UE Architecture 2

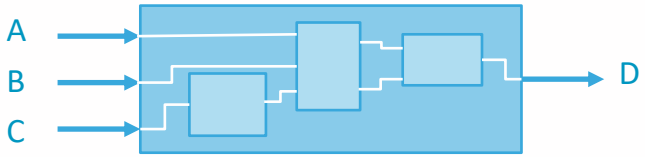
Organisation

- Rappels de logique combinatoire
- Le langage VHDL (1)
  - description de circuits combinatoires
- Logique séquentielle
- Le langage VHDL (2)
  - description de circuits séquentiels

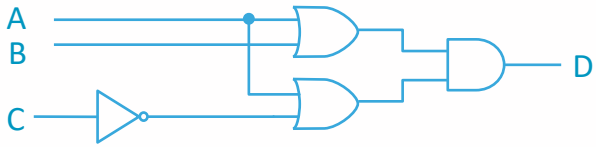
Flot de conception d'un circuit intégré

vision fonctionnelle

⋮



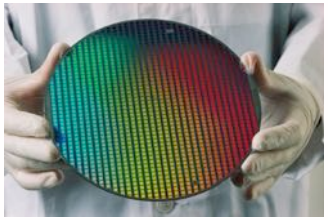
description structurelle ou équations



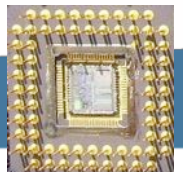
technologie cible

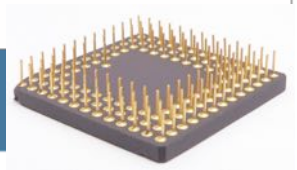
compilateur

circuit



wafer (Silicium)





VHDL ?

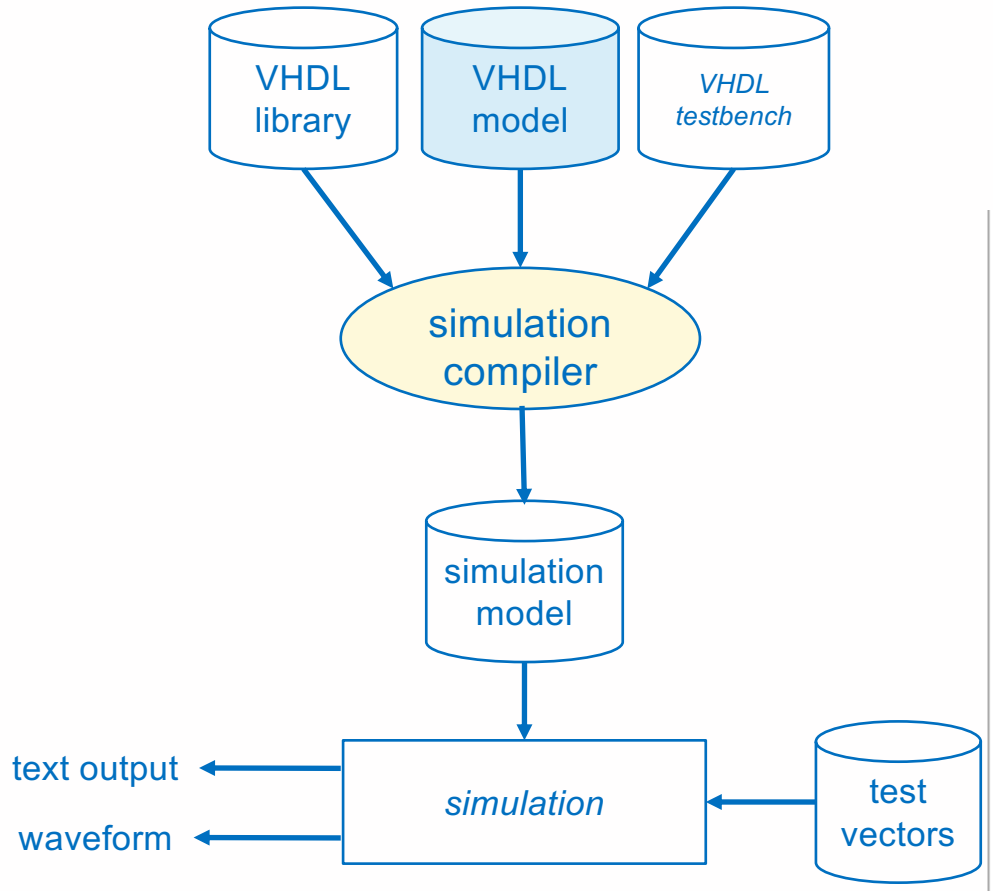
VHDL = VHSIC-HDL

- VHSIC : *Very High Speed Integrated Circuits*
- HDL : *Hardware Description Language*

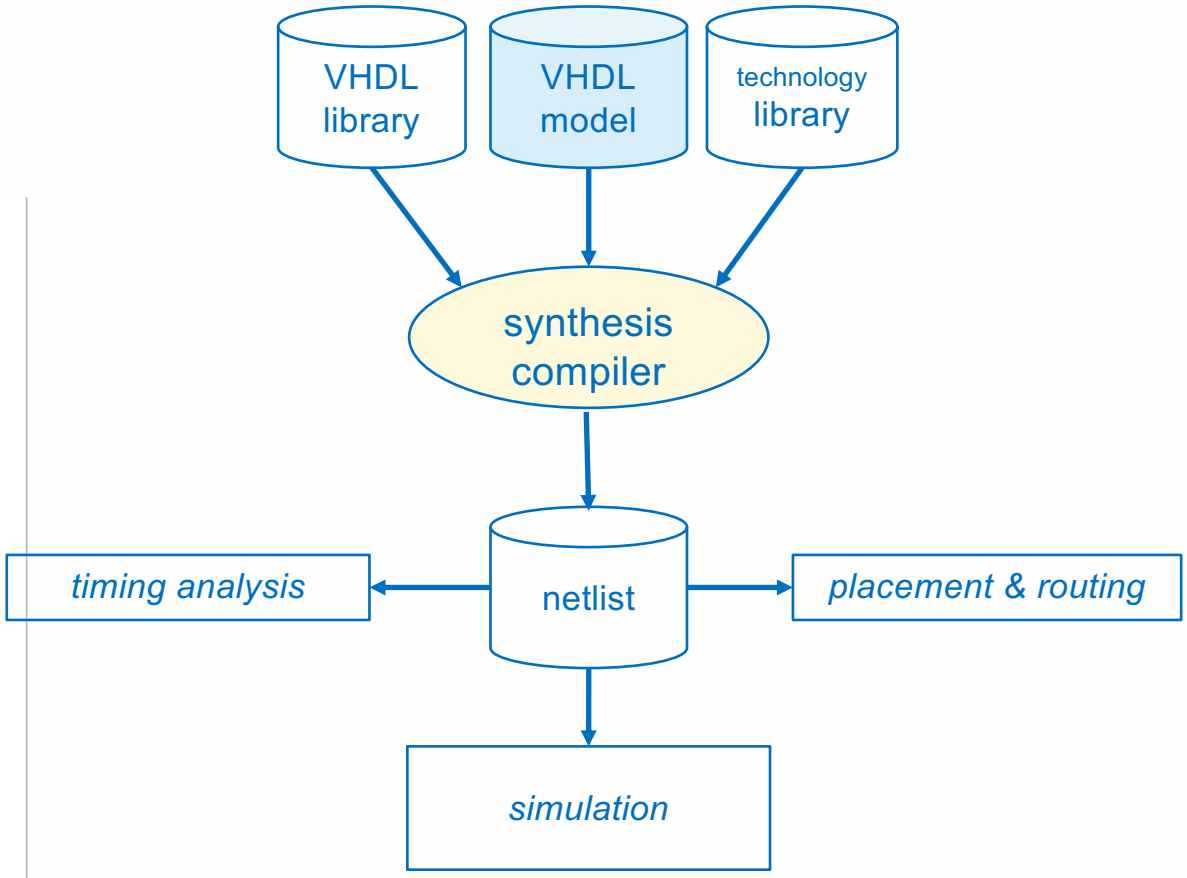
Objectifs

- un langage commun, depuis la description du système jusqu'à la réalisation du circuit
- un langage qui permet de valider la conception du système à tous les niveaux
- un langage qui permet de générer automatiquement les schémas d'implantation sur Silicium (ou FPGA)

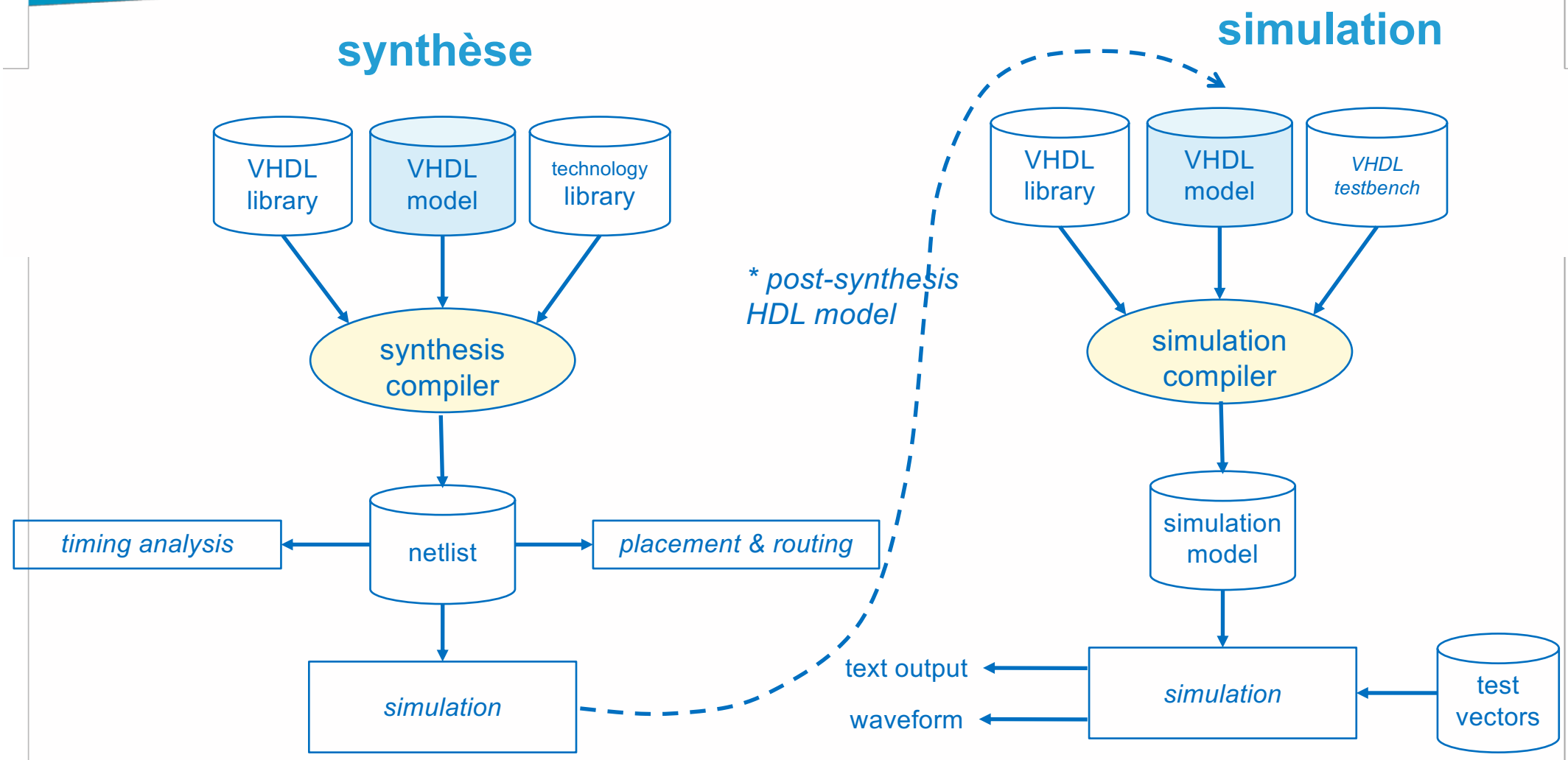
Simulation


VHDL
library
VHDL
model
VHDL
testbench
simulation
compiler
simulation
model
text output
waveform
simulation
test
vectors

Synthèse


VHDL library
VHDL model
technology library
synthesis compiler
netlist
timing analysis
placement & routing
simulation

Synthèse


synthèse
simulation
VHDL library
VHDL model
technology library
synthesis compiler
netlist
timing analysis
placement & routing
simulation
* post-synthesis HDL model
VHDL library
VHDL model
VHDL testbench
simulation compiler
simulation model
simulation
text output
waveform
test vectors

Description d'un circuit/composant

Entité : vue externe

- entrées et sorties (ports)

Architecture : fonctionnement interne

- flot de données
  - équations logiques
- comportementale
  - processus : instructions séquentielles (algorithme)
- structurelle
  - assemblage de composants

Déclaration d'une entité

```
ENTITY entite IS
  PORT(
    signal : mode type;
    ...
    signal : mode type
  );
END [ENTITY] [entite];
```

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY porte_ou IS
  PORT (e1 : IN STD_LOGIC;
        e2 : IN STD_LOGIC;
        s : OUT STD_LOGIC);
END ENTITY porte_ou;
```

mode : IN, OUT, INOUT, BUFFER

type : STD_LOGIC, STD_LOGIC_VECTOR(xx DOWNTO 0),
BIT, BOOLEAN, CHARACTER, INTEGER, REAL, TIME

Définition d'une architecture

```
ARCHITECTURE archi OF entite IS
  -- déclarations de signaux, constantes, …
  -- déclarations de composants
  BEGIN
    -- description de l'architecture
    --    flot de données
    --    et/ou comportementale (processus)
    --     et/ou structurelle
  END [ARCHITECTURE] [archi];
```

```
ARCHITECTURE porte_ou_fd OF porte_ou IS
  BEGIN
    s <= e1 OR e2;
  END ARCHITECTURE porte_ou_fd;
```

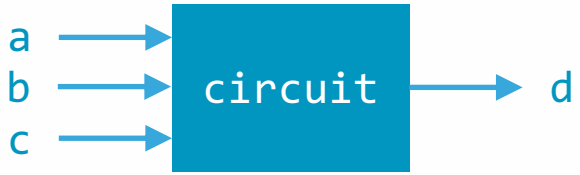
Description d'une architecture par flots de données

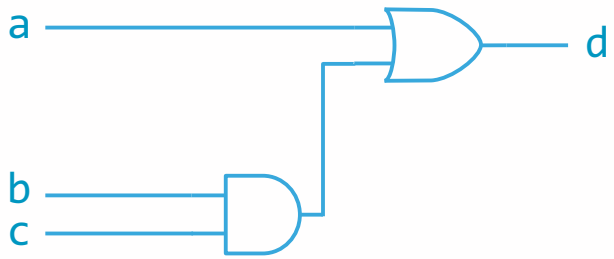
Ensemble d'affectations de signaux *concurrentes*

- elles correspondent aux couches d'opérateurs logiques

```
ENTITY circuit IS
  PORT (  a : IN STD_LOGIC;
          b : IN STD_LOGIC;
          c : IN STD_LOGIC;
          d : OUT STD_LOGIC);
END ENTITY circuit;

ARCHITECTURE a1 OF circuit IS
  BEGIN
    d <= a OR (b AND c);
  END ARCHITECTURE a1;
```

attention aux priorités !





Description d'une architecture par flots de données

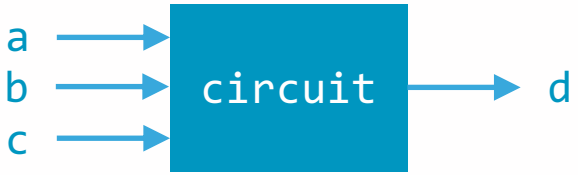
Ensemble d'affectations de signaux *concurrentes*

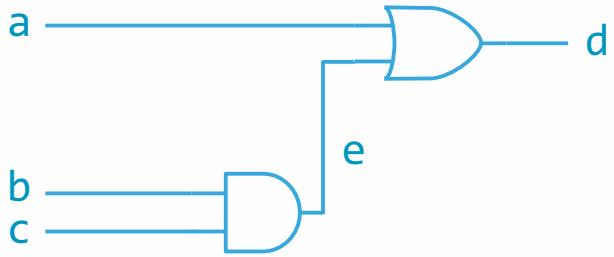
- elles correspondent aux couches d'opérateurs logiques

```
ARCHITECTURE a2 OF circuit IS
  SIGNAL e : STD_LOGIC;
  BEGIN
    d <= a OR e;
    e <= b AND c;
  END ARCHITECTURE a2;
```

=

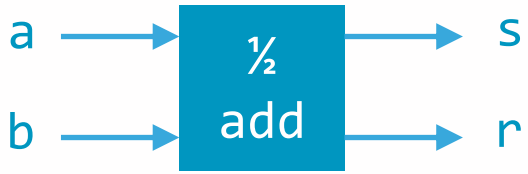
```
ARCHITECTURE a3 OF circuit IS
  SIGNAL e : STD_LOGIC;
  BEGIN
    e <= b AND c;
    d <= a OR e;
  END ARCHITECTURE a3;
```





Exercice : demi-additionneur
Architecture de type flot de données

Implémenter en VHDL un demi-additionneur : déclaration de l'entité et architecture de type flot de données



$$r = a.b$$

$$s = \bar{a}.b + a.\bar{b} = a \oplus b$$

Solution : demi-additionneur

Architecture de type flot de données

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY demi_add IS
  PORT( a : IN STD_LOGIC;
        b : IN STD_LOGIC;
        r : OUT STD_LOGIC;
        s : OUT STD_LOGIC);
END ENTITY demi_add;
```

Solution : demi-additionneur

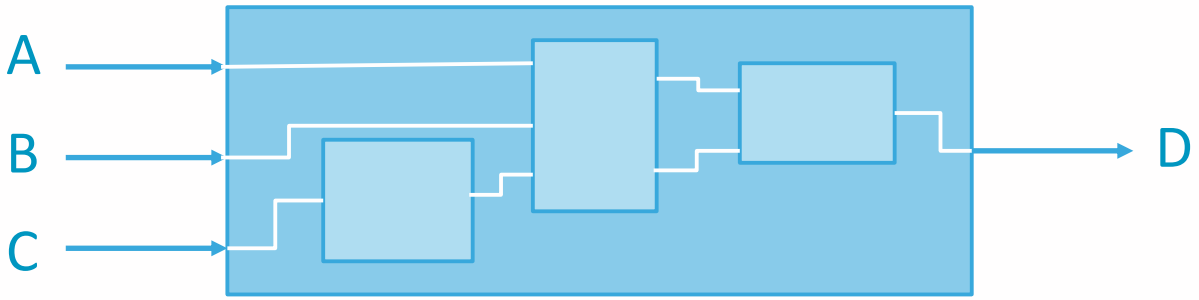
Architecture de type flot de données

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY demi_add IS
  PORT( a : IN STD_LOGIC;
        b : IN STD_LOGIC;
        r : OUT STD_LOGIC;
        s : OUT STD_LOGIC);
END ENTITY demi_add;

ARCHITECTURE demi_add_fd OF demi_add IS
  BEGIN
    r <= a AND b;
    s <= a XOR b;
  END ARCHITECTURE demi_add_fd;
```

Description structurelle d'une architecture

On décrit le circuit en indiquant :

- quels sont les composants utilisés
  - un composant est un circuit déjà défini par ailleurs
- comment ils sont connectés



Description structurelle d'une architecture

Déclaration d'un composant

- dans la partie « déclarations » de l'architecture utilisatrice (avant BEGIN)

```
COMPONENT composant [IS]
  PORT(
    signal : mode type;
    ...
    signal : mode type
  );
END COMPONENT [composant];
```

Description structurelle d'une architecture

Instanciation d'un composant

- dans le corps de l'architecture utilisatrice (après BEGIN)

```
instance: [COMPONENT] composant
    PORT MAP(
        port => signal,
        …
        port => signal);
```

Description structurelle d'une architecture

Instanciation d'un composant

- dans le corps de l'architecture utilisatrice (après BEGIN)

```
instance: [COMPONENT] composant
    PORT MAP(
        port => signal,
        …
        port => signal);
```

- par défaut, c'est l'entité du même nom que le composant qui est choisie (avec sa dernière architecture compilée)

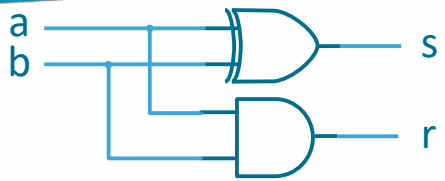
Exercice : demi-additionneur
Architecture structurelle

Implémenter en VHDL un demi-additionneur en proposant une description structurelle :

- définir des entités représentant les portes logiques ET et OUEX
- décrire la structure du demi-additionneur à partir de ces composants

Solution : demi-additionneur

Architecture structurelle

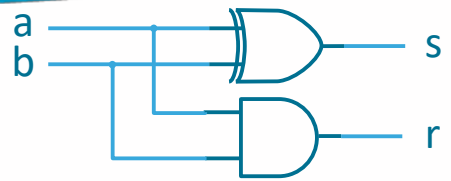



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY porte_et IS
  PORT( e1 : IN STD_LOGIC;
        e2 : IN STD_LOGIC;
        s : OUT STD_LOGIC);
END ENTITY porte_et;

ARCHITECTURE porte_et_fd OF porte_et IS
  BEGIN
    s <= e1 AND e2;
  END ARCHITECTURE porte_et_fd;
```

idem pour ouex …

Solution : demi-additionneur

Architecture structurelle




```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY porte_et IS
  PORT( e1 : IN STD_LOGIC;
        e2 : IN STD_LOGIC;
        s : OUT STD_LOGIC);
END ENTITY porte_et;

ARCHITECTURE porte_et_fd OF porte_et IS
  BEGIN
    s <= e1 AND e2;
  END ARCHITECTURE porte_et_fd;
```

idem pour ouex …

```
ARCHITECTURE demi_add_struct OF demi_add IS
  COMPONENT porte_et IS
    PORT( e1 : IN STD_LOGIC;
          e2 : IN STD_LOGIC;
          s : OUT STD_LOGIC);
  END COMPONENT;
  COMPONENT porte_ouex IS
    PORT( e1 : IN STD_LOGIC;
          e2 : IN STD_LOGIC;
          s : OUT STD_LOGIC);
  END COMPONENT;
  BEGIN
    et : porte_et
      PORT MAP (e1=>a, e2=>b, s=>r);
    ouex : porte_ouex
      PORT MAP (e1=>a, e2=>b, s=>s);
  END ARCHITECTURE demi_add_struct;
```

Description comportementale d'une architecture

Processus

- une suite d'instructions séquentielles
- exécutées dans une boucle sans fin
  - on peut mettre le processus en sommeil …
  - … et le réveiller lorsque des événements se produisent sur certains signaux (liste de sensibilité)
- la description comportementale d'une architecture peut contenir plusieurs processus indépendants (concurrents)

Exemple : demi-additionneur

Description comportementale

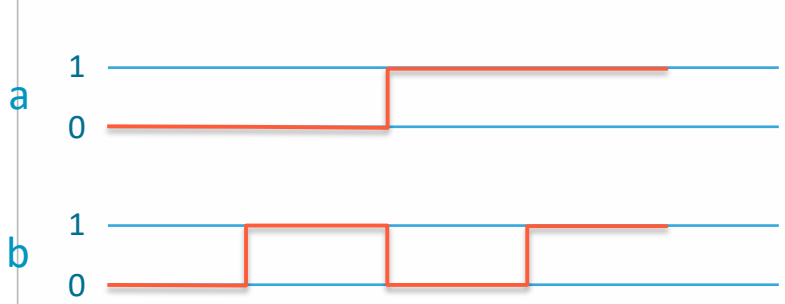
$$r = a.b$$

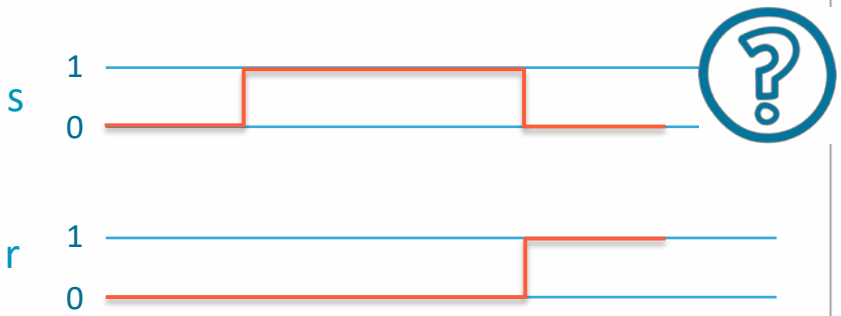
$$s = a \oplus b$$

```
ARCHITECTURE demi_add_comp OF demi_add IS
  BEGIN
    PROCESS(a,b)
      BEGIN
        r <= a AND b;
        s <= a XOR b;
      END PROCESS;
  END ARCHITECTURE demi_add_comp;
```

Simulation : banc de test

Objectif : tester un composant





Définition du banc de test

Entité

- pas d'entrées/sorties

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY demi_add_test IS
END ENTITY demi_add_test;
```

Architecture

- le banc de test inclut le composant à tester
  → description structurelle
- le comportement du banc de test est décrit de manière séquentielle
  → description comportementale

Description structurelle

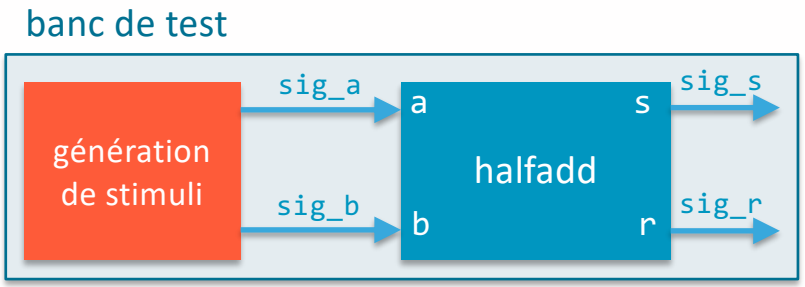
Déclaration du composant à tester : halfadd

```
ARCHITECTURE demi_add_test_arch OF demi_add_test IS
  COMPONENT demi_add IS
    PORT( a : IN STD_LOGIC;
          b : IN STD_LOGIC;
          r : OUT STD_LOGIC;
          s : OUT STD_LOGIC);
  END COMPONENT;
```

Description structurelle

Instanciation du composant

- connexion des signaux internes à l'architecture (du banc de test) aux ports du composant

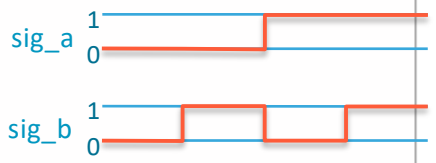



```
ARCHITECTURE demi_add_test_arch OF demi_add_test IS
  COMPONENT demi_add IS
    PORT( a : IN STD_LOGIC;
          b : IN STD_LOGIC;
          r : OUT STD_LOGIC;
          s : OUT STD_LOGIC);
  END COMPONENT;
  SIGNAL sig_a, sig_b, sig_r, sig_s : STD_LOGIC;
  BEGIN
    demi_additionneur: demi_add
      PORT MAP (  a=>sig_a,
                  b=>sig_b,
                  r=>sig_r,
                  s=>sig_s);
```

Banc de test pour le demi-additionneur

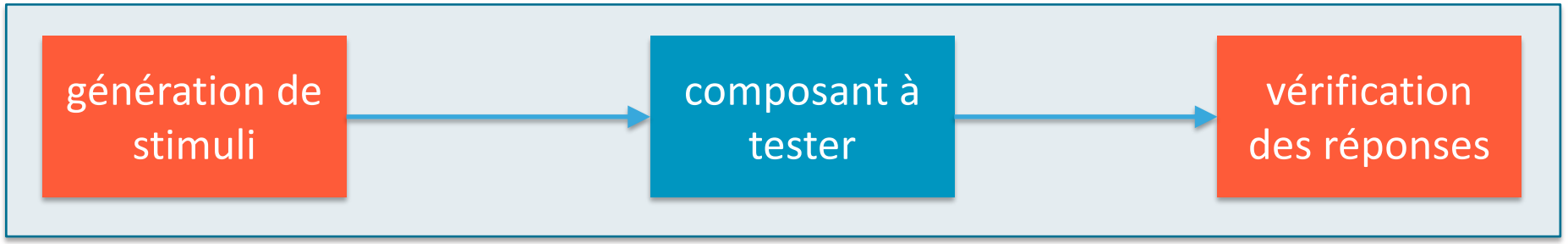
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY demi_add_test IS
END ENTITY demi_add_test;

ARCHITECTURE demi_add_test_arch OF demi_add_test IS
  COMPONENT demi_add IS
    PORT( a : IN STD_LOGIC;
          b : IN STD_LOGIC;
          r : OUT STD_LOGIC;
          s : OUT STD_LOGIC);
  END COMPONENT;
  SIGNAL sig_a, sig_b, sig_r, sig_s : STD_LOGIC;
  BEGIN
    demi_additionneur: demi_add
      PORT MAP (  a=>sig_a,
                  b=>sig_b,
                  r=>sig_r,
                  s=>sig_s);
```

```
    PROCESS
     BEGIN
      sig_a <= '0';
      sig_b <= '0';
      WAIT FOR 5 ns;
      ASSERT sig_r = '0';
      ASSERT sig_s = '0';

      sig_a <= '0';
      sig_b <= '1';
      WAIT FOR 5 ns;
      ASSERT sig_r = '0';
      ASSERT sig_s = '1';

      sig_a <= '1';
      sig_b <= '0';
      WAIT FOR 5 ns;
      ASSERT sig_r = '0';
      ASSERT sig_s = '1';

      sig_a <= '1';
      sig_b <= '1';
      WAIT FOR 5 ns;
      ASSERT sig_r = '1';
      ASSERT sig_s = '0';

      WAIT;
    END PROCESS;
END ARCHITECTURE demi_add_test_arch;
```



Vérification des réponses du composant

banc de test



Assertions

```
sig_a <= '0';
sig_b <= '0';
WAIT FOR 5 ns;
ASSERT sig_r = '0';
ASSERT sig_s = '0';
```

a	b	r	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Génération de traces

- utilisation de GtkWave

Vérification des réponses du composant

```
> ghdl -a gates.vhd
> ghdl -a half_adder.vhd
> ghdl -a half_adder_test.vhd
>
>
> ghdl -e halfadd_test
>
>
> ghdl -r halfadd_test --vcd=halfadd_test.vcd
>
>
> gtkwave halfadd_test.vcd
>
```

compile les modèles VHDL et génère des fichiers objets (.o)

-a : **a**nalysis

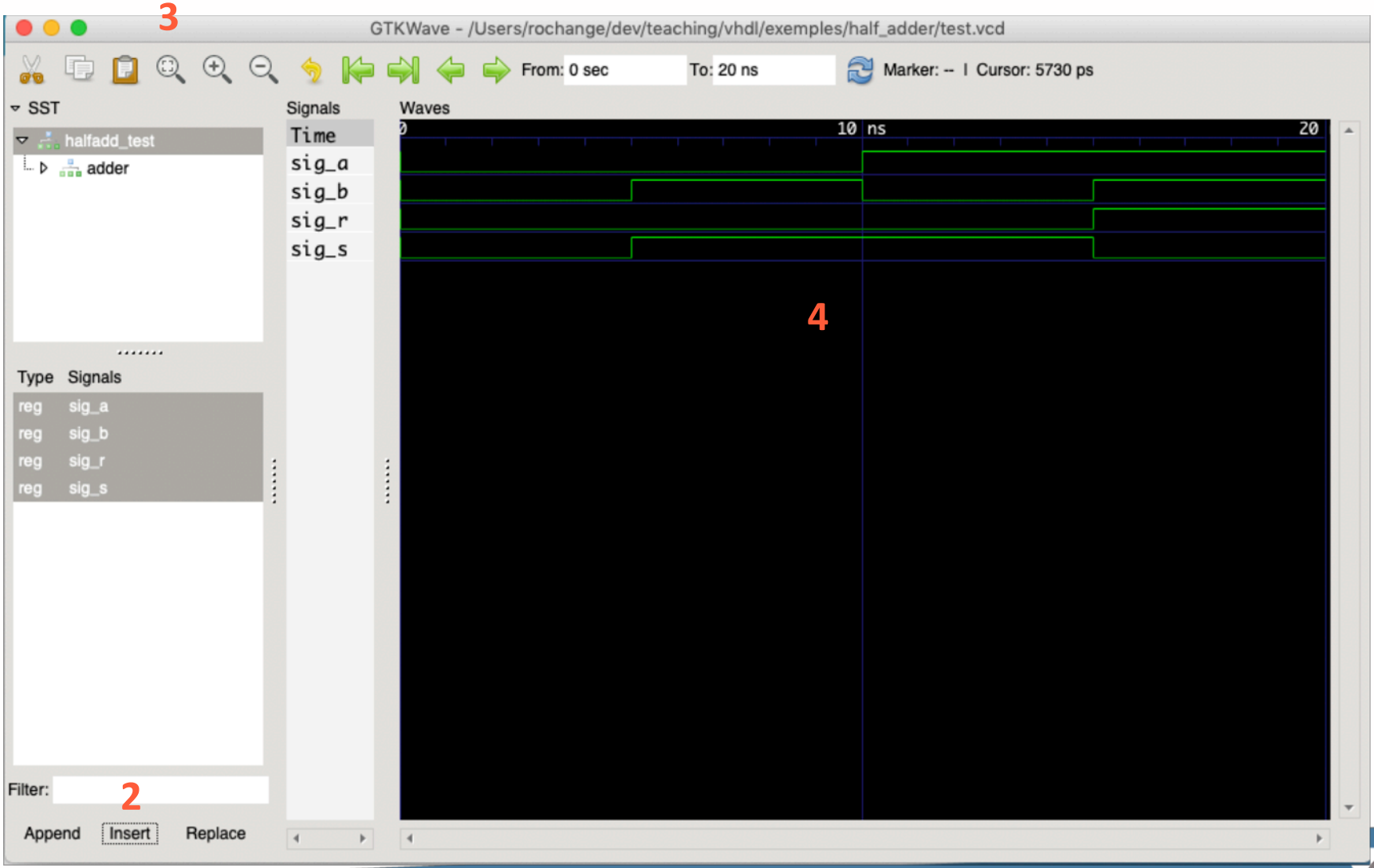
génère un executable VHDL

-e : **e**laboration

exécute/simule le modèle VHDL et génère un fichier de trace

visualise la trace de simulation

Vérification des réponses du composant



Vérification des réponses du composant

Assertions

- si on modifie le banc de test pour générer une erreur :

```
sig_a <= '0';
sig_b <= '0';
WAIT FOR 5 ns;
ASSERT sig_r = '0';
ASSERT sig_s = '0';
```

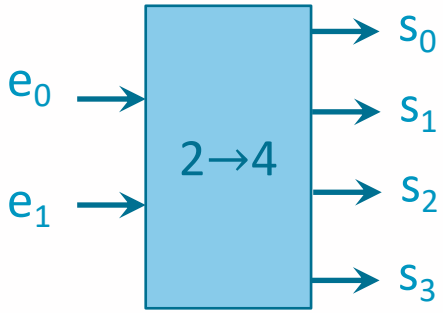
```
sig_a <= '0';
sig_b <= '0';
WAIT FOR 5 ns;
ASSERT sig_r = '1';
ASSERT sig_s = '0';
```

```
> ./halfadd_test
half_adder_test.vhd:34:11:@5ns:(assertion error): Assertion violation
>
```

Exercice : décodeur 2 vers 4

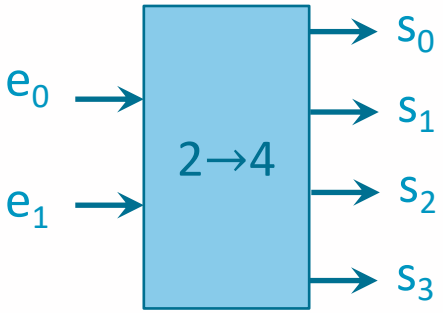
Architecture de type flot de données

Implémenter en VHDL un décodeur 2 vers 4, avec une architecture de type flot de données.



Solution : décodeur 2 vers 4

Architecture de type flot de données



$$s_0 = \overline{e_1}.\overline{e_0}$$

$$s_1 = \overline{e_1}.e_0$$

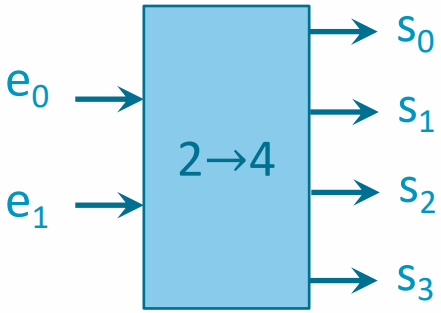
$$s_2 = e_1.\overline{e_0}$$

$$s_3 = e_1.e_0$$

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY decodeur IS
  PORT (  e0 : IN STD_LOGIC;
          e1 : IN STD_LOGIC;
          s0 : OUT STD_LOGIC;
          s1 : OUT STD_LOGIC;
          s2 : OUT STD_LOGIC;
          s3 : OUT STD_LOGIC);
END ENTITY decodeur;
```

Solution : décodeur 2 vers 4

Architecture de type flot de données



$$s_0 = \overline{e_1}.\overline{e_0}$$

$$s_1 = \overline{e_1}.e_0$$

$$s_2 = e_1.\overline{e_0}$$

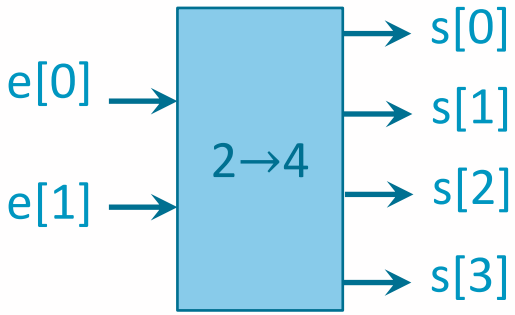
$$s_3 = e_1.e_0$$

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY decodeur IS
  PORT (  e0 : IN STD_LOGIC;
          e1 : IN STD_LOGIC;
          s0 : OUT STD_LOGIC;
          s1 : OUT STD_LOGIC;
          s2 : OUT STD_LOGIC;
          s3 : OUT STD_LOGIC);
END ENTITY decodeur;

ARCHITECTURE decodeur_fd OF decodeur IS
  BEGIN
    s0 <= NOT e0 AND NOT e1;
    s1 <= e0 AND NOT e1;
    s2 <= NOT e0 AND e1;
    s3 <= e0 AND e1;
  END ARCHITECTURE decodeur_fd;
```

Exercice : décodeur 2 vers 4

Architecture de type flot de données

Utilisation du type STD_LOGIC_VECTOR

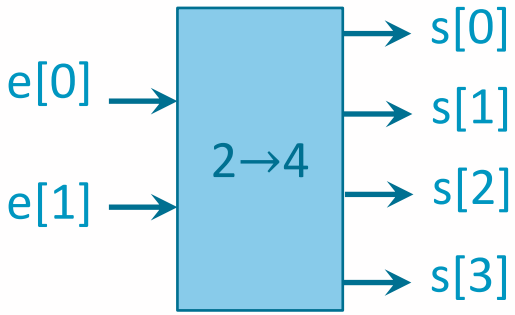



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY decodeur2 IS
  PORT( e : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        s : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
      );
END ENTITY decodeur2;
```

Exercice : décodeur 2 vers 4

Architecture de type flot de données

Utilisation du type STD_LOGIC_VECTOR




```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY decodeur2 IS
  PORT( e : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        s : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
      );
END ENTITY decodeur2;

ARCHITECTURE decodeur2_fd OF decodeur2 IS
  BEGIN
    s(0) <= NOT e(0) AND NOT e(1);
    s(1) <= e(0) AND NOT e(1);
    s(2) <= NOT e(0) AND e(1);
    s(3) <= e(0) AND e(1);
  END ARCHITECTURE decodeur2_fd;
```

Exercice : décodeur 2 vers 4

Architecture de type flot de données

Affectation sélective

e_1	e_0	s_3	s_2	s_1	s_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

```
ARCHITECTURE decodeur2_fd2 OF decodeur2 IS
  BEGIN
    WITH e SELECT
      s <=  "0001" WHEN "00",
            "0010" WHEN "01",
            "0100" WHEN "10",
            "1000" WHEN "11",
            "0000" WHEN OTHERS;
  END ARCHITECTURE decodeur2_fd2;
```

Exercice : décodeur 2 vers 4

Architecture de type flot de données

Affectation sélective

e_1	e_0	s_3	s_2	s_1	s_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

```
ARCHITECTURE decodeur2_fd2 OF decodeur2 IS
  BEGIN
    WITH e SELECT
      s <=  "0001" WHEN "00",
            "0010" WHEN "01",
            "0100" WHEN "10",
            "1000" WHEN "11",
            "0000" WHEN OTHERS;
  END ARCHITECTURE decodeur2_fd2;
```

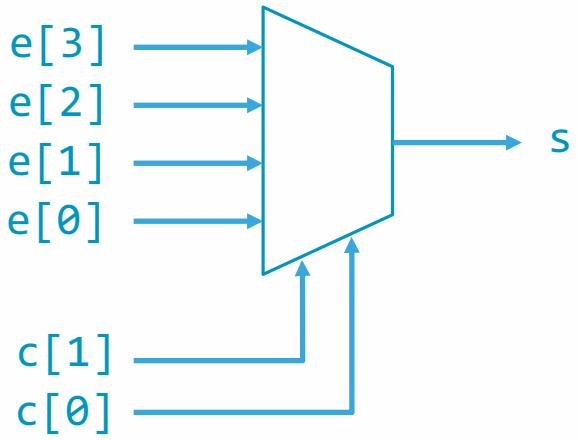
Affectation conditionnelle

```
ARCHITECTURE decodeur2_fd3 OF decodeur2 IS
  BEGIN
    s <=  "0001" WHEN e = "00" ELSE
          "0010" WHEN e = "01" ELSE
          "0100" WHEN e = "10" ELSE
          "1000" WHEN e = "11" ELSE
          "0000" ;
  END ARCHITECTURE decodeur2_fd3;
```

Exercice : multiplexeur 4 vers 1

Architecture de type flot de données

Implémenter en VHDL un multiplexeur 4 vers 1. Dans un premier temps, on demande une architecture de type flot de données.




```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY mux4_1 IS
  PORT( e : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        c : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        s : OUT STD_LOGIC);
END ENTITY mux4_1;
```

Solution : multiplexeur 4 vers 1

Architecture de type flot de données

```
ARCHITECTURE mux4_1_fd1 OF mux4_1 IS
  BEGIN
    s <=  e(0) WHEN c = "00" ELSE
          e(1) WHEN c = "01" ELSE
          e(2) WHEN c = "10" ELSE
          e(3) WHEN c = "11" else
          'X';
  END ARCHITECTURE mux4_1_fd1;
```

Solution : multiplexeur 4 vers 1

Architecture de type flot de données

```
ARCHITECTURE mux4_1_fd1 OF mux4_1 IS
  BEGIN
    s <=  e(0) WHEN c = "00" ELSE
          e(1) WHEN c = "01" ELSE
          e(2) WHEN c = "10" ELSE
          e(3) WHEN c = "11" else
          'X';
  END ARCHITECTURE mux4_1_fd1;
```

```
ARCHITECTURE mux4_1_fd2 OF mux4_1 IS
  BEGIN
    WITH c SELECT
      s <=  e(0) WHEN "00",
            e(1) WHEN "01",
            e(2) WHEN "10",
            e(3) WHEN "11",
            'X' WHEN OTHERS;
  END ARCHITECTURE mux4_1_fd2;
```

Solution : multiplexeur 4 vers 1

Architecture de type flot de données

```
ARCHITECTURE mux4_1_fd1 OF mux4_1 IS
  BEGIN
    s <=  e(0) WHEN c = "00" ELSE
          e(1) WHEN c = "01" ELSE
          e(2) WHEN c = "10" ELSE
          e(3) WHEN c = "11" else
          'X';
  END ARCHITECTURE mux4_1_fd1;
```

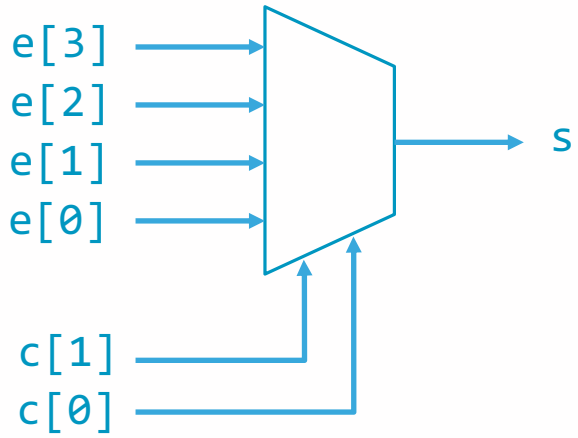
```
ARCHITECTURE mux4_1_fd2 OF mux4_1 IS
  BEGIN
    WITH c SELECT
      s <=  e(0) WHEN "00",
            e(1) WHEN "01",
            e(2) WHEN "10",
            e(3) WHEN "11",
            'X' WHEN OTHERS;
  END ARCHITECTURE mux4_1_fd2;
```

```
ARCHITECTURE mux4_1_fd3 OF mux4_1 IS
  BEGIN
    s <=  (e(0) AND NOT c(1) AND NOT c(0))
          OR
          (e(1) AND NOT c(1) AND c(0))
          OR
          (e(2) AND c(1) AND NOT c(0))
          OR
          (e(3) AND c(1) AND c(0));
  END ARCHITECTURE mux4_1_fd3;
```

Exercice : multiplexeur 4 vers 1

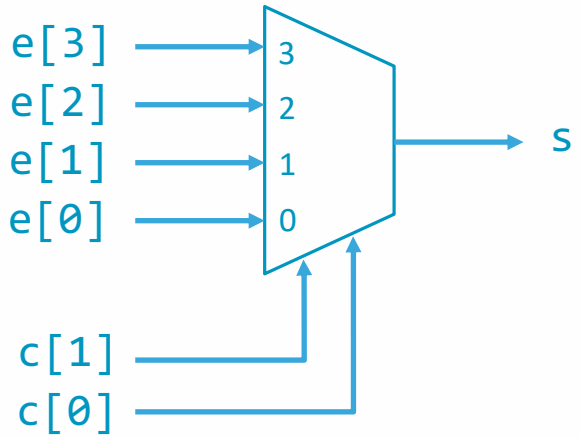
Architecture structurelle

Implémenter en VHDL un multiplexeur 4 vers 1 avec une architecture structurelle utilisant des multiplexeurs 2 vers 1 (à définir).

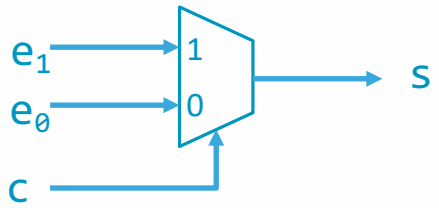


Solution : multiplexeur 4 vers 1

Architecture structurelle



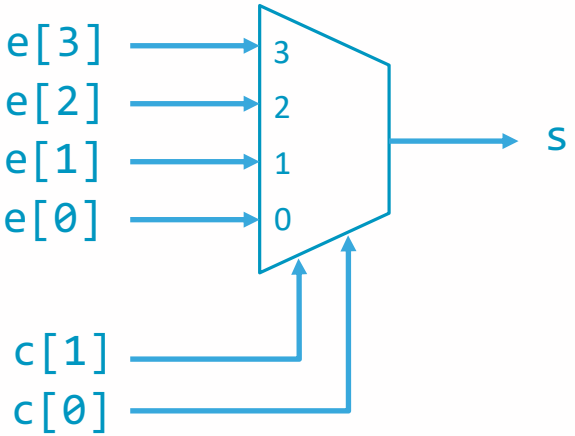
c[1]	c[0]	s
0	0	e[0]
0	1	e[1]
1	0	e[2]
1	1	e[3]

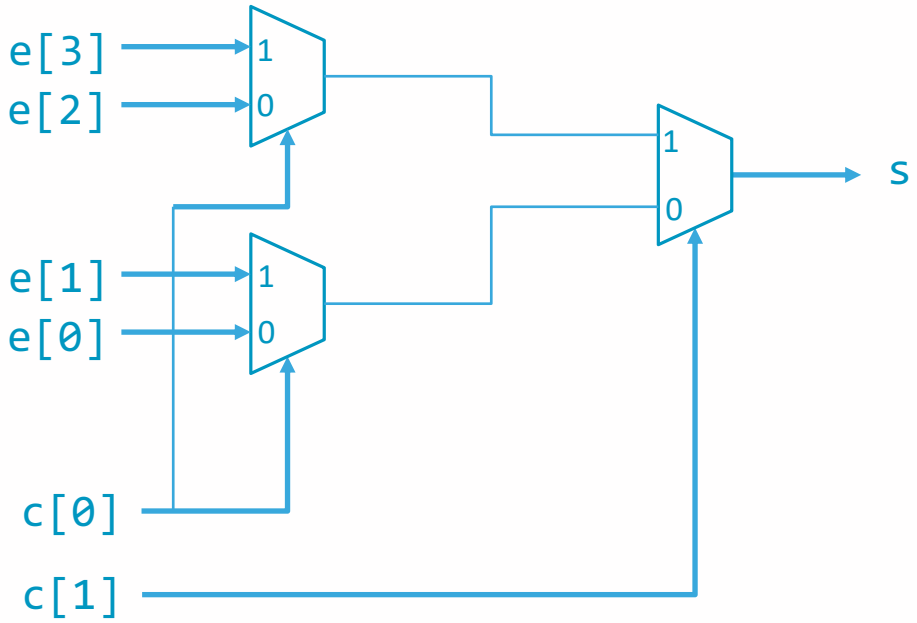


$$s = e_1.c + e_0.\bar{c}$$

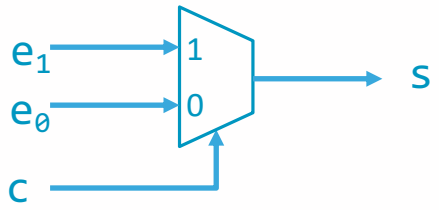
Solution : multiplexeur 4 vers 1

Architecture structurelle





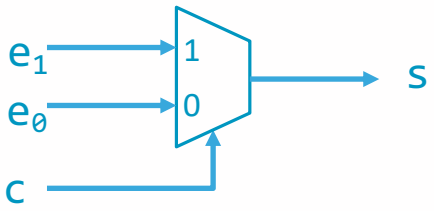
c[1]	c[0]	s
0	0	e[0]
0	1	e[1]
1	0	e[2]
1	1	e[3]



$$s = e_1.c + e_0.\bar{c}$$

Solution : multiplexeur 4 vers 1

Architecture structurelle

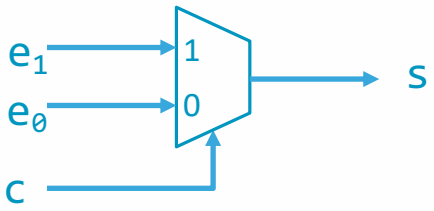


$$s = e_1.c + e_0.\bar{c}$$

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY mux2_1 IS
  PORT( e0 : IN STD_LOGIC;
        e1 : IN STD_LOGIC;
        c : IN STD_LOGIC;
        s : OUT STD_LOGIC);
END ENTITY mux2_1;
```

Solution : multiplexeur 4 vers 1

Architecture structurelle

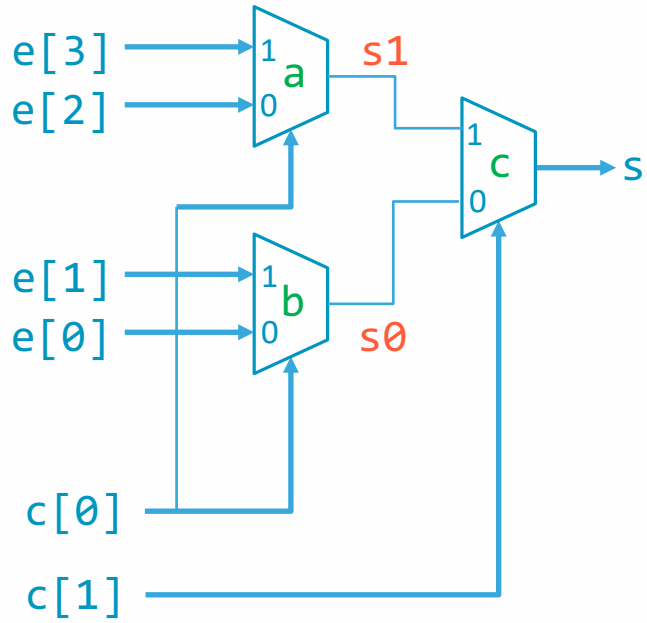


$$s = e_1.c + e_0.\bar{c}$$

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY mux2_1 IS
  PORT( e0 : IN STD_LOGIC;
        e1 : IN STD_LOGIC;
        c : IN STD_LOGIC;
        s : OUT STD_LOGIC);
END ENTITY mux2_1;

ARCHITECTURE mux2_1_fd OF mux2_1 IS
  BEGIN
    s <= (e1 AND c) OR (e0 AND NOT c);
  END ARCHITECTURE mux2_1_fd;
```

Solution : multiplexeur 4 vers 1

Architecture structurelle

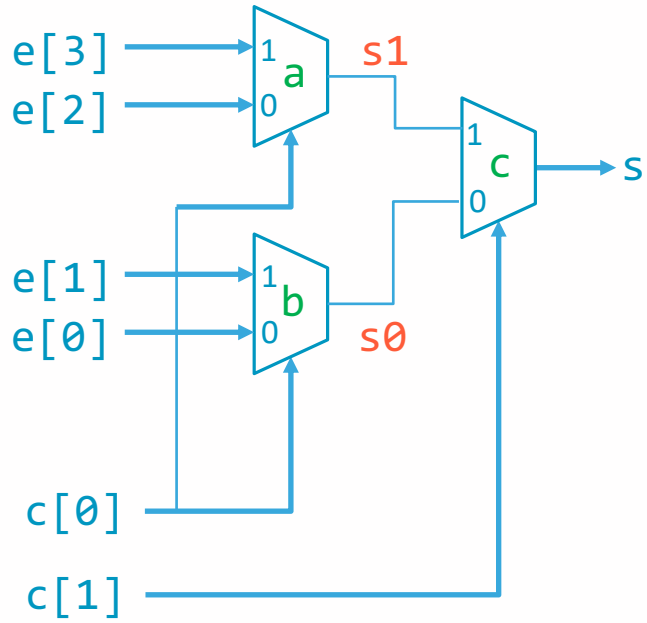



```
ARCHITECTURE mux4_1_struct OF mux4_1 IS
  SIGNAL s1, s0 : STD_LOGIC;
  COMPONENT mux2_1 IS
    PORT (  e0 : IN STD_LOGIC;
            e1 : IN STD_LOGIC;
            c : IN STD_LOGIC;
            s : OUT STD_LOGIC);
  END COMPONENT;
```

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY mux4_1 IS
  PORT( e : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        c : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        s : OUT STD_LOGIC);
END ENTITY mux4_1;
```

Solution : multiplexeur 4 vers 1

Architecture structurelle




```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY mux4_1 IS
  PORT( e : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        c : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        s : OUT STD_LOGIC);
END ENTITY mux4_1;
```

```
ARCHITECTURE mux4_1_struct OF mux4_1 IS
  SIGNAL s1, s0 : STD_LOGIC;
  COMPONENT mux2_1 IS
    PORT (  e0 : IN STD_LOGIC;
            e1 : IN STD_LOGIC;
            c : IN STD_LOGIC;
            s : OUT STD_LOGIC);
  END COMPONENT;
  BEGIN
    mux2a: mux2_1
      PORT MAP (e0=>e(2), e1=>e(3),
                c=>c(0), s=>s1);
    mux2b: mux2_1
      PORT MAP (e0=>e(0), e1=>e(1),
                c=>c(0), s=>s0);
    mux2c: mux2_1
      PORT MAP (e0=>s0, e1=>s1,
                c=>c(1), s=>s);
  END ARCHITECTURE mux4_1_struct;
```

Exercice : additionneur 1-bit complet

Implémenter en VHDL un additionneur 1-bit complet. Proposer deux architectures :

- flot de données
- structurelle (à partir de demi-additionneurs, déjà réalisés)

Solution : additionneur 1-bit complet

Architecture de type flot de données

r_e	a	b	r_s	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

r_s

r_e \ a b	00	01	11	10
0	0	0	1	0
1	0	1	1	1

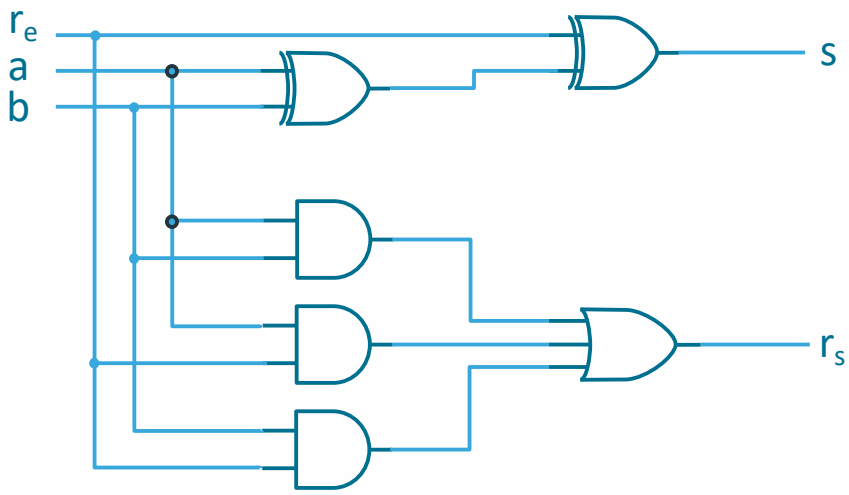
s

r_e \ a b	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$r_s = a.b + r_e.a + r_e.b$$

$$s = \bar{a}.\bar{b}.r_e + \bar{a}.b.\bar{r_e} + a.\bar{b}.\bar{r_e} + a.b.r_e$$

$$= a \oplus b \oplus r_e$$



Solution : additionneur 1-bit complet

Architecture de type flot de données

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY add_1b IS
  PORT (  a : IN STD_LOGIC;
          b : IN STD_LOGIC;
          re : IN STD_LOGIC ;
          s : OUT STD_LOGIC;
          rs : OUT STD_LOGIC);
END ENTITY add_1b;
```

Solution : additionneur 1-bit complet

Architecture de type flot de données

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY add_1b IS
  PORT (  a : IN STD_LOGIC;
          b : IN STD_LOGIC;
          re : IN STD_LOGIC ;
          s : OUT STD_LOGIC;
          rs : OUT STD_LOGIC);
END ENTITY add_1b;
```

$$r_s = a.b + r_e.a + r_e.b$$

$$s = a \oplus b \oplus r_e$$

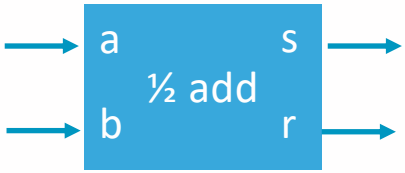
```
ARCHITECTURE add_1b_fd OF add_1b IS
  BEGIN
    s <= a XOR b XOR re;
    rs <=      (a AND b)
          OR   (a AND re)
          OR   (b AND re);
  END ARCHITECTURE add_1b_fd;
```

Solution : additionneur 1-bit complet

Architecture structurelle

Réalisation de l'additionneur complet à partir de demi-additionneurs

- somme de a et b, *puis* ajout de la retenue

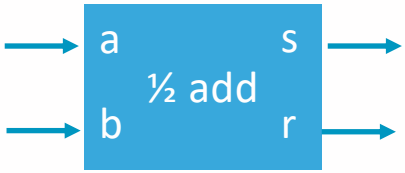


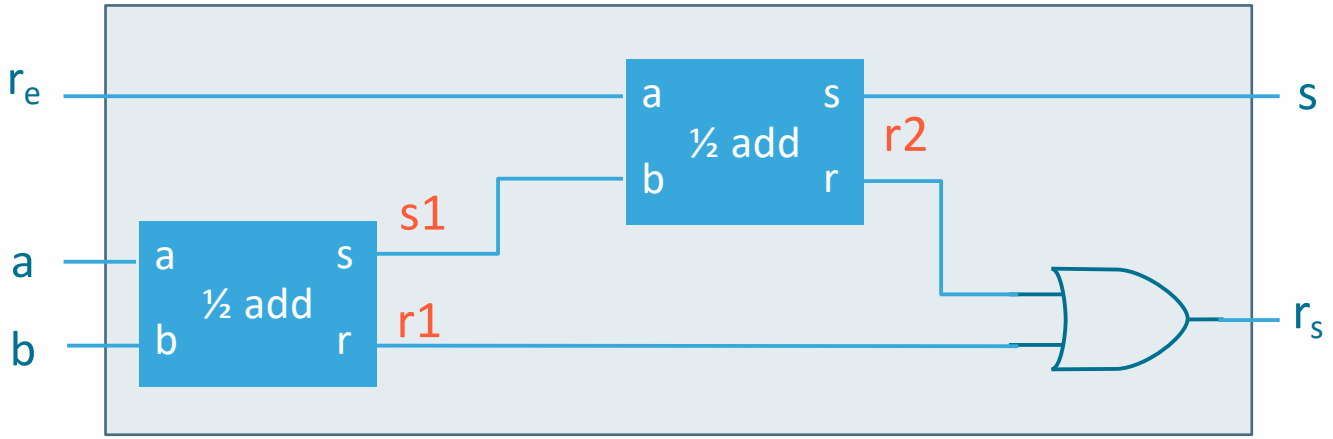
Solution : additionneur 1-bit complet

Architecture structurelle

Réalisation de l'additionneur complet à partir de demi-additionneurs

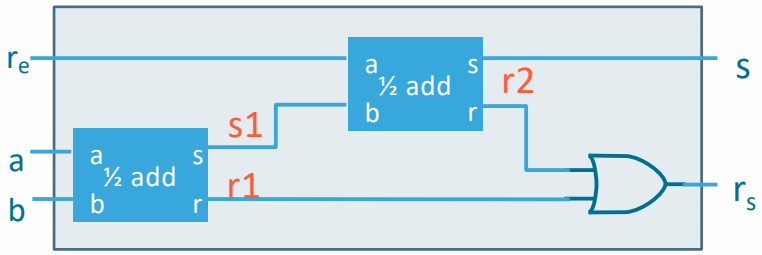
- somme de a et b, *puis* ajout de la retenue





Solution : additionneur 1-bit complet

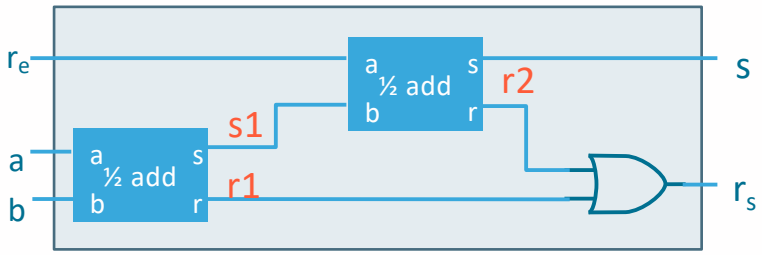
Architecture structurelle




```
ARCHITECTURE add_1b_struct OF add_1b IS
  SIGNAL s1, r1, r2 : STD_LOGIC;
  COMPONENT demi_add IS
    PORT (  a : IN STD_LOGIC;
            b : IN STD_LOGIC;
            r : OUT STD_LOGIC;
            s : OUT STD_LOGIC );
  END COMPONENT;
```

Solution : additionneur 1-bit complet

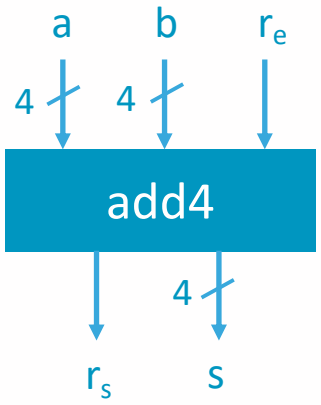
Architecture structurelle




```
ARCHITECTURE add_1b_struct OF add_1b IS
  SIGNAL s1, r1, r2 : STD_LOGIC;
  COMPONENT demi_add IS
    PORT (  a : IN STD_LOGIC;
            b : IN STD_LOGIC;
            r : OUT STD_LOGIC;
            s : OUT STD_LOGIC );
  END COMPONENT;
  BEGIN
    ha1 : demi_add
      PORT MAP (a=>a, b=>b, s=>s1, r=>r1);
    ha2 : demi_add
      PORT MAP (a=>re, b=>s1, s=>s, r=>r2);
    rs <= r1 OR r2;
  END ARCHITECTURE add_1b_struct;
```

Exercice : additionneur 4-bits

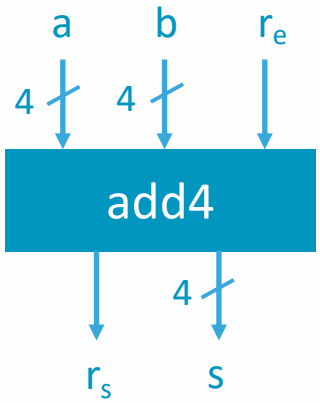
Implémenter en VHDL un additionneur 4-bits à partir d'additionneurs 1-bit complets (avec propagation de la retenue).

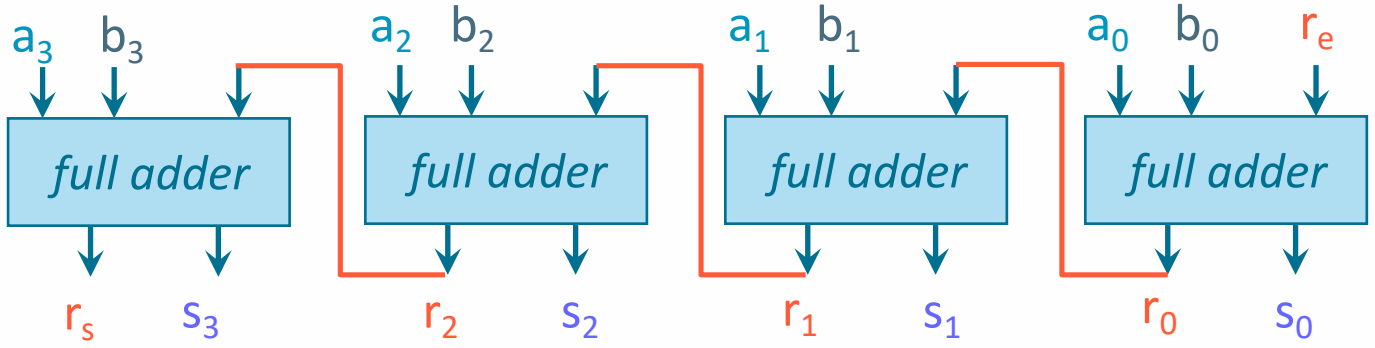



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY add_4b IS
  PORT (  a : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          b : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          re : IN STD_LOGIC ;
          s : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
          rs : OUT STD_LOGIC);
END ENTITY add_4b;
```

Exercice : additionneur 4-bits

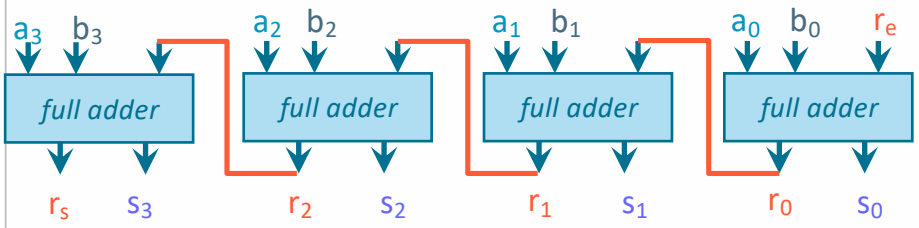
Implémenter en VHDL un additionneur 4-bits à partir d'additionneurs 1-bit complets (avec propagation de la retenue).





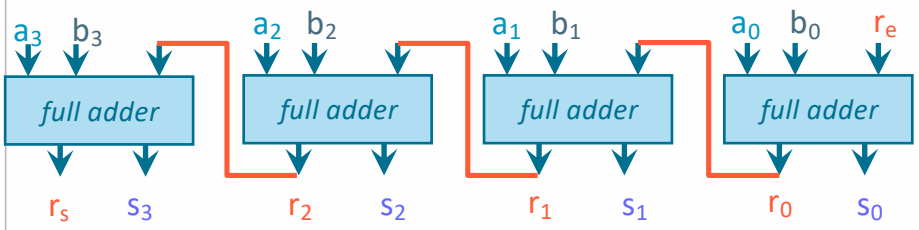
Solution : additionneur 4-bits

```
ARCHITECTURE add_4b_struct OF add_4b IS
  SIGNAL r : STD_LOGIC_VECTOR(4 DOWNTO 0);
  COMPONENT add_1b IS
    PORT (  a : IN STD_LOGIC;
            b : IN STD_LOGIC;
            re : IN STD_LOGIC ;
            s : OUT STD_LOGIC;
            rs : OUT STD_LOGIC);
  END COMPONENT;
```



Solution : additionneur 4-bits

```
ARCHITECTURE add_4b_struct OF add_4b IS
  SIGNAL r : STD_LOGIC_VECTOR(4 DOWNTO 0);
  COMPONENT add_1b IS
    PORT (  a : IN STD_LOGIC;
            b : IN STD_LOGIC;
            re : IN STD_LOGIC ;
            s : OUT STD_LOGIC;
            rs : OUT STD_LOGIC);
  END COMPONENT;
```




```
BEGIN
  r(0) <= re;
  add0 : add_1b
    PORT MAP (
      a=>a(0),
      b=>b(0),
      re=>r(0),
      s=>s(0),
      rs=>r(1)
    );
  add1 : add_1b
    PORT MAP (
      a=>a(1),
      b=>b(1),
      re=>r(1),
      s=>s(1),
      rs=>r(2)
    );
```

```
  add2 : add_1b
    PORT MAP (
      a=>a(2),
      b=>b(2),
      re=>r(2),
      s=>s(2),
      rs=>r(3)
    );
  add3 : add_1b
    PORT MAP (
      a=>a(3),
      b=>b(3),
      re=>r(3),
      s=>s(3),
      rs=>r(4)
    );
  rs <= r(4);
END ARCHITECTURE add_4b_struct;
```

Boucle concurrente (spatiale)

Permet de dupliquer une ou plusieurs instructions concurrentes

```
étiquette: FOR id IN intervalle GENERATE
    instruction concurrente;
    …
    instruction concurrente;
END GENERATE;
```

Boucle concurrente (spatiale)

Permet de dupliquer une ou plusieurs instructions concurrentes

```
étiquette: FOR id IN intervalle GENERATE
    instruction concurrente;
    …
    instruction concurrente;
END GENERATE;
```

Exemple :

```
SIGNAL vector1, vector2 : STD_LOGIC_VECTOR(31 DOWNTO 0);
```

```
vector1(0) <= vector2(8);
vector1(1) <= vector2(9);
vector1(2) <= vector2(10);
vector1(3) <= vector2(11);
```

≡

```
decale: FOR i IN 0 TO 3 GENERATE
  vector1(i) <= vector2(i+8);
END GENERATE decale;
```

Solution : additionneur 4-bits

```
BEGIN
  r(0) <= re;
  add0 : add_1b
    PORT MAP (
      a=>a(0),
      b=>b(0),
      re=>r(0),
      s=>s(0),
      rs=>r(1)
    );
  add1 : add_1b
    PORT MAP (
      a=>a(1),
      b=>b(1),
      re=>r(1),
      s=>s(1),
      rs=>r(2)
    );
  add2 : add_1b
    PORT MAP (
      a=>a(2),
      b=>b(2),
      re=>r(2),
      s=>s(2),
      rs=>r(3)
    );
  add3 : add_1b
    PORT MAP (
      a=>a(3),
      b=>b(3),
      re=>r(3),
      s=>s(3),
      rs=>r(4)
    );
  rs <= r(4);
END ARCHITECTURE add_4b_struct
```

```
ARCHITECTURE add_4b_struct2 OF add_4b IS
  SIGNAL r : STD_LOGIC_VECTOR(4 DOWNTO 0);
  COMPONENT add_1b IS
    PORT (  a : IN STD_LOGIC;
            b : IN STD_LOGIC;
            re : IN STD_LOGIC ;
            s : OUT STD_LOGIC;
            rs : OUT STD_LOGIC);
  END COMPONENT;
  BEGIN
    r(0) <= re;
    additionneurs : FOR i IN 0 TO 3 GENERATE
      addi : add_1b
        PORT MAP (
          a=>a(i),
          b=>b(i),
          re=>r(i),
          s=>s(i),
          rs=>r(i+1)
        );
    END GENERATE additionneurs;
    rs <= r(4);
  END ARCHITECTURE add_4b_struct2;
```

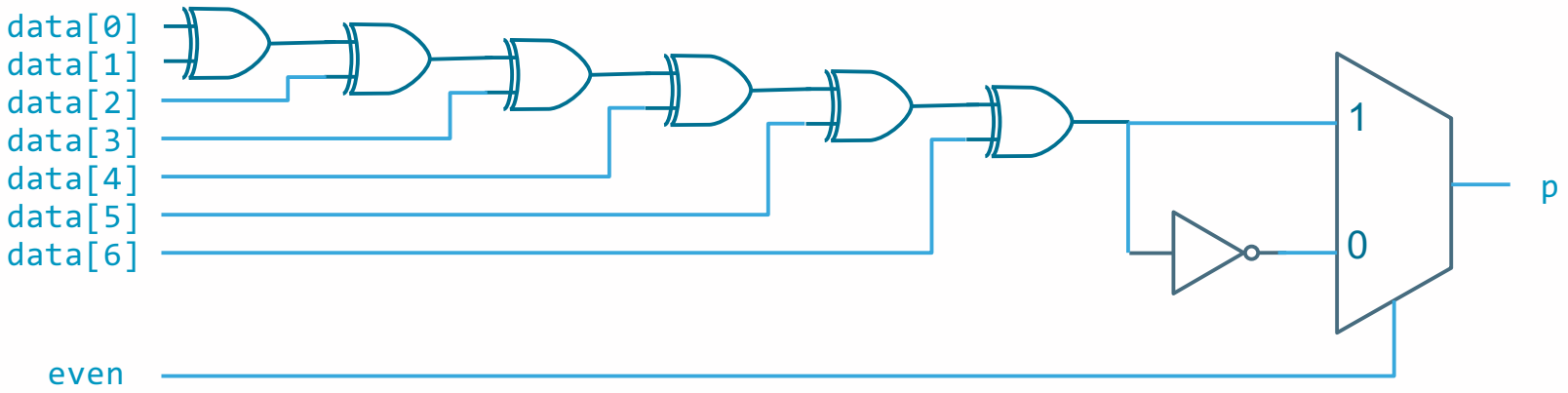
Exemple : générateur de parité 7-bits


```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY parityGen IS
  PORT (
    data : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
    even : IN STD_LOGIC;
    p: OUT STD_LOGIC
  );
END ENTITY parityGen;
```

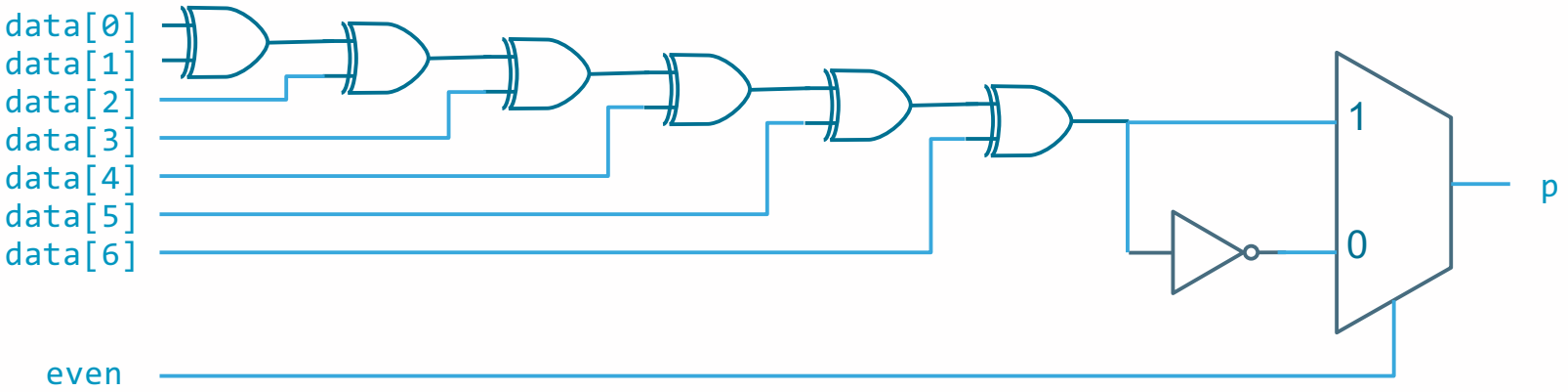
Exemple : générateur de parité 7-bits
Architecture flot de données

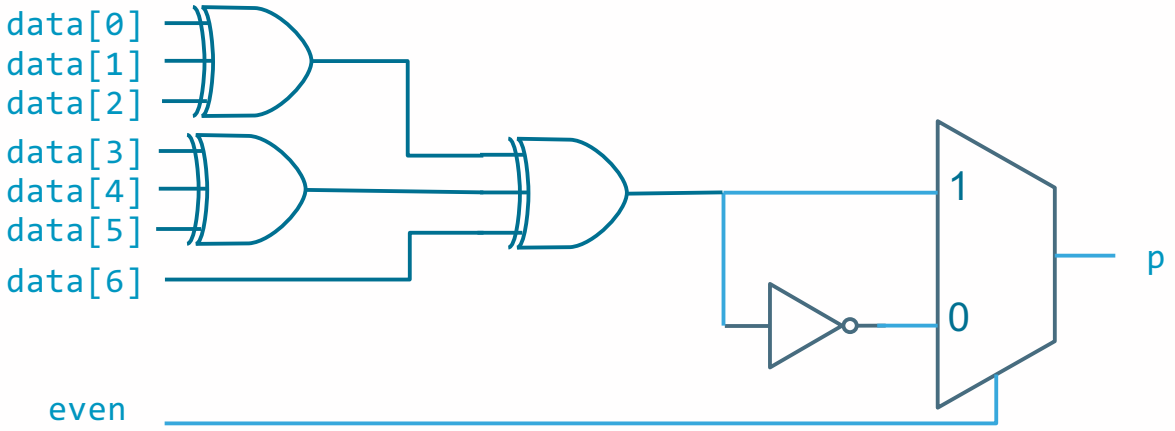

```
ARCHITECTURE parityGen_df OF parityGen IS
  SIGNAL par : STD_LOGIC;
  BEGIN
    par <= data(0) XOR data(1) XOR data(2) XOR data(3)
      XOR data(5) XOR data(6);
    WITH even SELECT
      p <=  par WHEN '1', -- parité paire
            NOT par WHEN '0',   -- parité impaire
            'X' WHEN OTHERS;
  END ARCHITECTURE parityGen_df;
```

Exemple : générateur de parité 7-bits
Architecture structurelle



Exemple : générateur de parité 7-bits
Architecture structurelle





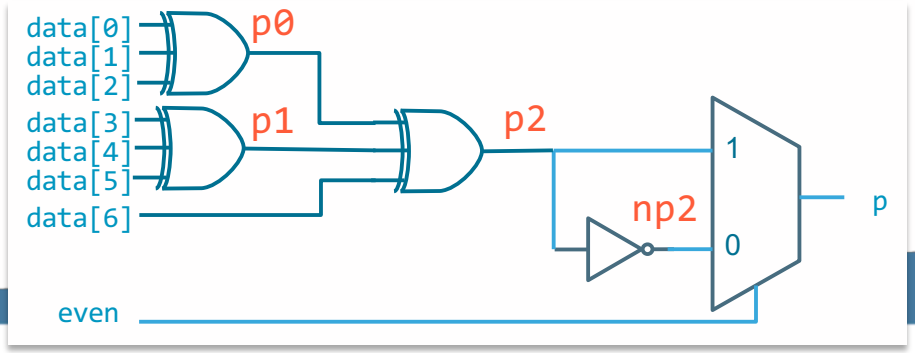
Exemple : générateur de parité 7-bits
Architecture structurelle

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY mux2_1 IS
  PORT( e0 : IN STD_LOGIC;
        e1 : IN STD_LOGIC;
        c : IN STD_LOGIC;
        s : OUT STD_LOGIC);
END ENTITY mux2_1;

ARCHITECTURE mux2_1_fd OF mux2_1 IS
  BEGIN
    s <= (e1 AND c) OR (e0 AND NOT c);
  END ARCHITECTURE mux2_1_fd;
```

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY xor3 IS
  PORT( a,b,c : IN STD_LOGIC;
        s : OUT STD_LOGIC);
END ENTITY xor3;

ARCHITECTURE xor3_fd OF xor3 IS
  BEGIN
    s <= a XOR b XOR c;
  END ARCHITECTURE xor3_fd;
```

Exemple : générateur de parité 7-bits
Architecture structurelle

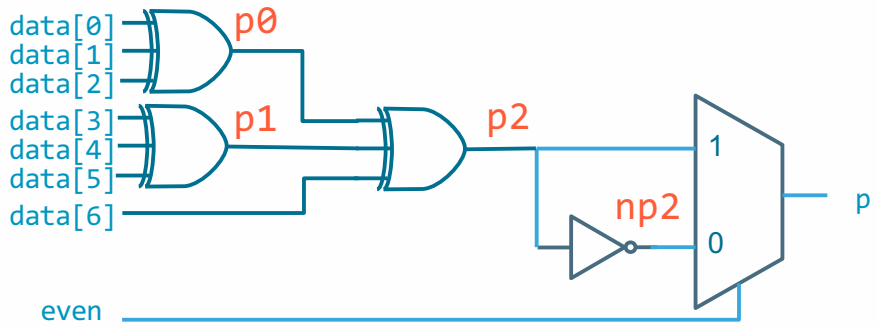
```
ARCHITECTURE parityGen_struct OF parityGen IS
  COMPONENT xor3 IS
    PORT( a,b,c : IN STD_LOGIC;
          s : OUT STD_LOGIC);
  END COMPONENT;
  COMPONENT mux2_1 IS
    PORT( e0,e1 : IN STD_LOGIC;
          c : IN STD_LOGIC;
          s : OUT STD_LOGIC);
  END COMPONENT;
  SIGNAL p0, p1, p2, np2 : STD_LOGIC;
```



Exemple : générateur de parité 7-bits
Architecture structurelle

```
ARCHITECTURE parityGen_struct OF parityGen IS
  COMPONENT xor3 IS
    PORT( a,b,c : IN STD_LOGIC;
          s : OUT STD_LOGIC);
  END COMPONENT;
  COMPONENT mux2_1 IS
    PORT( e0,e1 : IN STD_LOGIC;
          c : IN STD_LOGIC;
          s : OUT STD_LOGIC);
  END COMPONENT;
  SIGNAL p0, p1, p2, np2 : STD_LOGIC;
```

```
BEGIN
  xor3_1 : xor3
    PORT MAP (  a=>data(0), b=>data(1),
                c=>data(2), s=>p0);
  xor3_2 : xor3
    PORT MAP (  a=>data(3), b=>data(4),
                c=>data(5), s=>p1);
  xor3_3 : xor3
    PORT MAP (  a=>p0, b=>p1,
                c=>data(6), s=>p2);
  np2 <= NOT p2;
  mux : mux2_1
    PORT MAP (  e0=>np2, e1=>p2,
                c=>even, s=>p);
END ARCHITECTURE parityGen_struct;
```



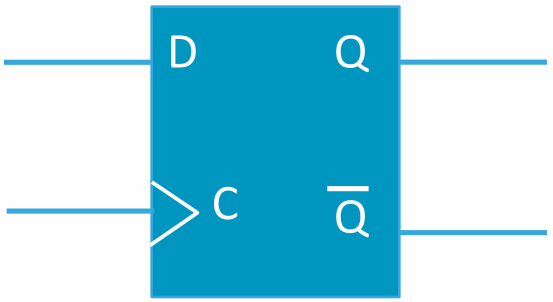
Exemple : générateur de parité 7-bits
Architecture comportementale

Par exemple :

```
ARCHITECTURE parityGen_comp OF parityGen IS
  BEGIN
    PROCESS(data,even)
      VARIABLE parityValue : STD_LOGIC;
    BEGIN
      parityValue := '0';
      FOR i IN data'RANGE LOOP
        parityValue := parityValue XOR data(i);
      END LOOP;
      IF even /= '1' THEN
        p <= NOT parityValue;
      ELSE
        p <= parityValue;
      END IF;
    END PROCESS;
  END ARCHITECTURE parityGen_comp;
```

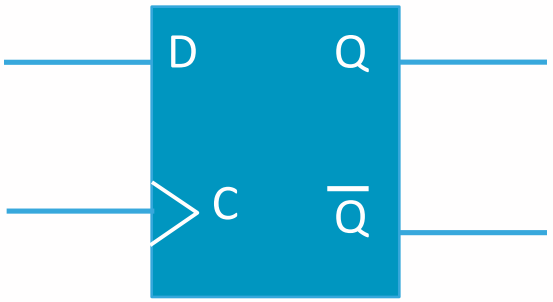
Description de composants séquentiels
Exemple : bascule D

```
ENTITY basculeD IS
  PORT( D, clk : IN STD_LOGIC;
        Q : INOUT STD_LOGIC;
        nQ : OUT STD_LOGIC);
END ENTITY basculeD;
```



Description de composants séquentiels
Exemple : bascule D

```
ENTITY basculeD IS
  PORT( D, clk : IN STD_LOGIC;
        Q : INOUT STD_LOGIC;
        nQ : OUT STD_LOGIC);
END ENTITY basculeD;
```



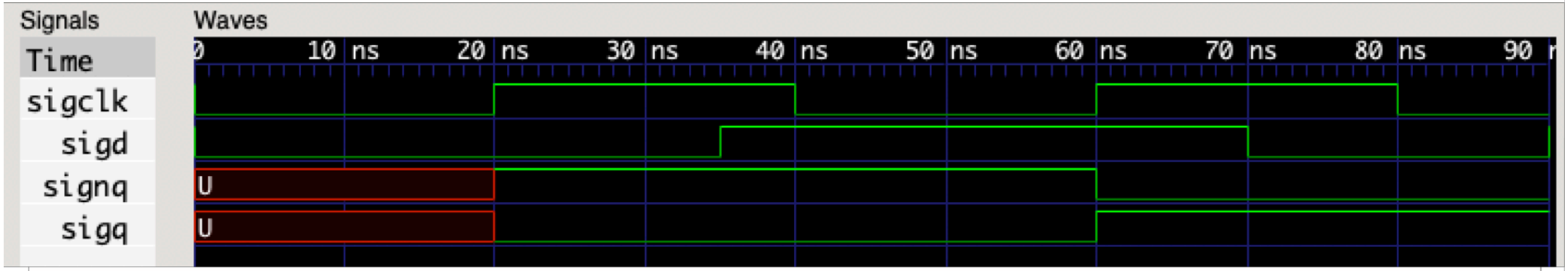

```
ARCHITECTURE compD OF basculeD IS
  BEGIN
    nQ <= NOT Q;
    PROCESS(clk)
      BEGIN
        IF RISING_EDGE(clk) THEN
          Q <= D;
        END IF;
      END PROCESS;
  END ARCHITECTURE compD;
```

Description de composants séquentiels
Exemple : test de la bascule D

```
ENTITY test_basculeD IS
END ENTITY;
```

```
ARCHITECTURE test OF test_basculeD IS
  SIGNAL sigD, sigQ, signQ, sigclk : STD_LOGIC := '0';
  BEGIN
    flipflop: ENTITY work.basculeD
      PORT MAP(sigD, sigclk, sigQ, signQ);
    clock: PROCESS
      BEGIN
        sigclk <= '0';
        WAIT FOR 20 ns;
        sigclk <= '1';
        WAIT FOR 20 ns;
      END PROCESS;
    input : PROCESS
      BEGIN
        sigD <= '1' AFTER 35 ns,'0' AFTER 70 ns, '1' AFTER 90 ns;
        WAIT;
      END PROCESS;
  END ARCHITECTURE test;
```

Description de composants séquentiels
Exemple : test de la bascule D

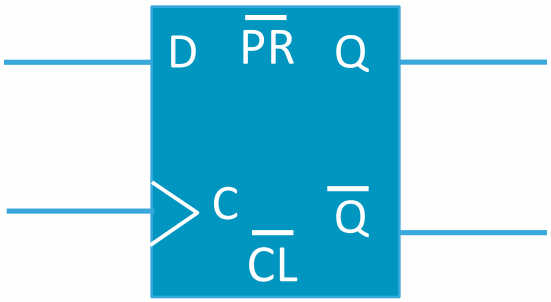



```
> ghdl -a bascule.vhd
> ghdl -e test_basculeD
> ./test_basculed --stop-time=90ns --vcd=basculeD.vcd
./test_basculed:info: simulation stopped by --stop-time @90ns
```

Description de composants séquentiels

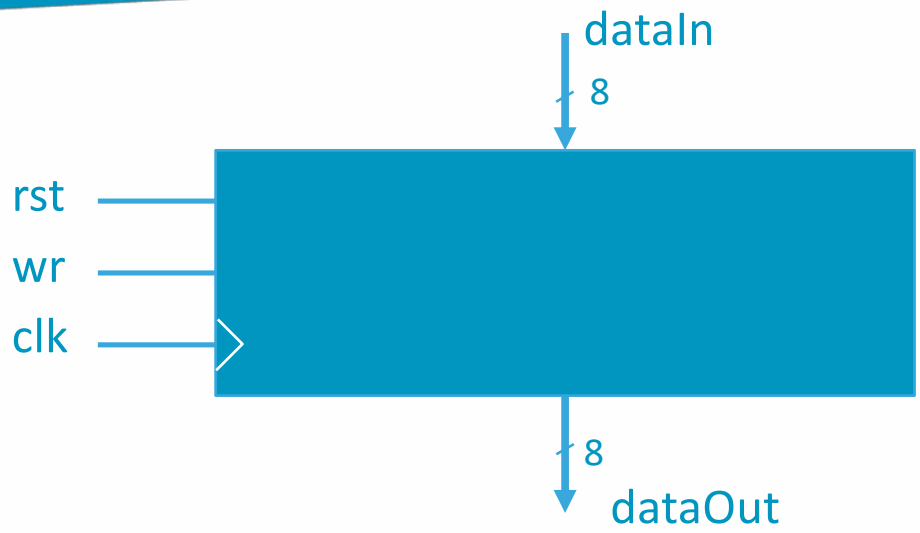
Exemple : bascule D **avec entrées asynchrones**

```
ENTITY basculeD IS
  PORT( D, clk , PR, CL: IN STD_LOGIC;
        Q : INOUT STD_LOGIC;
        nQ : OUT STD_LOGIC);
END ENTITY basculeD;
```




```
ARCHITECTURE compD OF basculeD IS
  BEGIN
    nQ <= NOT Q;
    PROCESS(clk, PR, CL)
      BEGIN
        IF (PR = '0') THEN Q <= '1';
        ELSIF (CL = '0') THEN Q <= '0';
        ELSIF RISING_EDGE(clk) THEN
          Q <= D;
        END IF;
      END PROCESS;
  END ARCHITECTURE compD;
```

Description de composants séquentiels
Exemple : registre 8 bits




```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY registre_8b IS
  PORT( clk, rst, wr : IN STD_LOGIC;
        dataIn : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        dataOut : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END ENTITY registre_8b;
```

Description de composants séquentiels
Exemple : registre 8 bits

```
ARCHITECTURE comp OF registre_8b IS
  BEGIN
    PROCESS(clk)
      BEGIN
        IF RISING_EDGE(clk)THEN
          IF (rst = '1') THEN
            dataOut <= "00000000";
          ELSIF (wr = '1') THEN
            dataOut <= dataIn;
          END IF;
        END IF;
      END PROCESS;
  END ARCHITECTURE comp;
```

Exercice : bascule JK

Décrire une bascule JK synchrone en VHDL.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY basculeJK IS
  PORT( J, K, clk : IN STD_LOGIC;
        Q : INOUT STD_LOGIC);
END ENTITY basculeJK;
```