

CHAPITRE 2

Structure d'une description VHDL

Plan du Chapitre

- Introduction au langage VHDL
- Structure d'une description VHDL
 - Déclaration des bibliothèques
 - Déclaration de l'entité type
 - Déclaration de l'architecture

Introduction au langage VHDL

VHDL : **V**HSIC **H**ardware **D**escription **L**anguage

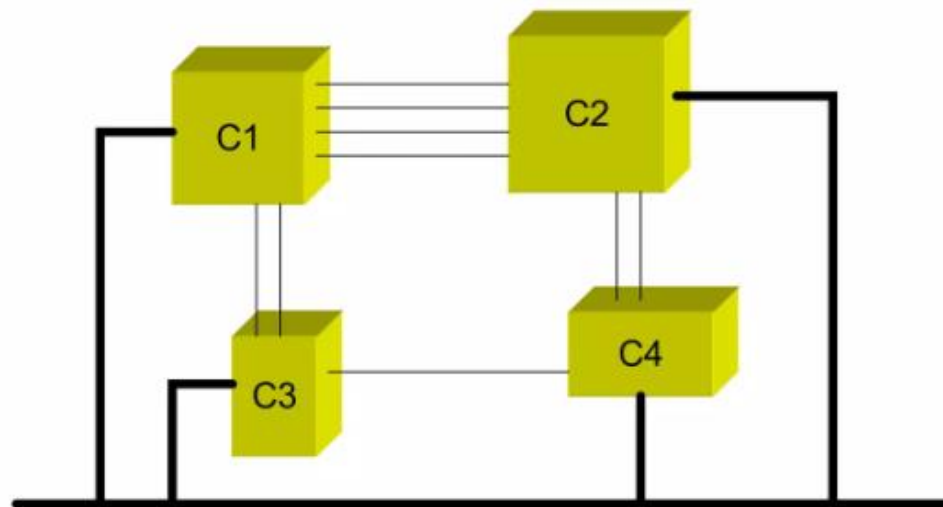
VHSIC: Very High Speed Integrated Circuit

HDL: c'est un moyen de décrire un système matériel:

qu'est qu'un système matériel ?

en général, il s'agit d'un schéma mettant en œuvre :

- un certain nombre de composants
- des connexion entre composants



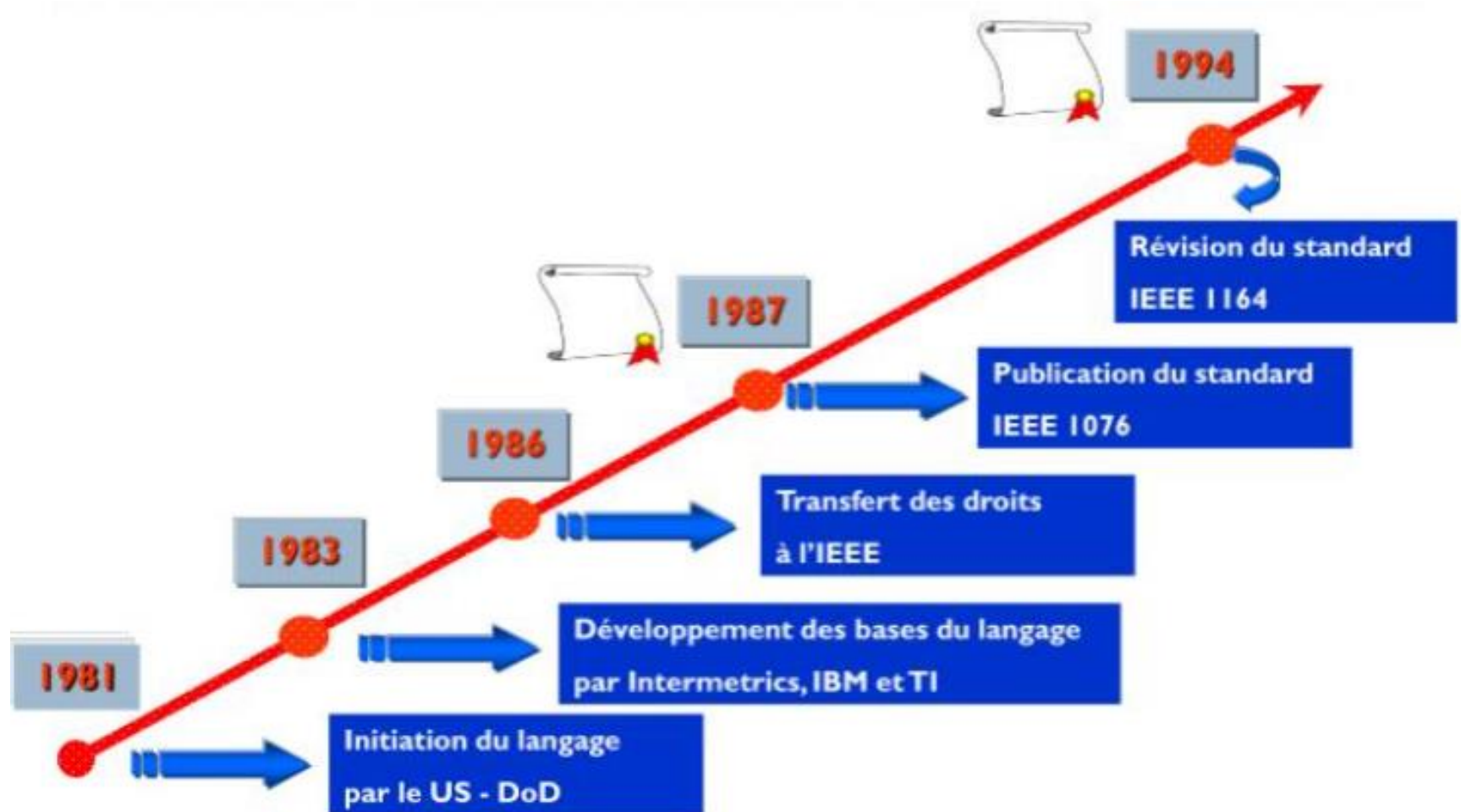
Introduction au langage VHDL

VHDL est une initiative de la Défense américaine dans les années 1980. C'est un langage de description de matériel utilisé pour modéliser et simuler des systèmes électroniques numériques.

- ✓ Il autorise plusieurs méthodologies de conception (comportemental, structurel, Flot de données) tout en étant d'un très haut niveau d'abstraction en électronique (il est indépendant de la technologie utilisée : FPGA, CPLD, ASIC, etc.). Cette abstraction permet d'ailleurs de le simuler sur ordinateur avant de programmer la moindre puce.
- ✓ Le VHDL est particulièrement apprécié pour sa capacité à décrire le comportement et l'architecture des systèmes électroniques de manière exécutable, ce qui permet une vérification par simulation avant la réalisation physique du circuit

Introduction au langage VHDL

Historique



Structure d'une description VHDL

Une description VHDL est composée de 3 parties :

- Bibliothèques
- L'entité (ENTITY), elle définit les entrées et sorties.
- L'architecture (ARCHITECTURE), elle contient les instructions VHDL permettant de réaliser le fonctionnement attendu du circuit logique par une description comportementale ou structurelle.


Déclaration des Bibliothèques

Toute description VHDL nécessitent la déclaration d'une ou plusieurs bibliothèques qui incluent la définition des types de signaux et des opérations que l'on peut effectuer entre ceux-ci. L'IEEE (Institut of Electrical and Electronics Engineers) les a normalisées et plus particulièrement la bibliothèque **IEEE1164**. les bibliothèques contiennent les définitions des types de signaux électroniques, des fonctions et des sous programmes permettant de réaliser des opérations arithmétiques et logiques,...

```
Library ieee;  
Use ieee.std_logic_1164.all;  
Use ieee.numeric_std.all;  
Use ieee.std_logic_unsigned.all;  
-- cette dernière bibliothèque est souvent utilisée pour l'écriture de compteurs
```

La directive **Use** permet de sélectionner les bibliothèques à utiliser.

Déclaration de l'entité

Elle permet de définir le **NOM** de la description VHDL ainsi que les entrées et sorties utilisées, comme suit: 

```
Syntaxe:  
entity NOM_DE_L_ENTITE is  
    port ( Description des signaux d'entrées /sorties ... );  
end NOM_DE_L_ENTITE;  
  
Exemple :  
entity SEQUENCEMENT is  
port (  
    CLOCK      : in std_logic;  
    RESET       : in std_logic;  
    Q           : out std_logic_vector(1 downto 0)  
);  
end SEQUENCEMENT;
```

Remarque : Il faut toujours vérifier de ne pas oublier le point virgule

L'instruction **port** décrit les entrées / sorties de circuit logique :

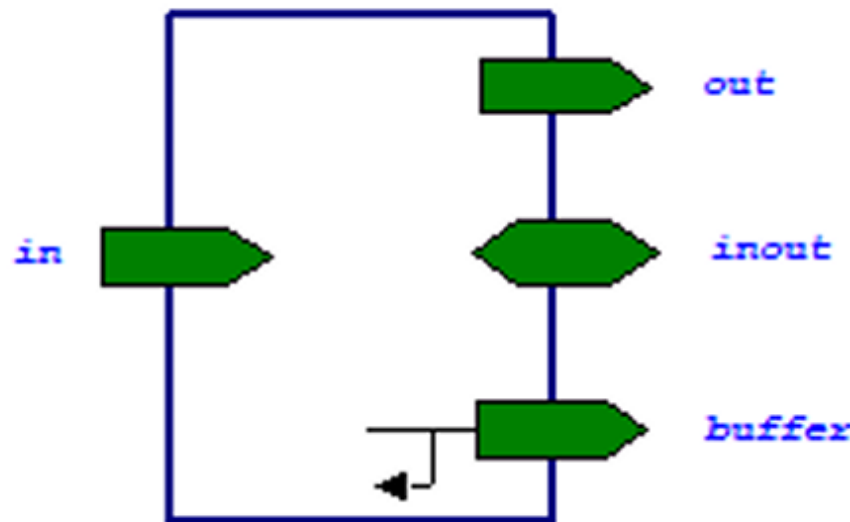
Syntaxe: **NOM_DU_SIGNAL**: **sens** **type**;

Exemple: **CLOCK** : in std_logic;

BUS : out std_logic_vector (7 downto 0);

On doit définir pour chaque signal : le **NOM_DU_SIGNAL**, le **sens** et le **type**.

- **Le NOM_DU_SIGNAL**: Il est composé de caractères dont le premier caractère doit être une lettre et sa longueur ne doit pas dépasser une ligne de code. VHDL n'est pas sensible à la « casse », c'est à dire qu'il ne fait pas la distinction entre les majuscules et les minuscules.
- **Le SENS du Signal**:
 - in : pour un signal en entrée.
 - out : pour un signal en sortie.
 - inout : pour un signal en entrée/sortie.
 - buffer : pour un signal en sortie mais utilisé comme entrée dans la description.



- Le **TYPE** utilisé pour les signaux d'entrées/sorties est :
- le **std_logic** pour un signal.
 - le **std_logic_vector** pour un bus composé de plusieurs signaux.

Exemple: un bus bidirectionnel de 5 bits s'écrira :

LATCH: inout std_logic_vector (4 downto 0) ;

Où LATCH(4) correspond au MSB et LATCH(0) correspond au LSB.

Les valeurs que peuvent prendre un signal de type **std_logic** sont :

- '0' ou 'L' : pour un niveau bas.
- '1' ou 'H' : pour un niveau haut.
- 'Z' : pour état haute impédance.
- '-' : Quelconque, c'est à dire n'importe quelle valeur.

Déclaration de l'Architecture

L'architecture décrit le fonctionnement souhaité pour un circuit ou une partie du circuit. En effet, le fonctionnement d'un circuit est, généralement, décrit par plusieurs modules VHDL.

→ **Un module c'est le couple ENTITE/ARCHITECTURE.**

Dans le cas de simples programmes on trouve souvent un seul module. L'architecture établit, à travers les instructions, les relations entre les entrées et les sorties. On peut avoir un fonctionnement purement combinatoire ou séquentiel voire les deux séquentiel et combinatoire.

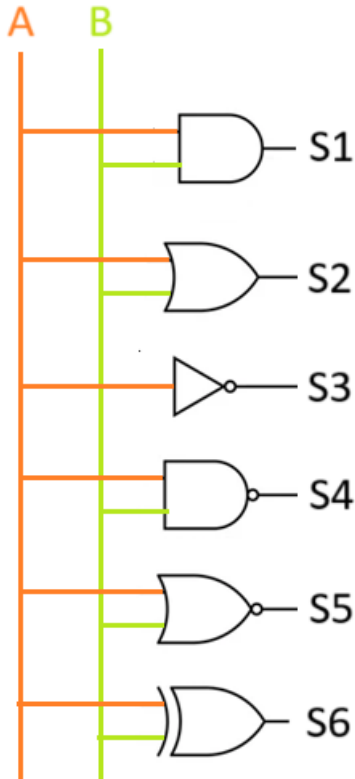
Syntaxe: `architecture Nom_de_L_Architecture of Nom_de_L_ENTITE is`
`begin`
Description de fonctionnement des signaux sorties en fonction des entrées;
`end Nom_de_L_Architecture;`

Exemple: addition des deux entrées A et B et le stockage de résultat dans la sortie S.

```
architecture Addition of Entity is
begin
    S<= A + B;
end Addition;
```

➤ **On passe à l'architecture après avoir déclaré l'entité.**

Exemple 1: Description fonctionnelle des portes logiques:



```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.STD_LOGIC_ARITH.ALL;  
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity Porte is
```

```
    Port ( A , B : in  STD_LOGIC;  
          S1 : out  STD_LOGIC;  
          S2 : out  STD_LOGIC;  
          S3 : out  STD_LOGIC;  
          S4 : out  STD_LOGIC;  
          S5 : out  STD_LOGIC;  
          S6 : out  STD_LOGIC);
```

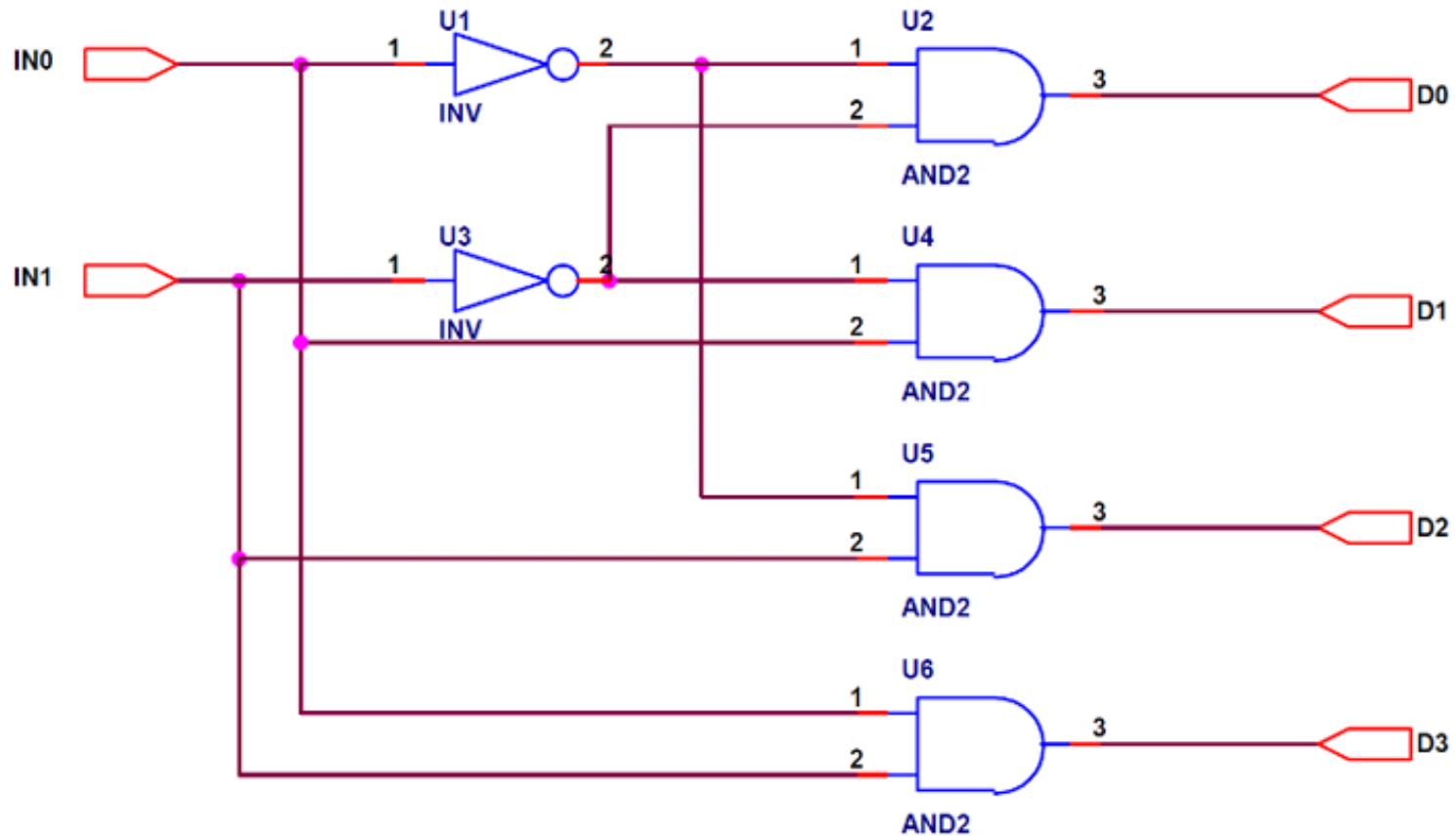
```
end Porte;
```

```
architecture Arch_Porte of Porte is  
begin
```

```
    S1 <= A and B;  
    S2 <= A or B;  
    S3 <= not (A);  
    S4 <= A nand B;  
    S5 <= A nor B;  
    S6 <= A xor B;
```

```
end Arch_Porte;
```

Exemple 2: Schéma d'un décodeur 2 vers 4 à base des portes logiques:



Ecrire un programme VHDL pour crée ce décodeur.

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

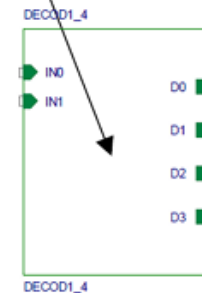
Déclaration des bibliothèques

```
-- décodeur
-- Un parmi quatre
```

Commentaires, en VHDL ils commencent par --

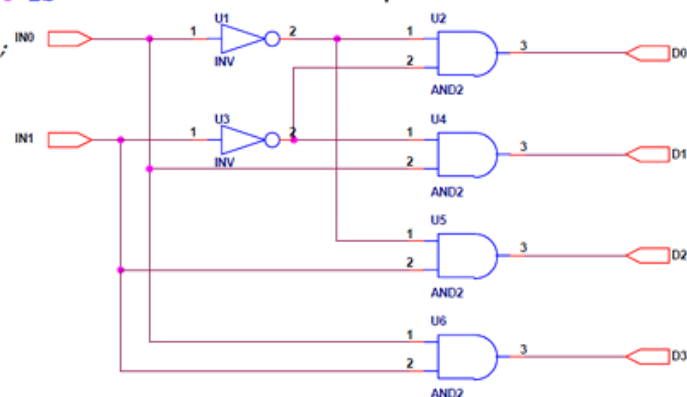
Déclaration de l'entité du décodeur
Correspondance schématique

```
entity DECOD1_4 is
    port(IN0, IN1: in std_logic;
          D0, D1, D2, D3: out std_logic);
end DECOD1_4;
```



Déclaration de l'architecture du décodeur
Correspondance schématique

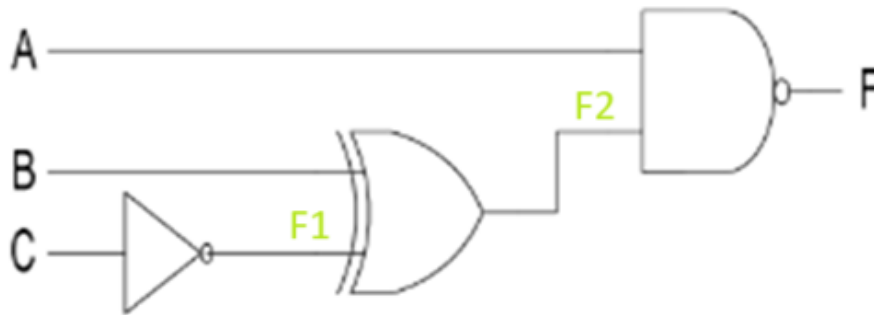
```
architecture DESCRIPTION of DECOD1_4 is
begin
    D0 <= (not(IN1) and not(IN0));
    D1 <= (not(IN1) and IN0);
    D2 <= (IN1 and not(IN0));
    D3 <= (IN1 and IN0);
end DESCRIPTION;
```



Un décodeur 1 parmi 4

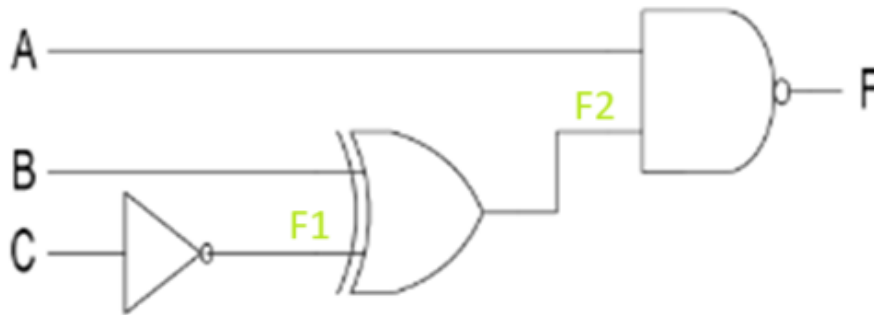
Application 1:

Déterminer le code **VHDL** de ce circuit logique:



Application 1:

Déterminer le code **VHDL** de ce circuit logique:



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Application1 is
    Port ( A , B, C : in STD_LOGIC;
          F : out STD_LOGIC);
end Application1;

architecture Flot_donnee1 of Application1 is
begin
    F <= A nand (B xor (not (C)));
end Flot_donnee1;
```

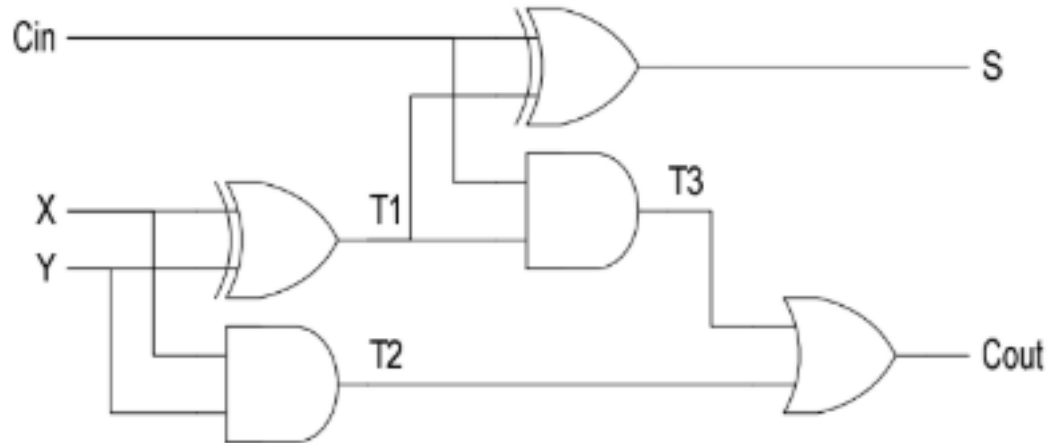
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Application1 is
    Port ( A , B, C : in STD_LOGIC;
          F : out STD_LOGIC);
end Application1;

architecture Arch_App of Application1 is
    Signal F1, F2: STD_LOGIC;
begin
    F1 <= not (C);
    F2 <= A xor B;
    F <= F2 nand A;
end Arch_App;
```

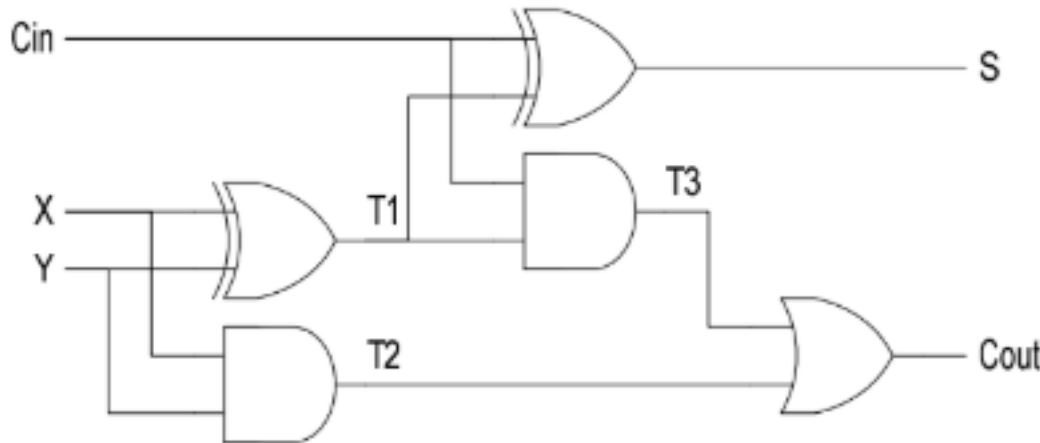

Application 2:

Déterminer le code **VHDL** de ce circuit logique:



Application 2:

Déterminer le code **VHDL** de ce circuit logique:



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Application2 is
    Port ( Cin,X,Y: in STD_LOGIC;
          S,Cout : out STD_LOGIC);
end Application2;

architecture Flot_donnee2 of Application2 is
Begin
    S <= (X xor Y) xor Cin;
    Cout <= (Cin and (X xor Y)) or (X and Y);
end Flot_donnee2;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Application2 is
    Port ( Cin,X,Y: in STD_LOGIC;
          S,Cout : out STD_LOGIC);
end Application2;

architecture Arch_App2 of Application2 is

    Signal T1, T2, T3: STD_LOGIC;

begin
    T1 <= X xor Y;
    T2 <= X and Y;
    T3 <= Cin and T1;
    S <= T1 xor Cin;
    Cout <= T3 or T2;

end Arch_App2;
```