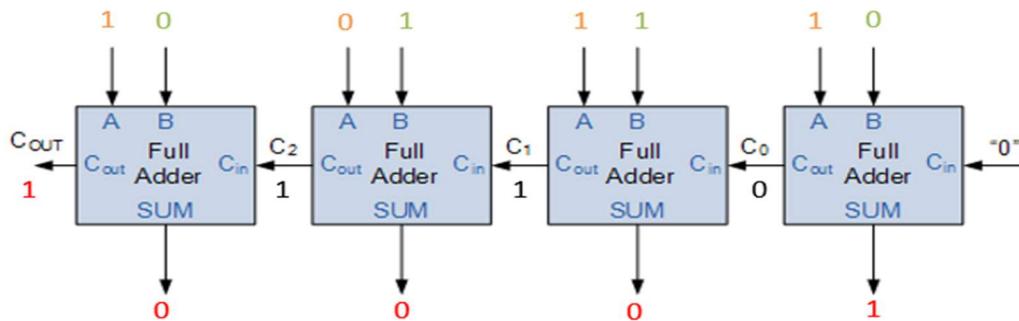
2.2.1.7 Exercice 7

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

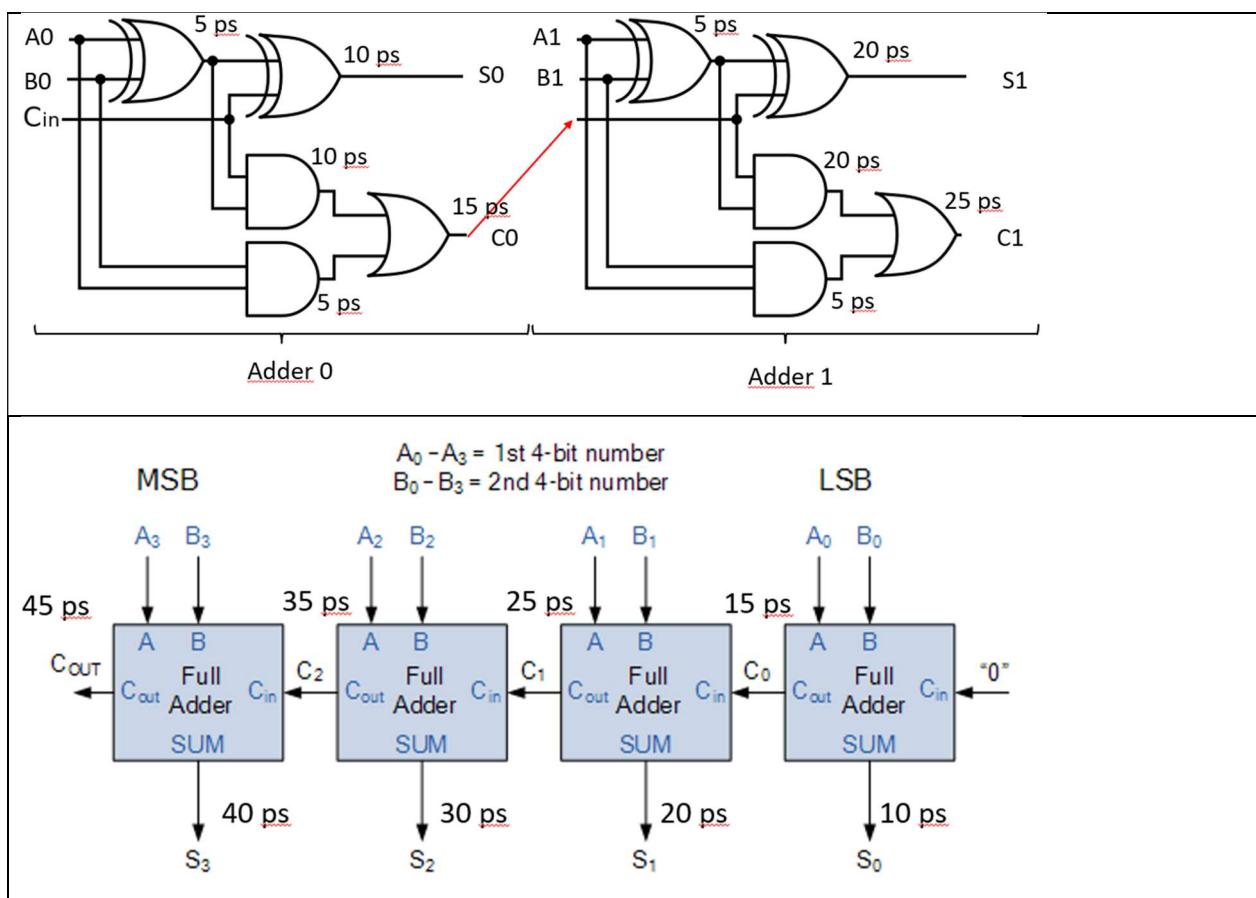
Soit $A = 1011_{(b2)} \Leftrightarrow B(b16) \Leftrightarrow 11_{(b10)}$ et $B = 0110_{(b2)} \Leftrightarrow 6(b16) \Leftrightarrow 6_{(b10)}$

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



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

Pour chaque chemin on compte le nombre de porte logique que l'on traverse, on ajoute 5ps à chaque passage. La latence pour obtenir S est de 10ps et pour Cout de 15 ps

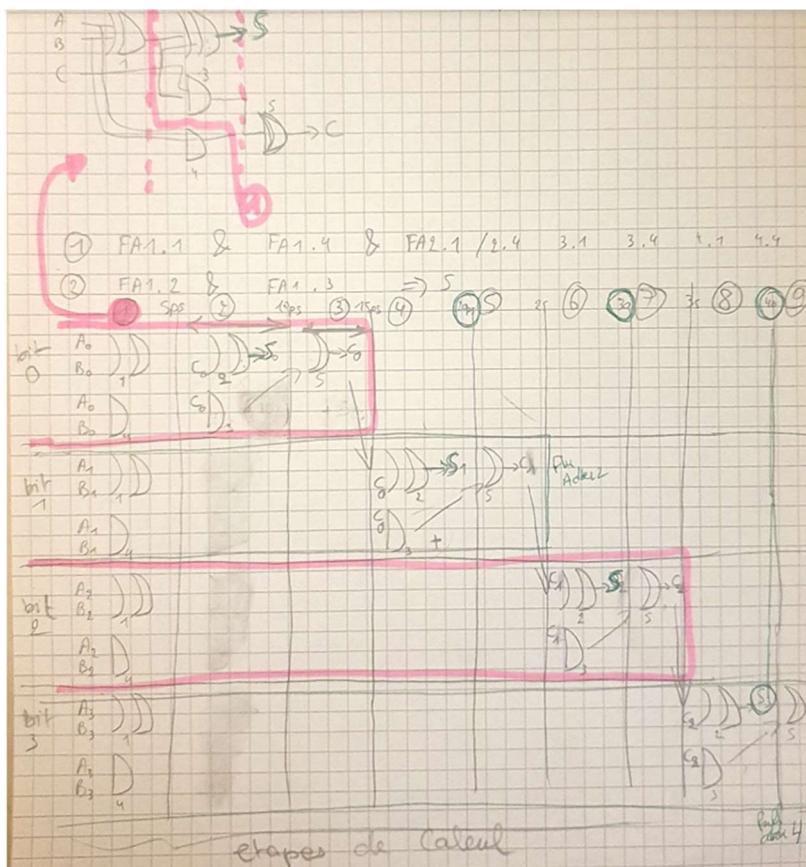


Sur 4 bits, nous avons besoin de 4 full-adders chainés autrement dit en pipeline.

Notre latence est donc => L s = 40 ps et L cout = 45 ps

Sur 500 itérations notre temps d'exécution est de Lx500 (code serait alors de $500 \times 45\text{ps} = 22.5\text{us}$)

Ma compréhension sur papier :



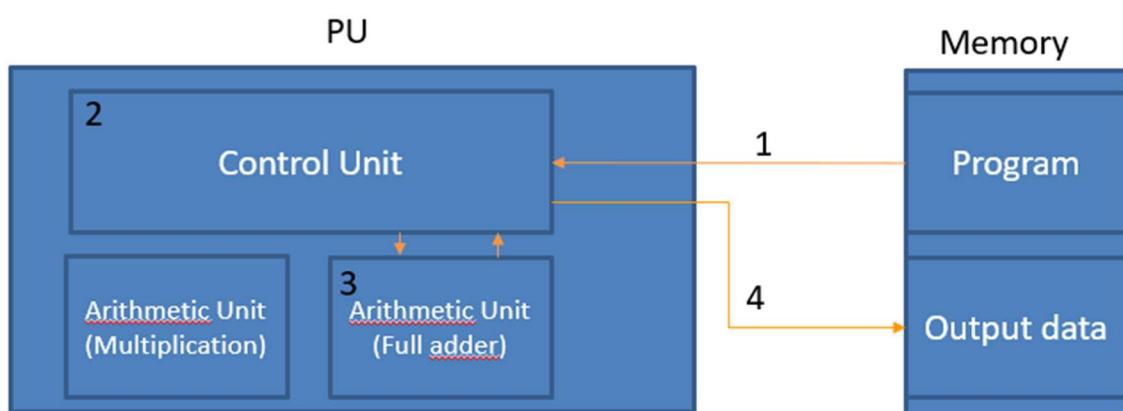
2.2.2 Travaux pratiques - TP1 – Full adder

Voir compte rendu TP1-FULL ADDER - 26/04/2023

2.3 Base d'architecture d'un calculateur

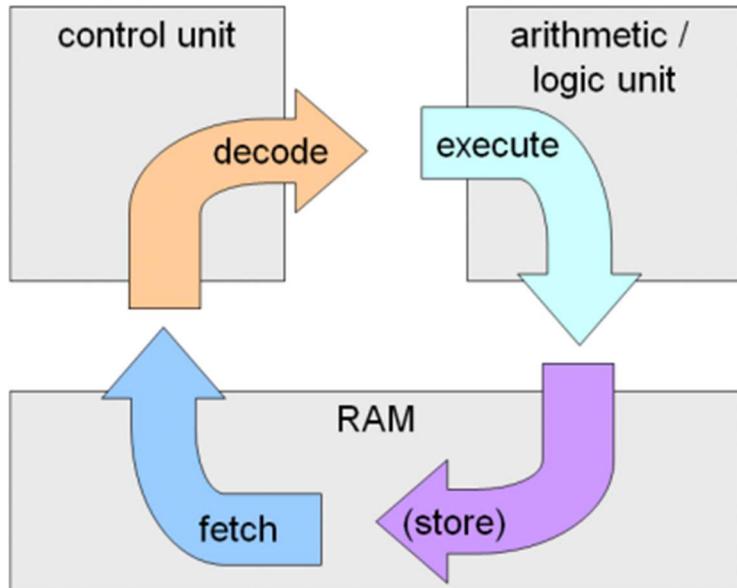
2.3.1 Rappel sur les Process unit

Les opérations d'un calculateur type CPU, GPU ou microcontrôleur



2.3.1.1 Jargon à retenir :

- Lorsque les instructions sont enregistrées dans la mémoire → c'est la programmation
- Les instructions sont exécutées dans la PU → c'est l'inférence

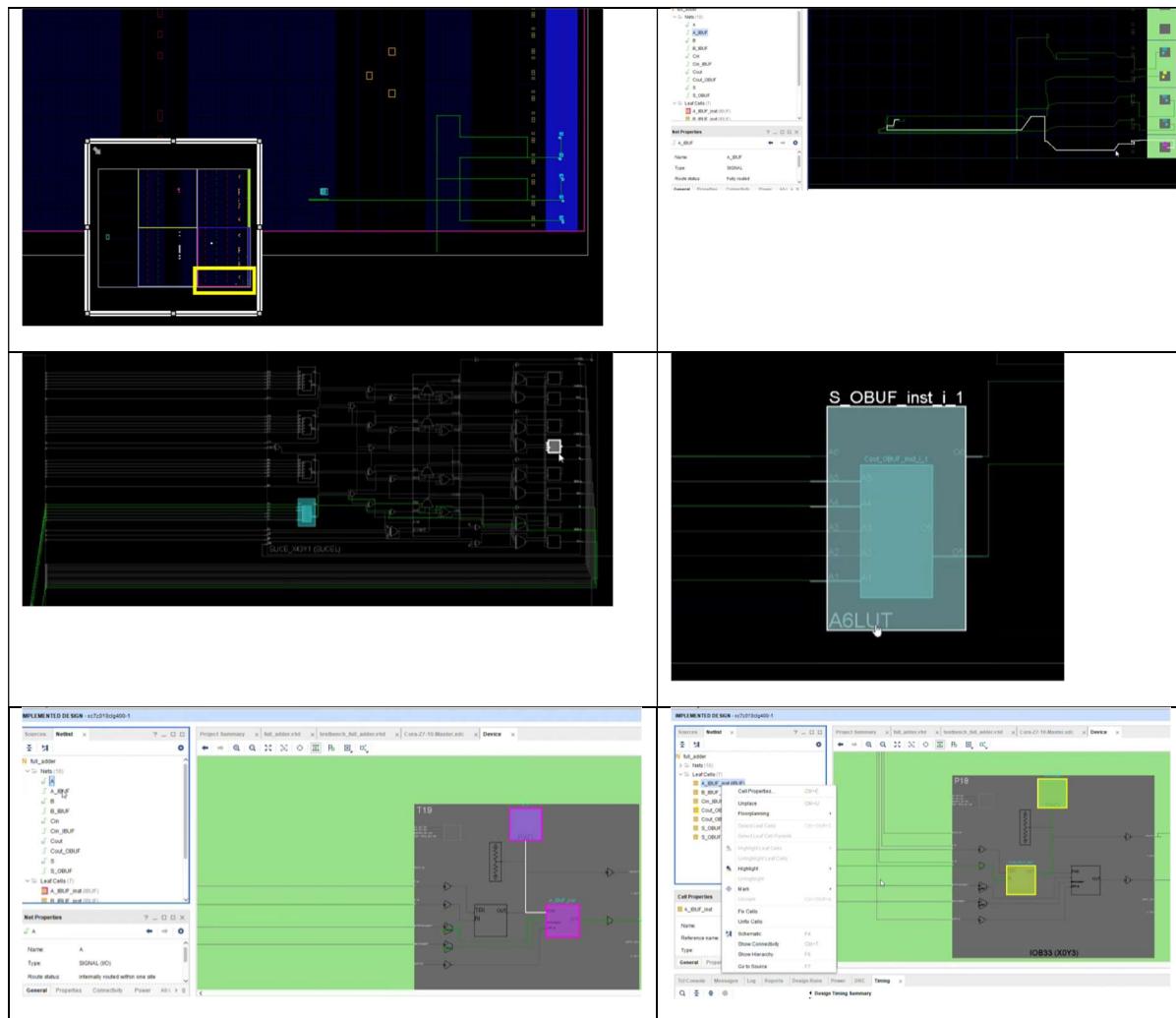


Ce mode d'exécution **est très versatile**, cependant, **chaque étape prend du temps** qui augmente la latence d'inférence (« ça RAM! ») par ailleurs chaque étape à un **coût en énergie non négligeable** lorsque répété plusieurs fois.

- **Le fetch** = c'est l'étape entre la mémoire et le CU : récupérer l'instruction depuis la mémoire
- **decode** : Décoder l'instruction, quelle opération nous devons réaliser
- le **chemin critique** : le chemin le plus long pour lier des portes logiques
- en **pipeline** : en série : exemple : 4 full-adders chainés
- **Logical Elements (LE)** : On utilise deux fois plus de Process Units, aussi appelées Logical Elements (LE), dans le contexte d'un FPGA (slide 97)
- **BRAM = emBdedded RAM**
- Slice = est un regroupement de composants logiques bas niveau, Chaque FPGA possède sa propre architecture de Slice
- **LUT** est une **Look Up Table**, elle permet de **reproduire le comportement d'une porte logique**
- **Flip Flop** = La **FF** est un **registre** et permet la mémorisation d'un signal. Chaque FF permet de mémoriser 1 bit, c'est un composant **synchrone**
- La **Carry4** : c'est un composant ajouté par Xilinx dans la série zynq7010 et permet d'optimiser les fonctions de comptage très récurrentes dans les designs FPGA

2.3.1.2 Exercice 8

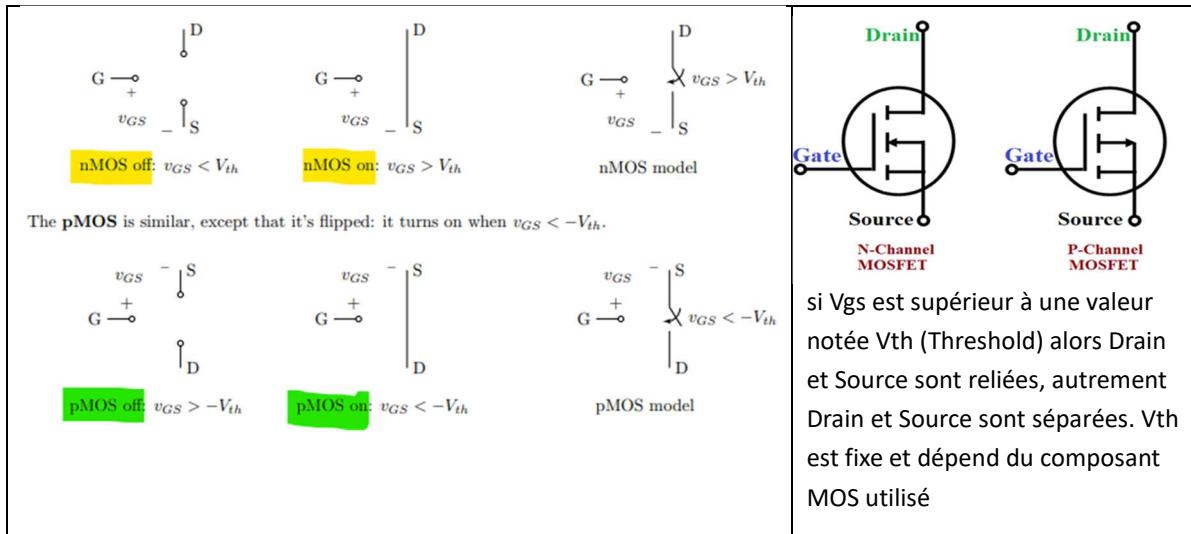
TD : Exploration guidée, étude des composants internes du zynq7010, focus sur les sous-composants du FPGA ; les Logical Elements ainsi que la mémoire embarquée (emBedded RAM ou BRAM)



A retenir :

- on retrouve toujours des LUTs, des MUX et des registres dans un FPGA.
- Chaque fabricant a sa propre architecture de fonction par slice ...
- Une Slice parfois appelée aussi « Logic Cell »
- Une slice va toujours contenir :
 - ✓ Des LUTs
 - ✓ Des multiplexeurs (MUX)
 - ✓ Des registres (FF)

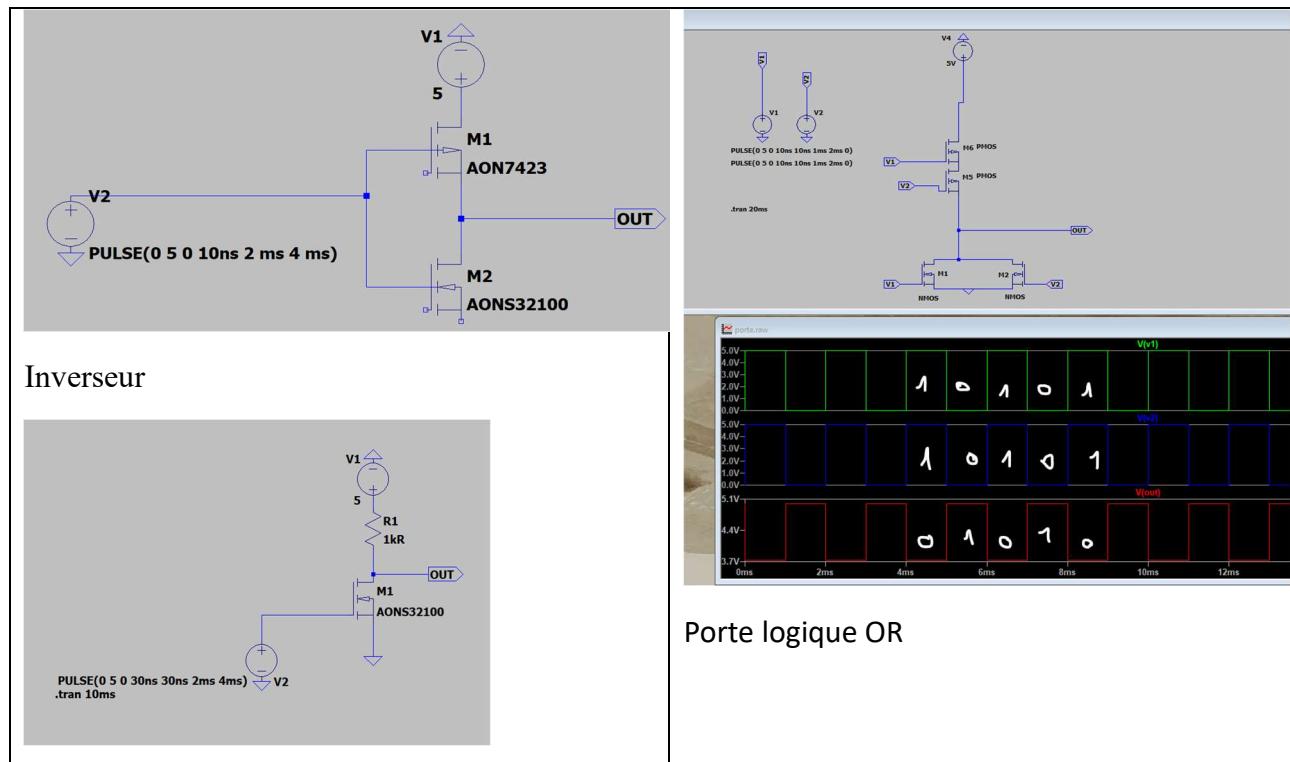
2.3.2 Rappel sur les MOSFET (Electronique numérique fondamentale)

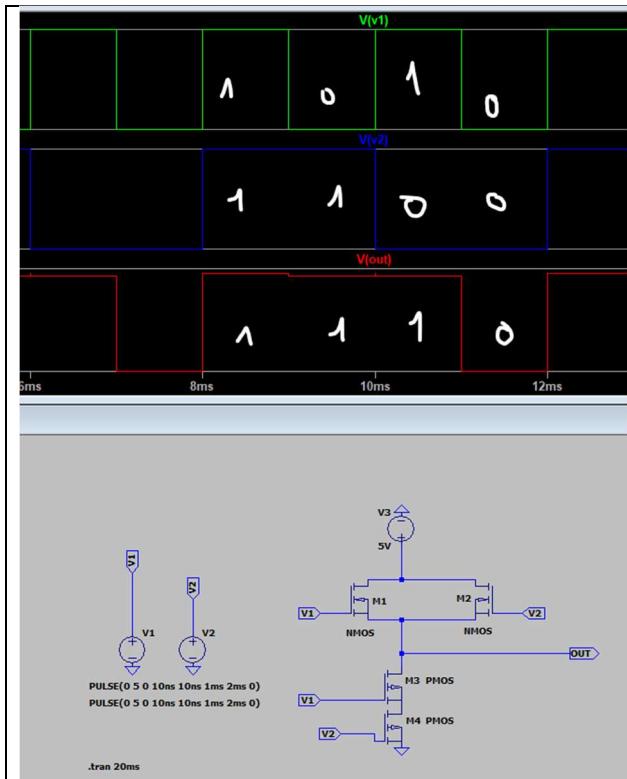


2.3.3 Travaux pratiques – TP2 – portes logiques et simulation sous Ltspice

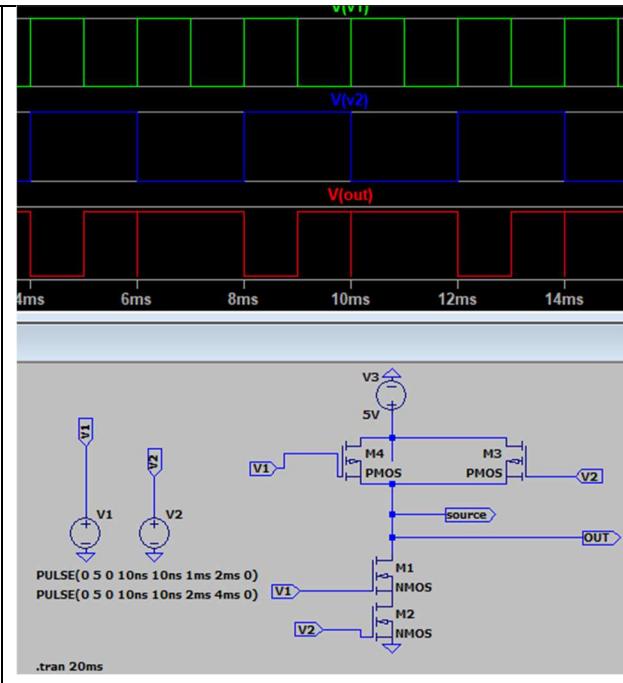
Objectif est de faire des modélisations de portes logiques et simulation sous Ltspice.

➔ Voir CR-TP 27/04/2023





Circuit représente la porte logique NOR



Circuit représente la porte logique NAND

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**

2.3.3.1 Exercice 9

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

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

 $S = A$ $S = B$

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

 $S = A$ $S = B$

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

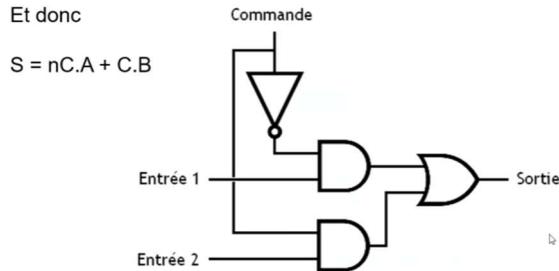
$$\text{Donc, } S = nC \cdot (A \cdot nB + A \cdot B) + C \cdot (nA \cdot B + A \cdot B)$$

$$\text{Donc, } S = nC \cdot (A + A) + C \cdot (B + B)$$

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

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

Et donc



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

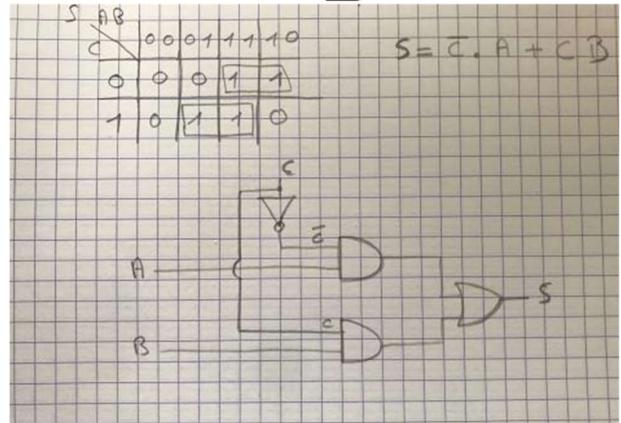
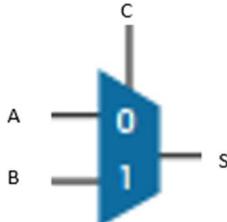
tableau de ka-raugh

S	CB	nCb	CnB	NCNB
A	1	1	0	1
Na	1	0	0	0

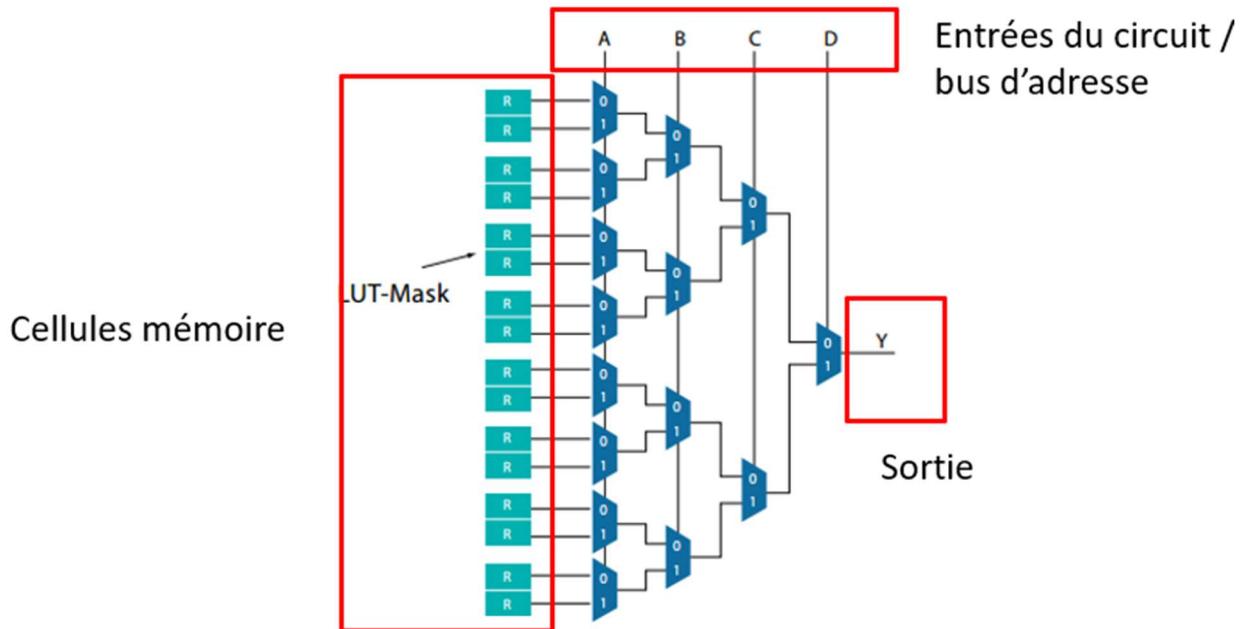
C	AB			
	.00	.01	.11	.10
0	0	0	0	1
1	0	1	1	0

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

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



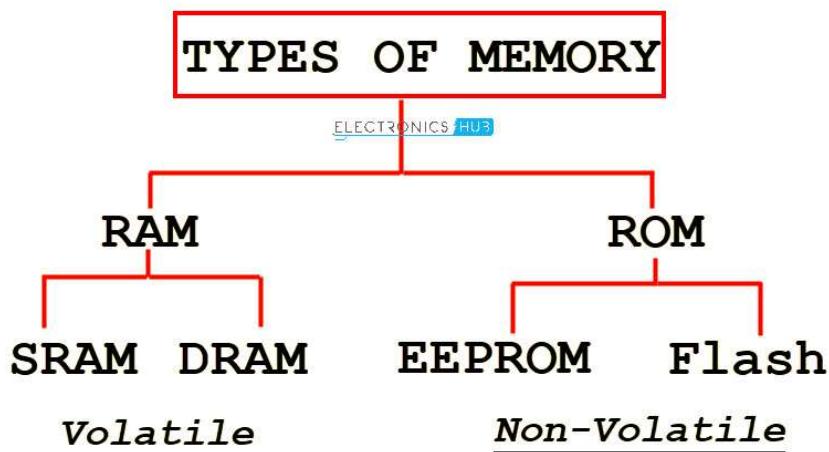
→ La LUT étant une mémoire, possède un bus d'adresse sur lequel on connecte nos entrées



2.3.4 Rappel sur les Mémoires

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



2.3.4.1 Exercice 10

Construire un full-substractor 4bits

- Construire la table de vérité
- Réaliser le schéma électrique
- Réaliser l'analyse de chemin critique
- Simuler le circuit sous LT-spice, prouver son fonctionnement

INPUT			OUTPUT	
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	0
1	1	0	0	0
1	1	1	1	1

On représente un chiffre négatif par le code suivant :

Pour 2 bits, $0x3 = 11 = -2 + 1 = -1$

$S = C_{in} + C_{in}(A \oplus B) + \bar{C}_{in}AB$

$Bout = C_{in} + C_{in}(A \oplus B) + \bar{C}_{in}AB$

$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}C_{in} + ABC$

$Bout = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}C_{in} + BCA$

par méthode

$S = C_{in}(\bar{A}\bar{B} + AB) + \bar{C}_{in}(\bar{A}B + A\bar{B})$

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

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

$Bout = \bar{A}B(C_{in} + C_{in}) + C_{in}(AB + A\bar{B})$

$C_{out} = \bar{A}B + C_{in}(A \oplus B)$

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

$C_{out} = C_{in} \cdot (A \oplus B) + \bar{A} \cdot B$

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

Sps.

2.4 Rappel sur la Logique numérique synchrone

Nous avons dans un circuit Combinatoire, les sorties dépendent directement des entrées. Et dans un circuit Séquentiel, les sorties dépendent des entrées, ainsi que d'un état.

Autrement dit, la notion de mémorisation apparaît avec la logique séquentielle.

On a par exemple des composants synchrones :

- La bascule (latch)
- Le registre, (flip flop)
- Les compteurs et registres à décalage
- Machines à état, (Finite State Machine)

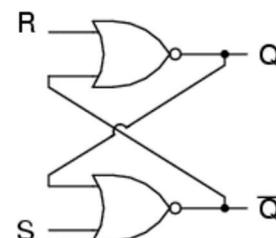
2.4.1 La bascule (le latch)

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

- ✓ La mise à 1 de S (Set) force la sortie à 1.
- ✓ La mise à 1 de R (Reset) force la sortie à 0.
- ✓ Si R et S sont à 0, on conserve en sortie la valeur précédente (mémorisation)
- ✓ La mise à 1 simultanée de R et S n'est pas possible.

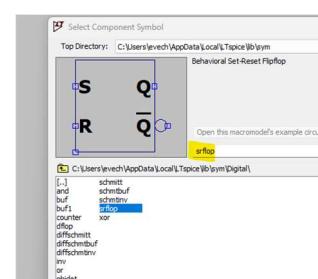
En logique séquentielle, il faut nécessairement considérer un **état initial** du fait de cette notion de mémorisation.

➔ La mémorisation résulte du rebouclage de la sortie sur les entrées des portes logiques.



Équation Logique : $Q_{\text{sortie}} = S + \overline{R} \cdot Q$

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



Symbol sur LT-Spice

2.4.1.1 Exercice 11

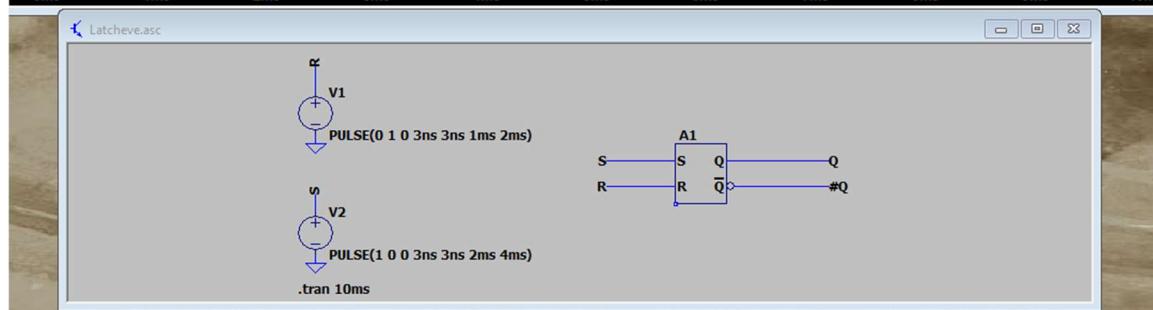
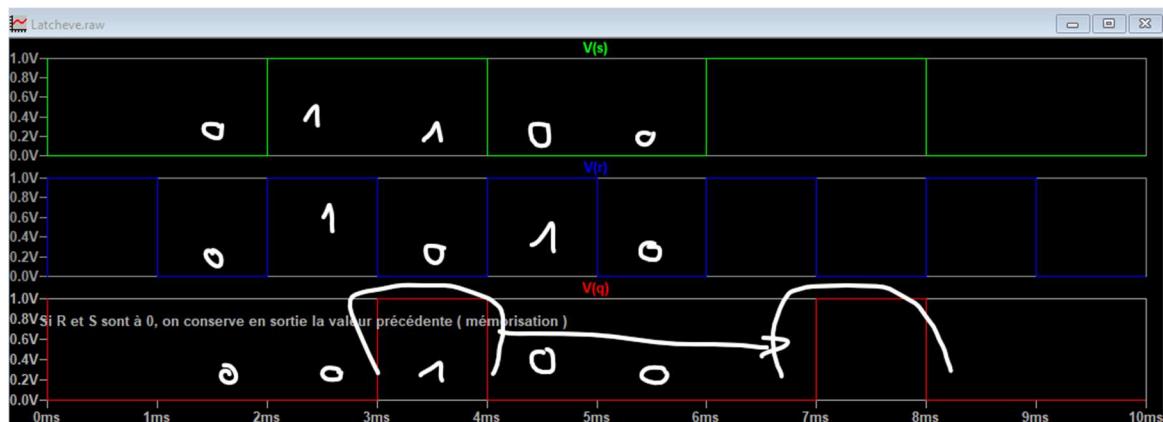
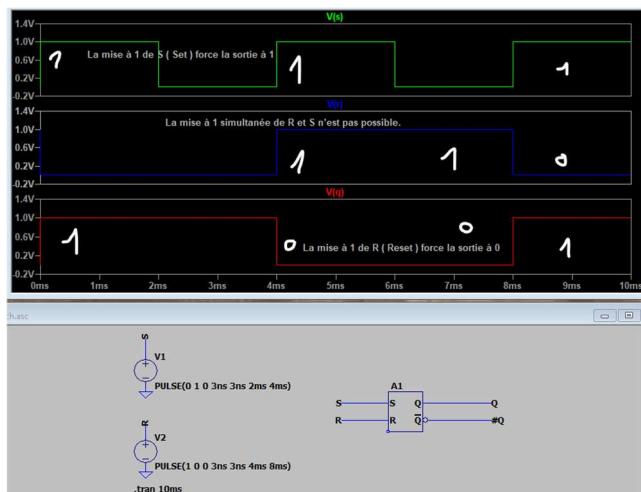
Sous Ltspice, utiliser le symbole de « sriflop » de la librairie digital

Réaliser les deux cas suivants :

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

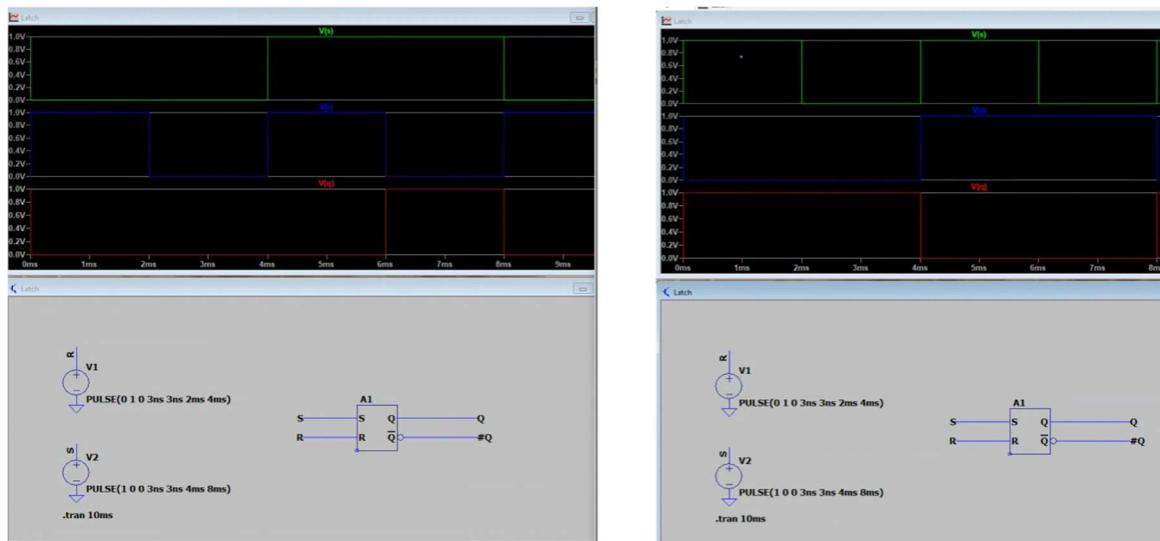
$S = 1$ et $R = 0$ puis $S = 0$ et $R = 0$

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



S	R	Q	Q	remarque
0	0	q	q	mémorisation
0	1	0	1	mise à 0
1	0	1	0	mise à 1
1	1	0	0	cas particulier= état « non légal »

→ La mémorisation résulte du rebouclage de la sortie sur les entrées des portes logiques : La valeur présente en sortie de la bascule est présente en entrée un coup d'horloge avant.



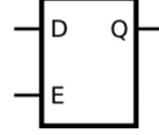
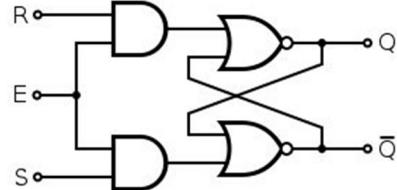
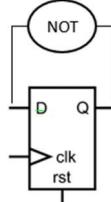
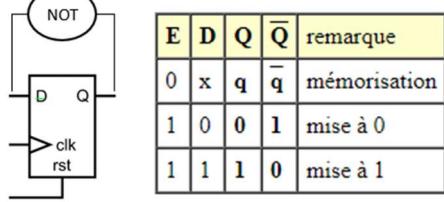
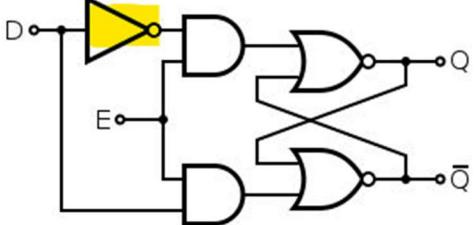
Conclusion :

Les composants synchrones sont des composants dont les valeurs évoluent uniquement sur les fronts montants d'une horloge.

Dans un système complexe mettant en parallèle plusieurs composants, cela permet de mettre tout le monde d'accord.

2.4.2 Les registres

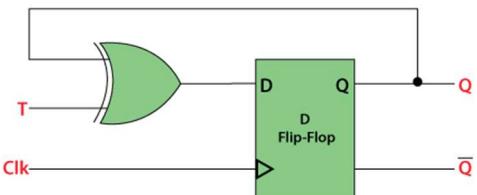
On appelle registre un ensemble de bascules avec une même commande d'horloge

<p>Un registre est basé sur le principe d'une bascule.</p> <ul style="list-style-type: none"> ✓ La bascule (latch) : Le latch se symbolise de la façon suivante et permet de réaliser la fonction de mémorisation (de 1bit). <p>1) Les registres (flip flop)</p> <p>Un registre est basé sur le principe d'une bascule</p> <ul style="list-style-type: none"> ✓ On ajoute un étage de portes AND ✓ On ajoute d'un bit de commande d'écriture : l'entrée Enable E (clk dans le schémas) <p>➔ Le Set ou le Reset ne sont possibles que lorsque Enable est à 1.</p>	  <p>Table de vérité :</p> <table border="1" data-bbox="865 774 1167 1021"> <thead> <tr> <th>E</th> <th>S</th> <th>R</th> <th>Q</th> <th>\bar{Q}</th> <th>remarque</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>x</td> <td>x</td> <td>q</td> <td>\bar{q}</td> <td>mémorisation</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>q</td> <td>\bar{q}</td> <td>mémorisation</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>0</td> <td>1</td> <td>mise à 0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>1</td> <td>0</td> <td>mise à 1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>0</td> <td>0</td> <td>cas particulier</td> </tr> </tbody> </table>	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
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																																
<p>a) Le registre D (flip flop)</p> <p>Si l'on veut mémoriser un état logique D, il faut dans le schéma ci-dessus que S=D et R=not D ; c'est ce qui est réalisé dans le circuit suivant :</p> <p>le registre va alterné 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.</p> <p>➔ L'écriture de D se fait lorsque Enable est à 1 : nous avons enfin une cellule mémoire avec commande d'écriture</p> <p>A noter que : Cette bascule peut être utilisée comme registre à décalage pour les conversions de transmission parallèle/série ; ou encore comme compteur en rebouclant Q sur D.</p>	  <p>Table de vérité :</p> <table border="1" data-bbox="889 1325 1365 1482"> <thead> <tr> <th>D</th> <th>CLK</th> <th>Q</th> <th>\bar{Q}</th> <th>remarque</th> </tr> </thead> <tbody> <tr> <td>d</td> <td>↗</td> <td>d</td> <td>\bar{d}</td> <td>Q recopie D</td> </tr> <tr> <td>X</td> <td>1,0,front descendant</td> <td>q</td> <td>\bar{q}</td> <td>mémorisation</td> </tr> </tbody> </table> 	D	CLK	Q	\bar{Q}	remarque	d	↗	d	\bar{d}	Q recopie D	X	1,0,front descendant	q	\bar{q}	mémorisation																					
D	CLK	Q	\bar{Q}	remarque																																	
d	↗	d	\bar{d}	Q recopie D																																	
X	1,0,front descendant	q	\bar{q}	mémorisation																																	

b) **Le registre T (flip flop)**

- Il s'agit d'une variante de la D-FlipFlop

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



La table de vérité associée :

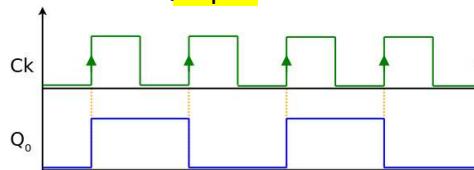
CLK	T	Q_{n+1}
↑	0	Q_n
↑	1	Q_n'

2.4.2.1 Différence entre les bascules D et T

D FLIP-FLOP

La sortie de la bascule suit l'entrée avec un retard d'une impulsion d'horloge.

Ici → on a Q copie D

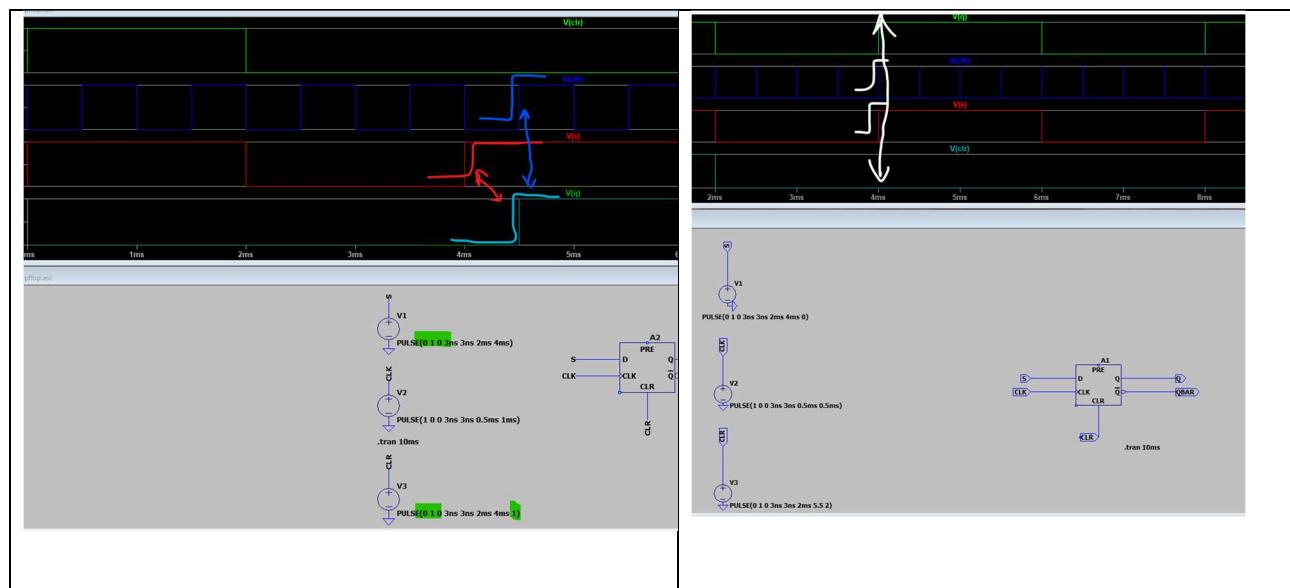


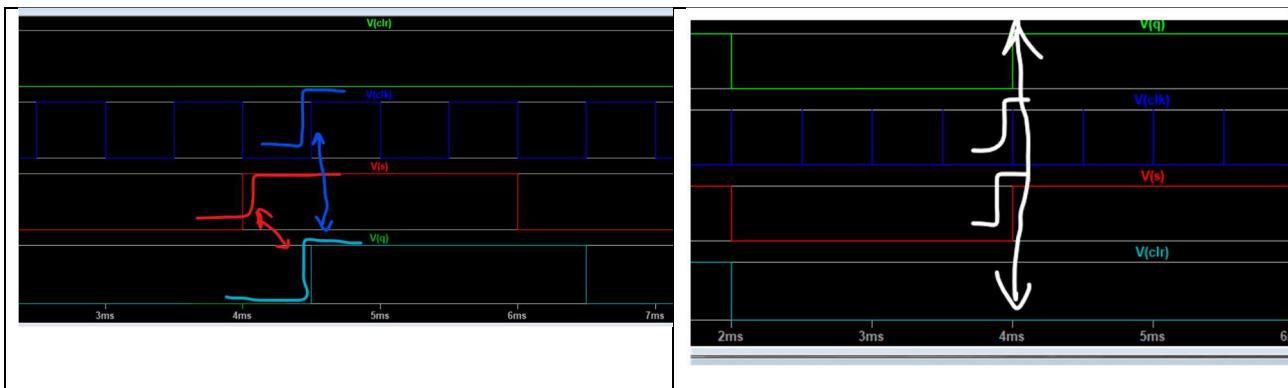
T FLIP-FLOP

Si $T_i=1 \rightarrow$ on a $Q_i = \text{not}(Q_{i-1})$
Si $T_i=0 \rightarrow$ on a $Q_i = Q_{i-1}$

2.4.2.2 Exercice 12

Reprendre les cas précédemment étudiés avec le composant sriflop cette fois ci avec le composant « dflop »





2.4.3 Les Compteurs :

En électronique, selon la définition très générale de la norme IEC 6050, un compteur est un circuit dont une mémoire contient un nombre, auquel il ajoute un nombre entier relatif constant lors d'un événement logique à son entrée. Dans le cas simple et courant, le compteur est un circuit intégré numérique, dont la mémoire contient un nombre entier auquel il ajoute ou soustrait 1 à chaque impulsion appliquée à son entrée. Il est composé d'un certain nombre de bascules D, T ou autre...

(source : <https://fr.wikipedia.org/wiki/Compteur>)

2.4.3.1 Compteur binaire

Le compteur le plus simple est obtenu en mettant en cascade une série de bascules, le signal à compter étant appliqué à l'entrée de la première bascule ; la sortie de cette bascule pilote l'entrée de la deuxième bascule et ainsi de suite. Le résultat du comptage apparaît sous forme de nombre binaire, la première bascule indiquant le chiffre le moins significatif.

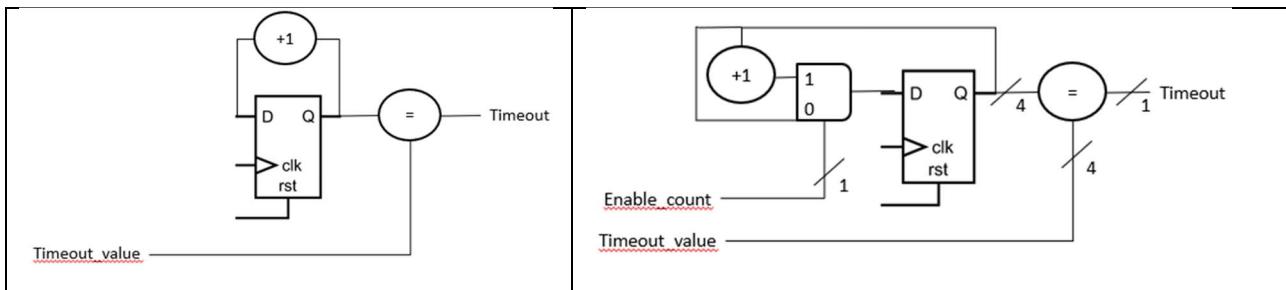
Exemple : 01010111 indique 87.

A retenir :

Les compteurs où l'horloge est appliquée simultanément à toutes les bascules sont appelés compteurs **synchrones** (synchronous counters), les autres, compteurs **asynchrones** (asynchronous ou ripple counters).

Exemple d'un compteur : **watchdogs** qui est un compteur qui permet d'interrompre un processus lorsque ce dernier ne répond pas au bout d'un certain temps.

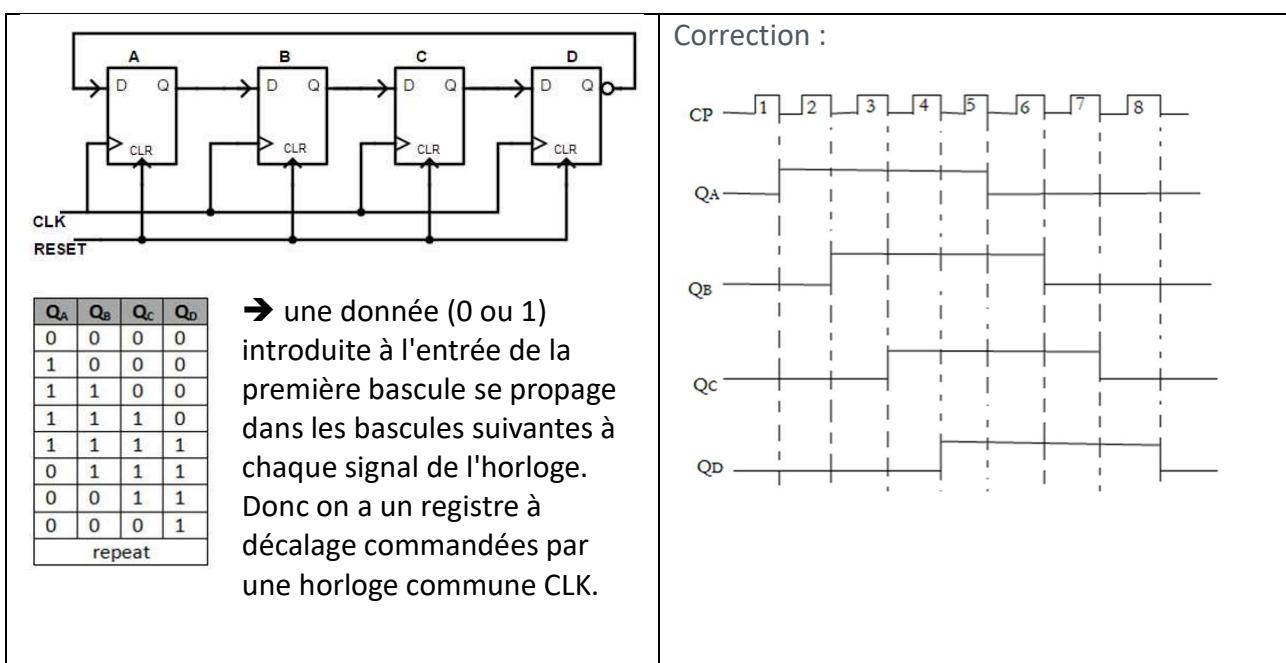
Si on souhaite observer lorsque notre compteur, arrive à une valeur donnée on va utiliser un comparateur à la sortie. Si les deux valeurs d'entrées sont égales, alors la sortie est à 1 sinon elle est à 0. ➔ il s'agit d'un timer ou watchdog



2.4.3.2 Exercice 13

Compteur 4bit, schéma RTL et chronogramme

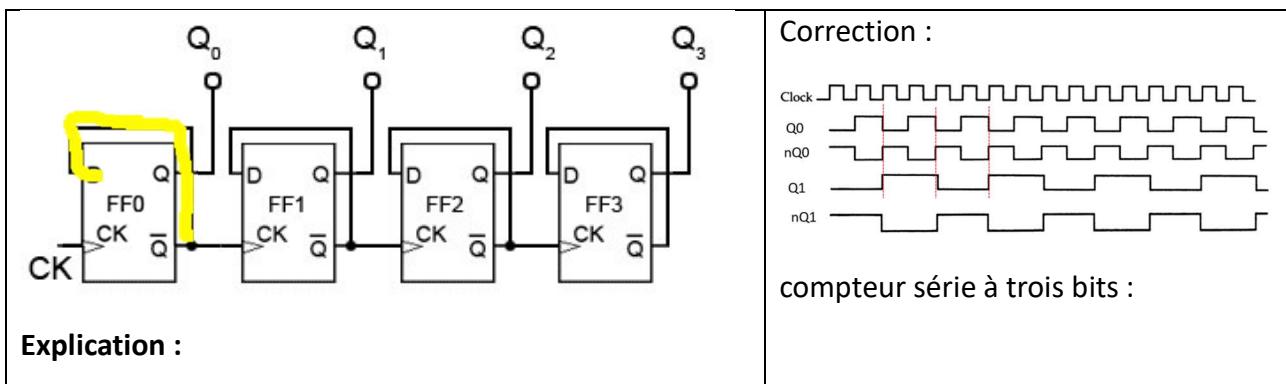
→ Réaliser le chronogramme de ce circuit, on suppose que les registres sont initialisés à zéro.



2.4.3.3 Exercice 14

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

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



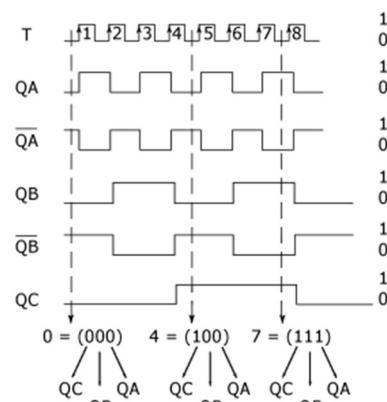
à l'état initial, les sorties des bascules Q0, Q1 et Q2, Q3 sont toutes à l'état logique 0.

Ainsi, les sorties complémentaires des bascules (not Q0 , not Q1, not Q2, not Q3) sont à l'état logique 1.

Par conséquent, les entrées des bascules D0,D1, D2 et D3 sont à l'état logique 1.

Comme la bascule "D" ne réagit que sous le signal de l'horloge, les sorties restent inchangées jusqu'à l'arrivée de ce signal. A cette condition initiale correspond le code binaire (Q0 Q1 Q2 Q3) = (0000), donc 0 décimal

- ✓ C'est un compteur série fournit des valeurs **décimales codées en binaires**



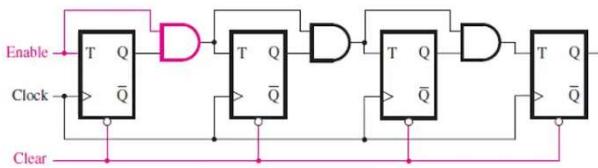
A noter : Le nombre de bascules "D" utilisées dans la conception du compteur série détermine le nombre maximal de comptage :

Ce nombre est égal à '2' exposants le nombre de bascules utilisées, avec trois bascules, le nombre maximal des chiffres comptés à la sortie du compteur est 2 exposant 3 soit 8 (0 à 7 décimal).

2.4.3.4 Exercice 15

Réaliser le chronogramme de ce circuit pour Q0..Q3 . On suppose que les registres sont initialisés à zéro. On suppose l'entrée « enable" à 1

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

**Explication :**

On va se baser sur l'idée que nous allons prendre une photo à chaque front montant de l'horlage : donc il y a des bits qui seront activés à 1 (dans ce cas) et d'autres gardent leur valeur précédente i-1

Init t = 0

$Q_0 = 0$,

$Q_1 = 0$,

$Q_2 = 0$,

$Q_3 = 0$

$T_1 = 1$ er front montant de clock

⇒ Snapshot des entrées

$T_0 = 1$

$T_1 = Q_0 \& \text{Enable} = 0 \& 1 = 0$

$T_2 = Q_1 \& T_1 = 0 \& 0 = 0$

$T_3 = Q_2 \& T_2 = 0 \& 0 = 0$

$T_0 = 1$ donc $Q_0 = \text{not } Q_0$ précédent = $\text{not } 0 = 1$

$T_1 = 0$ donc $Q_1 = Q_1$ précédent = 0

$T_2 = 0$ donc $Q_2 = Q_2$ précédent = 0

$T_3 = 0$ donc $Q_3 = Q_3$ précédent = 0

$T_2 = 2$ ième front montant de clock

⇒ Snapshot des entrées

$T_0 = 1$

$T_1 = Q_0 \& \text{Enable} = 1 \& 1 = 1$

$T_2 = Q_1 \& T_1 = 0 \& 1 = 0$

$T_3 = Q_2 \& T_2 = 0 \& 0 = 0$

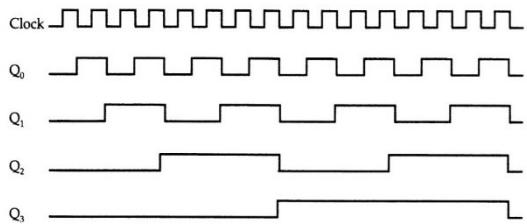
$T_0 = 1$ donc $Q_0 = \text{not } Q_0$ précédent = $\text{not } 1 = 0$

$T_1 = 1$ donc $Q_1 = \text{not } Q_1$ précédent = $\text{not } 0 = 1$

$T_2 = 0$ donc $Q_2 = Q_2$ précédent = 0

$T_3 = 0$ donc $Q_3 = Q_3$ précédent = 0

Sortie des Bascules				Entrée des Bascules			
Q3	Q2	Q1	Q0	T3	T2	T1	T0
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	1
0	0	1	0	0	0	0	1
0	0	1	1	0	1	1	1
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	1
0	1	1	0	0	0	0	1
0	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1
1	0	0	1	0	0	1	1
1	0	1	1	0	1	1	1
1	1	0	0	0	0	0	1
1	1	0	1	0	0	1	1
1	1	1	0	0	0	0	1
1	1	1	1	1	1	1	1

Correction :

$T_3 = 3$ ième front montant de clock

⇒ Snapshot des entrées

$T_0 = 1$

$T_1 = Q_0 \& \text{Enable} = 0 \& 1 = 0$

$T_2 = Q_1 \& T_1 = 0 \& 0 = 0$

$T_3 = Q_2 \& T_2 = 0 \& 0 = 0$

$T_0 = 1$ donc $Q_0 = \text{not } Q_0$ précédent = $\text{not } 0 = 1$

$T_1 = 0$ donc $Q_1 = Q_1$ précédent = 1

$T_2 = 0$ donc $Q_2 = Q_2$ précédent = 0

$T_3 = 0$ donc $Q_3 = Q_3$ précédent = 0

⇒ Snapshot des entrées

$T_0 = 1$

$T_1 = Q_0 \& \text{Enable} = 1 \& 1 = 1$

$T_2 = Q_1 \& T_1 = 1 \& 1 = 1$

$T_3 = Q_2 \& T_2 = 0 \& 1 = 0$

On retrouve :

Si $T_i=1 \rightarrow$ on a $Q_i = \text{not}(Q_{i-1})$

Si $T_i=0 \rightarrow$ on a $Q_i = Q_{i-1}$

Le registre T (flip flop)

2.4.4 Les Machines d'états

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.

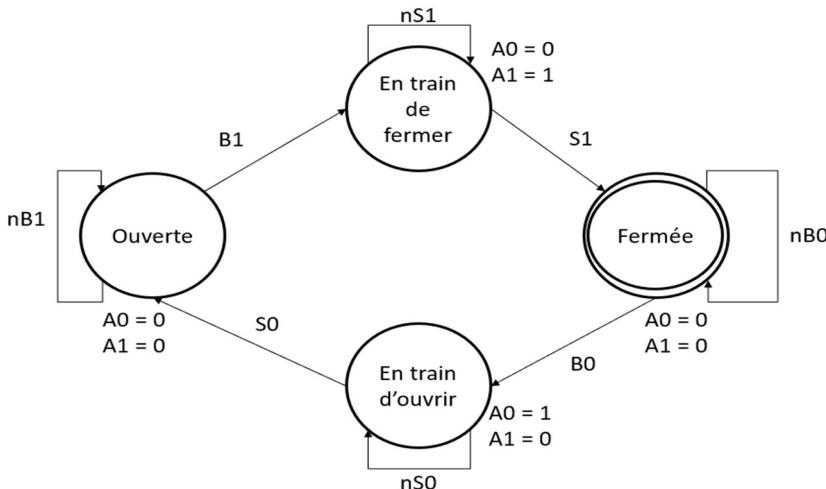
- Il existe deux types de machine à état, les machines de Mealy et les machines de moore
 - FSM Moore : l'état suivant dépend uniquement de l'état courant
 - FSM Mealy : l'état suivant dépend de l'état courant et aussi de signaux d'entrées

2.4.4.1 Exercice 16

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.

2.4.4.2 Exercice 17

Cahier des charges :

Considérons un système de verrouillage à code. Le système doit avoir quatre boutons, chacun étiqueté de 0 à 9, pour entrer le code de verrouillage.

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

<p>Un jeu d'état possible pour ce problème est</p> <ul style="list-style-type: none"> • 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 	<p>*O = 0 Ec = 0 (Ec \Leftrightarrow Enable counter)</p> <ul style="list-style-type: none"> - 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 24bits, ce dernier devra compter 10 000 000 cycles d'horloges (10 secondes d'attente); nous utiliserons un comparateur pour déterminer la fin de comptage.
--	---

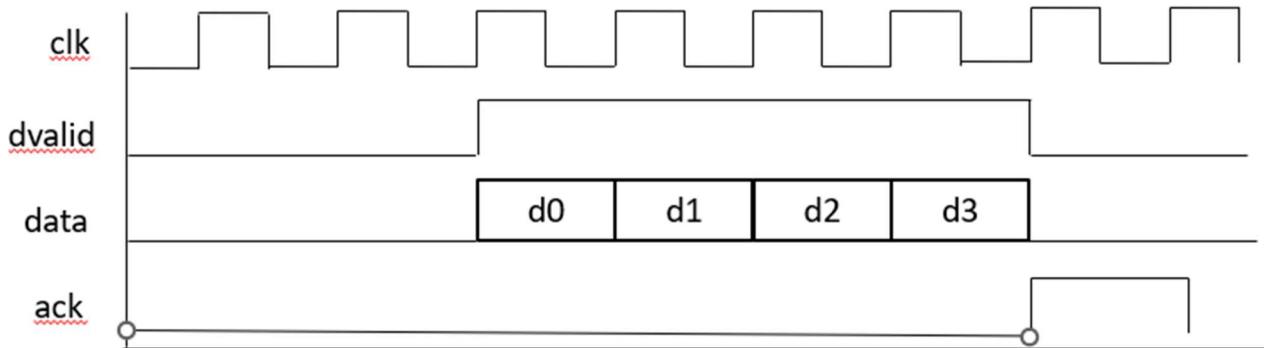
2.4.4.3 Exercice 18

Lors de la transmission, un mot de 4 bit est transmission sur la ligne data. Une horloge est générée telle que sur le chronogramme.

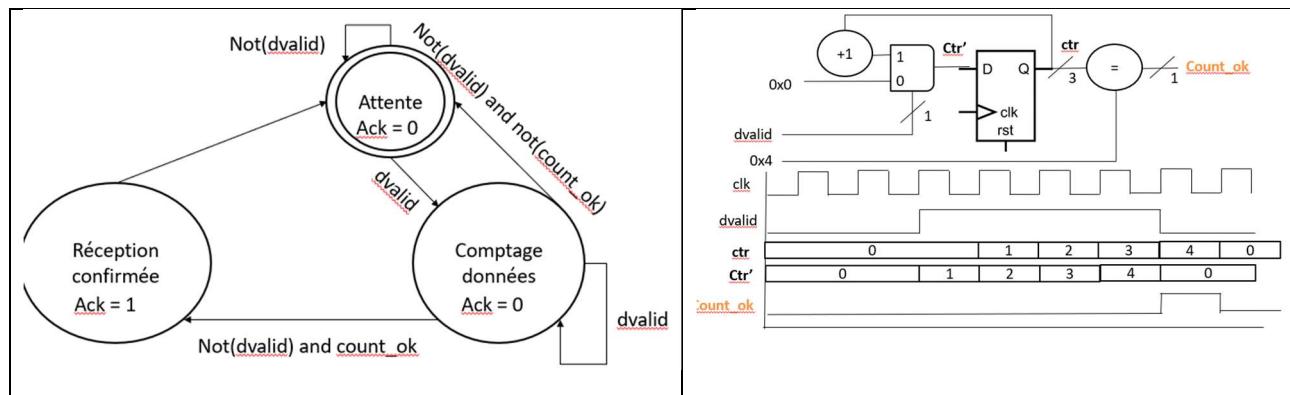
Une pulsation ack est générée afin de signifier que la transmission du mot est complète.

⇒ Réaliser un design FPGA qui permet la réception des données de ce protocole.

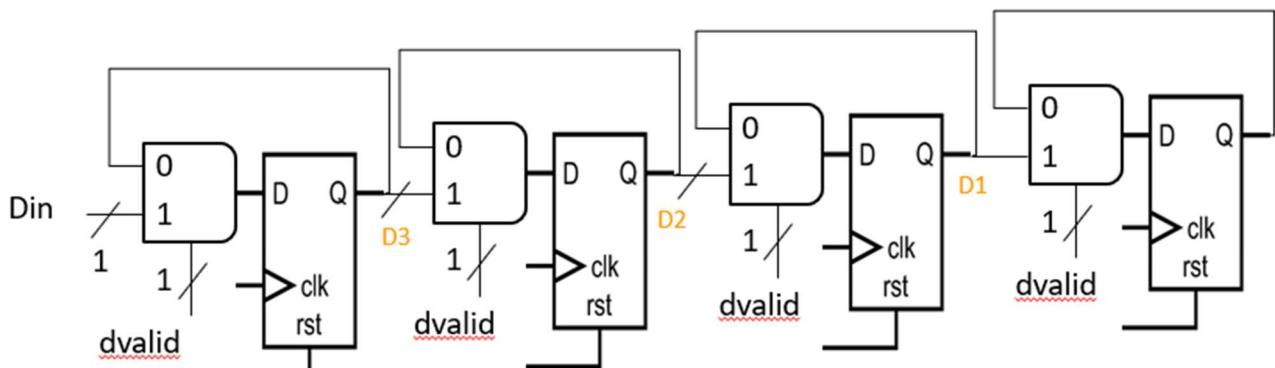
⇒ Déterminer la machine à état du système ainsi que les parties opératives.



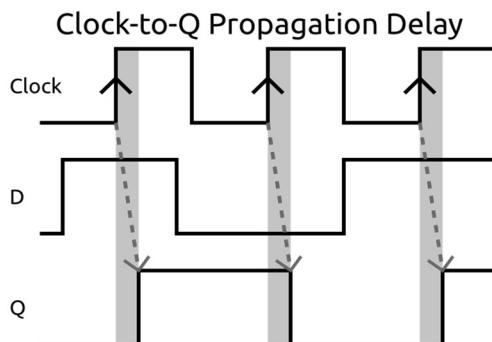
Correction :



Dans ce cas un **registre à décalage** permet de récupérer la donnée : c'est un compteur constitué de bascules telles que la sortie de l'une est reliée à l'entrée de la bascule suivante



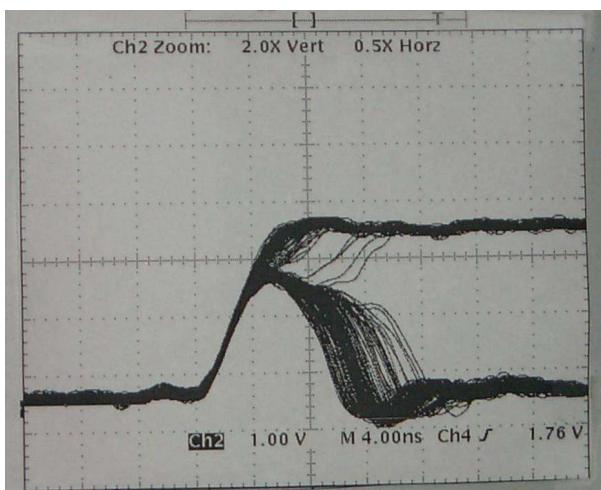
2.4.5 Gestion de timing



C'est une problématique courante en micro-électronique.
On résoud 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**.

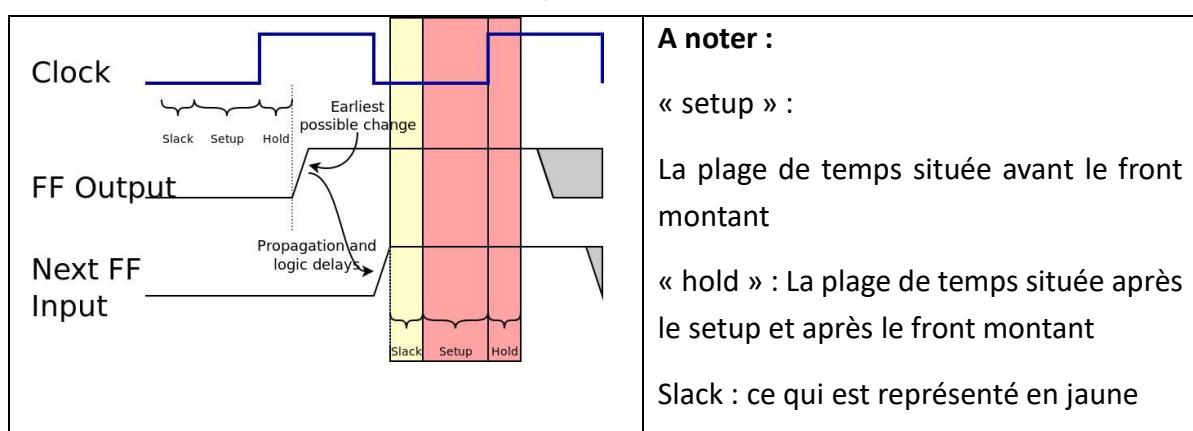
La conséquence de cela est un signal non stable dans le registre de sortie. On dit qu'il est **métastable**.



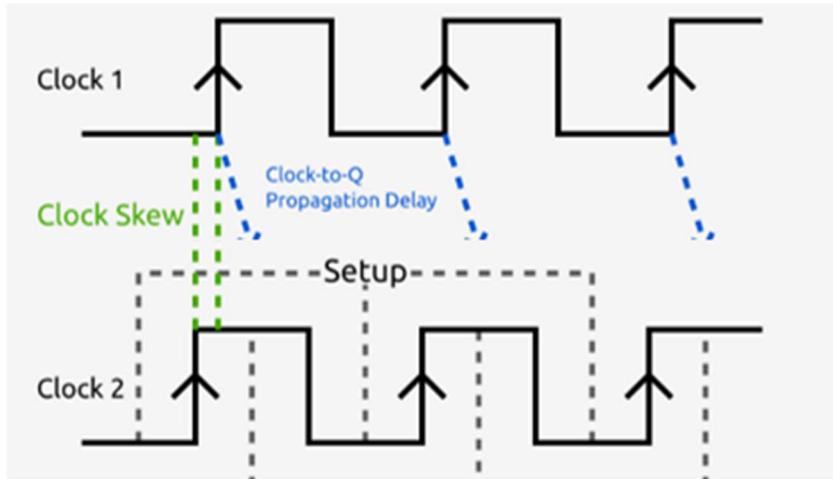
2.4.5.1 Exemple de Métastabilité

La métastabilité fait référence à l'état où la sortie n'est pas déterministe. Cela peut provoquer des oscillations, des transitions peu claires dans les circuits. Par exemple, flip Flop fait face au problème de la métastabilité ; cela arrive à une bascule lorsque l'impulsion d'horloge et les données changent au même moment, ce qui entraîne un comportement imprévisible du résultat.

2.4.5.2 Notions de Gestion de timing



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



→ 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)

A noter :

- 1) 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.
- 2) Voici 3 facteurs qui peuvent augmenter 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) → speed grade : le code de vitesse de la puce (1 indique la vitesse la plus élevée).

Device	Speed Grade and Junction Temperature Range		
	Commercial (C) 0°C to +85°C	Extended (E) 0°C to +100°C	Industrial (I) -40°C to +100°C
XC7Z007S			
XC7Z012S	-1	-2	-1, -2
XC7Z014S			
XC7Z010			
XC7Z015	-1	-2, -3	-1, -2, -1L
XC7Z020			
XC7Z030			
XC7Z035	-1	-2, -3	-1, -2, -2L
XC7Z045			
XC7Z100	-1	-2	-1, -2, -2L

2.4.5.3 Jargon à retenir :

La bascule (le latch) : permet de réaliser la fonction de mémorisation (de 1bit)

Un registre : On appelle registre un ensemble de bascules avec une même commande d'horloge

Un registre à décalage (c'est un compteur) : est constitué de bascules telles que la sortie de l'une est reliée à l'entrée de la bascule suivante

Le registre D (flip flop) : Cette bascule peut être utilisée comme registre à décalage pour les conversions de transmission parallèle/série ; ou encore comme compteur en rebouclant Q sur D.

Métastabilité : La métastabilité fait référence à l'état où la sortie n'est pas déterministe. Cela peut provoquer des oscillations, des transitions peu claires dans les circuits. Par exemple, flip Flop fait face au problème de la métastabilité ; cela arrive à une bascule lorsque l'impulsion d'horloge et les données changent au même moment, ce qui entraîne un comportement imprévisible du résultat.

compteurs synchrones : Les compteurs où l'horloge est appliquée simultanément à toutes les bascules

watchdogs : c'est un compteur qui permet d'interrompre un processus lorsque ce dernier ne répond pas au bout d'un certain temps

Les compteurs : peuvent être classifiés suivant plusieurs groupes :

- ⇒ selon leur comptage maximal,
- ⇒ de leur mode d'opération (synchrone ou asynchrone),
- ⇒ et de leur mode de fonctionnement (permanent ou à arrêt automatique).

Compteur série : Le compteur série est construit par la mise en cascade de plusieurs bascules "D". Le nombre de ces bascules détermine le nombre maximal que le compteur est capable de compter. Le compteur série fournit des valeurs décimales codées en binaires. Chaque bit de ces codes binaires correspond à la sortie d'une bascule "D".

Compteur parallèle : Un compteur parallèle est un compteur synchrone qui compte de telle façon que la valeur binaire des bits de codage croît ou décroît d'une unité à la fois.

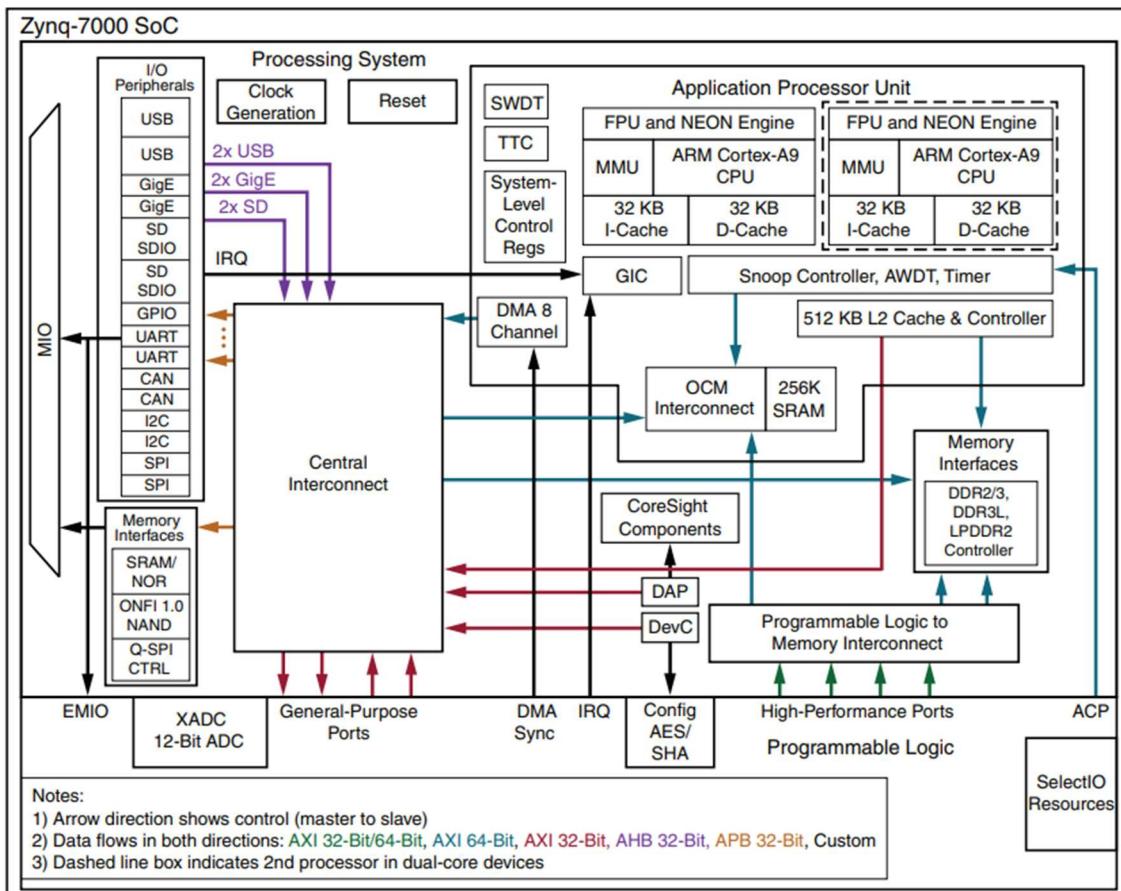
« setup » :

La plage de temps située avant le front montant

« hold » : La plage de temps située après le setup et après le front montant

Le clock skew : Cela se produit lorsque le signal d'horloge n'arrive pas en même temps aux registres
l'arbre d'horloge (clock tree) : Pour limiter le clock skew, les horloges sont routées sur un réseau particulier spécialement conçu pour de la faible latence,

3 ANNEXE



Architectural Overview

Xc7z010-1CLG400C

