TP Chapitre 01 : Construction des Portes Logiques

Objectifs pratiques

- Écrire du code HDL
- Implémenter les portes élémentaires
- Tester avec le simulateur
- Construire le Mux et DMux

Durée estimée : 2h

Prérequis : TD Chapitre 01 terminé

Préparation

Accès au Simulateur

👉 **Ouvrir le Simulateur HDL**

Allez dans la section **HDL Progression** pour accéder aux exercices.

Alternative locale :

```
cd web
npm install
npm run dev
# → http://localhost:5173
```

Rappel : Structure HDL

```
entity NomPuce is
  port(
    entree1 : in bit;
    entree2 : in bit;
    sortie  : out bit
  );
end entity;

architecture rtl of NomPuce is
  component AutrePuce
    port(a : in bit; b : in bit; y : out bit);
  end component;
  signal fil_interne : bit;
begin
  u1: AutrePuce port map (a => entree1, b => entree2, y => fil_interne);
  sortie <= fil_interne;
end architecture;
```

Exercice 1 : NOT (Inverseur)

Objectif : Première porte en HDL

Spécification

in	out
0	1
1	0

Indice

Connectez le même signal aux deux entrées d'un NAND.

👉 **Ouvrir l'exercice Inv**

Code à compléter

```
entity Inv is
  port(
    a : in bit;
    y : out bit
  );
end entity;

architecture rtl of Inv is
  component Nand
    port(a : in bit; b : in bit; y : out bit);
  end component;
begin
  -- TODO: Compléter l'implémentation
  -- u1: Nand port map (a => ?, b => ?, y => ?);
end architecture;
```

Validation

✅ Tous les tests passent dans le simulateur

▸ Solution

Exercice 2 : AND

Objectif : Utiliser NOT et NAND

Spécification

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

Indice

```
AND(a, b) = NOT(NAND(a, b))
```

👉 **Ouvrir l'exercice And2**

Code à compléter

```
entity And2 is
  port(
    a : in bit;
    b : in bit;
    y : out bit
  );
end entity;

architecture rtl of And2 is
  component Nand
    port(a : in bit; b : in bit; y : out bit);
  end component;
  component Inv
    port(a : in bit; y : out bit);
  end component;
  signal w : bit;
begin
  -- TODO: Compléter
end architecture;
```

Validation

✅ Tous les tests passent

▸ Solution

Exercice 3 : OR

Objectif : Appliquer De Morgan

Spécification

a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

Indice

`OR(a, b) = NAND(NOT(a), NOT(b))`

👉 **Ouvrir l'exercice Or2**

Code à compléter

```
entity Or2 is
  port(
    a : in bit;
    b : in bit;
    y : out bit
  );
end entity;

architecture rtl of Or2 is
  component Nand
    port(a : in bit; b : in bit; y : out bit);
  end component;
  component Inv
    port(a : in bit; y : out bit);
  end component;
  signal not_a, not_b : bit;
begin
  -- TODO: Compléter
end architecture;
```

▸ Solution

Exercice 4 : XOR

Objectif : Circuit plus complexe

Spécification

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

Formule

```
XOR(a, b) = (a AND NOT b) OR (NOT a AND b)
```

👉 **Ouvrir l'exercice Xor2**

Code à compléter

```
entity Xor2 is
  port(
    a : in bit;
    b : in bit;
    y : out bit
  );
end entity;

architecture rtl of Xor2 is
  component Inv port(a : in bit; y : out bit); end component;
  component And2 port(a, b : in bit; y : out bit); end component;
  component Or2 port(a, b : in bit; y : out bit); end component;
  signal not_a, not_b, w1, w2 : bit;
begin
  -- TODO: Compléter
end architecture;
```

▸ Solution

Exercice 5 : Multiplexeur (Mux)

Objectif : L'aiguilleur de données

Spécification

- Si `sel = 0` alors `y = a`
- Si `sel = 1` alors `y = b`

Formule

```
y = (a AND NOT sel) OR (b AND sel)
```

👉 **Ouvrir l'exercice Mux**

Code à compléter

```
entity Mux is
  port(
    a   : in bit;
    b   : in bit;
    sel : in bit;
    y   : out bit
  );
end entity;

architecture rtl of Mux is
  component Inv port(a : in bit; y : out bit); end component;
  component And2 port(a, b : in bit; y : out bit); end component;
  component Or2 port(a, b : in bit; y : out bit); end component;
  signal not_sel, w1, w2 : bit;
begin
  -- TODO: Compléter
end architecture;
```

▸ Solution

Exercice 6 : Démultiplexeur (DMux)

Objectif : Le routeur de signal

Spécification

- Si `sel = 0` alors `a = in, b = 0`
- Si `sel = 1` alors `a = 0, b = in`

Formules

```
a = in AND NOT sel
b = in AND sel
```

👉 **Ouvrir l'exercice DMux**

Code à compléter

```
entity DMux is
  port(
    input : in bit;
    sel   : in bit;
    a     : out bit;
    b     : out bit
  );
end entity;

architecture rtl of DMux is
  component Inv port(a : in bit; y : out bit); end component;
  component And2 port(a, b : in bit; y : out bit); end component;
  signal not_sel : bit;
begin
  -- TODO: Compléter
end architecture;
```

▸ Solution

Exercice 7 : Or8Way (Défi)

Objectif : Porte multi-entrées

Spécification

Sortie = 1 si **au moins un** des 8 bits d'entrée vaut 1.

```
Or8Way(a[0..7]) = a[0] OR a[1] OR ... OR a[7]
```

Structure en arbre

```
Niveau 1: Or2(a[0],a[1])→t0  Or2(a[2],a[3])→t1 ...
Niveau 2: Or2(t0, t1)→t4     Or2(t2, t3)→t5
Niveau 3: Or2(t4, t5)→y
```

👉 **Ouvrir l'exercice Or8Way**

Code à compléter

```
entity Or8Way is
  port(
    a : in bits(7 downto 0);
    y : out bit
  );
end entity;

architecture rtl of Or8Way is
  component Or2 port(a, b : in bit; y : out bit); end component;
  signal t0, t1, t2, t3, t4, t5 : bit;
begin
  -- Niveau 1
  -- TODO
  -- Niveau 2
  -- TODO
  -- Niveau 3
  -- TODO
end architecture;
```

▸ Solution

Exercice 8 : And32 avec Generate (Défi)

Objectif : Utiliser la boucle `for generate`

Spécification

AND bit-à-bit sur deux bus de 32 bits.

```
entity And32 is
  port(
    a : in bits(31 downto 0);
    b : in bits(31 downto 0);
    y : out bits(31 downto 0)
  );
end entity;

architecture rtl of And32 is
  component And2 port(a, b : in bit; y : out bit); end component;
begin
  gen: for i in 0 to 31 generate
    u: And2 port map (a => a(i), b => b(i), y => y(i));
  end generate;
end architecture;
```

👉 **Ouvrir l'exercice And32**

Question

Combien de portes And2 ce circuit contient-il ?

▸ Réponse

Récapitulatif

Portes implémentées

Porte	Composants utilisés	Difficulté
Inv (NOT)	1 NAND	★
And2	1 NAND + 1 NOT	★
Or2	2 NOT + 1 NAND	★
Xor2	2 NOT + 2 AND + 1 OR	★★
Mux	1 NOT + 2 AND + 1 OR	★★
DMux	1 NOT + 2 AND	★★
Or8Way	7 OR	★★
And32	32 AND (generate)	★★★

Validation Finale

Checklist

Avant de passer au chapitre suivant, vérifiez :

- [ ] Tous les tests de Inv passent
- [ ] Tous les tests de And2 passent
- [ ] Tous les tests de Or2 passent
- [ ] Tous les tests de Xor2 passent
- [ ] Tous les tests de Mux passent
- [ ] Tous les tests de DMux passent

Prochaine étape

➡️ **Chapitre 02 : Arithmétique** — Construire un additionneur et une ALU !

📚 **Référence :** Livre Seed, Chapitre 01 - Logique Booléenne