Synthèse avec VHDL

Structure générale (exemple):

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

entity AddLog is
  port (x1, x0, y1, y0 : in std_logic;
        s2, s1, s0     : out std_logic);
end AddLog;

architecture equations of AddLog is
begin
  s2 <= (x1 and y1) or (x1 and x0 and y0) or (x0 and y1 and y0);
  s1 <= (x0 and y0) xor (x1 xor y1);
  s0 <= x0 xor y0;
end equations;
```

Signaux

Types:
std_logic (valeurs utilisées: 'X', '0', '1', 'Z') ou **std_logic_vector**

Tous les signaux sont globaux

Variables

locales à un processus
ne représentent rien dans la réalité
interdites!

Façons d'implémenter le corps:

structurelle
dataflow
comportementale

Processus implicites / explicites:

```
architecture toto of test is
begin
c <= a and b;
z <= c when oe='1' else 'Z';
seq: process (clk, reset)
     begin
       .
       .
     end process;
end toto;
```

processus implicites

processus explicite

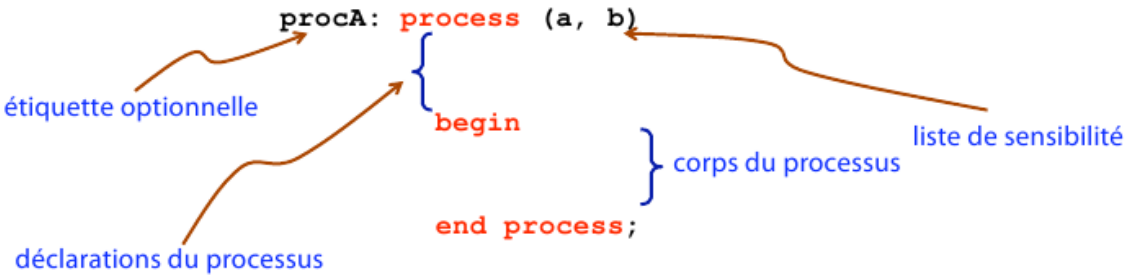
Processus implicites:

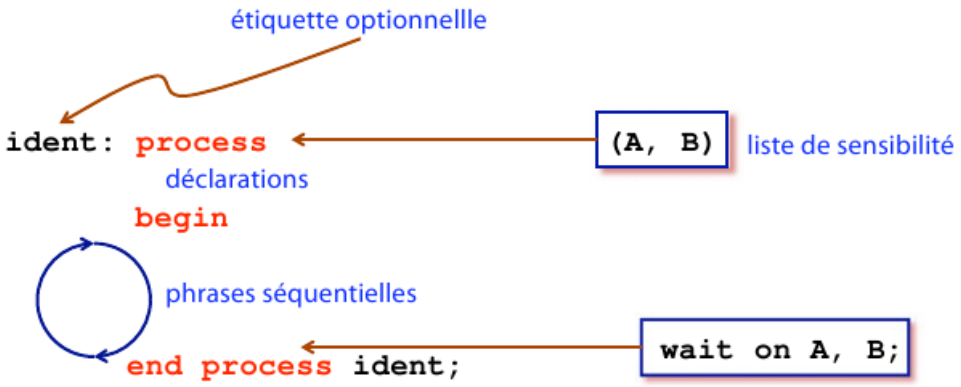
```
toto <= x when a=1 else
        y when a=2 else
        "0000";
```

le **generate** (équivalent d'un **for**).
Exemple:

```
G1: for i in 0 to 5 generate
      result(i) <= ena and InputSignal(i);
    end generate;
```

Processus explicites:





Un **wait on** à la fin d'un processus peut être remplacé par une liste de sensibilité, placée juste après le mot clé **process**. La liste de sensibilité est incompatible avec un **wait**: c'est l'un ou l'autre

Phrases séquentielles:

```
if condition then
 phrases
  { elsif condition then
 phrases }
  [ else
 phrases ]
end if;
```

```
case opcode is
      when X"00" => add;
      when X"01" => subtract;
      when others => illegal_opcode;
    end case;
```

```
 loop
       faire;
     end loop;
```

```
 while toto < tata loop
       toto := toto + 1;
     end loop;
```

```
     for item in 1 to last_item loop
       table(item) := 0;
     end loop;
```

```
     next [ label ] [ when condition ];
```

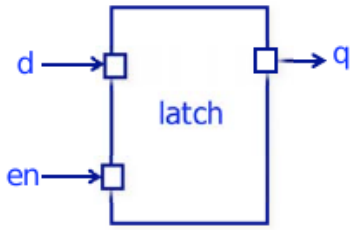
```
     exit [ label ] [ when condition ];
```

Opérateurs:

- Opérations logiques:
  - `and or nand nor xor xnor`
- Opérations de comparaison:
  - `= /= < <= > >=`
- Opérations de décalage
  - `sll srl sla sra rol ror`
- Opérations d'addition:
  - `+ - &`
- Opérations de signe:
  - `+ -`
- Opérations de multiplication:
  - `* / mod rem`
- Opérations diverses:
  - `not abs **`

Latch: Interdit!

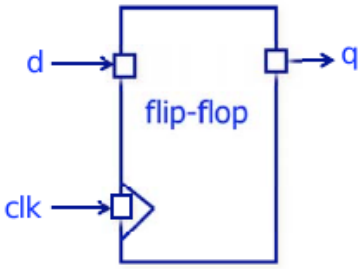
```
process (en, d)
  begin
  if en='1'
    then q <= d;
  end if;
end process;
```



et tout processus similaire aussi, car il va créer un latch

flip-flop (sans reset, mauvais)

```
process (clk)
begin
  if clk'event and clk='1'
      then q <= d;
  end if;
end process;
```



Ne pas oublier le reset:

Reset asynchrone:

```
process (clk, reset)
begin
  if reset='1'
    then q <= '0';
    elsif clk'event and clk='1'
      then q <= d;
  end if;
end process;
```

Reset synchrone:

```
process (clk)
begin
  if clk'event and clk='1'
    then if reset='1'
           then q <= '0';
           else q <= d;
         end if;
  end if;
end process;
```

plus facile: utiliser

```
if rising_edge(clk)
```

au lieu de

```
if clk'event and clk='1'
```