

Note : 1er cours

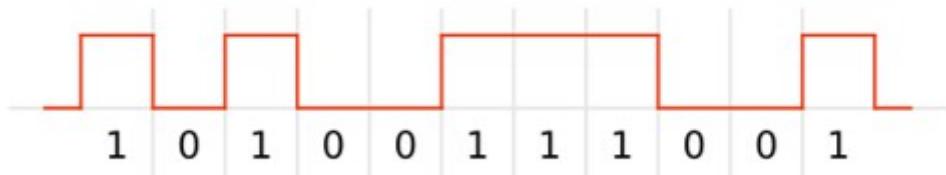
Un FPGA va se comporter comme un petit CPU. On va fonctionner avec les différentes bases : binaire, décimale et hexadécimale.

Rappel des différentes bases de calcul en informatique

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

Td (à la fin du premier cours)

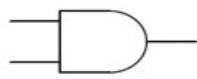
À l'aide de ces chiffres binaires, on va obtenir des signaux (le chronogramme).



Pour réaliser les opérations de calculs on va utiliser des opérateurs booléen.

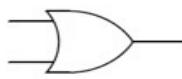
On a le AND, OR, XOR (ou exclusif), NAND, NOR et XNOR. La table de vérité définit le comportement de ces portes.

Voici quelques rappel de portes logiques :



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



XNOR

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

RTL(Register Transfer Langage) : Niveau le plus bas de calcul

TD1

Exercice 1

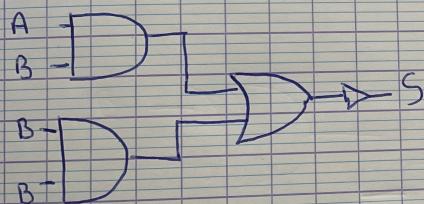
Base 2	Base 16
$18 = 0001\ 001$	$0x12$
$40 = 1010\ 0000$	$0x28$
$64 = 0100\ 0000$	$0x40$

Exercice 2

$$A = 0 \text{ et } B = 1$$

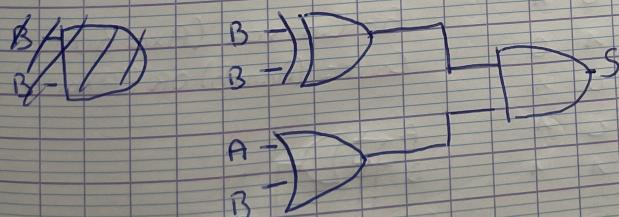
$$- \text{NOT}((A \text{ and } B) \text{ OR } (B \text{ and } B))$$

$$X = A \text{ and } B = 0 \quad Y = B \text{ and } B = 1 \quad X \text{ or } Y = 1 \quad S = \text{Not}(X \text{ or } Y) = 0$$



$$- (B \text{ XOR } B) \text{ And } (A \text{ OR } B)$$

$$X = B \text{ XOR } B = 0 \quad Y = A \text{ OR } B = 1 \quad S = X \text{ and } Y = 0$$

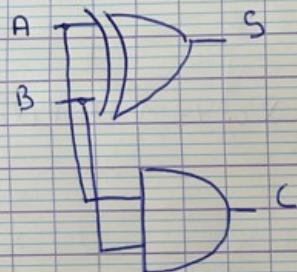


Exercice 3

Table de vérité

A	B	base 10	base 2	C (retenue)	S
0	0	0	00	0	0
1	0	1	01	0	1
0	1	1	01	0	1
1	1	2	10	1	0

$$C = A \text{ and } B \quad \text{et} \quad S = A \text{ xor } B$$



Exercice 4

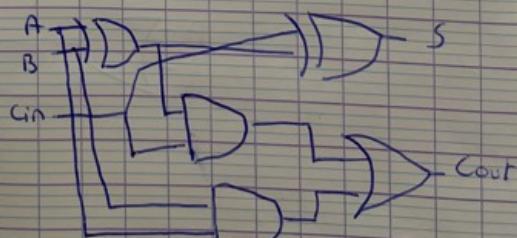
Table de vérité

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

En repartant de l'exercice précédent, je détermine les équations suivantes:

$$S = \text{Cin xor } (\text{A xor B})$$

$$\text{Cout} = (\text{A and B}) \text{ OR } ((\text{A xor B}) \text{ and Cin})$$



Un additionneur 3 bits peut se résumer à une cascade d'additionneur. La même chose pour un additionneur 4 Bits on va mettre des full Ladder en cascade.

Exercice 4

Pour la réalisation un additionneur 4 bits, il suffit de mettre plusieurs additionneur en cascade (4 en l'occurrence)

Note Cours 2

Explication de l'additionneur 4Bits :

Il faut prendre toute les équations correspondantes à S=1 et simplifier l'algèbre booléen.

On utilise une écriture plus synthétique afin de voir les choses.

Tableau récapitulatif :

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

Il existe également des simplifications :

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

D'après les calculs réalisés sur excel $\text{XOR} = \text{A} \oplus \text{B}$

Autre cas en diagonale

$$\begin{array}{c} x \quad \bar{a} \quad a \\ \bar{b} \quad 0 \quad 1 \\ b \quad 1 \quad 0 \end{array} \quad \begin{array}{l} \text{On a alors } x = A \oplus B \\ \text{car on a } x = a \cdot \bar{b} + \bar{a} \cdot b \text{ (forme déjà vu)} \end{array}$$

Pour plusieurs entrées mettre le dernier en avant dernier
c'est à dire 11 avant 10
On aurait 00 01 1110

$$\begin{array}{cccc} A & \cancel{B \text{in}} & \bar{B} \text{in} & \bar{B} \text{cin} \\ \cancel{A} & 0 & 1 & 0 \\ A & 1 & 0 & \cancel{1} \\ & & & 0 \end{array} \quad \begin{array}{l} \text{On a:} \\ S = \bar{B} \cdot \bar{C}_{\text{in}} \cdot A + \bar{B} \cdot \text{Cin} \cdot \bar{A} + B \cdot \bar{C}_{\text{in}} \cdot \bar{A} \\ S = \bar{B} \cdot (\bar{C}_{\text{in}} \cdot A + \text{Cin} \cdot \bar{A}) + B \cdot \bar{C}_{\text{in}} \cdot \bar{A} \\ S = \bar{B} \cdot \text{Cin} \oplus A + B \cdot \bar{C}_{\text{in}} \cdot \bar{A} \end{array}$$

$$S = A \cdot \bar{B} \cdot \bar{C} \oplus \bar{A} \cdot \bar{B} \cdot C \oplus A \cdot B \cdot C \oplus \bar{A} \cdot B \cdot \bar{C} = \text{Cin} \oplus A \oplus B$$

Pour Cout

$$\begin{array}{ccccc} A & \cancel{B \text{in}} & \bar{B} \cdot \bar{C}_{\text{in}} & \bar{B} \text{cin} & B \cdot \bar{C}_{\text{in}} \\ \cancel{A} & 0 & 0 & 1 & 0 \\ A & 0 & \boxed{1} & 1 & 1 \end{array}$$

$$\begin{aligned} \text{Cout} &= AB + BC \oplus CA \\ &= B \cdot \text{Cin} + A \cdot \bar{B} \cdot C + A \cdot B \\ &= B \cdot C + A(\bar{B} \cdot C + BC) \\ &= B \cdot C + A(B \oplus C) \\ &= B \cdot C + AB \oplus AC \end{aligned}$$

Développement pour

$$\begin{aligned} S &= A \cdot \bar{B} \cdot \bar{C} \oplus \bar{A} \cdot \bar{B} \cdot C \oplus A \cdot B \cdot C \oplus \bar{A} \cdot B \cdot \bar{C} \\ &\quad - \cancel{A(\bar{B} \cdot \bar{C} \oplus BC)} \oplus \bar{A}(\bar{B} \cdot \bar{C} \oplus B \cdot \bar{C}) \\ &= C \cdot (\bar{A} \cdot \bar{B} \oplus A \cdot B) \text{ XOR } \bar{C} \cdot (\bar{A} \cdot \bar{B} \oplus \bar{A} \cdot B) \\ &= C \cdot (A \text{ Xnor } B) \oplus \bar{C} \cdot (A \oplus B) \\ &= C \oplus A \oplus B \end{aligned} \quad \text{exo précédent}$$

à retenir :

$$/A \cdot B + A \cdot /B = AxorB$$

$$/A \cdot /B + A \cdot B = /(AxorB)$$

$$/C(X) + C(/X) = CxorX$$

$$/a + /b = /(a \cdot b)$$

$$/A \cdot /B \text{ xor } A \cdot B = C \text{ xor } A \text{ xor } B$$

Rappel pour la carte

Les composants d'une carte :

-PU (process Unit)

-Memory

Ces deux unités sont le plus proche possible pour que le signal soit le plus intégrer (moins corrompu). Cela permet également d'augmenter la cadence de lecture et d'écriture.

Il existe des cartes dans lesquels on va graver directement la mémoire dans la puce.

Un CPU exécute les instructions de manière séquentiel alors qu'un GPU traite plusieurs instructions d'un coup.

-Gestion d'interface

-Gestion d'alimentation du circuit

-Shipset : PU ou mémoire

Arduino et Apple par exemple mettent la mémoire et le PU dans le même composants.

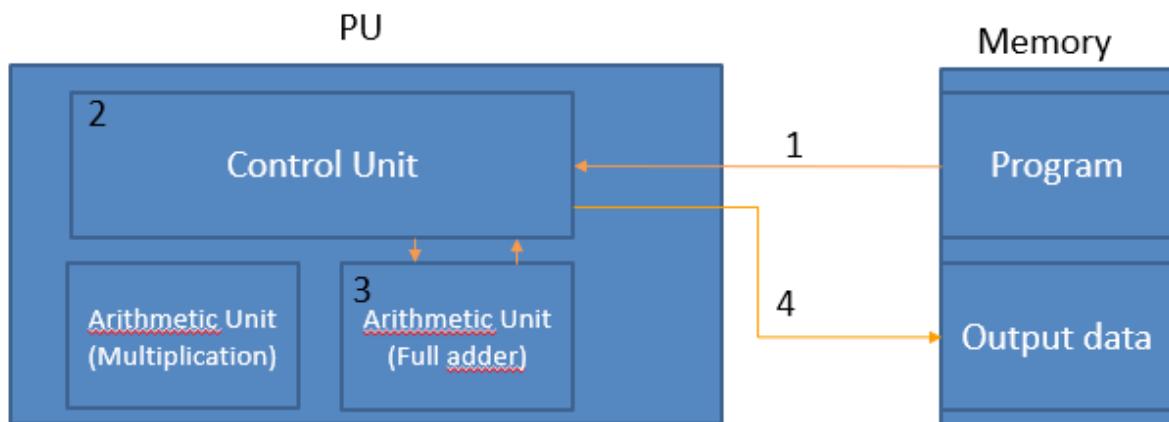
FPGA : PU + Mémoire reconfigurable au niveau matériel.

L'analyse de chemin critique permet de savoir si on va utiliser du synchrone ou du synchrone (avec horloge ou non)

Bram Mémoire volatile, elle perd sa mémoire quand on coupe le courant. C'est une très petite mémoire 10kbit (pas fait pour stockage de masse mais pour données temporaires). Très efficace pour accéder au données critique sur les Bram.

Un Pu peut être composer de GPU, CPU, unité de calcul...

Le G de GPU= Graphical pour le traitement 2D que ce soit de l'image ou des données (exemple : Tableau, données financières...)



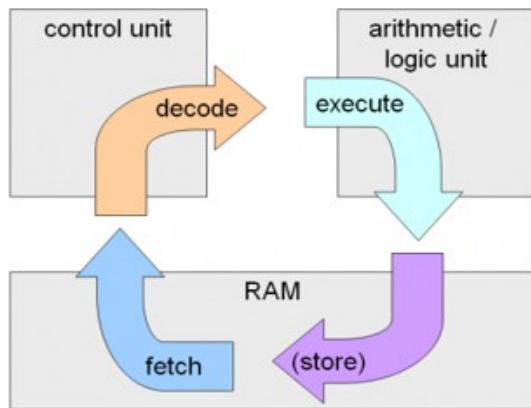
L'exécution des programme se fait dans la mémoire non volatile.

Le programme va dire au control unit dans la Pu quoi faire (Transfert physique mémoire vers PU). Ensuite la communication va se faire dans les deux sens entre notre control Unit et nos unités arithmétiques. Cette information qui va remonter va être stocker dans la mémoire.

On suit un cycle : Fetch (on récupère l'info), décode (on l'a décode), exécute et on store.

Instructions enregistrées = programmation

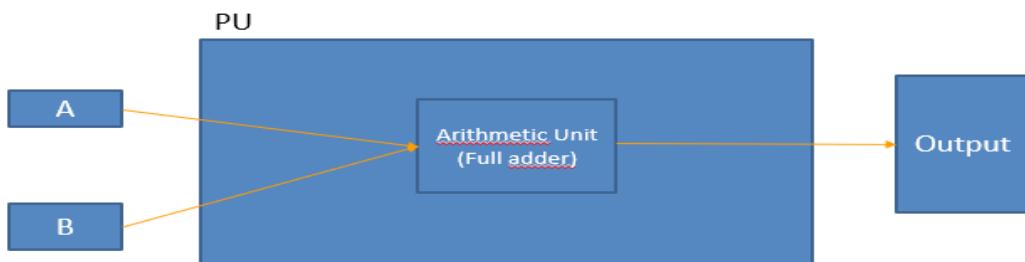
Instructions exécutées = L'inférence



C'est un mode d'exécution versatile mais qui prend beaucoup de temps. Cela nécessite une certaine énergie.

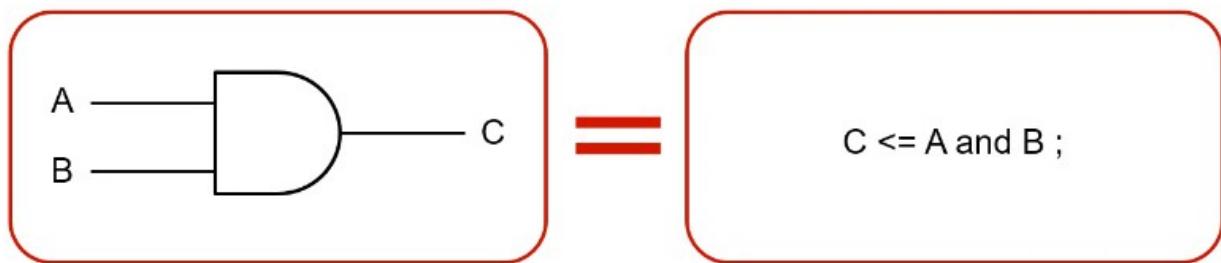
Pour lire des valeurs il faut utiliser des GPIO.

Le FPGA peut permettre de faire des calculs pour décharger la charge de travail du CPU et permet de consommer moins d'énergie.



Cours 3

Le langage de description matériel est un langage d'assez bas niveau. 3 types de langage : HDL, verilog et system verilog. Ils vont décrire l'architecture d'un point de vu numérique.



VHDL : Couramment utilisé, simple pour la simulation

Verilog : plus bas niveau mais plus complexe que le VHDL pour le design.

System Verilog : extension du verilog.

On va choisir son langage par affinité en fonction de ce que l'on a l'habitude d'utiliser.

Référence de notre carte : xc7z010clg400-1 (active)

Sur le logiciel Top = Hiérarchie

Contrainte : permet d'associer les éléments aux éléments de la carte

La synthèse permet de traduire ce qu'on écrit en code en architecture utilisable sur carte.

.v : Verilog

.vhdl:Vhdl

BitStream : permet d'envoyer le programme sur la carte

On doit créer une entity avec ces entrées et sorties ensuite dans l'architecture, on va décrire notre programme et nos signaux internes.

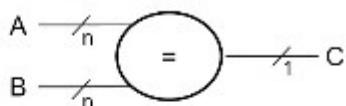
```
entity Example is
    -- Déclaration des entrées et des sorties
    Port (
        input_port      : in  std_logic;
        output_port     : in  std_logic
    );
end Example;
```

```
-- Définition de l'architecture
architecture Behavioral of Example is
    -- Déclaration d'un signal interne
    signal intern_signal : std_logic;

begin
    output_port <= input_port and intern_signal ;
end Behavioral;
```

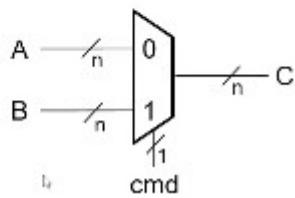
Dans les tests de conditions des schéma RTL, le petit n représente le nombre de bit.
Il y a différents types de test : les conditions ($=, >, <$), les multiplexeurs

- Les conditions ($=, >, <$)



**Si $A = B$ alors $C = 1$,
sinon $C = 0$**

- Les multiplexeurs



Si $cmd = 0$ alors $C = A$, sinon $C = B$

La logique séquentielle nécessite un signal d'horloge (temporisation), le résultat va dépendre du résultat précédent (ex : compteur, machine à état, ou mémoire...)

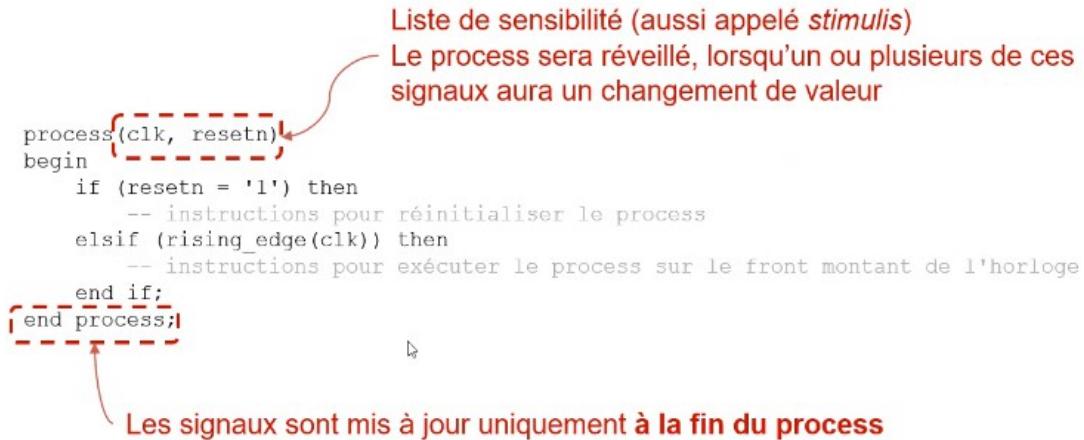
La logique combinatoire ne dépend que de l'état actuel des signaux(ex : portes logiques, multiplexeurs...)

Instructions concurrentes : Tout ce qui n'est pas dans le process, nous n'avons pas d'ordre de priorité sur les instructions donc attention aux conflits.

Instructions séquentielles : C'est dans le process, les instructions se font à la suite donc pas de conflit.

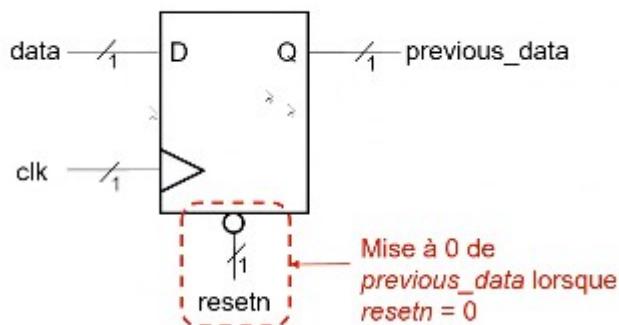
Le process est un ensemble d'instructions, on va rentrer dans ce process par rapport à la valeur des signaux d'entrées, si on a un changement sur les signaux d'entrées on va réaliser les instructions du process. Le process n'interrompt pas les autres événements sur la carte.

Les Process



L'exemple ci-dessous représente un process synchrone car on dépend de CLK.

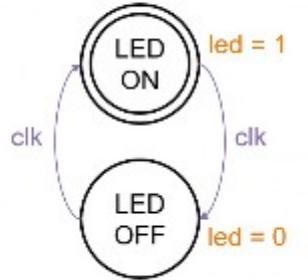
Un registre permet de mémoriser une valeur (bloc mémoire), il va prendre en mémoire data qui va être mise à jour à chaque front montant de clk. Q permet un reset.



Là on est logique séquentielle. Resetn permet de préciser qu'on est actif à l'état bas. Sur le schéma pour plusieurs bit, on peut représenter un registre avec le nombre de bit mais pas oublier qu'il y en a plusieurs.

Machine à état : on a une bulle par état, dans chaque bulle préciser l'état des signaux. Led=1 et led=0 affectation , clk changement d'état, Led ON et Led OFF=état du système.

- Permet de représenter les différents modes de fonctionnement du système
- Doit y être représenté : les états, les signaux de passage d'un état à l'autre et les **valeurs des signaux** dans l'état



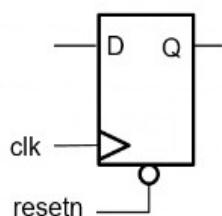
En VHDL, on peut écrire not ou ~.

Les Process

Process synchrone

```

process(clk, resetn)
begin
  if (resetn = '1') then
  elsif (rising_edge(clk)) then
  end if;
end process;
  
```



Process combinatoire

```

process(-----)
begin
end process;
  
```

La liste de sensibilité doit contenir **tous** les signaux d'entrée de ce bloc combinatoire

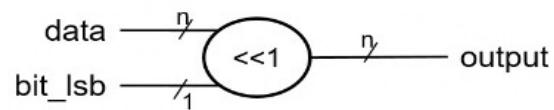


Les décalages : Il existe des symbole RTL pour faire les décalages. Qu'on on décrit un système à décalage, on va lire le MSB .

Décalage

- Vers la gauche avec insertion

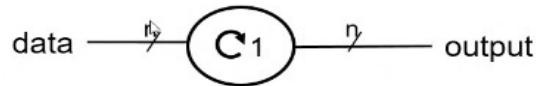
```
output <= data[n-2:0] & bit_lsb;
```



$\text{data} = 1001110$ et $\text{bit_lsb} = 1 \rightarrow \text{output} = 0011101$

- Vers la gauche avec rotation

```
output <= data[n-2:0] data[0];
```



$\text{data} = 1001110 \rightarrow \text{output} = 0011101$

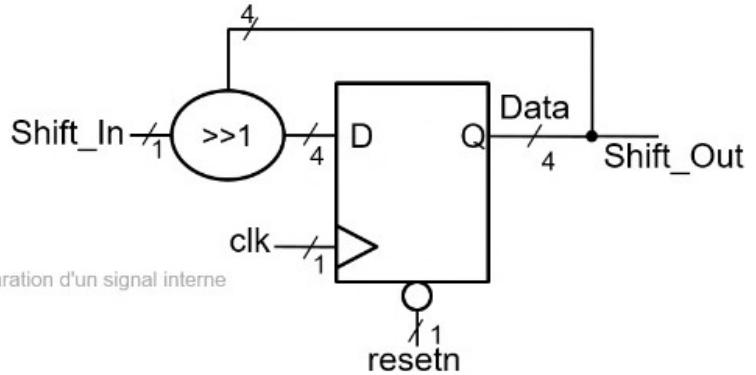
Exemple d'architecture (Registre à décalage) :

```
entity Shift_Register is
  Port (
    -- Déclaration des entrées et des sorties
    Shift_In : in std_logic;
    clk       : in std_logic;
    resetn   : in std_logic;
    Shift_Out : out std_logic_vector(3 downto 0)
  );
end Shift_Register;

-- Définition de l'architecture
architecture Behavioral of Shift_Register is
  signal Data : std_logic_vector(3 downto 0); -- Déclaration d'un signal interne

begin
  -- Processus du registre à décalage
  process(clk, resetn)
  begin
    if (resetn = '0') then -- Réinitialisation
      Data <= (others => '0');
    elsif (rising_edge(clk)) then -- Front montant de l'horloge
      Data <= Shift_In & Data(3 downto 1); -- Décalage vers la gauche avec insertion de Shift_In en MSB
    end if;
  end process;

  -- Sortie du registre à décalage
  Shift_Out <= Data;
end Behavioral;
```



Other : permet de remettre tous les signaux à 0.

Les synoptiques sont importants quand on a des systèmes complexes.(Pour le projet fixer les étapes avec).

Le flot de développement à suivre :

Flot de développement

1. Commencer par le schéma RTL / synoptique
2. Retranscrire le schéma RTL en code VHDL
3. Fixer les contraintes du système
4. Vérifier le comportement du système en simulation
5. Etudier la synthèse
6. Placer des sondes avec l'ILA
7. Etudier le placement routage (Place And Route, PAR)
8. Vérifier le comportement du système sur carte

vérifier les différentes contraintes : contrainte de timing, contrainte de placement, contrainte de composant.

Faire l'autre niveau de test avec des sondes pour vérifier l'état des signaux.

La simulation : on va tester toutes nos entrées et sorties (en pilotant nos entrées avec toutes les possibilités).

Le testbench permet de tester le comportement de l'architecture.

Lors de la simulation, il faut créer nos clk. Partout où on place notre curseur on peut avoir l'état de nos variables.

TP1:Note

- 1) $S = \text{Cin} \text{ xor } A \text{ xor } B$
 $C_0 = \text{Cin} \cdot (A \text{ xor } B) + A \cdot B$
 - 2) Les entrées : A,B et Cin
Sorties : S, Cout
- 4)

```

library ieee;
use ieee.std_logic_1164.all;

entity full_adder is
    Port (
        --Exemple d'entrees
        A : in std_logic;
        B : in std_logic;
        Ci : in std_logic;

        --Exemple de sorties
        S : out std_logic;
        Co : out std_logic
    );
end full_adder;

architecture behavior of full_adder is
begin
    S <= Ci xor A xor B;  --Affectation d'une sortie
    Co <= Ci and ( A xor B ) or (A and B);
end behavior;

--Valeurs des sorties attendues :
--  Cout = 0
--  S = 1

A <= '1';
B <= '0';
Cin <= '0';
wait for 10 ns;

--Valeurs des sorties attendues :
--  Cout = 0
--  S = 1

A <= '0';
B <= '1';
Cin <= '0';
wait for 10 ns;

--Valeurs des sorties attendues :
--  Cout = 0
--  S = 1

A <= '1';
B <= '1';
Cin <= '0';
wait for 10 ns;

--Valeurs des sorties attendues :
--  Cout = 1
--  S = 0

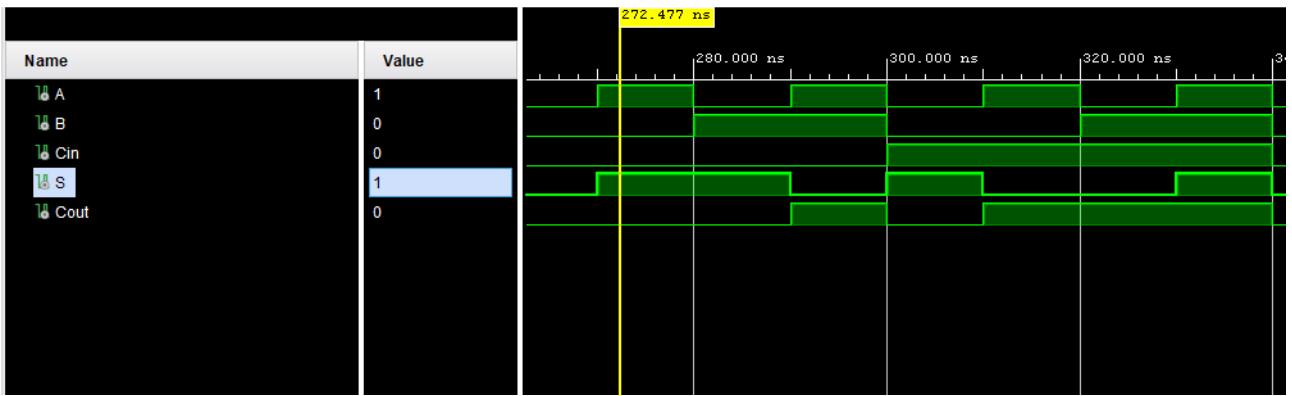
A <= '0';
B <= '0';
Cin <= '1';
wait for 10 ns;

--Valeurs des sorties attendues :
--  Cout = 1
--  S = 0

A <= '1';
B <= '1';
Cin <= '1';
wait for 10 ns;

--Valeurs des sorties attendues :
--  Cout = 1
--  S = 1

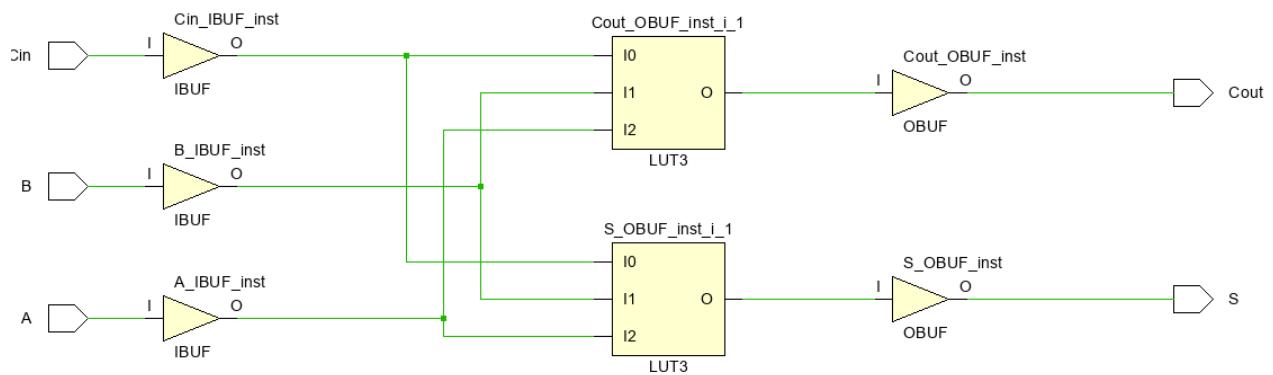
```



On obtient bien les mêmes résultats que sur le test bench

Pour automatiser on ajoute les asserts et les reports.

On obtient le schéma en exécutant une architecture de notre synthèse :



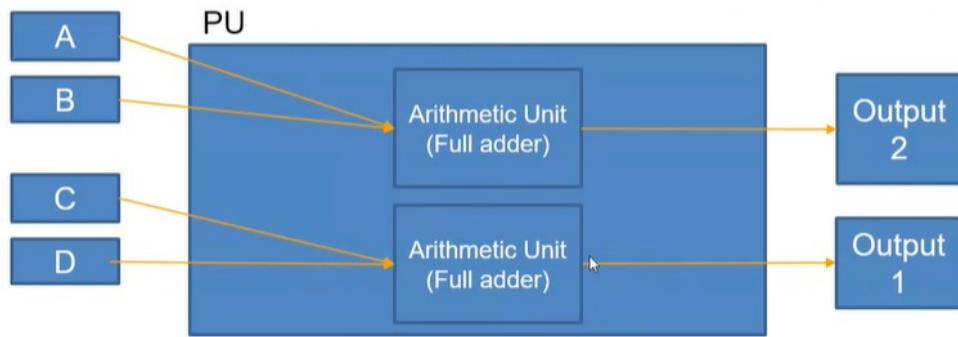
On peut voir que l'on a pas de portes logiques car on a des lut (Table de vérité).

Cours 4

Il est important d'évaluer la criticité (contraintes, exigences clients...) de nos signaux afin que les plus critiques soient envoyés le plus rapidement possible.

Sur un système le chemin le plus long est le chemin critiques (exemple : le Cout sur le full Adder).

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

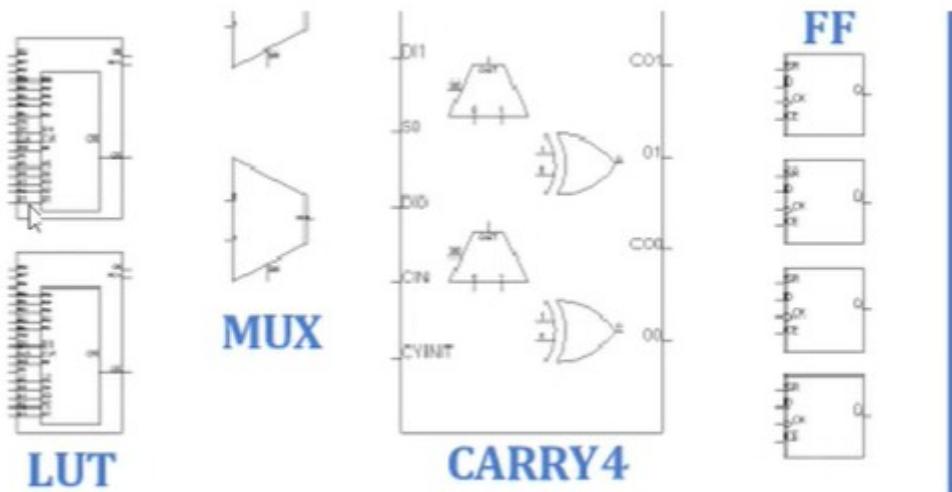


Il suffit de paralléliser les traitements.

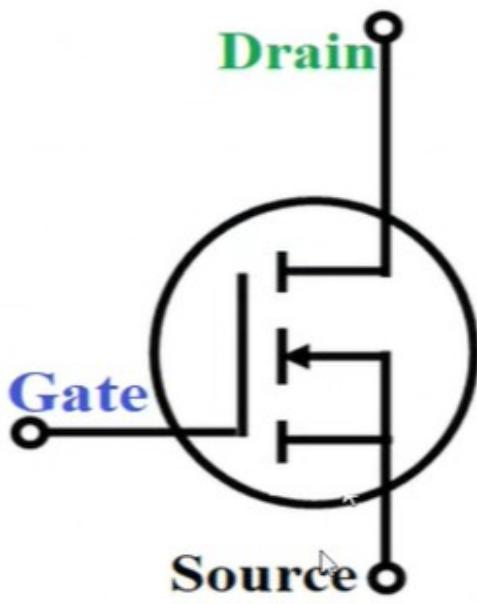
La latence reste alors inchangée. On utilisera deux fois plus de Process Units, aussi appelées Logical Elements (LE) dans le contexte d'un FPGA

Concernant la vu design, le fpga va créer notre routage de manière automatique mais on peut ajouter des contraintes pour placer des signaux seuls en fonction des contraintes. On a 4 luts par Slice, des registres et des multiplexeurs. Les cartes sont composées de boites de routage, de bloc avec (éléments logiques), des entrées/sorties. Les buffers permet de passer du niveau de tension dit de cœur vers le niveau de tension d'interface.

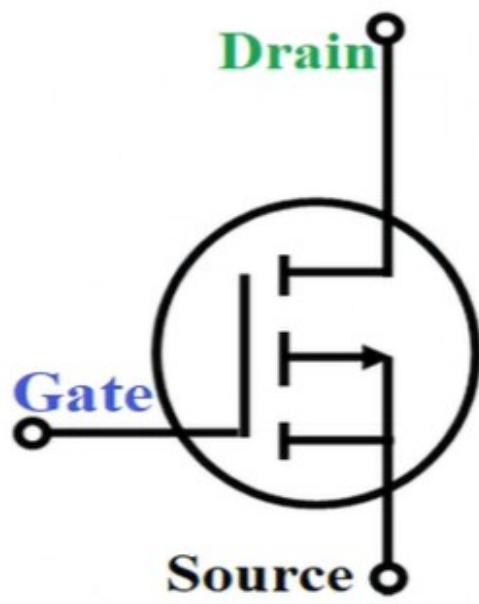
Chaque FPGA va avoir sa propre architecture de slice. Chaque architecture permet d'avoir des performances différentes. On va avoir des luts table, des mux(petite boîte de routage), les flip flop (qui permettent de mémoriser un bit) et des registres. Le carry 4 permet d'optimiser les fonctions de comptage.



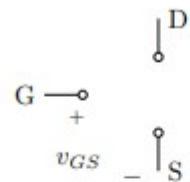
Les Mosfet : tripôle (gate, drain et source). Deux types Mosfet P et Mosfet N. Selon la tension appliquée sur la broche gate on va déclencher Drain et source. Si on est inférieur à la valeur de tension de gate, drain et source sont pas reliés sinon ils sont reliés pour le mos N pour le mos P c'est l'inverse.



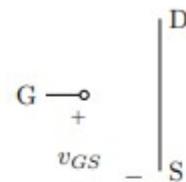
**N-Channel
MOSFET**



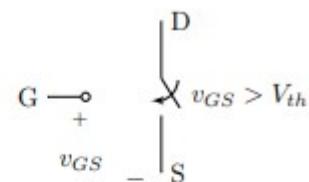
**P-Channel
MOSFET**



nMOS off: $v_{GS} < V_{th}$

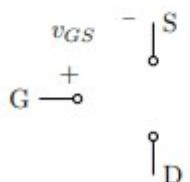


nMOS on: $v_{GS} > V_{th}$

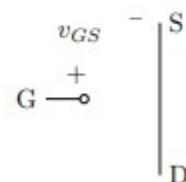


nMOS model

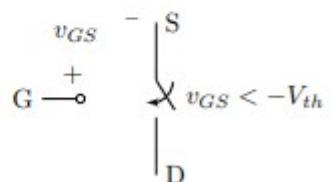
The pMOS is similar, except that it's flipped: it turns on when $v_{GS} < -V_{th}$.



pMOS off: $v_{GS} > -V_{th}$



pMOS on: $v_{GS} < -V_{th}$



pMOS model

Ce sont les lutts qui permettent de reconfigurer nos FPGA :



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**

C'est l'outil de synthèse qui permet de passer des portes logiques au LUT. Il va transformer toutes les portes logiques en mettant des adresses dans notre Ram (Dans le Lut). Asic= Application specific integrated circuit), un asic n'est pas reconfigurable. C'est un multiplexer qui se charge de faire l'aiguillage.

Td Multiplexer

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	

$S = A$

$S = B$

$$\text{Donc, } S = A \cdot \bar{n}B \cdot \bar{n}C + A \cdot B \cdot \bar{n}C + \bar{n}A \cdot B \cdot C + A \cdot B \cdot C$$

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

A	B	$\bar{n}A$	$\bar{n}B$	$A \cdot \bar{n}B$	$A \cdot B$	$\bar{n}A \cdot B$	$A \cdot \bar{n}B + A \cdot B$	$\bar{n}A \cdot B + A \cdot B$
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	0	1	1	1

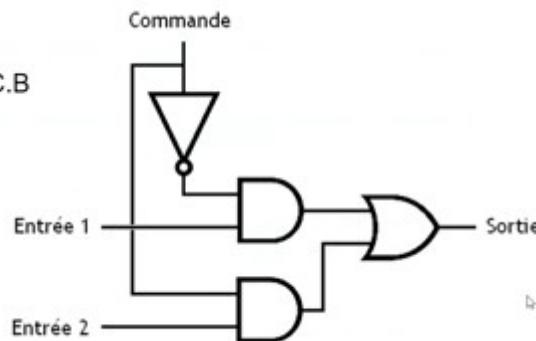
On remarque que

$A \cdot \bar{n}B + A \cdot B$	A
0	0
0	=
1	1
1	1

$\bar{n}A \cdot B + A \cdot B$	B
0	0
1	=
0	0
1	1

Et donc

$$S = \bar{n}C \cdot A + C \cdot B$$



La lut utilise de la Sram Ou du Flash.(Sram technologie de Ram). Les avantages de la Flash par rapport à la ram, c'est qu'on aura toujours la description matérielle dans les luts. Cela permet d'éviter le temps de reconfiguration de la carte.

C'est le FTDI qui fait le pont entre la carte et l'ordinateur.

La Sram est différente du Rom car au niveau de la microélectronique, on a des meilleures performances de traitement. La cellule Flash est plus grande qu'une cellule Ram. Le sillicium va être beaucoup plus étalé. Le Flash plus risqué que la Ram car il faut chiffrer notre BitStream. Il existe donc des FPGA volatile et non volatile.

TD substractor

Table de vérité –

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

		B Bin			
		00	01	11	10
		0	1	0	1
		1	1	0	0

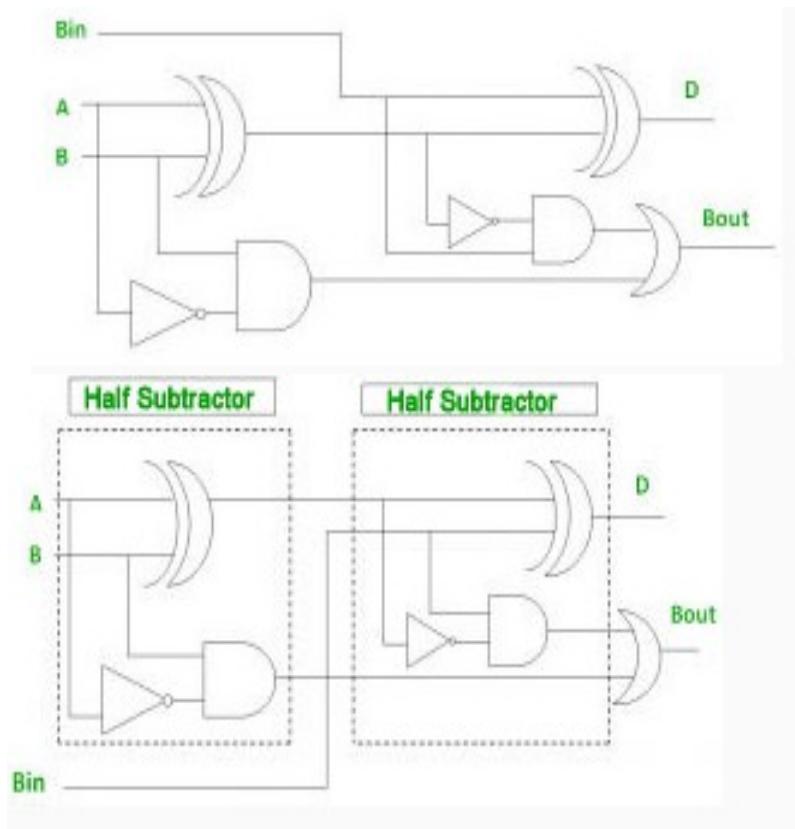
$$D = A'B'Bin + AB'Bin' + A'BBin' + ABBin$$

		B Bin			
		00	01	11	10
		0	1	1	1
		1	0	1	0

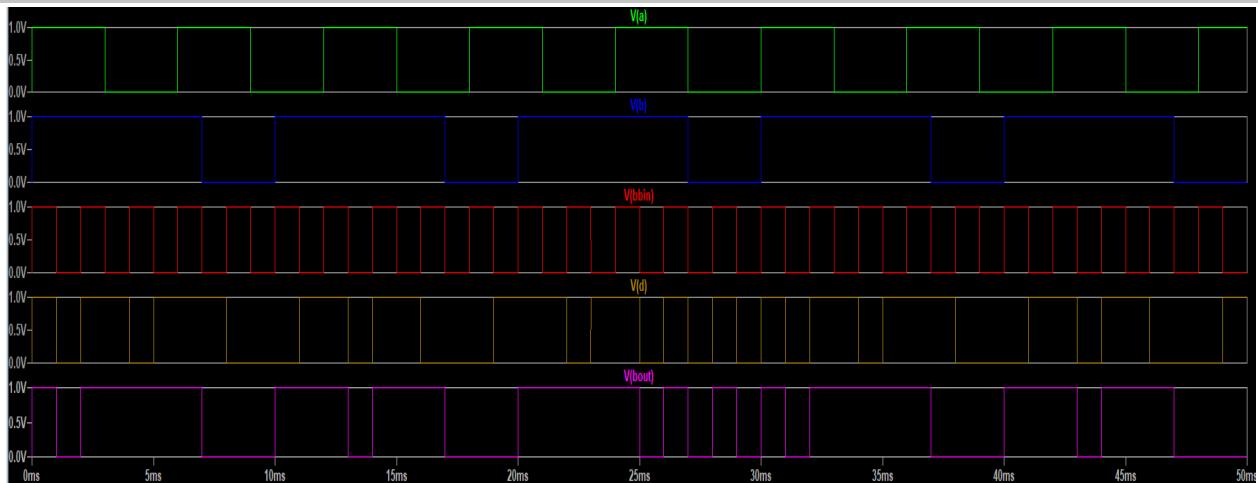
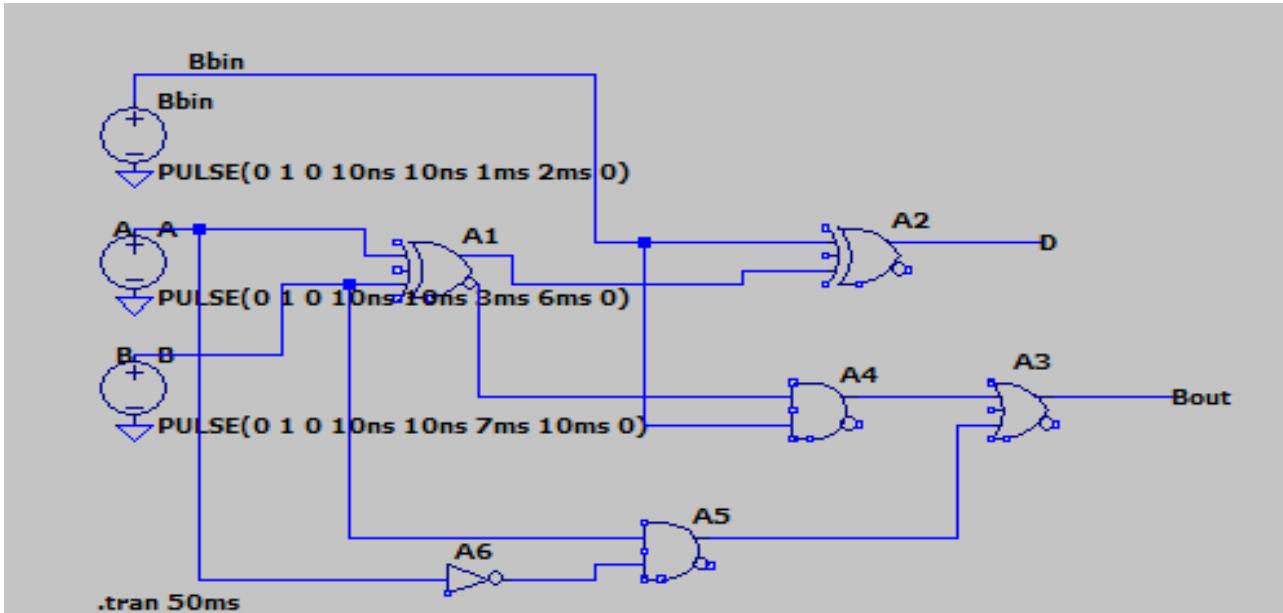
$$Bout = A'Bin + A'B + BBin$$

$$\begin{aligned}
D &= A'B'Bin + A'BBin' + AB'Bin' + ABBin \\
&= Bin(A'B' + AB) + Bin'(AB' + A'B) \\
&= Bin(A \text{ XNOR } B) + Bin'(A \text{ XOR } B) \\
&= Bin(A \text{ XOR } B)' + Bin'(A \text{ XOR } B) \\
&= Bin \text{ XOR } (A \text{ XOR } B) \\
&= (A \text{ XOR } B) \text{ XOR } Bin
\end{aligned}$$

$$\begin{aligned}
Bout &= A'B'Bin + A'BBin' + A'BBin + ABBin \\
&= Bin(AB + A'B') + A'B(Bin + Bin') \\
&= Bin(A \text{ XNOR } B) + A'B \\
&= Bin(A \text{ XOR } B)' + A'B
\end{aligned}$$

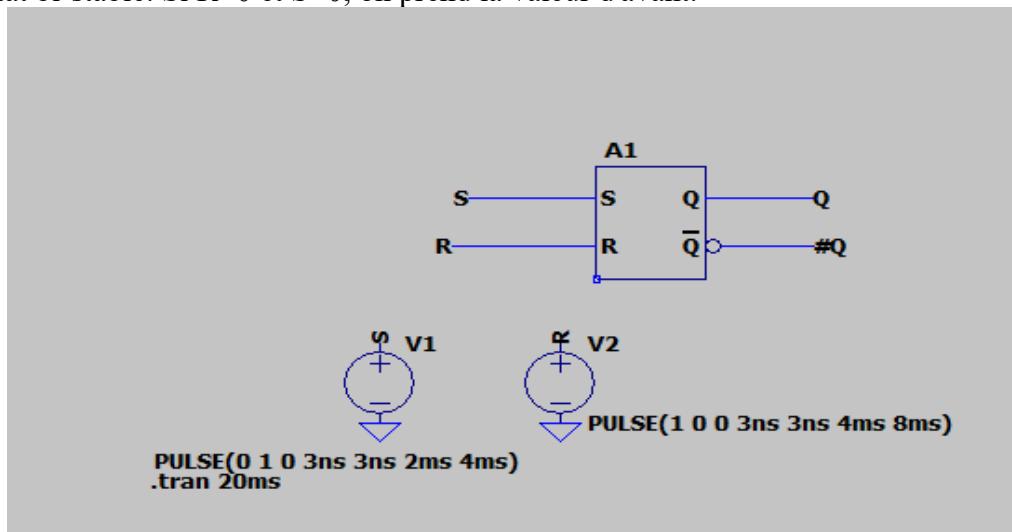


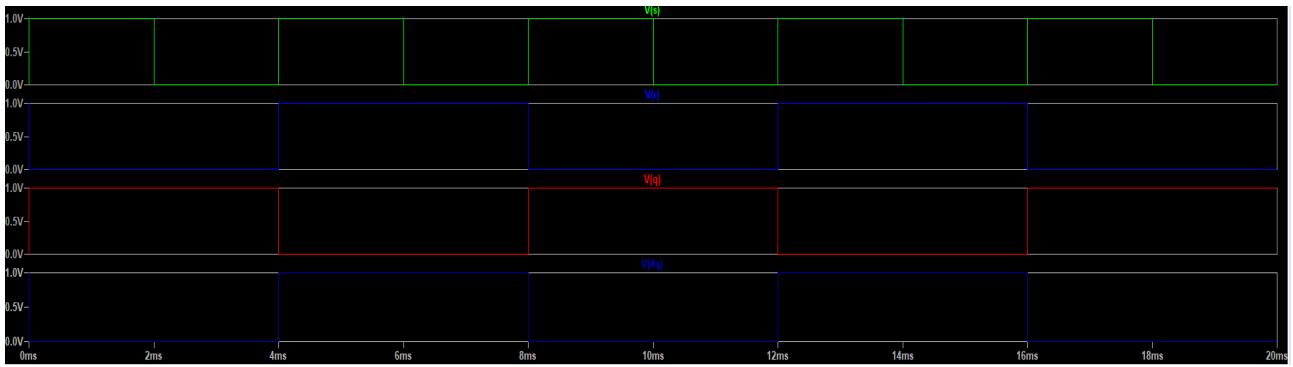
Réalisation sur LtSpice :



Cours 5

La logique synchrone : bascule, registre, les compteurs et registres à décalage et les machines à état.
 La Bascule (le latch), est un élément qui a deux états qui permettent de stocker des informations.
 On a un état bi-stable. Si $R=0$ et $S=0$, on prend la valeur d'avant.

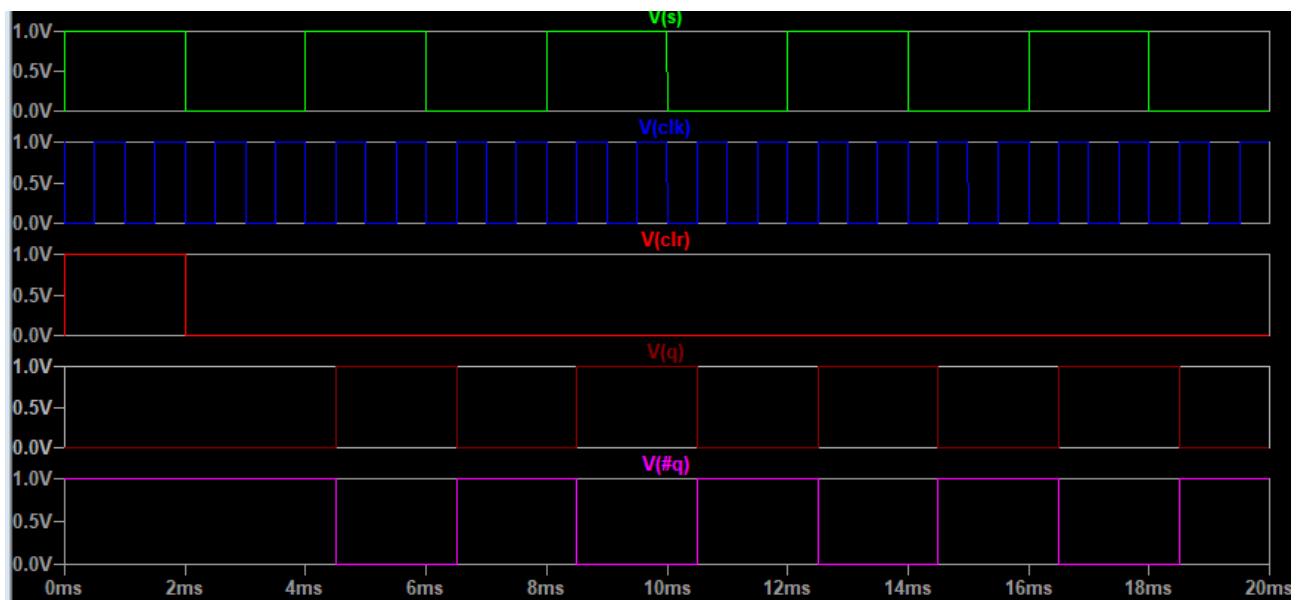
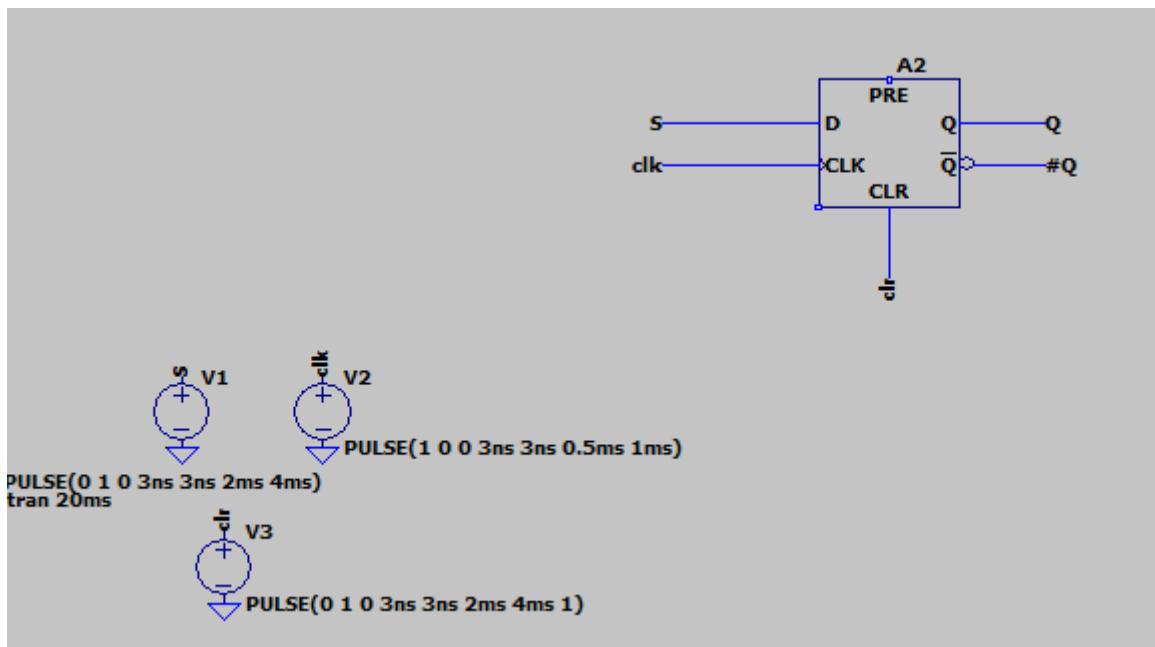




Le cas $S=1$ et $R=1$ est un cas dit illégal car il est redondant.

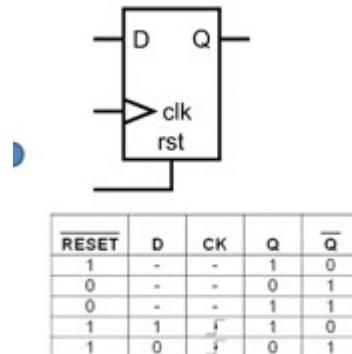
Ici, nous ne sommes pas en logique synchrone, ce circuit permet juste de sauvegarder un bit à l'instant t.

Registre Flip Flop

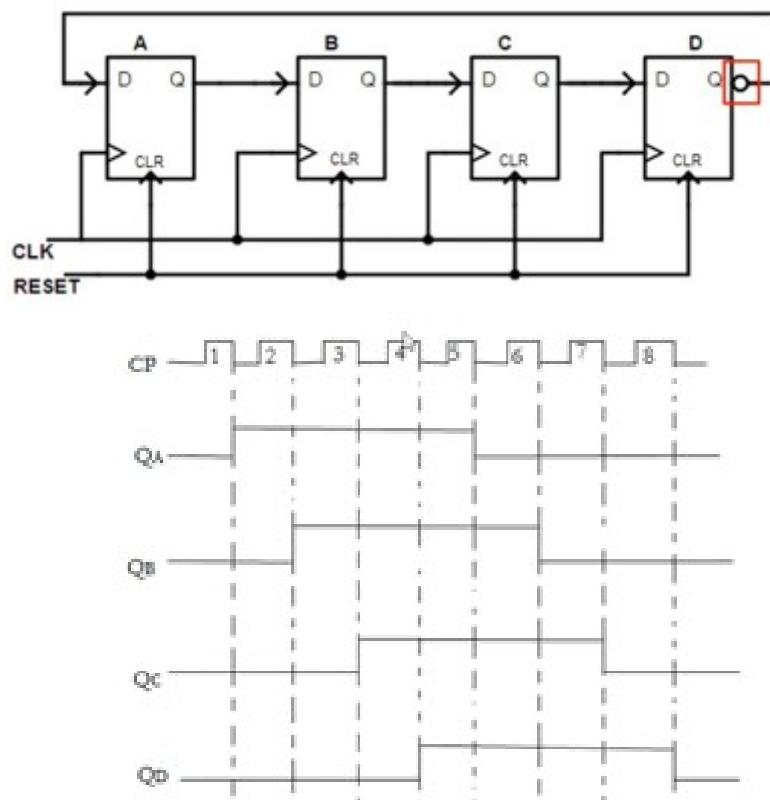


Lorsque le clear est actif on a notre sortie Q à 0 ensuite lorsque le clear est à 0 on doit avoir un

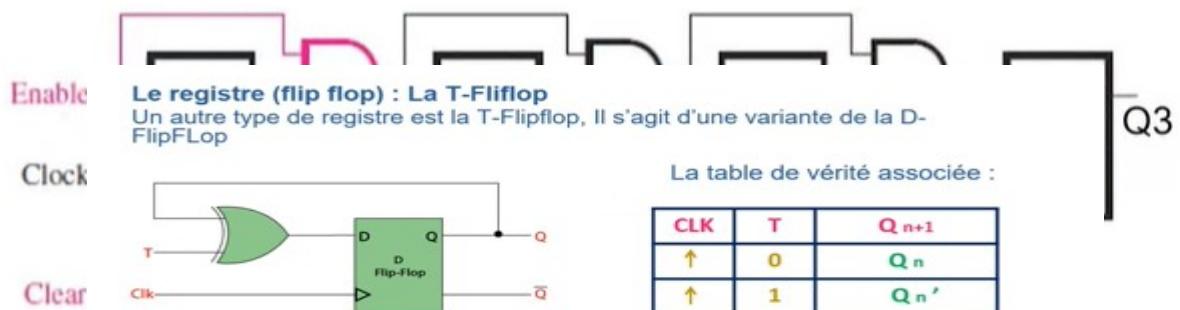
front montant d'horloge et notre entrée pour que Q passe à 1 et inversement. L'horloge est notre port de synchronisation et le clr permet une remise à zéro.



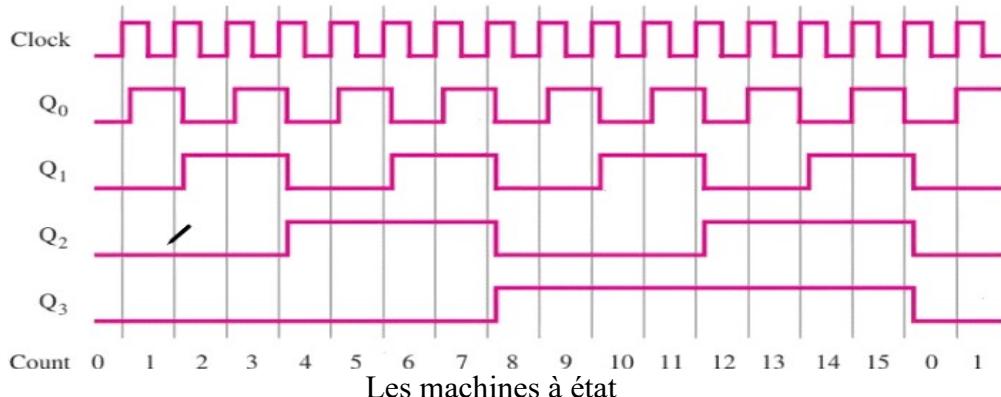
Compteur



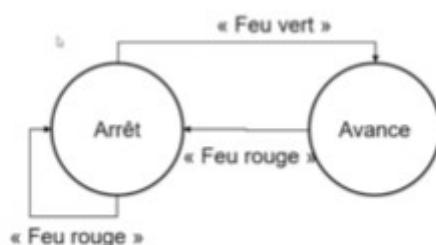
On peut observer un décalage car chaque valeur dépend de la précédente et des fronts montant de clk(ici on est sur front descendant).



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.



Les machines à état permettent de réaliser des calculs un peu plus complexe.(machine à café, contrôle de caméra...). Elle va être définie par un nombre d'état finis. Chaque état va avoir une sortie associée.



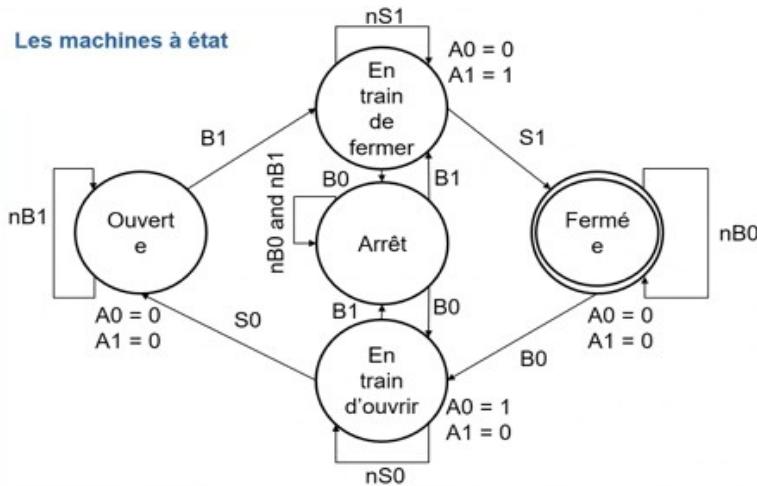
Deux types de diagramme : diagramme de Moore (l'état suivant dépend de l'état courant) et de Mealy(l'état dépend de l'état courant et des entrées).

Exercice :

On a 5 état s: Ouverture, en train de s'ouvrir, fermeture, en train de se fermer, fermeture et arrêt.

Les transitions :

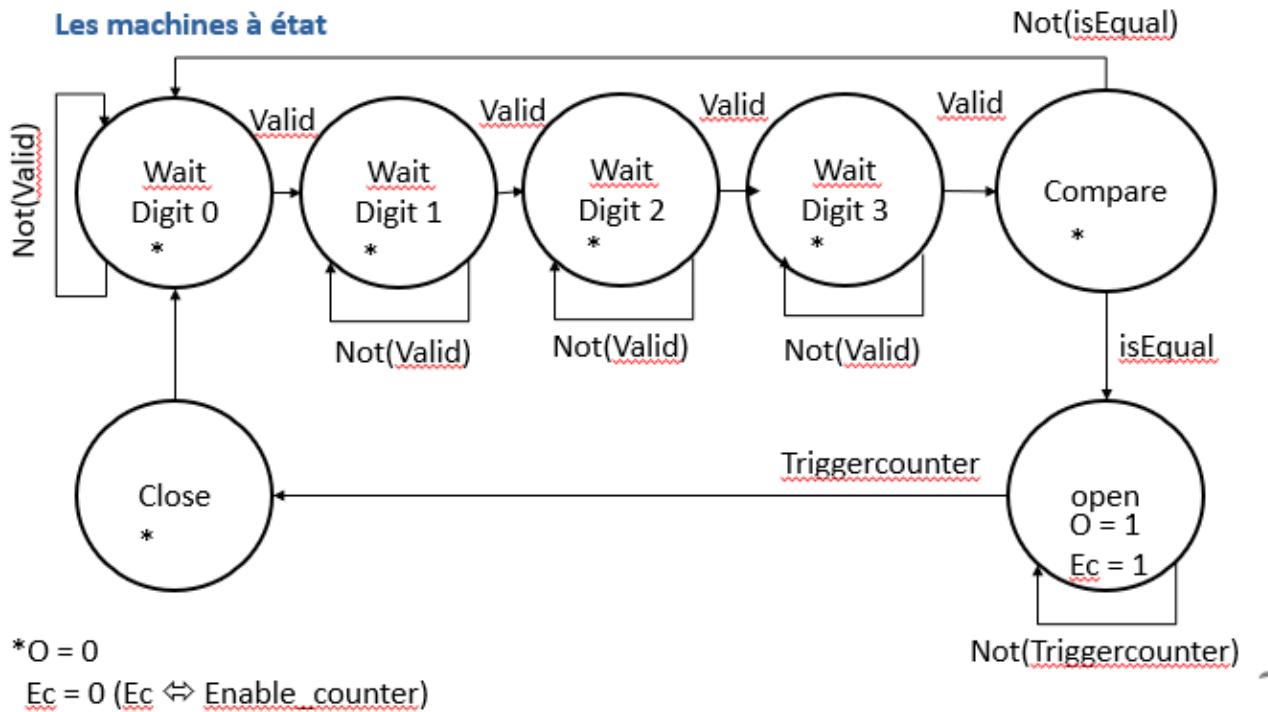
- De l'état "fermée" à l'état "en train de s'ouvrir" si le bouton d'ouverture est enfoncé et que la porte n'est pas déjà en train de s'ouvrir.
- De l'état "en train de s'ouvrir" à l'état "ouverte" lorsque le détecteur de fin de course indique que la porte est complètement ouverte.
- De l'état "ouverte" 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.

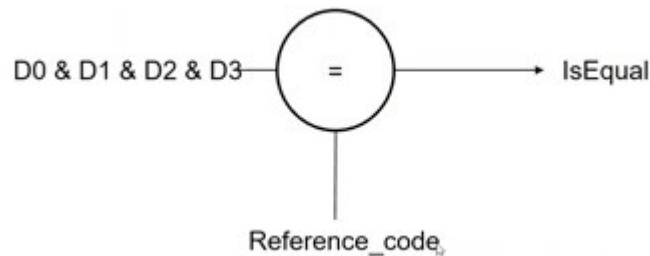
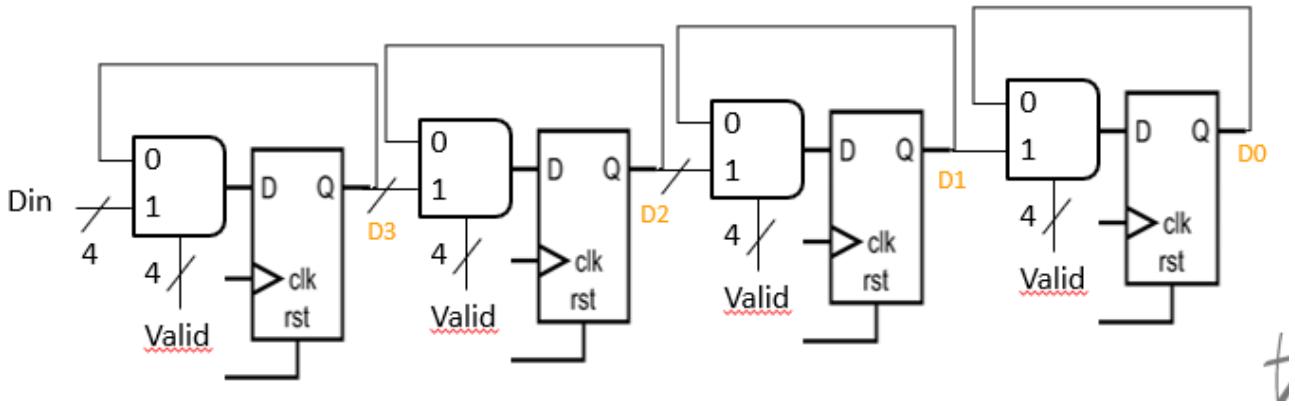


2ème exercice

Les états :

- saisie du premier chiffre
- saisie du deuxième chiffre
- saisie du troisième chiffre
- saisie du quatrième chiffre
- Validation du code
- ouverture de la porte
- fermeture de la porte





Registre à décalage

Il faut réaliser une gestion de timing car un temps de délais peut apparaître et les calculs seront faux durant ces périodes. La métastabilité peut se propager sur tout notre calcul. L'analyse des timing est très importante.

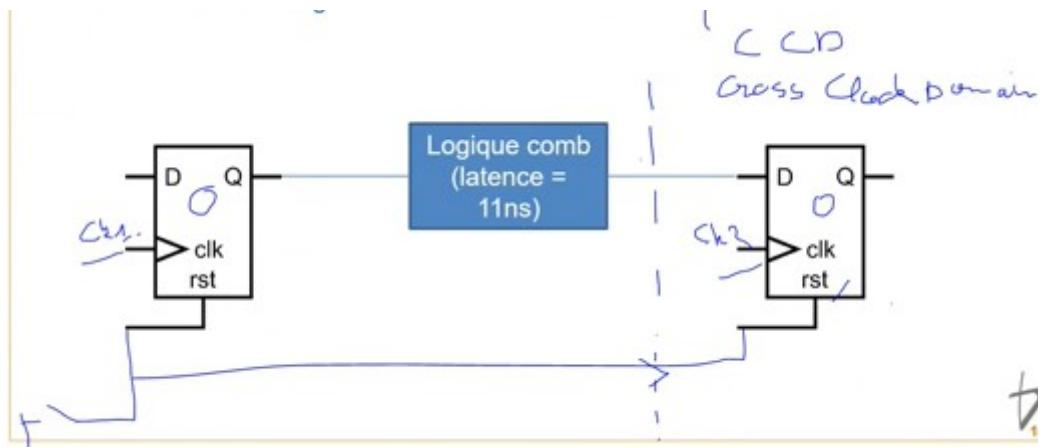
La règle : tout signal, en entrée d'un registre, doit être dans un état stable durant la plage de temps s'étalant avant et après le front montant d'horloge.

C'est lors du routage qu'on va régler ce problème de timing.

La plage où le signal doit être stable est la plage de setup et la plage après le front montant est la plage de « hold ». La mesure de temps est le slack. Lorsqu'on a un slack de 2ns on est stable avant la période de setup. Si le slack est négatif, ce n'est pas bon. Le slack c'est par rapport à la zone de setup et à la zone de hold.

Pour régler ce soucis de métastabilité il est possible de diviser la logique combinatoire en deux rajouter un registre.

Lorsqu'il y a deux clk on peut rajouter plusieurs registres en sortie.



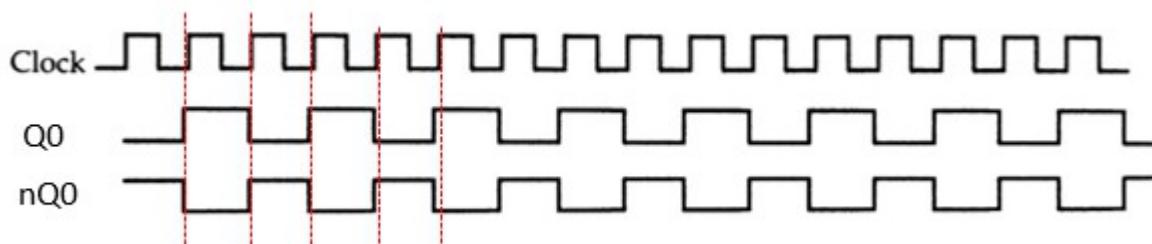
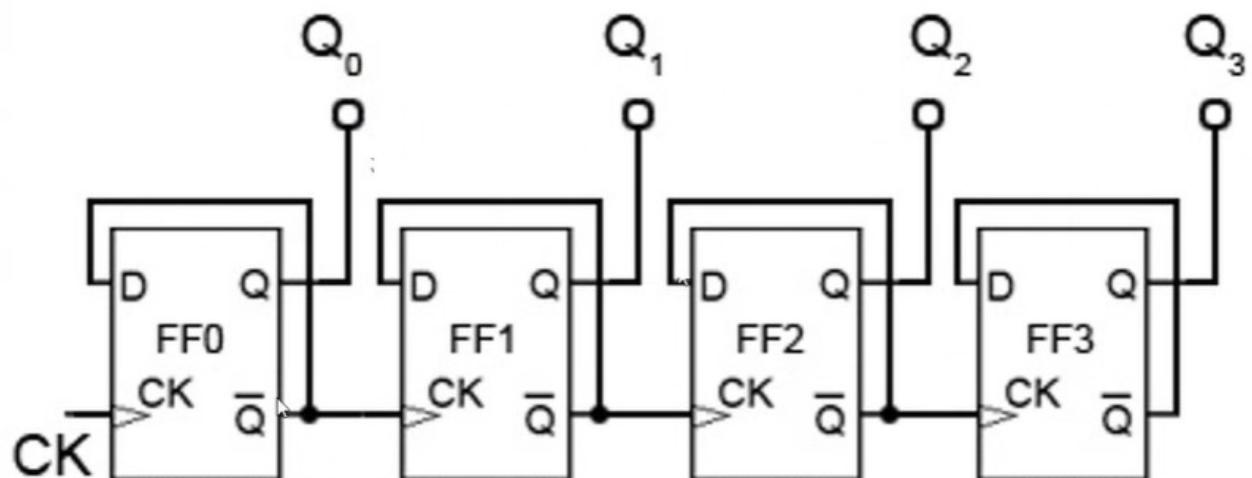
Pour régler ce problème de cross clock domain, pour régler cela il faut dire au logiciel de ne pas analyser ce chemin (Création d'un faux chemin). POR=Power on reset.

L'horloge fait un plus long chemin sur le routage pour arriver au même moment que nos signaux de registre.

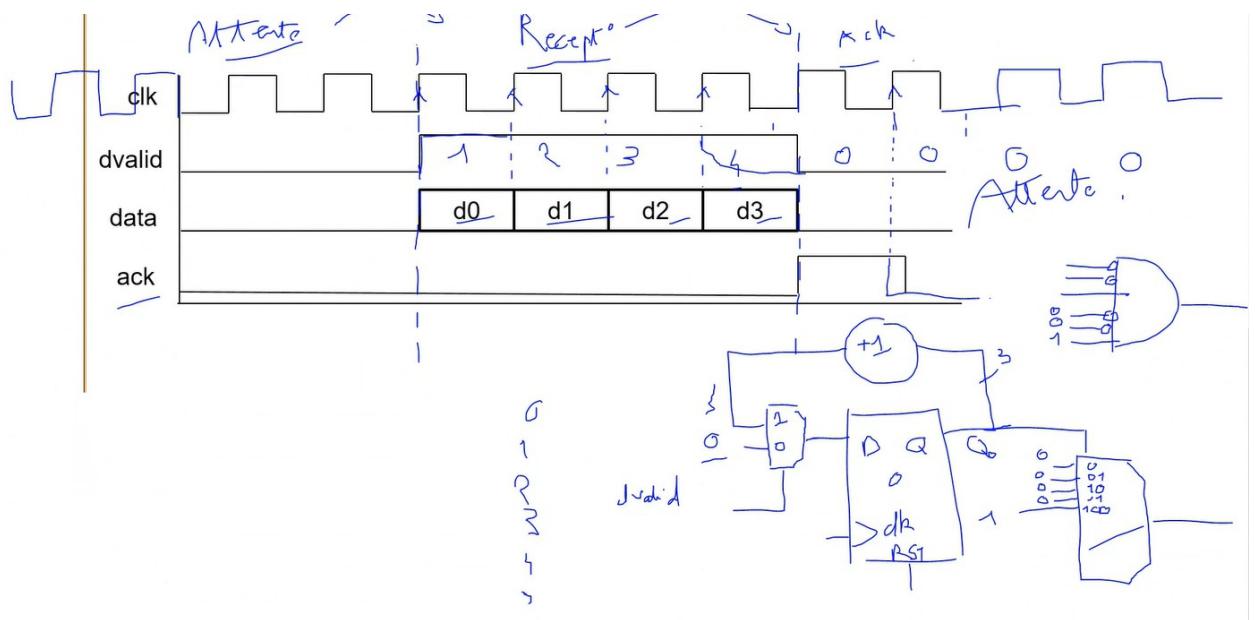
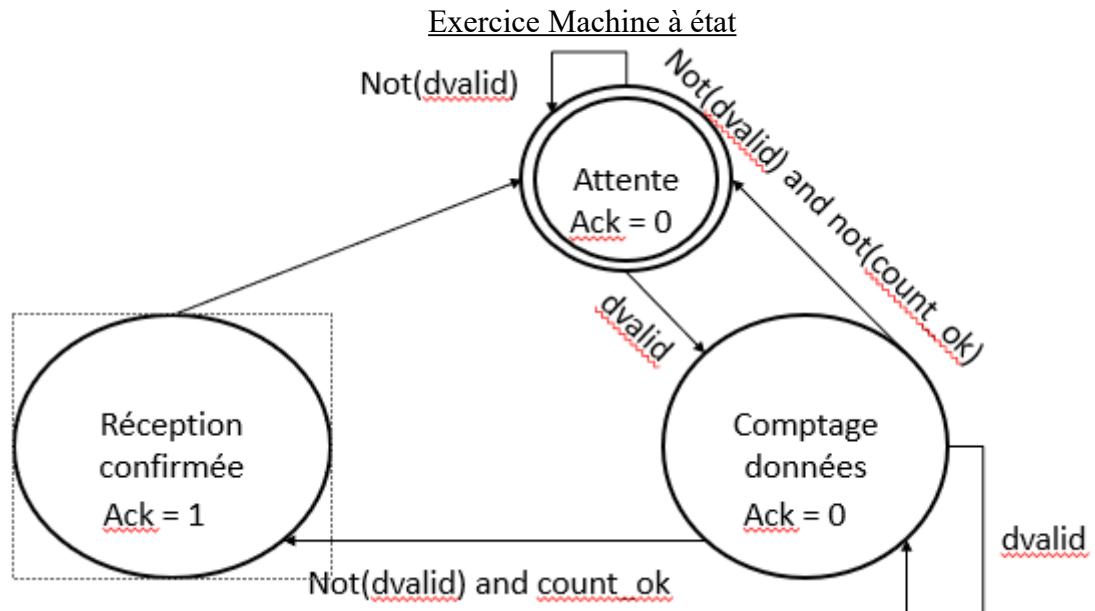
Pour limiter le clock skew, les horloges sont routées sur un arbre d'horloge.

Il ne faut pas dépasser les 80% sur les ressources que ce soit lut, slice embedded ram..

Ripple Counter



Cette fonction s'implémente mais il faut faire attention au nombre de piste de l'arbre d'horloge.



Pour éviter le CCD cross clock domain au niveau d'un bus de données, on peut utiliser un fifo (first in first out) qui possède une clock d'écriture et une clock de lecture.

L'ILA permet de placer des sondes sur le circuit et de vérifier avec des sondes que le circuit fonctionne bien. L'ILA intervient après le banc de test car l'ILA permet de tester le routage.