Circuits numériques programmables

AP4 S8 2022-2023 BX

Email: antoine.pirog@junia.com

Plan du cours

I. Le microcontrôleur et ses périphériques

I. Rappels : Généralités sur les microcontrôleurs et le langage C

II. Configuration d'un microncontrôleur

II. Le FPGA

I. Architecture des circuits logiques programmables

II. Le langage de description matériel VHDL

Organisation du cours

- **Module de circuits numériques programmables AP4**
  (65% de l'UE Systèmes électroniques et Télécommunications)

SÉANCES DE COURS

	Volume horaire
Cours/TD	10h (6h + 4h)
TP	42h (18h30 + 23h30)

EVALUATION

	Durée	**Nombre**	**Pondération**
DS	1h30	2	20% + 30%
TP/Projet	3h - 3h30	6	40%
TP	3h30 – 4h	6	10%
TOTAL			100%

Organisation du cours

- **Module de circuits numériques programmables AP4**
  (65% de l'UE Systèmes électroniques et Télécommunications)
- 2ème partie : FPGA

SÉANCES DE COURS

	Nombre de séances	Volume horaire
Cours/TD	1	4
TP	6	23h30

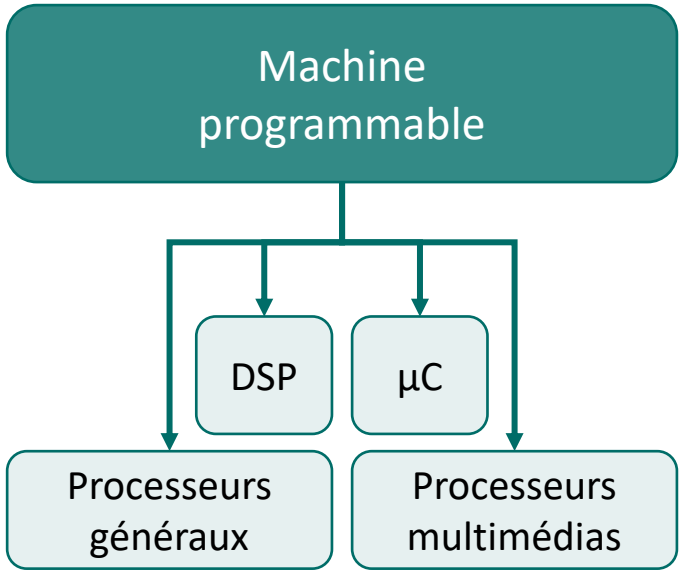
EVALUATION

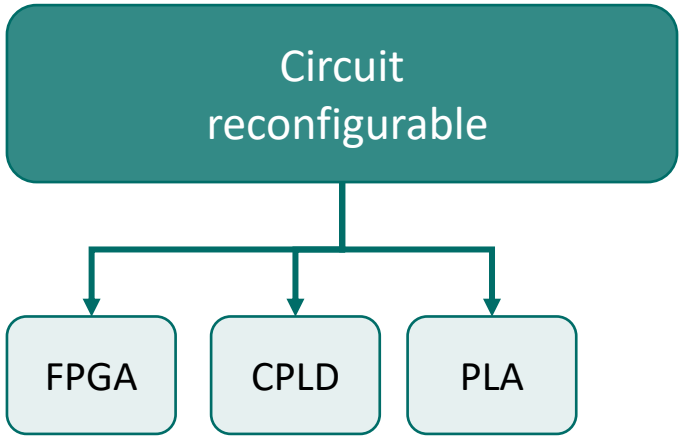
	Durée	Nombre	Pondération
DS	1h30	1	?
TP	3h30 - 4h	6	?
TOTAL			

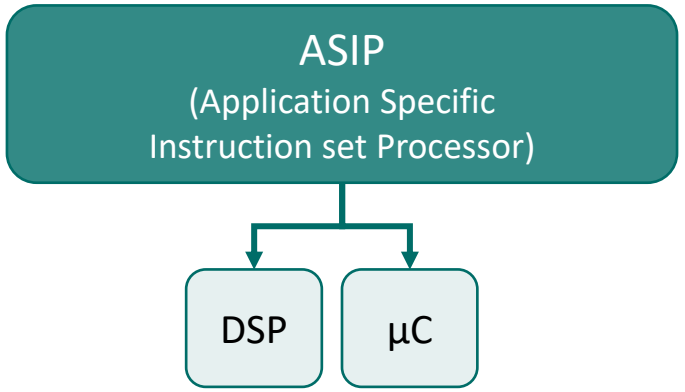
Introduction

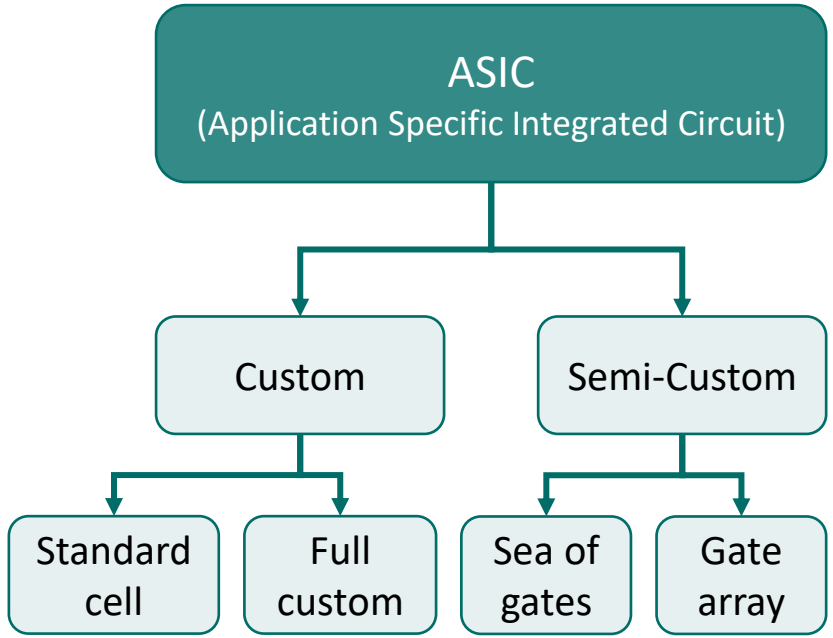
Les circuits numériques programmables

Solutions d'intégration en électronique









Implications des différentes solutions d'intégration

ASIC

- Développement long
- Coût de fabrication en augmentation
- Performances maximales (full custom)
- Fabrication en grande série

Circuit programmable

- Développement rapide
- Coût à l'unité en diminution
- Performances limitées par la technologie choisie
- Prototypage rapide

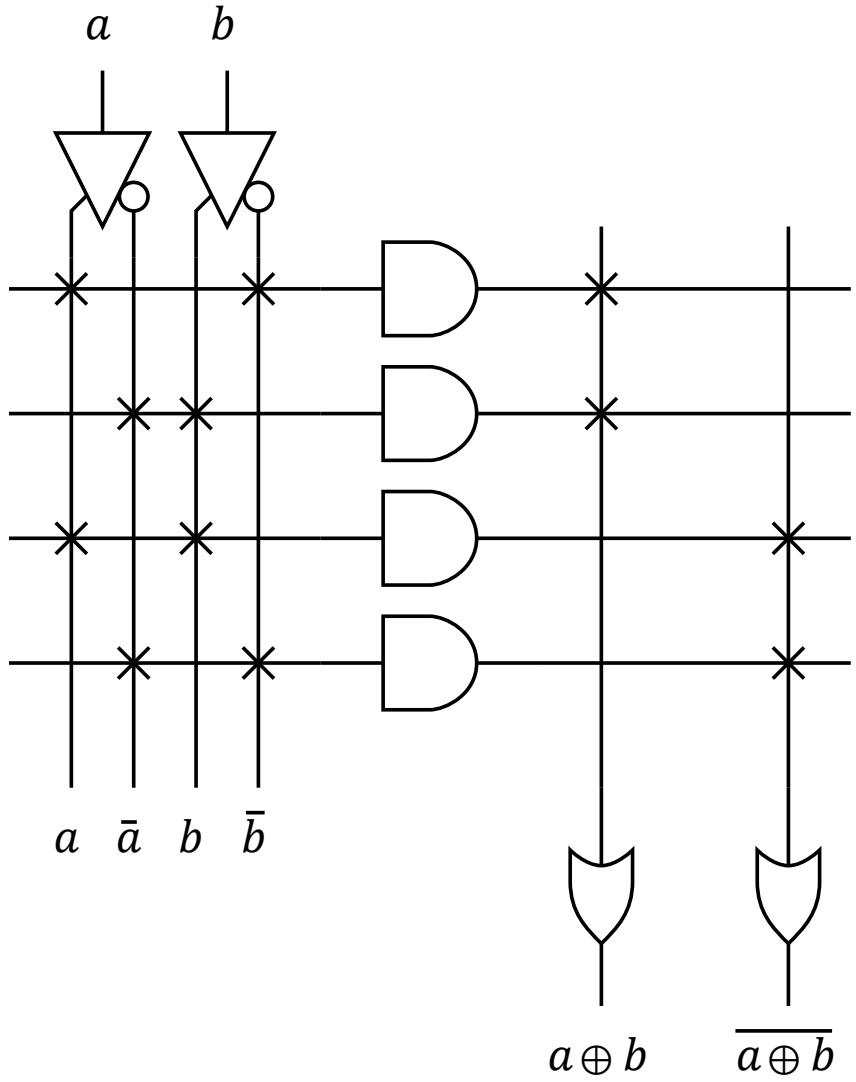
Les circuits configurables de type PLA

- **PLA** : Programmable Logic Array
- Typ. 100-200 portes logiques

1ère couche : Entrées et entrées complémentées

2ème couche : Réseau de ET **programmables**

3ème couche : Réseau de OU **programmables**

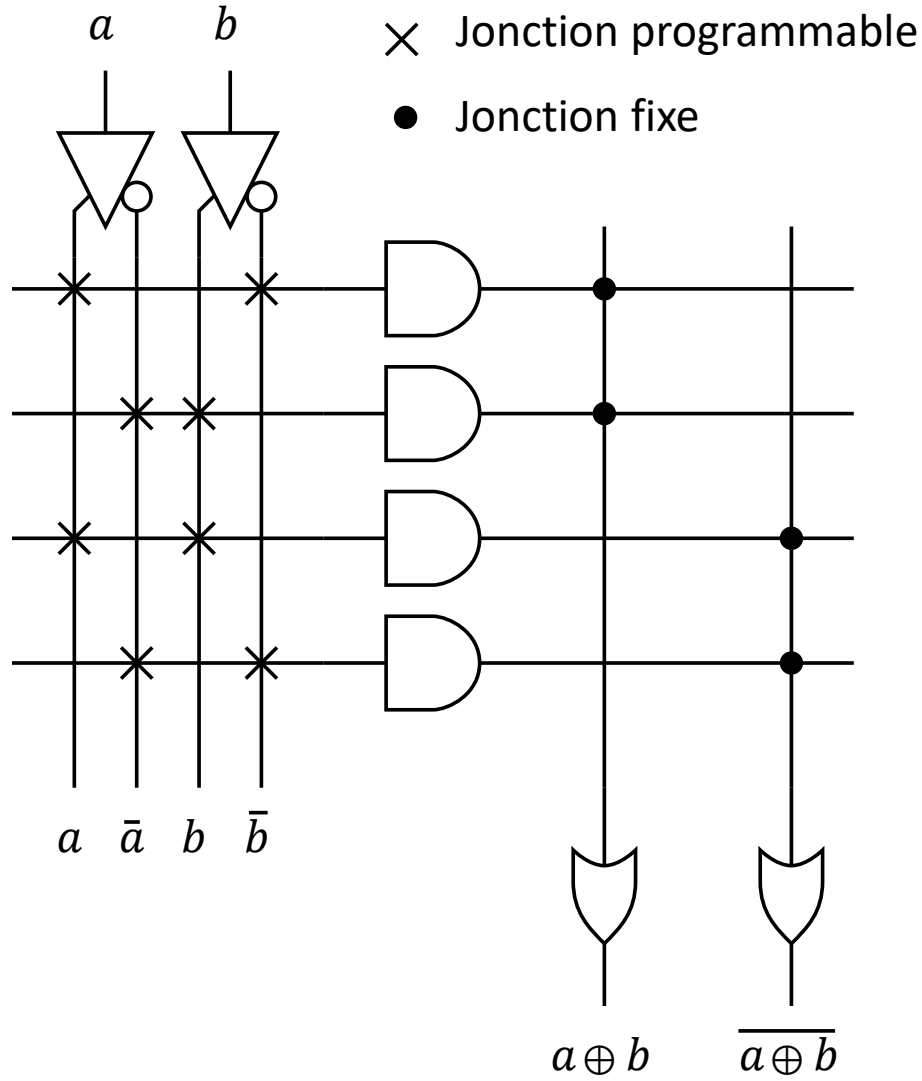


Les circuits configurables de type PLD

- **PLD** : Programmable Logic Device
- Typ. 100-200 portes logiques

Semblable aux PLA, moins flexible :

- 1 réseau **programmable** (ET)
- 1 réseau **non programmable** (OU)

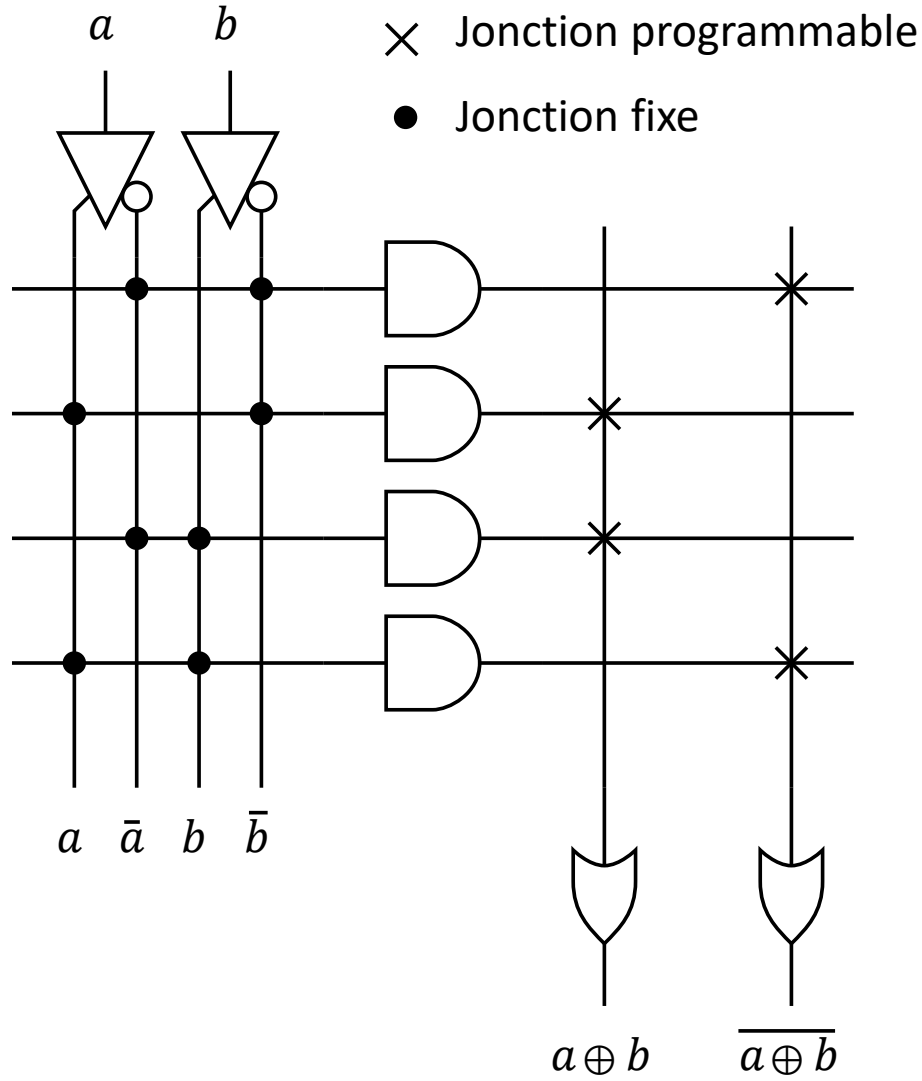


Les circuits configurables de type LUT

- **LUT** : Lookup Table
- Typ. 100-200 portes logiques

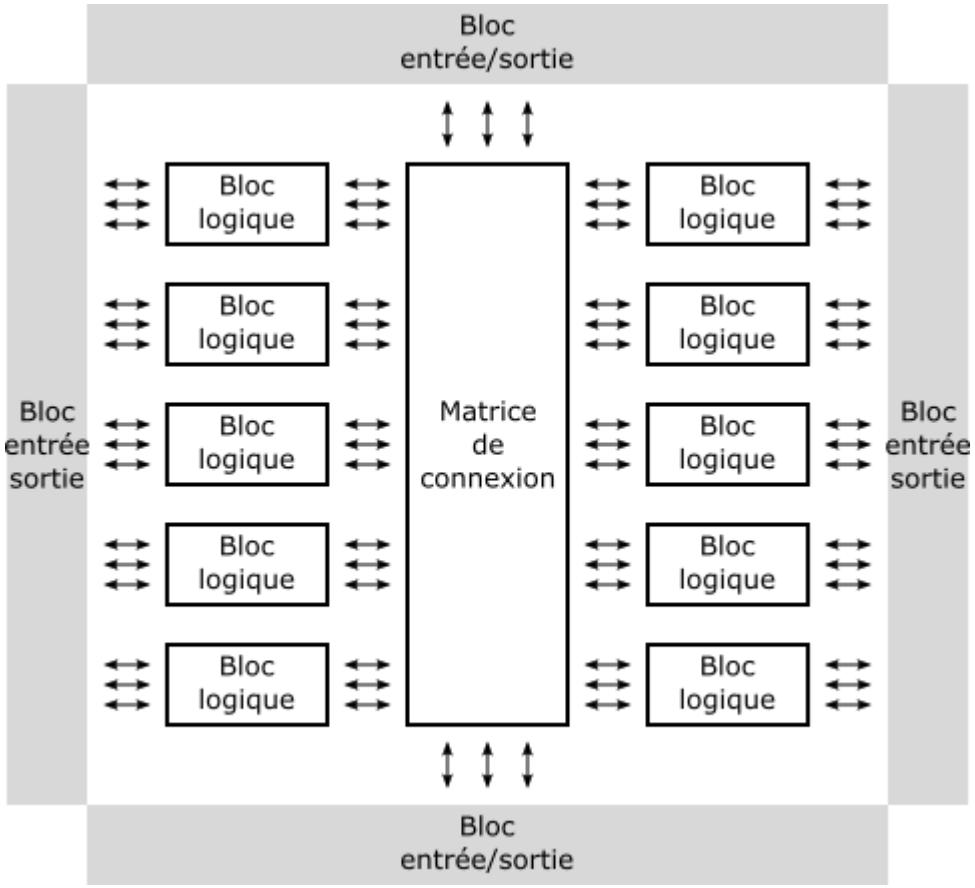
Semblable aux PLA, moins flexible :

- 1 réseau **programmable** (OU)
- 1 réseau **non programmable** (ET)



Les circuits configurables de type CPLD

- **CPLD** : Complex PLD
- Interconnexion de PLA à l'aide d'une matrice
- Typ. 5k-10k portes logiques



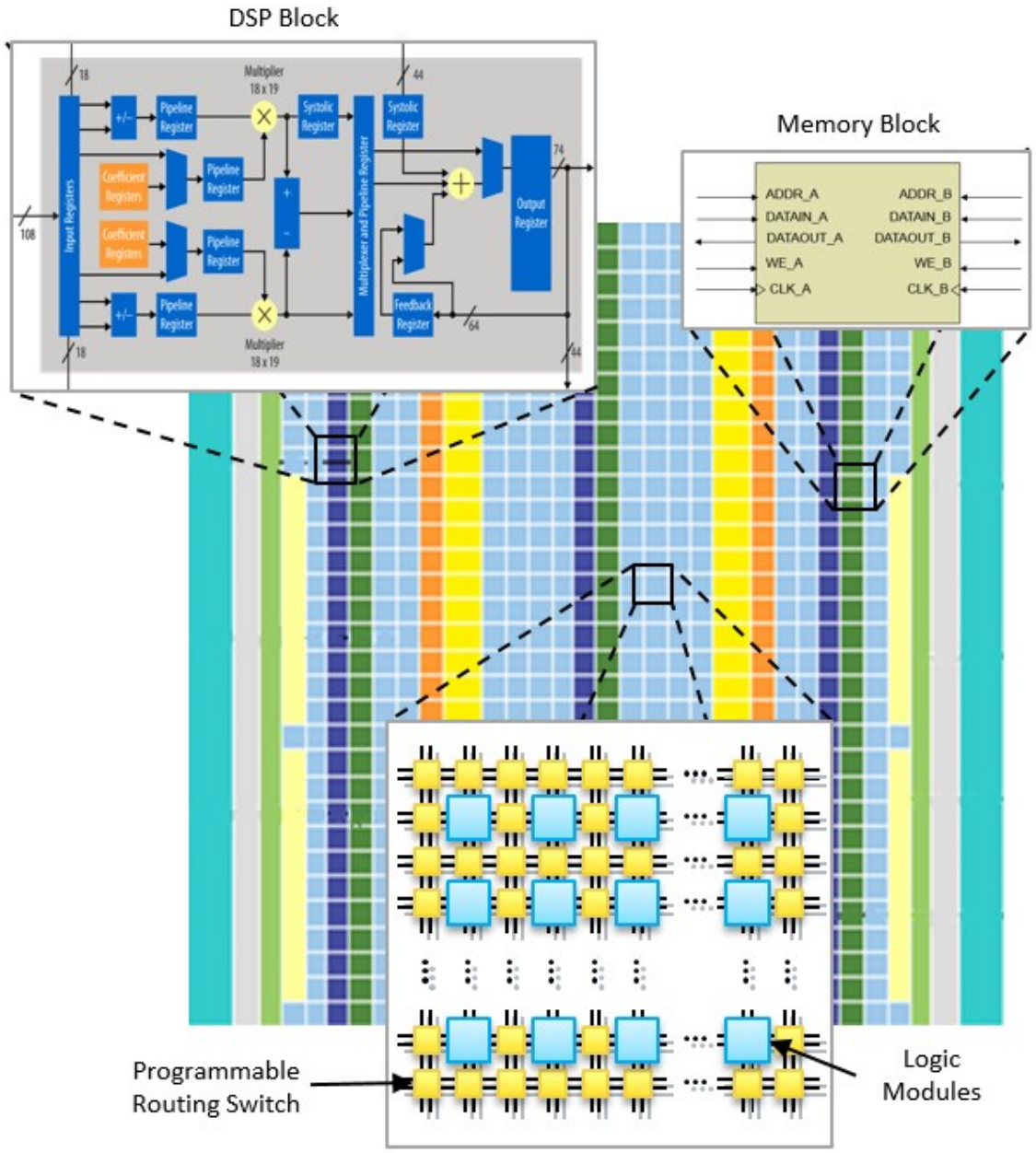
Les circuits configurables de type FPGA

FPGA : Field Programmable Gate Array

> Circuit intégré **programmable** permettant **l'implémentation matérielle** de **fonctions logiques combinatoires et séquentielles**

Fonctionne grâce à l'interconnexion configurable de milliers/millions de **blocs logiques élémentaires** configurables (LUT,MUX, RAM, DSP, etc.)

Typ. 10M portes logiques



Applications des FPGA

- Traitement de l'image
- Intelligence artificielle
- Accélération matérielle
- Sécurité matérielle
- Systèmes d'assistance à la conduite

1.

Calcul **parallélisé**

2.

Calcul **faible latence**

3.

Systèmes dont **l'implémentation matérielle** est régulièrement **mise à jour**

4.

Prototypage de puces numériques

Architecture des FPGA

Remarque

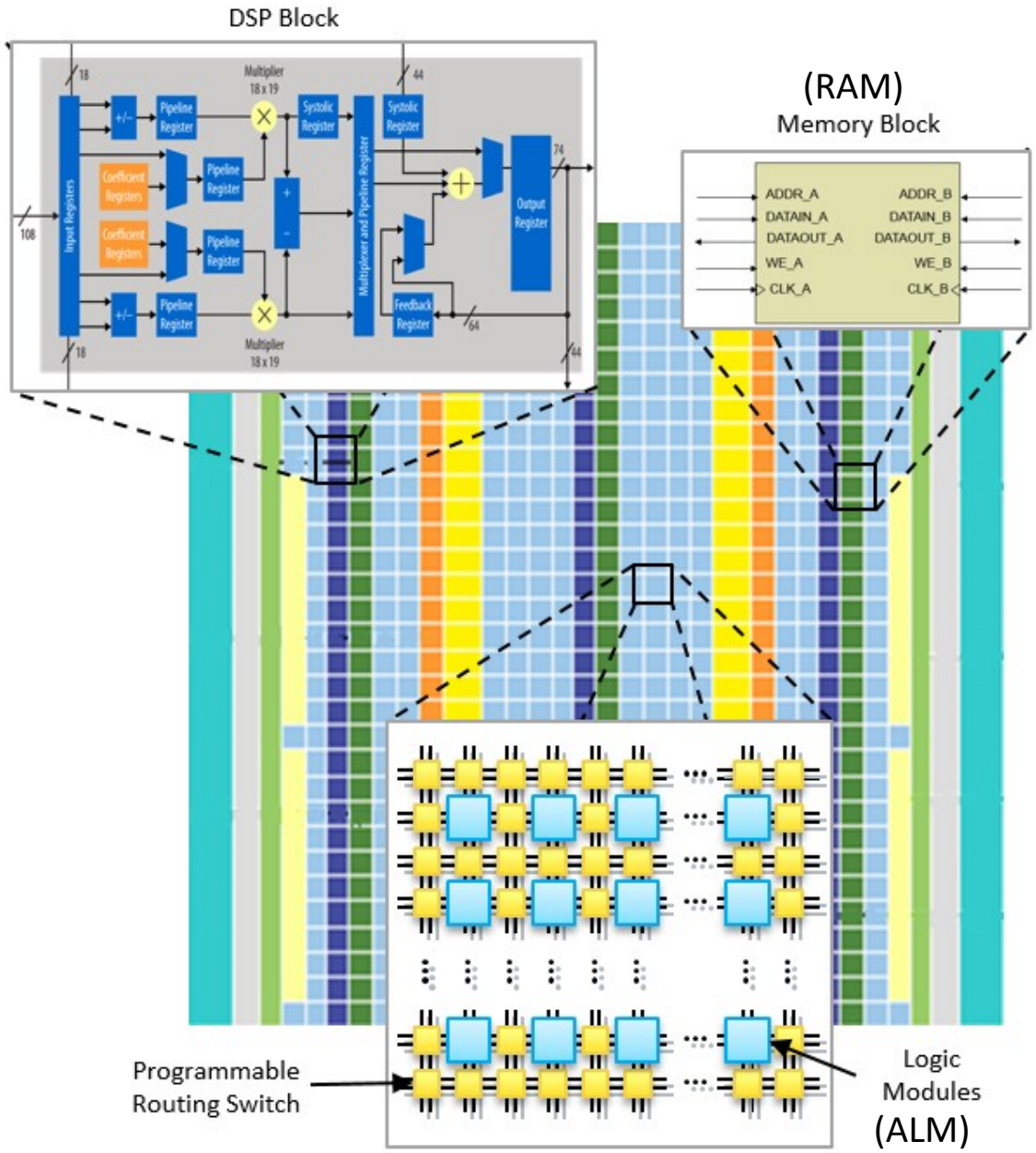
On utilisera dans cette partie du cours l'exemple concret de l'architecture des FPGA Intel Cyclone V.

La source des informations présentées peut être consultée sur :

- https://www.intel.com/content/www/us/en/docs/programmable/683152/22-1/fpga-architecture-overview.html

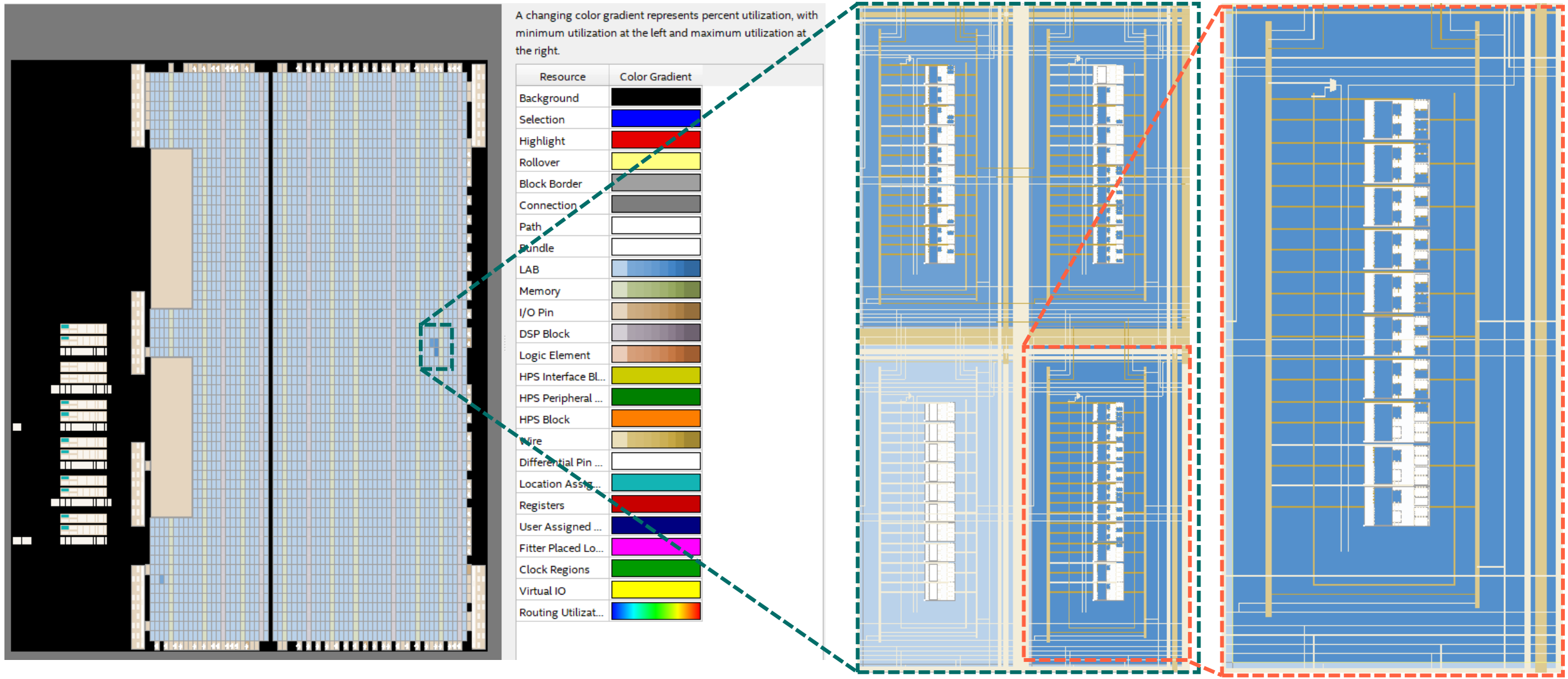
Architecture

Cyclone V 5CGXFC5C6F27C7	
ALMs	29080
I/Os	364
GPIOs	336
Memory bits	4 567 040
DSP Blocks	150
Fractional PLLs	6
Global clocks	16



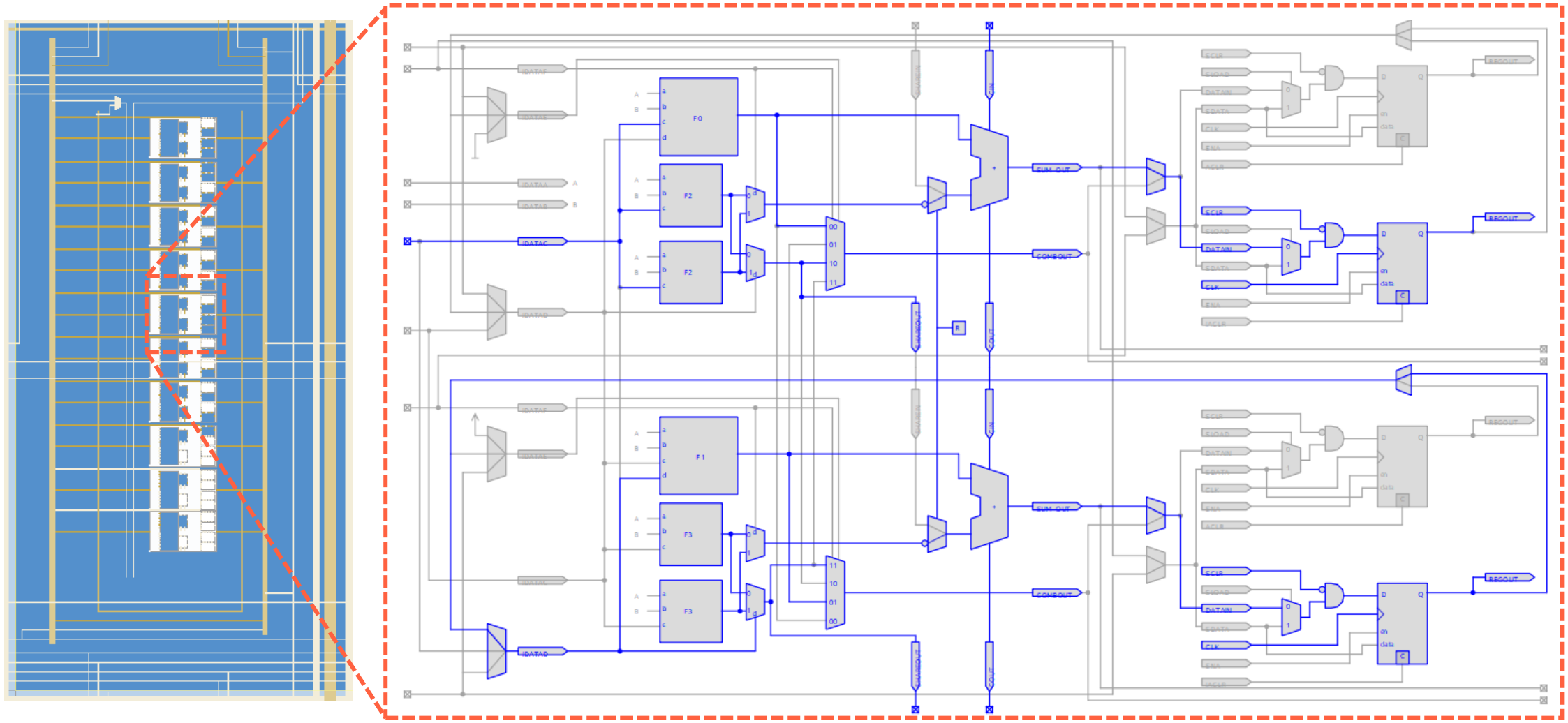
Note : Les horloges sont distribuées par des circuits spécifiques, leur chemin ne doit pas être défini par le concepteur

Vue du *Chip Planner*



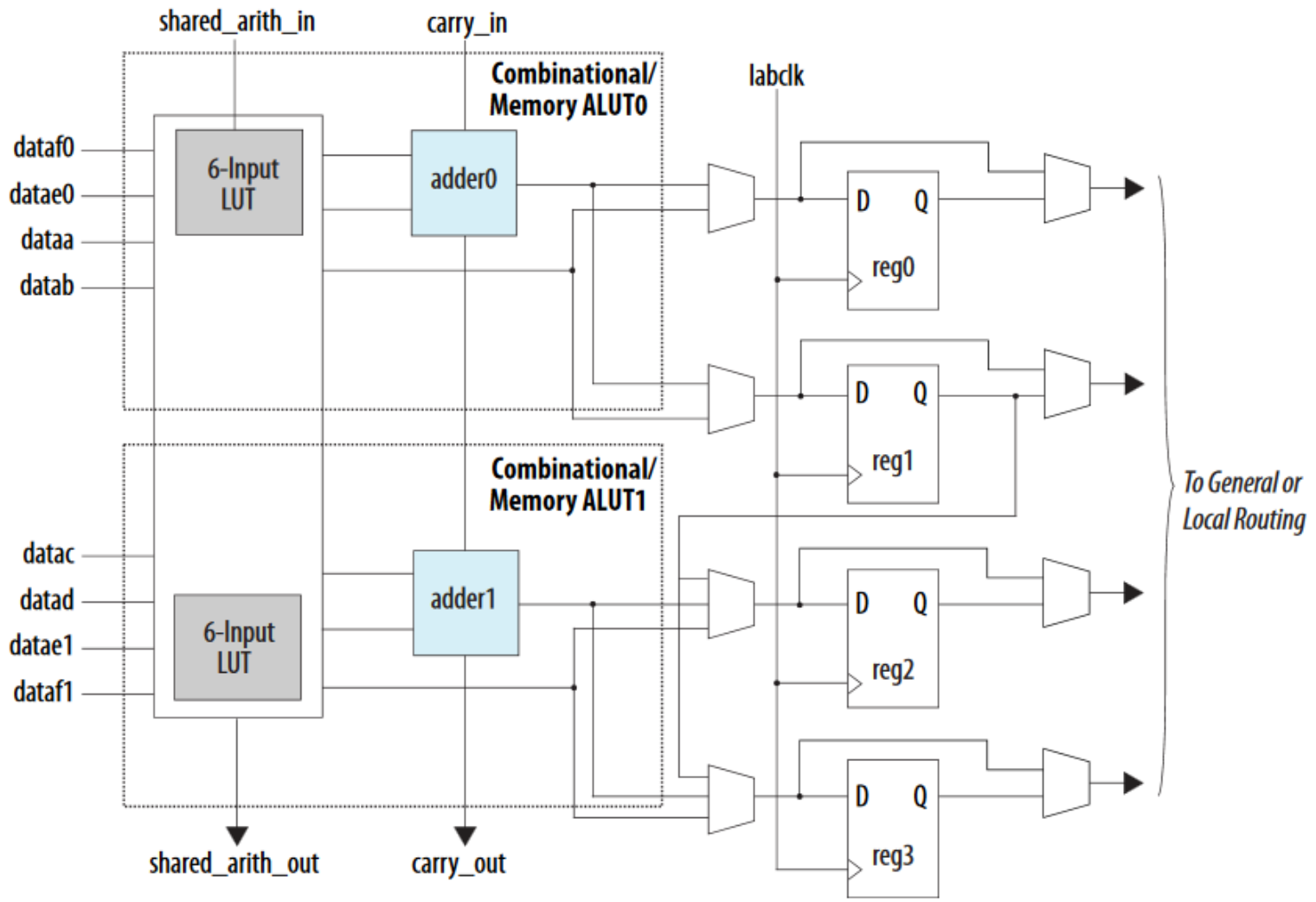
Vue du *Chip Planner :*

Exemple d'une configuration ALM sur le Quartus Chip planner

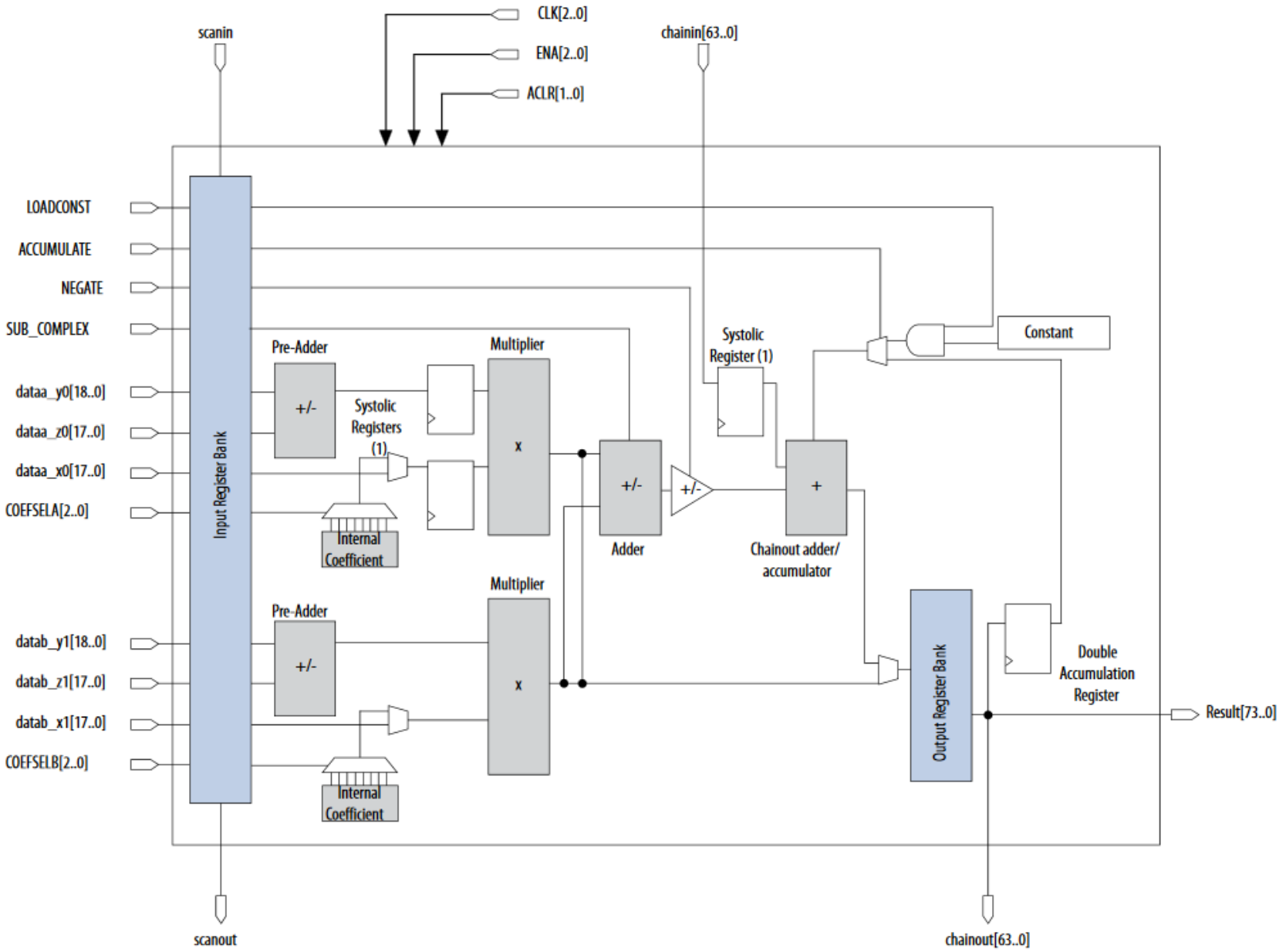


ALM : Adaptive Logic Module

Figure 1-5: ALM High-Level Block Diagram for Cyclone V Devices



DSP : Digital Signal Processor



Ressources du Cyclone V

Cyclone V 5CGXFC5C6F27C7	
ALMs	29080
I/Os	364
GPIOs	336
Memory bits	4 567 040
DSP Blocks	150
Fractional PLLs	6
Global clocks	16

Le langage de description matériel VHDL

1. Introduction
2. Structure d'un fichier VHDL, entités et architectures
3. Les opérateurs
4. Les instructions
5. Procédés séquentiels
6. Les *testbench*

Le langage de description matériel VHDL

1. **Introduction**
2. Structure d'un fichier VHDL, entités et architectures
3. Les opérateurs
4. Les instructions
5. Procédés séquentiels
6. Les *testbench*

Le langage VHDL

- **VHDL** : VHSIC Hardware Description Language
- **VHSIC** : Very High Speed Integrated Circuit

- Développé dans les années 80
  - D'abord pour la simulation de circuit électroniques
  - Ensuite, étendu à la synthèse de circuits numériques

- Langage de « haut niveau » (n'est pas lié à la cible technologique)
- Le VHDL décrit ou définit le fonctionnement d'un système

Le langage VHDL

Contextualisé dans les différents niveaux de description :

Description Comportementale

Algorithmique
- Algorigramme, pseudo-code, ou syntaxe Python, C ...
- Mathématiquement correct, mais non synthétisable

Architecturelle
- Schémas-blocs
- Description des entrées/sorties de chaque bloc
- Aucune description comportementale des blocs

Description Structurelle

Niveau RTL (Register-Transfer-Level)
- Description comportementale d'un système
- Description de chaque signal
- Prise en compte des horloges
- Abstraction des ressources propres au PLD
- **C'est le niveau du VHDL**

Niveau portes logiques
- Description de l'implémentation à l'échelle des ressources disponibles (portes logiques, registres, multiplexeurs, etc.)

Niveau transistor
- Description de l'implémentation à l'échelle physique
- Réservé à la conception d'ASICs

Le langage VHDL

Vous connaissez des cours d'électronique numérique différentes descriptions équivalentes de fonctions logiques combinatoires et séquentielles :

Logique combinatoire :

- Logigramme (représentation graphique)
- Table de vérité
- Equation logique

Logique séquentielle :

- Logigramme
- Diagramme d'états
- Chronogramme
- Tables de transition
- Equations caractéristiques

A même titre, le VHDL est une description syntaxique du comportement d'un système combinatoire ou séquentiel, strictement équivalente à ses autres descriptions

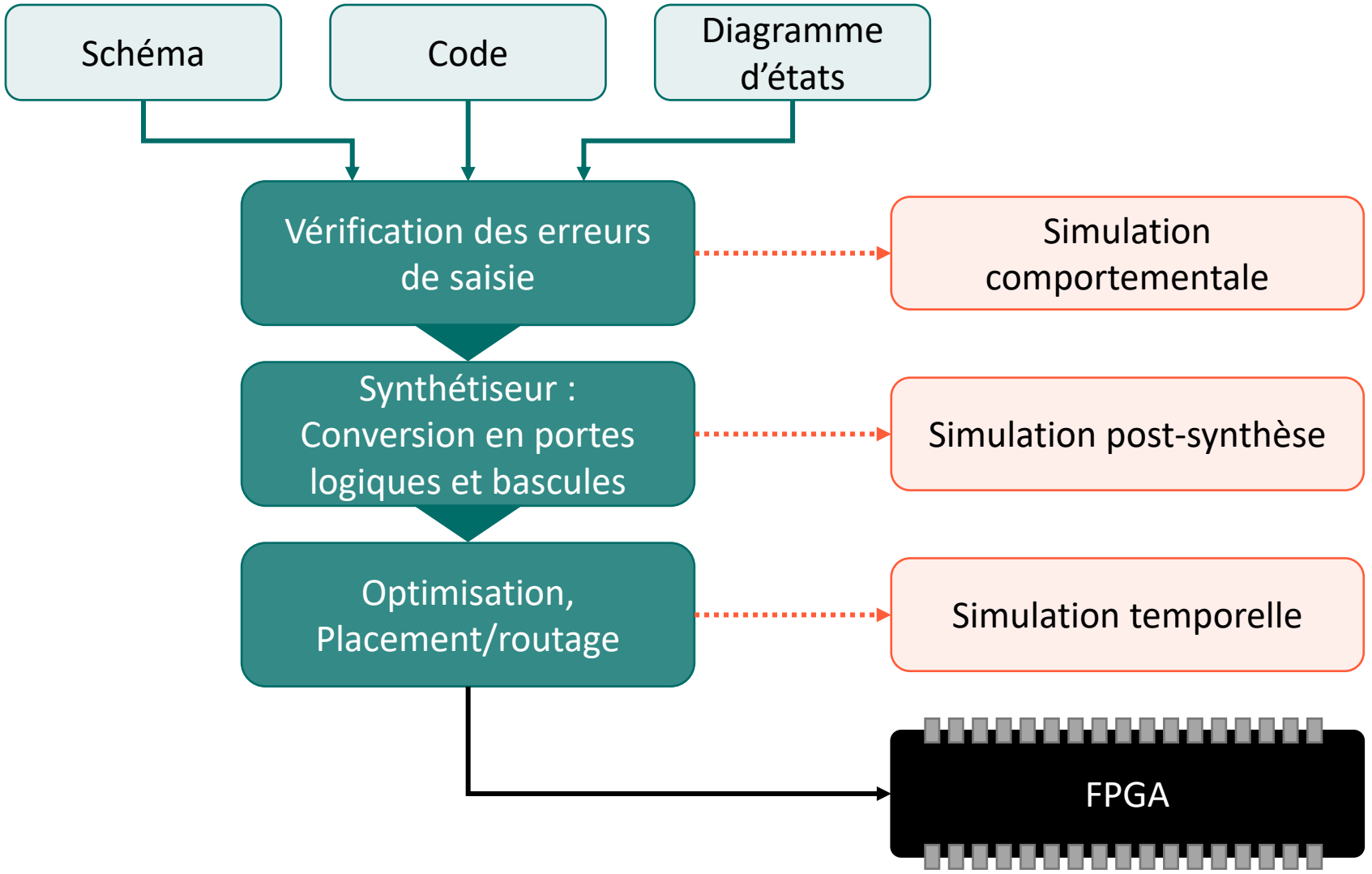
Pour une documentation détaillée du langage VHDL, voir :
https://vhdlguide.readthedocs.io/en/latest/index.html

Développement d'un PLD

Pour réaliser la description d'un circuit, on fera appel à tous les niveaux de conception vus précédemment :

- **Description algorithmique** et simulation comportementale
- Partitionnement de l'architecture en **blocs**
- Description **RTL de chaque bloc**
  - Définition des **Entrées/Sorties**
  - Description **comportementale**
- **Synthèse** de l'ensemble de l'architecture en fonction des ressources du PLD
- Génération d'une **netlist** propre aux ressources du PLD

Développement d'un PLD



Le langage de description matériel VHDL

1. Introduction
2. **Structure d'un fichier VHDL, entités et architectures**
3. Les opérateurs
4. Les instructions
5. Procédés séquentiels
6. Les *testbench*

Structure d'un fichier VHLD

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

-- Multiplexeur 2 vers 1

entity MUX2 is
     port (
          a : in std_logic;
          b : in std_logic;
          s : in std_logic;
          y : out std_logic
     );
end MUX2;

architecture behavioral of MUX2 is
begin
     y <= (a and not(s)) or (b and s);
end architecture;
```

Déclaration des **bibliothèques**
A inclure dans chaque fichier

Commentaire, déclaré par **--**

Déclaration de **l'entité** du composant

Déclaration de **l'architecture** du composant

Attention, l'entité et l'architecture sont deux parties **indissociables** d'une description VHDL d'un composant

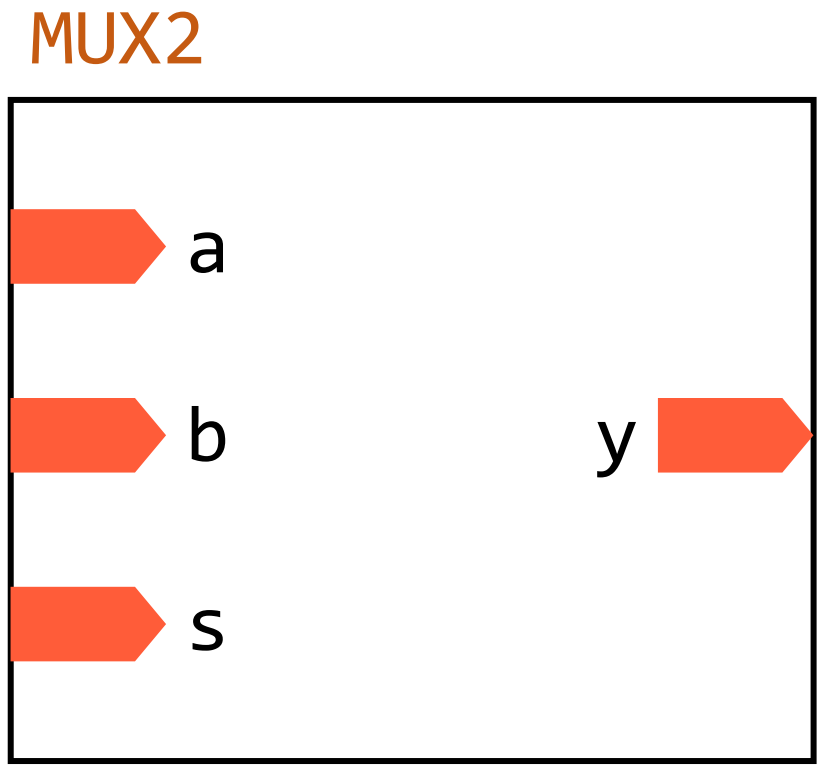
Zoom sur la déclaration d'entité

```
...

entity NOM_DE_L_ENTITE is
    port (
        -- Description des signaux d'entrée/sortie
    );
end NOM_DE_L_ENTITE;

...
```

Une **entité** = un bloc défini par ses **entrées/sorties**

Zoom sur la déclaration d'entité

```
...

entity MUX2 is
    port (
        a : in std_logic;
        b : in std_logic;
        s : in std_logic;
        y : out std_logic
    );
end MUX2;

...
```



Remarque : Pas de **;** à la dernière ligne de l'instruction port

L'instruction **port**

direction :

> In : entrée
> Out : sortie
> Inout : entrée/sortie
> Buffer : sortie, utilisée comme entrée dans la description

type :

> std_logic : signal à 1 bit
> std_logic_vector : bus à plusieurs bits

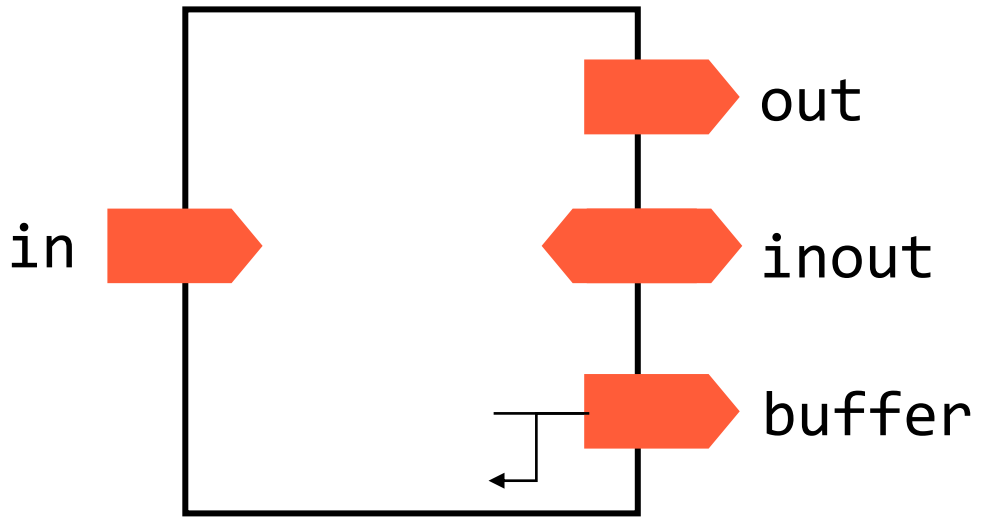
Un signal de type std_logic peut prendre les valeurs :

> '0' ou 'L' : niveau bas
> '1' ou 'H' : niveau haut
> 'Z' : état haute impédance
> '-' : état quelconque

Exemples :

```
A : in  std_logic;
B : out std_logic_vector (4 downto 0);
```

```
...
entity NOM_ENTITE is
    port (
        nom_signal : direction type;
        ...
    );
end NOM_ENTITE;
...
```



Zoom sur la déclaration d'architecture

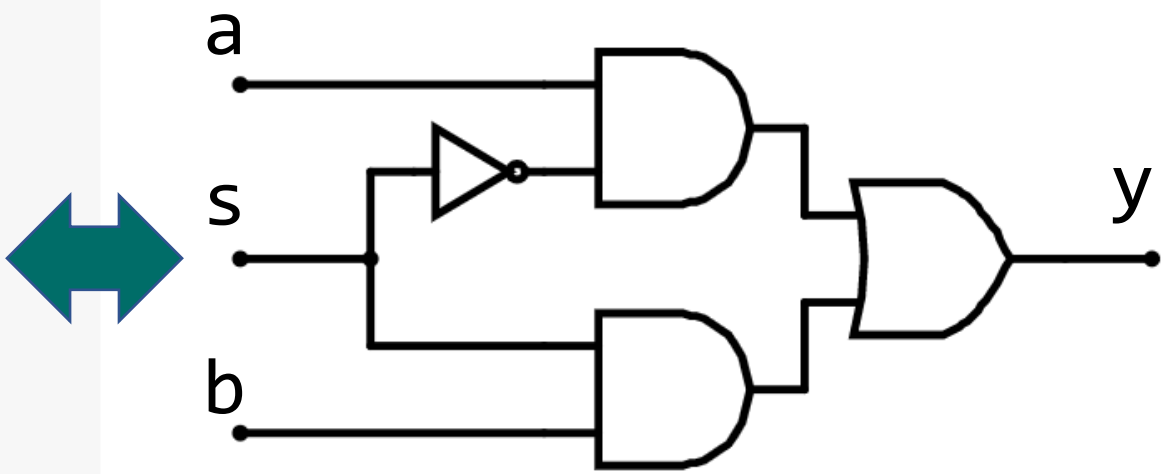
```
...

architecture nom_architecture of nom_entite is
-- Déclaration des signaux
begin
    -- Description de la logique
end architecture;
...
```

Une **architecture** = la description du fonctionnement d'une entité

Zoom sur la déclaration d'architecture

Entité MUX2 définie précédemment

```
...

architecture behavioral of MUX2 is
begin
    y <= (a and not(s)) or (b and s);
end architecture;

...
```



Note importante sur l'architecture

- Les instructions utilisées dans l'architecture sont dites **concurrentes** :
  - **Elles sont « évaluées » simultanément, indépendamment de leur ordre d'écriture**

- Exemples d'instructions concurrentes (voir chapitre suivant) :
  - Assignation de signal
  - Assignation conditionnelle
  - Assignation sélective
  - Instantiation de composants
  - Process

Le langage de description matériel VHDL

1. Introduction
2. Structure d'un fichier VHDL, entités et architectures
3. **Les opérateurs**
4. Les instructions
5. Procédés séquentiels
6. Les *testbench*

Opérateur **d'affectation**

L'opérateur d'affectation est la flèche : **<=**

Sur des signaux de 1 bit on pourra assigner :
'1' pour le niveau haut
'0' pour le niveau bas
'Z' pour l'état haute impédance
'-' pour un état intéterminé

Sur des signaux composés de plusieurs bits (bus) :
- On utilisera les guillemets : "01001"
- On pourra assigner en base **binaire**, **hexadécimale**, ou **octale** mais pas en décimal

Exemples :

```
Y  <= A and B; -- Porte ET
X1 <= "1100";  -- Assignation de 12 en binaire
X2 <= X"C";    -- Assignation de 12 en hexadécimal
X3 <= O"14";   -- Assignation de 12 en octal
```

Opérateurs de **concaténation**

L'opérateur de concaténation est :

La **concaténation** permet de joindre plusieurs signaux entre eux

Exemple :

```
-- Soit A = '1' de type 1 bit
-- Soit B = "000" de type 3 bits
-- Soit C = "11" de type 2 bits
-- Soit Y de type 8 bits

Y <= A & B & C & "01"; -- Y vaut "10001101" après affectation
--   |   |   |    |
--   1  000  11   01
```

Opérateurs **logiques**

Opérateur	Symbole VHDL
Et	`and`
Non et	`nand`
Ou	`or`
Non ou	`nor`
Ou exclusif	`xor`
Non ou exclusif	`xnor`
Non	`not`
Décalage à gauche	`sll`
Décalage à droite	`srl`
Rotation à gauche	`rol`
Rotation à droite	`ror`

Opérateurs **arithmétiques**

Opérateur	Symbole VHDL
Addition	+
Soustraction	-
Multiplication	*
Division	/

Opérateurs **relationnels**

Opérateur	Symbole VHDL
Egal	=
Différent	/=
Inférieur à	<
Inférieur ou égal à	<=
Supérieur à	>
Supérieur ou égal à	>=

Le langage de description matériel VHDL

1. Introduction
2. Structure d'un fichier VHDL, entités et architectures
3. Les opérateurs
4. **Les instructions**
5. Procédés séquentiels
6. Les *testbench*

Instructions de base : **affectation conditionnelle**

Modifie l'affectation d'un signal en fonction d'une condition logique.

Syntaxe :

```
signal <= expression when condition
          [else expression when condition]
          [else expression];
```

Exemples :

```
-- MUX 2:1
S <= E0 when (SEL='0') else E1;
```

```
-- MUX 4:1
S <= E0 when (SEL="00") else
     E1 when (SEL="01") else
     E2 when (SEL="10") else
     E3 when (SEL="11")
     else '0';
```

Instructions de base : **affectation sélective**

Permet d'affecter différentes valeurs selon un signal de sélection :

Syntaxe :

```
with signal_de_selection select
     signal <= expression when valeur_de_selection,
     [signal <= expression when valeur_de_selection,]
     [expression when others];
```

Exemple :

```
-- MUX 4:1
with SEL select
     S <= E0 when "00",
          E1 when "01",
          E2 when "10",
          E3 when "11",
          '0' when others;
```

Attention : Ne s'utilise que dans des descriptions combinatoires hors des process

Instructions de base : **instanciation**

Permet d'instancier une entité décrite par ailleurs

Syntaxe :

```
nom_instance : nom_entite port map(
        [ port_name => ] expression, ...
        [ port_name => ] expression, ...
        );
```

Exemple :

```
MUX_1 : entity work.MUX2 port map (
    S => S,
    X0 => X0,
    X1 => X1,
    Y => Y
);
```

Le langage de description matériel VHDL

1. Introduction
2. Structure d'un fichier VHDL, entités et architectures
3. Les opérateurs
4. Les instructions
5. **Procédés séquentiels**
6. Les *testbench*

Les `process`

- Un `process` décrit une partie d'un circuit dont les instructions sont exécutées **séquentiellement**.

- L'exécution d'un process est **déclenchée** par un ou plusieurs changement(s) d'état(s) (**event**) de signaux logiques. Ceux-ci sont définis dans la **liste de sensibilités** lors de la déclaration du process.

Les process

Exécute séquentiellement des instructions, sur déclenchement selon des signaux définis dans une liste de sensibilités

Syntaxe :

```
[nom_du_process :] process(liste_des_sensibilites)
begin
-- instructions du process
end process [nom_du_process] ;
```

Exemple :

```
entity DFlipFlop is
    port (
        D,clk : in std_logic;
        S     : out std_logic);
end DFlipFlop;
```

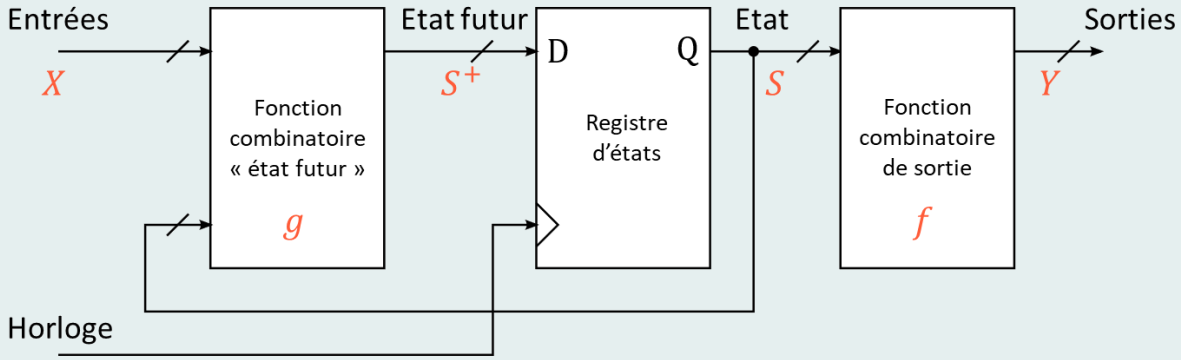
```
architecture behavioral of DFlipFlop is
    begin
        proc_dff: process (clk)
            begin
             if (clk'event and clk ='1') then
                  S <= D;
              end if;
    end process proc_dff;
end behavioral;
```

Déclenchement sur front

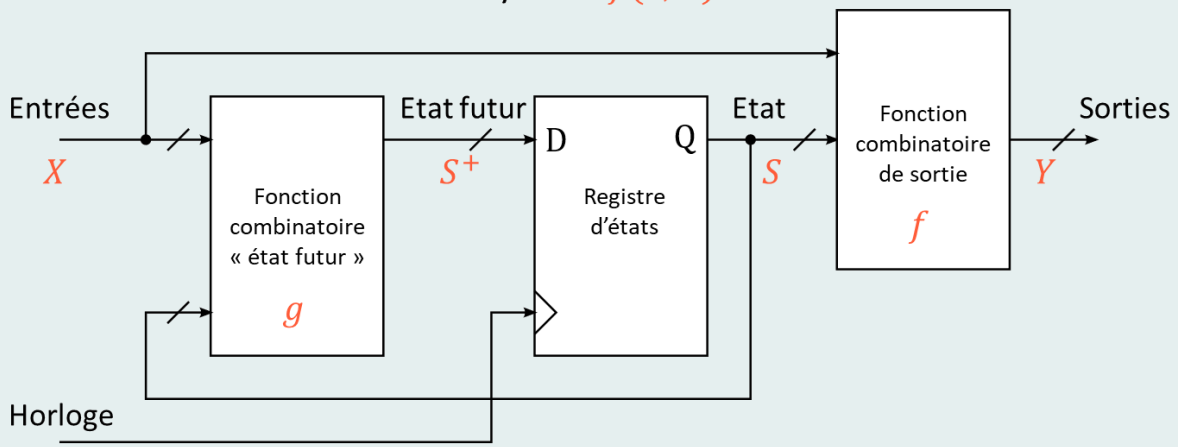
Les machines à états finis

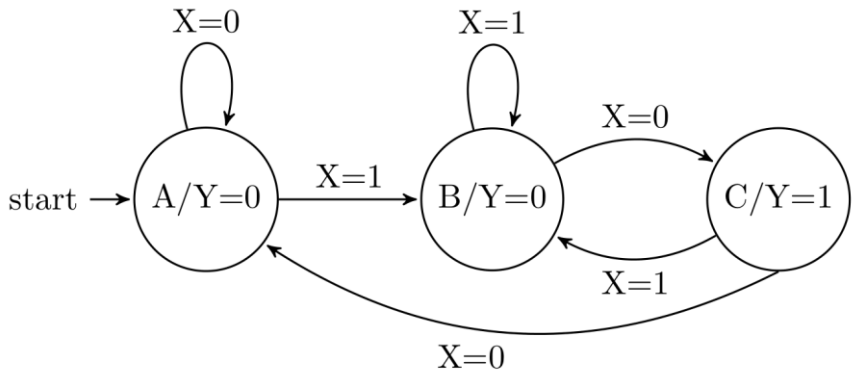
- Rappel : Machine de Moore et Machine de Mealy

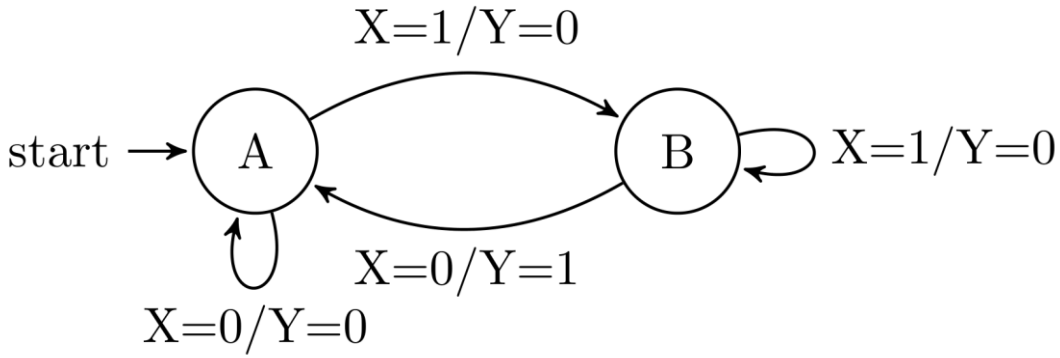
Structure d'une machine de Moore : $Y = f(S)$



Structure d'une machine de Mealy : $Y = f(S, X)$





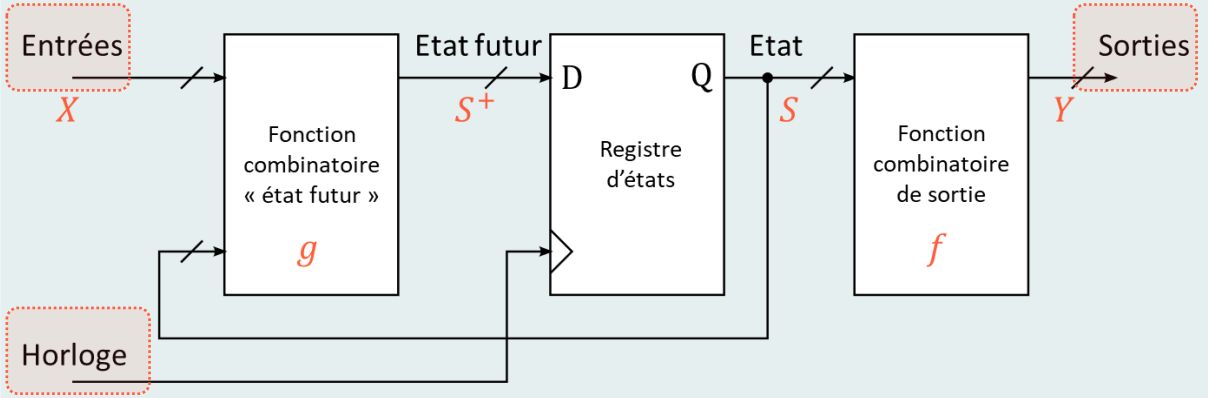


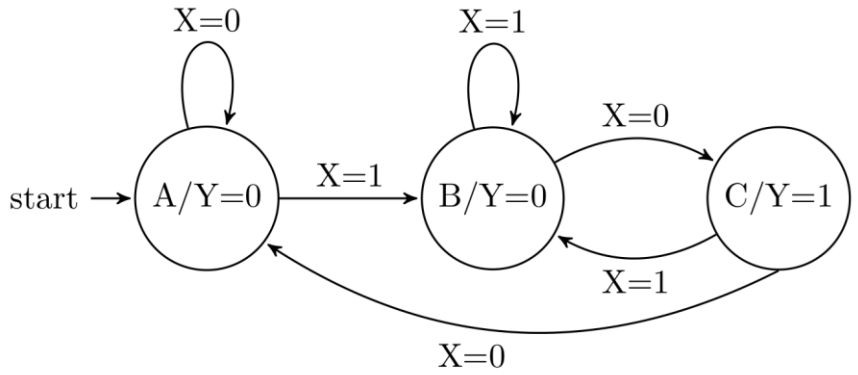
Implémentation d'une MEF

- Exemple d'un détecteur de séquence 10 (machine de Moore)

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

-- Description de l'entité
entity fsm_10detect_moore is
    port(
        clk : in std_logic;
        rst : in std_logic;
        X   : in std_logic;
        Y   : out std_logic
    );
end entity;
-- Suite diapo suivante
```

Structure d'une machine de Moore : $Y = f(S)$



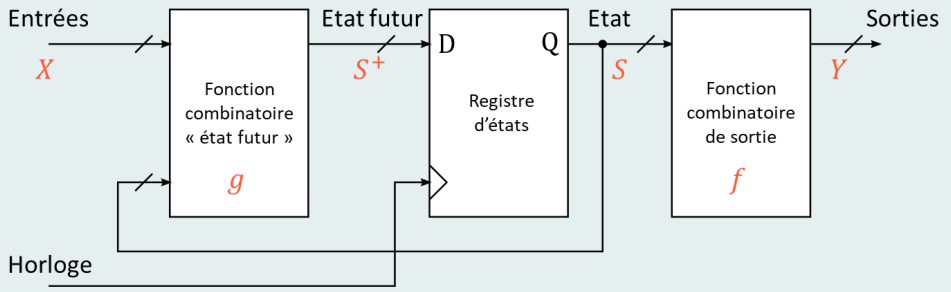


Implémentation d'une MEF

```
architecture behavioral of fsm_10detect_moore is
    type state_type is (A, B, C);
    signal current_state : state_type;
    signal future_state  : state_type;

    begin

    -- Description du registre d'états
    -- Fonction SEQUENTIELLE
    -- (Voir diapos suivantes [Bloc 1])

    -- Description du calcul de l'état futur
    -- Fonction COMBINATOIRE
    -- (Voir diapos suivantes [Bloc 2])

    -- Description du calcul des sorties
    -- Fonction COMBINATOIRE
    -- (Voir diapos suivantes [Bloc 3])
end behavioral;
```

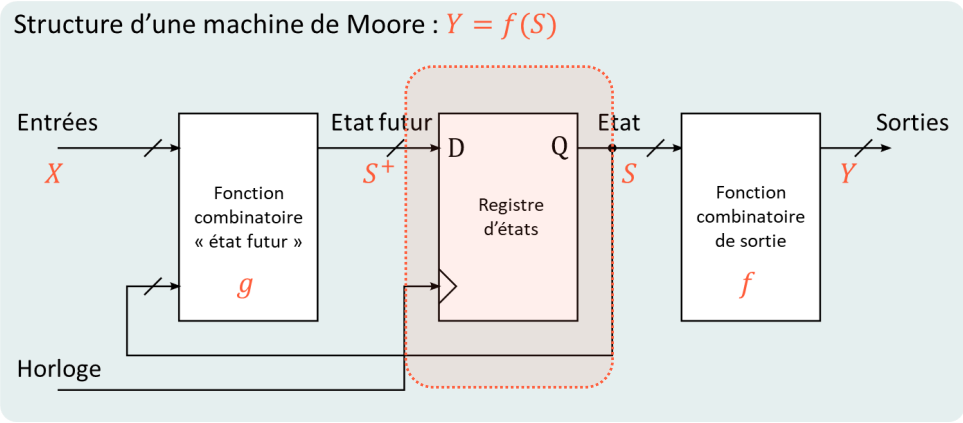
Déclaration des **états possibles**, et des signaux **état présent** et **état futur**

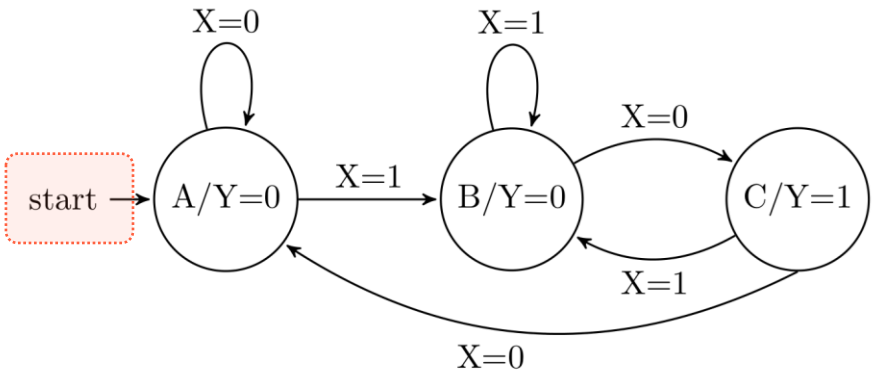
Structure d'une machine de Moore : $Y = f(S)$



Implémentation d'une MEF [Bloc 1]

```
-- Description du registre d'états
-- Fonction SEQUENTIELLE
process(clk, rst)
begin
    if rst = '1' then
        current_state <= A;
    elsif (clk'event and clk ='1') then
        current_state <= future_state;
    end if;
end process;
```

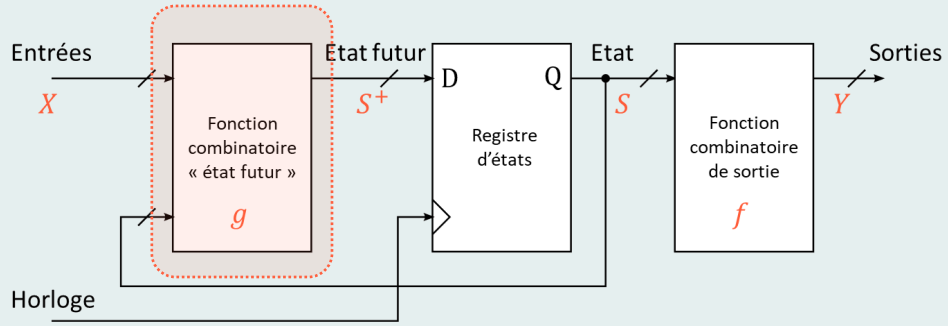


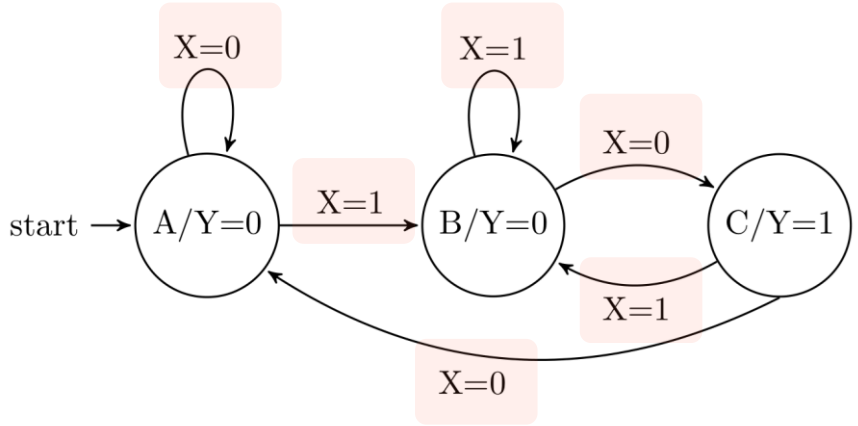


Implémentation d'une MEF [Bloc 2]

```
-- Description du calcul de l'état futur
-- Fonction COMBINATOIRE
process(X, current_state)
begin
    case current_state is
        when A =>
            if X = '0' then future_state <= A;
            else            future_state <= B;
            end if;
        when B =>
            if X = '0' then future_state <= C;
            else            future_state <= B;
            end if;
        when C =>
            if X = '0' then future_state <= A;
            else            future_state <= B;
            end if;
        when others =>
            future_state <= current_state;
    end case;
end process;
```

Structure d'une machine de Moore : $Y = f(S)$





Implémentation d'une MEF [Bloc 3]

```
-- Description du calcul des sorties
-- Fonction COMBINATOIRE
process(current_state)
begin
    case current_state is
        when A => Y <= '0';
        when B => Y <= '0';
        when C => Y <= '1';
        when others => Y <= 'X';
    end case;
end process;
```

Structure d'une machine de Moore : $Y = f(S)$