VHDL

Ahcène Bounceur

2009

Plan de travail

1. Introduction au langage
2. Prise en main
3. Machine à état
4. Implémentation sur FPGA

1

VHDL

Introduction au langage

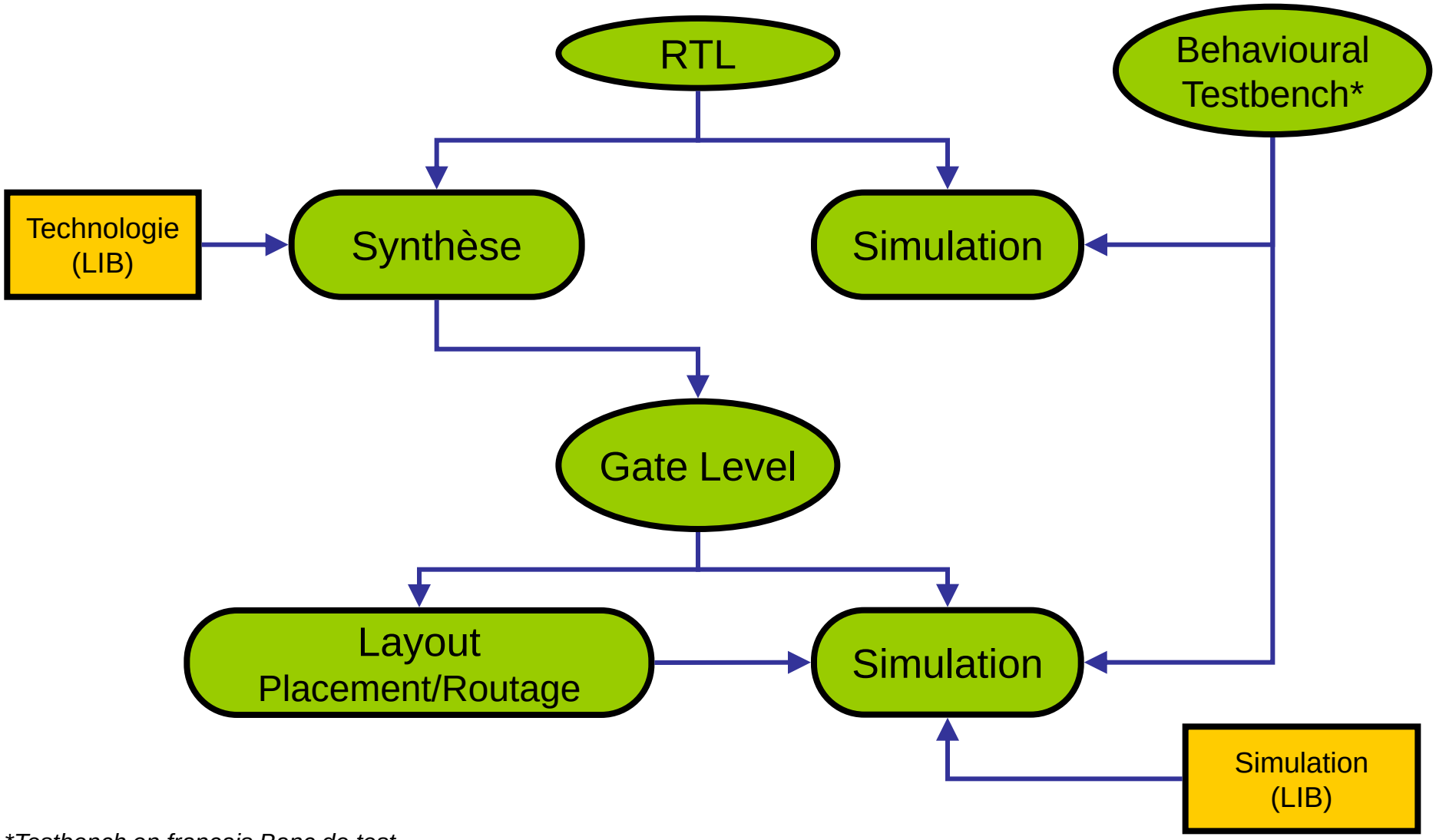
Ahcène Bounceur

VHDL

- **V**HIC (**V**ery **H**igh-speed **I**ntegration **C**ircuit)
- **H**ardware
- **D**escription
- **L**anguage

1987 normalisé IEEE
1998 IEEE std 1076
1993 revu
1998 revu à nouveau

Flot de conception



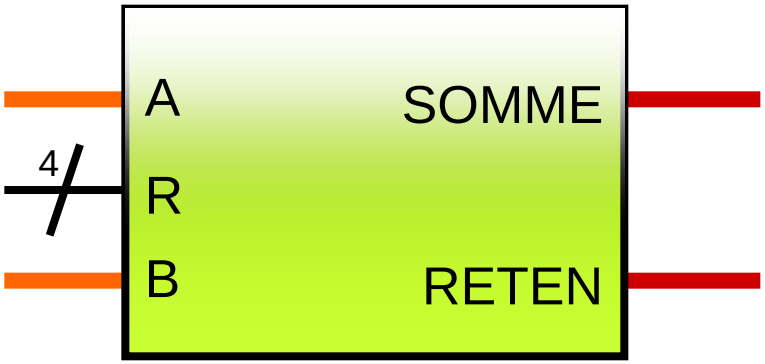
*Testbench en français Banc de test

L'En-tête

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
```

L'Entité

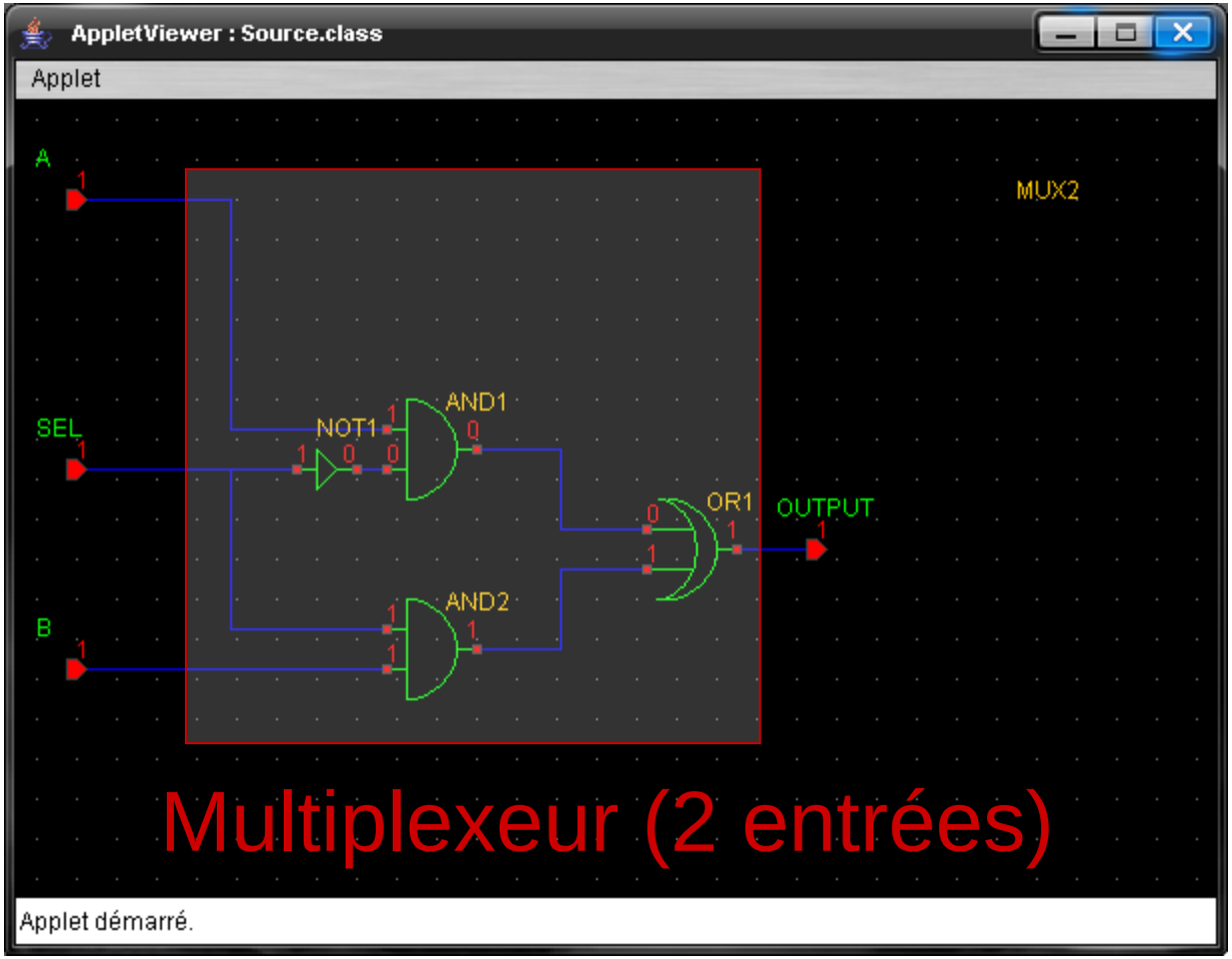
```
entity DEMIADD is
port (
      A,B : in std_logic;
      R : in std_logic_vector(3 downto 0);
      SOMME, RETEN : out std_logic;
);
end  DEMIADD;
```



L'Architecture

```
architecture ARCHI of DEMIADD is
-- déclarations préalables
begin
    SOMME <= A xor B;
    RETEN <= A and B;
end ARCHI;
```

Le Process


AppletViewer : Source.class
Applet
A
SEL
B
MUX2
NOT1
AND1
AND2
OR1
OUTPUT
Multiplexeur (2 entrées)
Applet démarré.

Le Process

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity MUX is
port (
          SEL,A,B : in std_logic;
          OUTPUT : out std_logic
);
end  MUX;

architecture ARCHI of MUX is
begin
   MUXPROCESS : process(A,B,SEL)
   begin
        if SEL='1' then
              OUTPUT <= B;
        else
              OUTPUT <= A;
        end if;
   end process MUXPROCESS;
end ARCHI;
```

Le Process

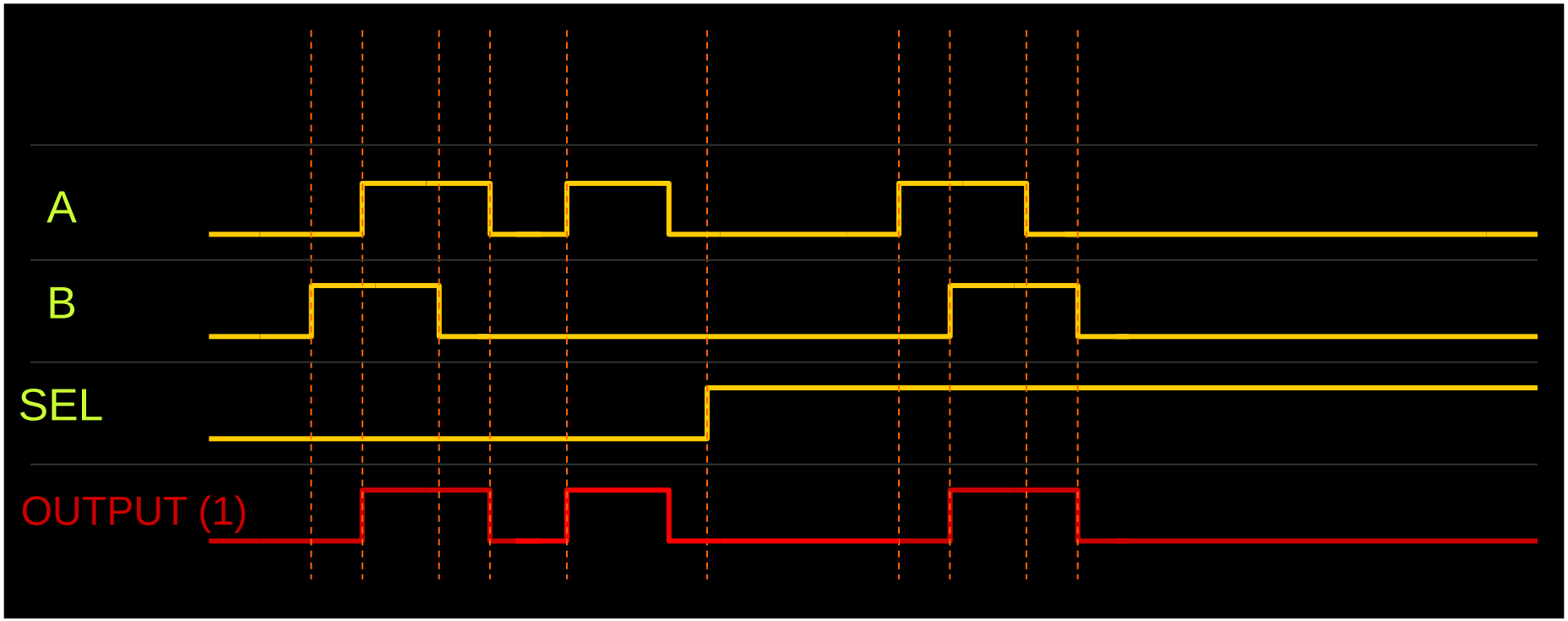
```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity MUX is
port (
        SEL,A,B : in std_logic;
        OUTPUT : out std_logic
);
end  MUX;

architecture A of MUX is
begin
   MUXPROCESS : process(A,B,SEL)
   begin
       if SEL='1' then
             OUTPUT <= B;
       else
             OUTPUT <= A;
       end if;
   end process MUXPROCESS;
end A;
```

Le Process

```
MUXPROCESS : process(A,B,SEL)
```



Le Process

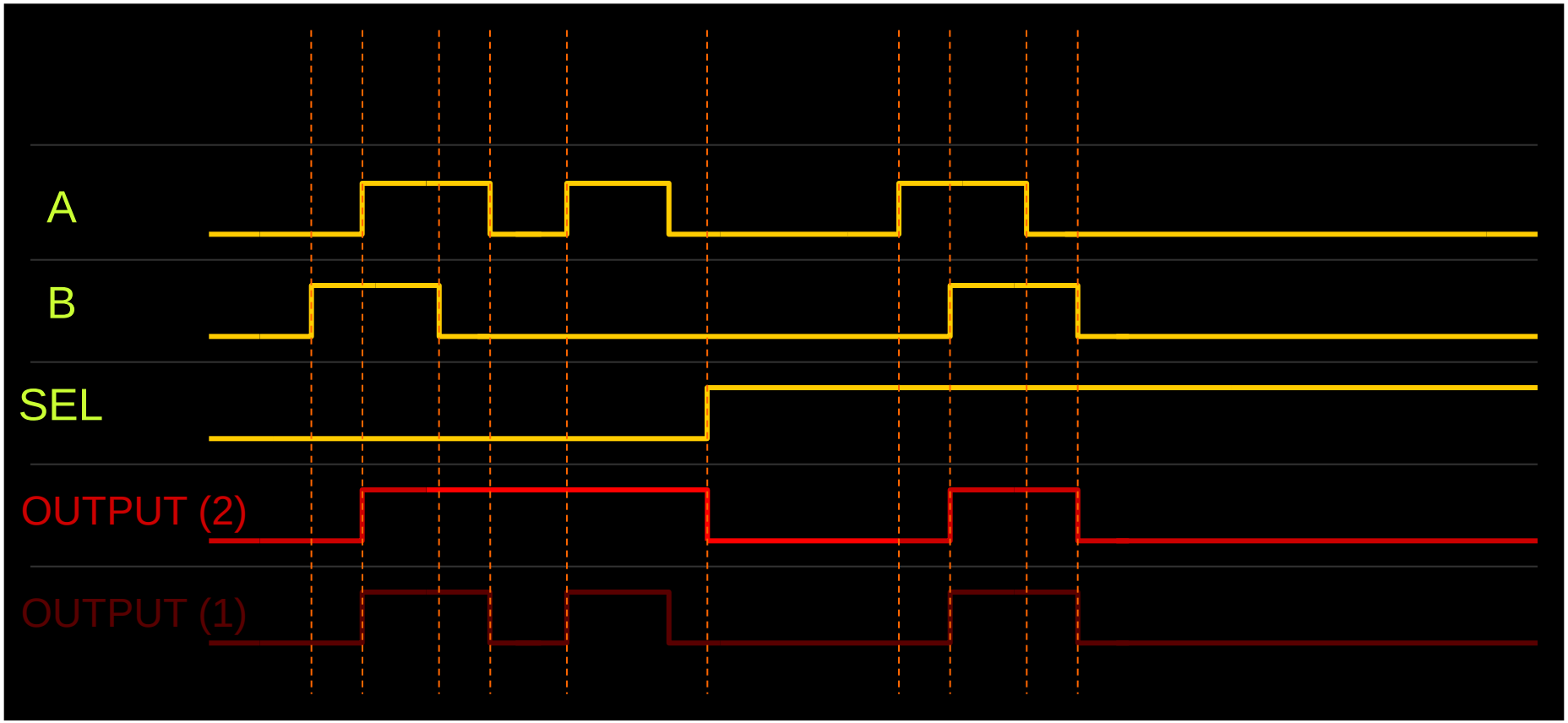
```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity MUX is
port (
        SEL,A,B : in std_logic;
        OUTPUT : out std_logic
);
end  MUX;

architecture A of MUX is
begin
   MUXPROCESS : process(A,B,SEL)
   begin
       if SEL='1' then
             OUTPUT <= B;
       else
             OUTPUT <= A;
       end if;
   end process MUXPROCESS;
end A;
```

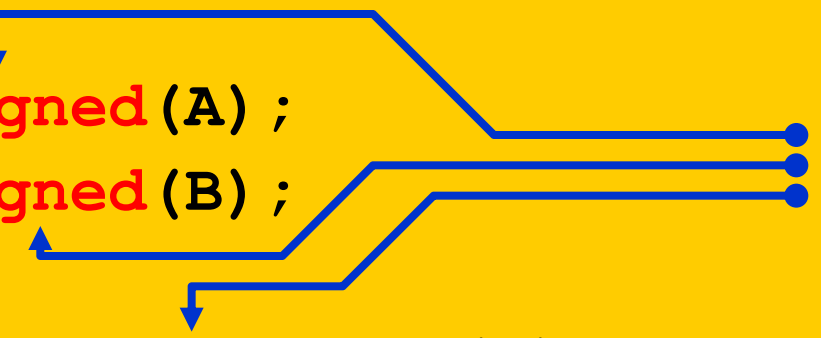
Le Process

```
MUXPROCESS : process(B,SEL)
```



Librairie std_logic_arith

```
architecture archi of my_entity is
signal A,B,C : std_logic_vector(5 downto 0);
signal D,E,F : unsigned(5 downto 0)
begin
  D <= unsigned(A);
  E <= unsigned(B);
  F <= D+E;
  C <= std_logic_vector(F);
end;
```



| + | - | * | / | = |

| < | <= | > | >= | = | /= |

Agrégats et Concaténation

```
architecture archi of my_entity is
signal A,B,C : std_logic;
signal Z : std_logic_vector(1 downto 0)
signal S : std_logic_vector(7 downto 0)
Begin
  Z <= (A,B);
  S <= (7=>'1' , 5 downto 4=>Z , 6=>C , others=> '0');
  S <= '1' & C & Z & "0000";
  S(7) <= '1';
  S(6) <= C;
  S(5 downto 4) <= Z;
  S(3 downto 0) <= "0000";
end;
```

Attributs des signaux et Vecteurs

```
signal A : std_logic_vector(5 downto 0);
```

- **A'high** = 5
- **A'low** = 0
- **A'left** = 5
- **A'right** = 0
- **A'range** = 5 downto 0
- **A'reverse_range** = 0 downto 5
- **A'length** = 6
- **A'ascending** = false

Attributs des signaux et Vecteurs

```
signal X : std_logic;
```

X'event = true ***quand X change de valeur***

Règles d'écriture

- Un signal est défini par son type et sa taille
- Tout objet doit être déclaré avant d'être utilisé
- Les noms de signaux, les labels, etc. doivent respecter les mêmes règles que les variables en langage soft (C/C++ et Java) et pas d'underscore à la fin
- Le VHDL est cas insensitive : abc, Abc ou ABC représentent le même nom
- Utiliser -- pour introduire un commentaire et pas de commentaire sur plusieurs lignes
- Na jamais utiliser X <= X + Y; si X et Y ne figurent pas dans la liste des signaux de la liste de sensibilité

L'écriture séquentielle et parallèle

if

```
if condition then
  -- séquence d'opérations
end if;
```

```
if condition then
  -- séquence 1 d'opérations
else
  -- séquence 2 d'opérations
end if;
```

L'écriture séquentielle et parallèle

if

```
if condition1 then
  -- séquence 1 d'opérations
elseif condition2 then
  -- séquence 2 d'opérations
elseif condition3 then
  -- séquence 3 d'opérations
else
  -- séquence n d'opérations
end if;
```

L'écriture séquentielle et parallèle

case

```
case expression is
when valeur1 =>
  -- séquence 1 d'opérations
when valeur2 to valeur5 =>
  -- séquence 2 d'opérations
when valeur6 | valeur8 =>
  -- séquence 3 d'opérations
when others =>
  -- séquence 4 d'opérations
end case;
```

L'écriture séquentielle et parallèle

for

```
signal X,Y : std_logic_vector(3 downto 0);
--
for I in 0 to 3 loop
  Y(I) <= IP(3-I);
end loop;
```

Assignation Concurrente

```
process(A,B,C,T)
begin
   if T>5 then
      Y <= A;
   elseif T<5 then
      Y <= B;
   else
      Y <= C;
   end if;
end process;
```

if

dans un process

hors process

```
Y <= A when T>5 else B when T<5 else C;
```

Assignation Concurrente

```
process(A,B,C,T)
begin
   case T is
   when 0 to 4 =>
      Y <= B;
   when 5 =>
      Y <= C;
   when others =>
      Y <= A;
   end case;
end process;
```

case

dans un process

hors process

```
with T select
Y <= B when 0 to 4, C when 5, A when others;
```

Variables vs Signaux

variables

```
…
signal A,B,P :
  integer;
begin
  process(A,B)
  variable vM,vN : integer
  begin
      vM := A;
      vN := B;
      P <= vM + vN;
  end process;
end ARCHI;
```

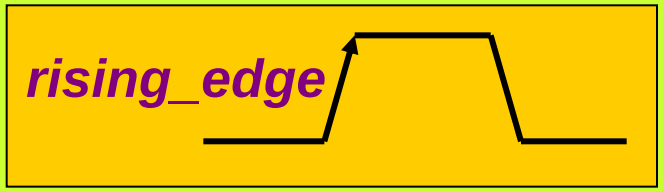
signaux

```
…
signal A,B,P : integer;
signal sM,sN : integer;
begin
  process(A,B,sM,sN)
  begin
      sM <= A;
      sN <= B;
      P <= sM + sN;
  end process;
end ARCHI;
```

Process CLOCKé

```
Library IEEE;
…
entity …
architecture …
signal CLK,RESET :std_logic;
signal count : unsigned(3 downto 0);
begin
process(CLK)
begin
   if CLK'event and CLK='1' then
        if RESET='1' then
                count <= "0000";
        elsif count>=9 then
                count <= "0000";
        else
                count <= count + 1;
        end if;
   end if;
end process;
end architecture;
```

- Machine à état SYNCHRONE
- Process réagissant sur front d'horloge



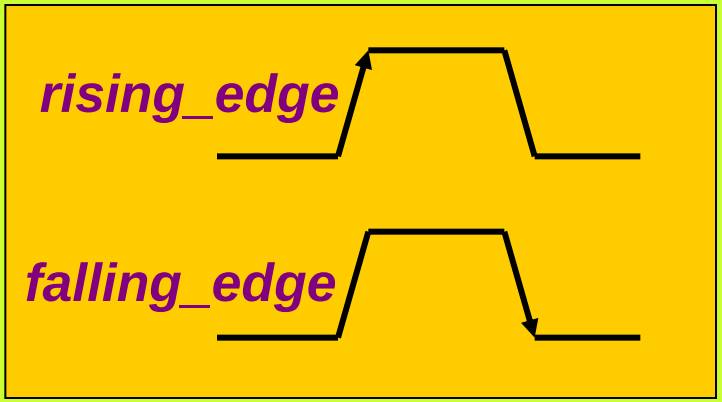

```
if rising_edge(CLK) then
```

Process CLOCKé

Éviter les if (pour la synthèse)

```
Library IEEE;
…
entity …
architecture …
signal CLK,RESET :std_logic;
signal count : unsigned(3 downto 0);
begin
process(CLK)
begin
   wait until CLK'event and CLK='1';
        if RESET='1' then
                count <= "0000";
        elsif count>=9 then
                count <= "0000";
        else
                count <= count + 1;
        end if;

end process;
end architecture;
```




```
wait until rising_edge(CLK);
```

Reset Asynchrone

```
begin
process(CLK,RESET)
Begin
   if RESET='1' then
        count <= "0000";
   elsif (CLK'event and CLK='1') then
        …
   end if;
end process;
```

```
begin
process(CLK,RESET)
Begin
   if RESET='1' then
        count <= "0000";
   end if;
   wait until CLK'event and CLK='1';
        …
end process;
```

Quelques Remarques

- Ne pas considérer le cas rising_edge et falling edge au même temps
- Faire attention de bien écrire un process synchrone

```
wait until CLK'event and (CLK='1' or CLK='0');
if CLK'event and (CLK='1' or CLK='0') then

wait until CLK'event
if CLK'event then

wait until CLK='1';
if CLK='1' then
```

Un grand merci pour votre attention