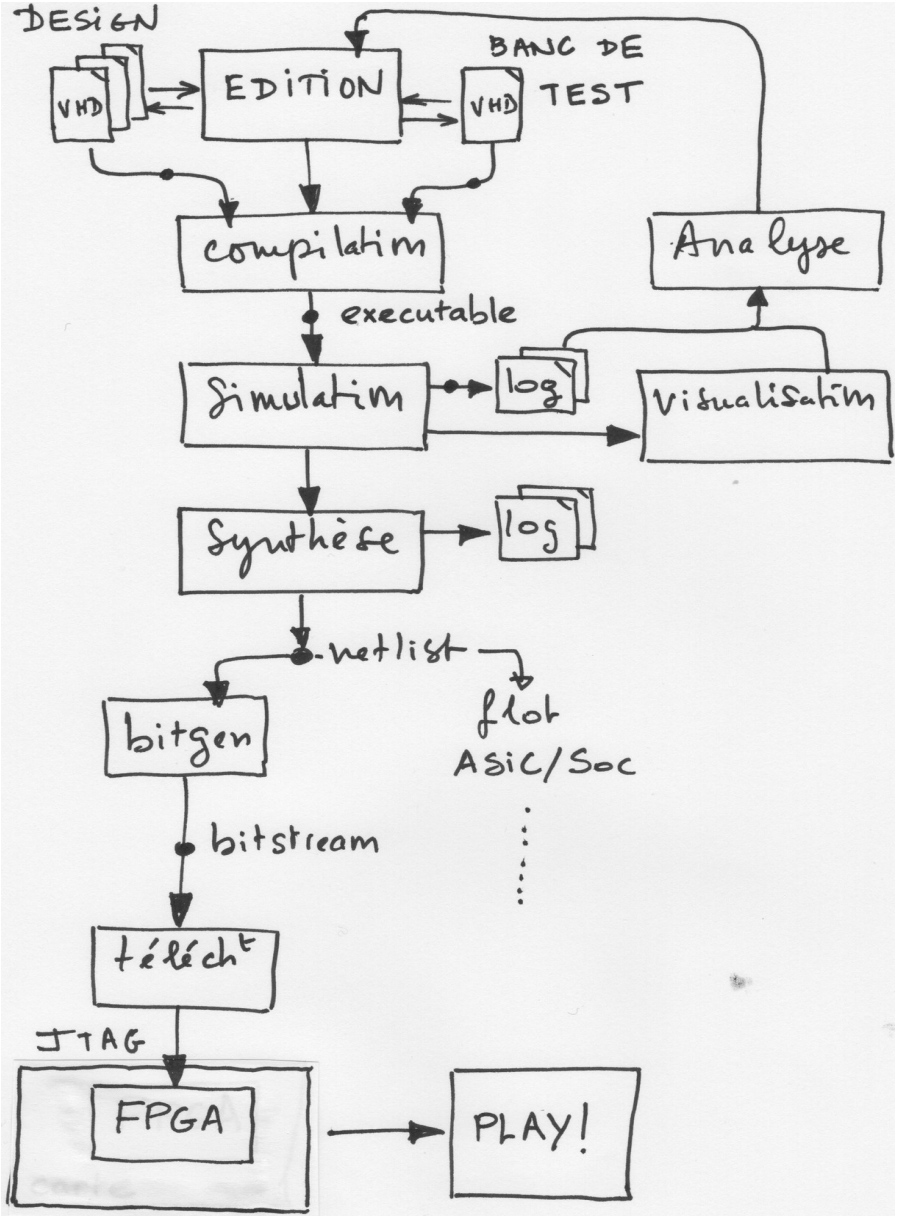
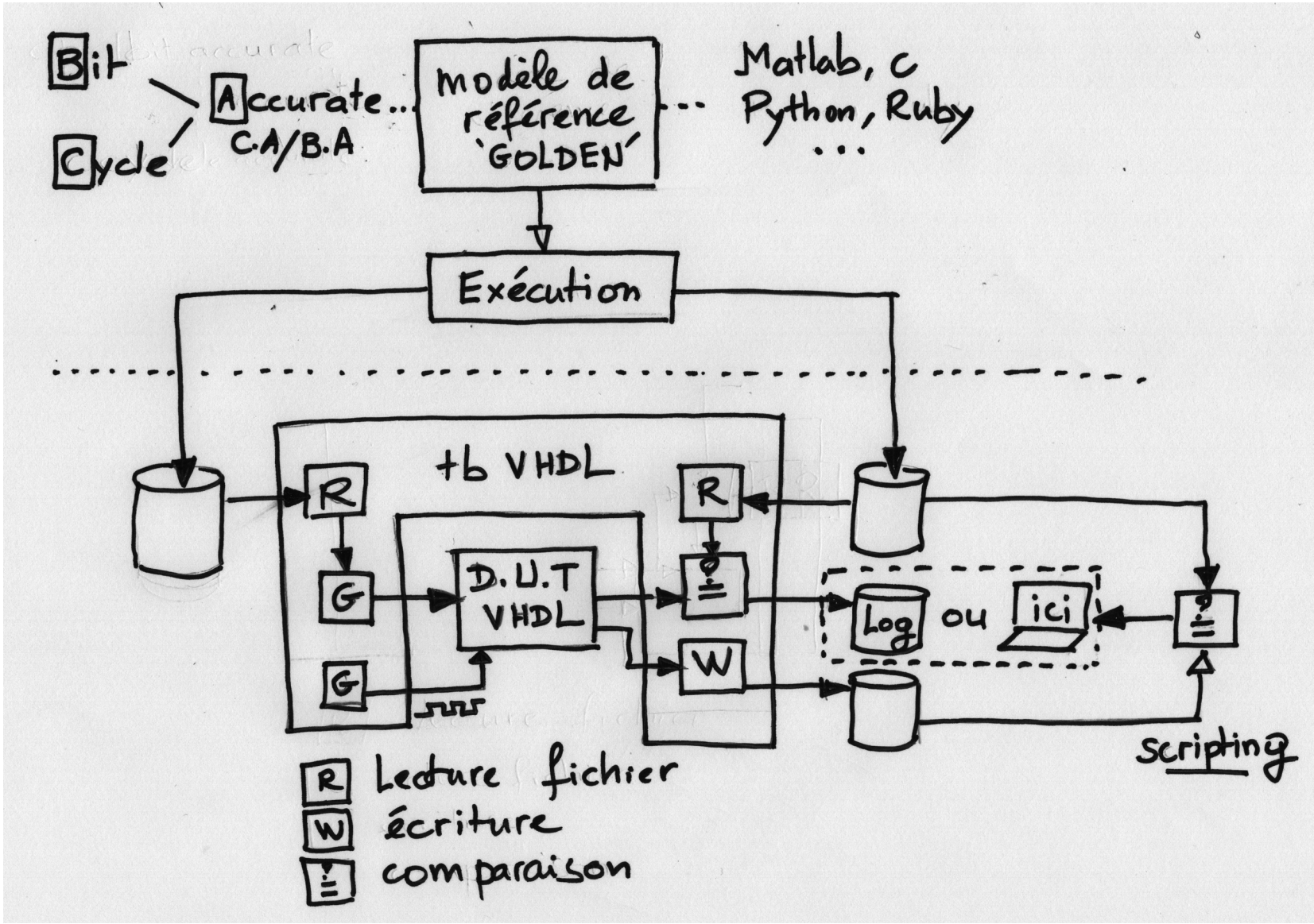
Introduction à VHDL

Flot de conception général



Importance des testbenchs



Histoire

- Le **V**HSIC **H**ardware **D**escription **L**anguage (VHSIC = Very High Speed Integrated Circuit)
- normalisé en 1987 par l'IEEE,
- Modification en 1993 IEEE 1076-93.
  - version du langage qui est majoritairement supportée par les outils du marché.
- Evolutions « mineures », et pas forcement supportés par les logiciels.

Histoire

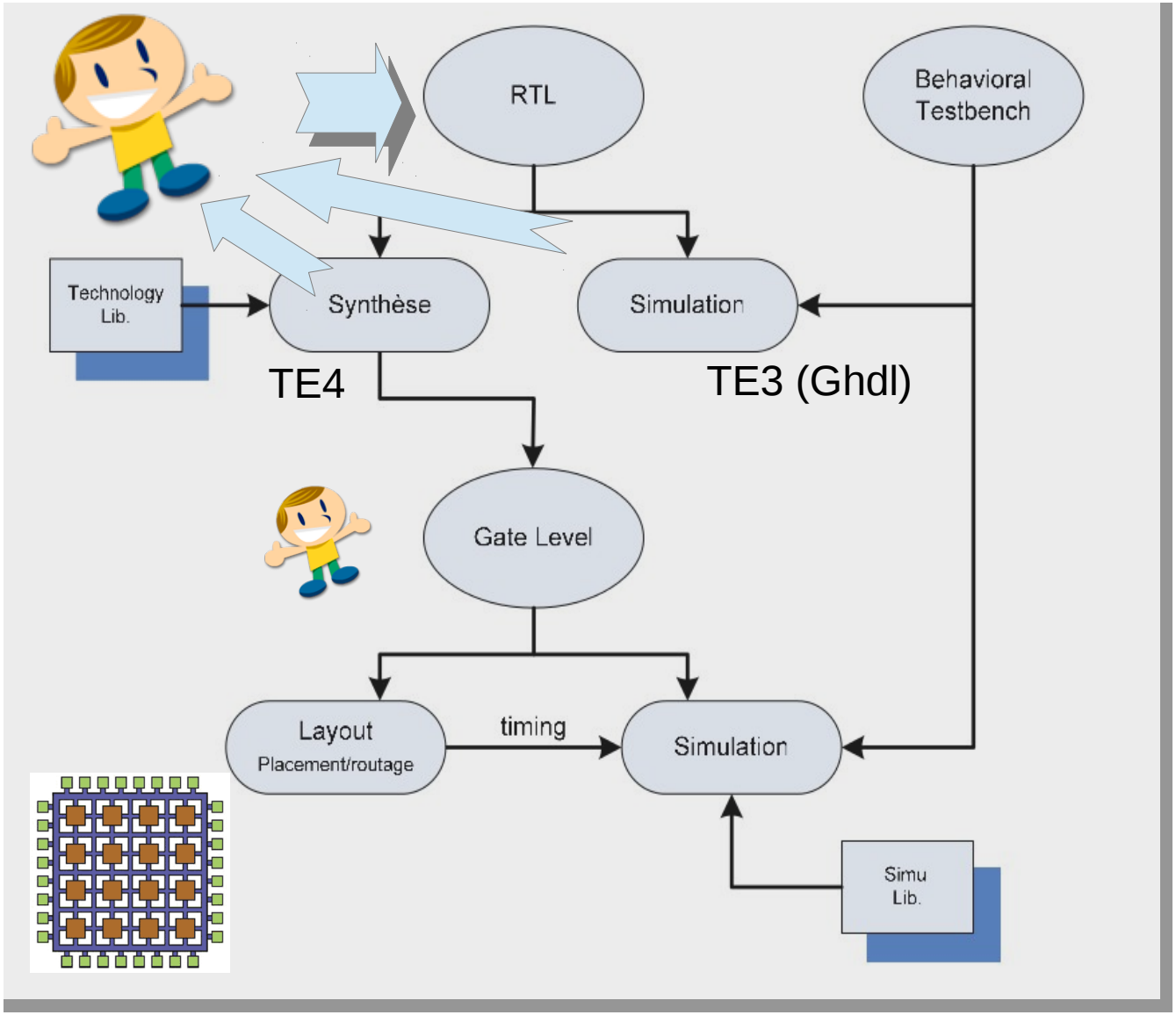
- Le VHDL est un langage de description matériel, qui ne peut pas être considéré comme un langage « software » comme le C ou le Java.
- A partir de ce langage, on peut définir un système comme une **structure hiérarchique** matérielle, et préciser son **comportement temporel**

VHDL : un langage riche !

- **Différents niveaux d'abstraction possibles**
  - 1) Abstraction très élevée, algorithme distribué.
  - 2) RTL : register transfer level. Définir le système par une architecture de type machine de Moore ou Mealy et des éléments de calculs (datapath).
  - 3) Abstraction basse : un niveau proche du matériel, où l'on décrit le système par un ensemble de porte logique et d'interconnexion (« gate level »).
  - etc
- C'est le **niveau RTL** que l'on utilise le plus quand on fait de la synthèse.
  - niveau algorithmique n'est pas forcement synthétisable, il est plutôt utilisé pour faire des bancs de tests ou de la simulation.
  - Le niveau « gate level » est souvent lourd à décrire quand le système devient un tant soit peu complexe...
- L'avantage du code RTL est qu'il est **indépendant des technologies utilisées** dans la cible à programmer, alors qu'au niveau « gate », il faut prendre en compte la techno du circuit cible, car tout n'est pas permis.

Flot de conception VHDL

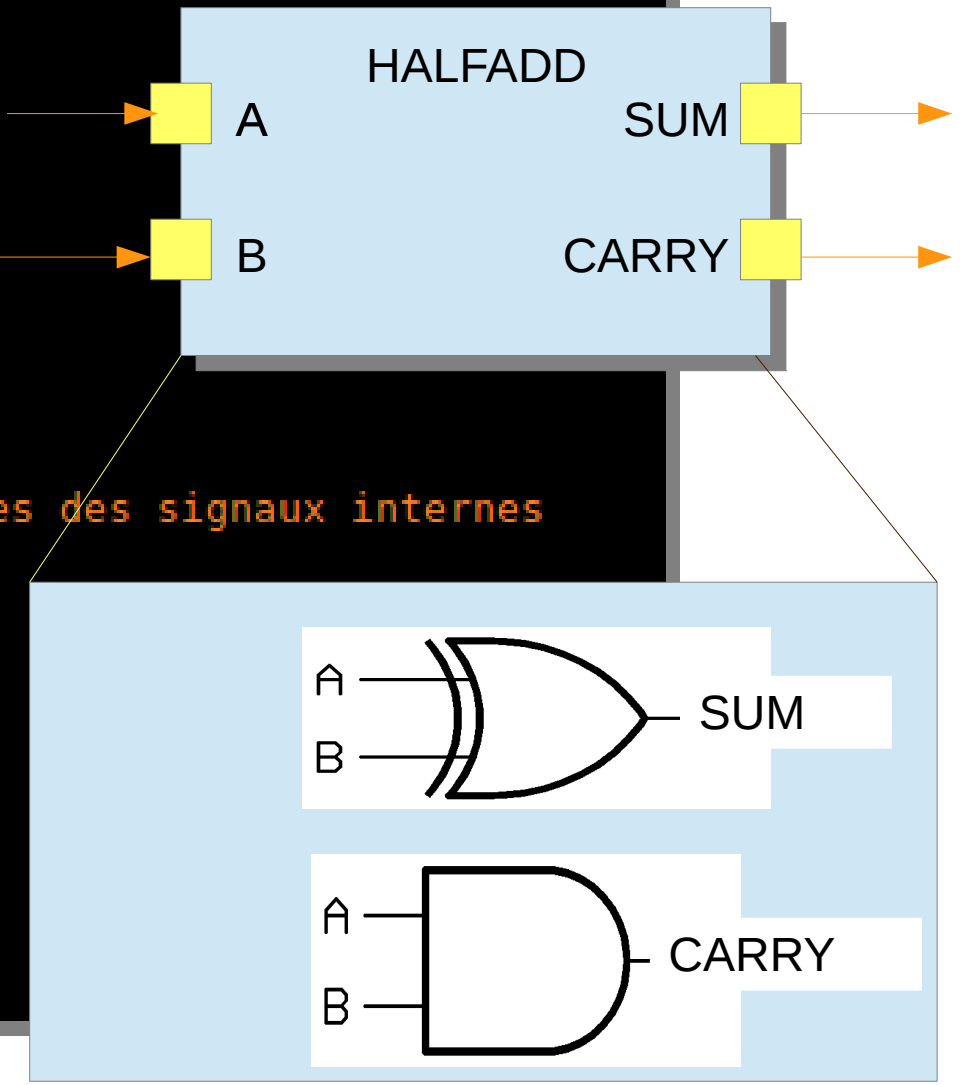
le plus fréquent



FPGA
SOC

A quoi ça ressemble ?

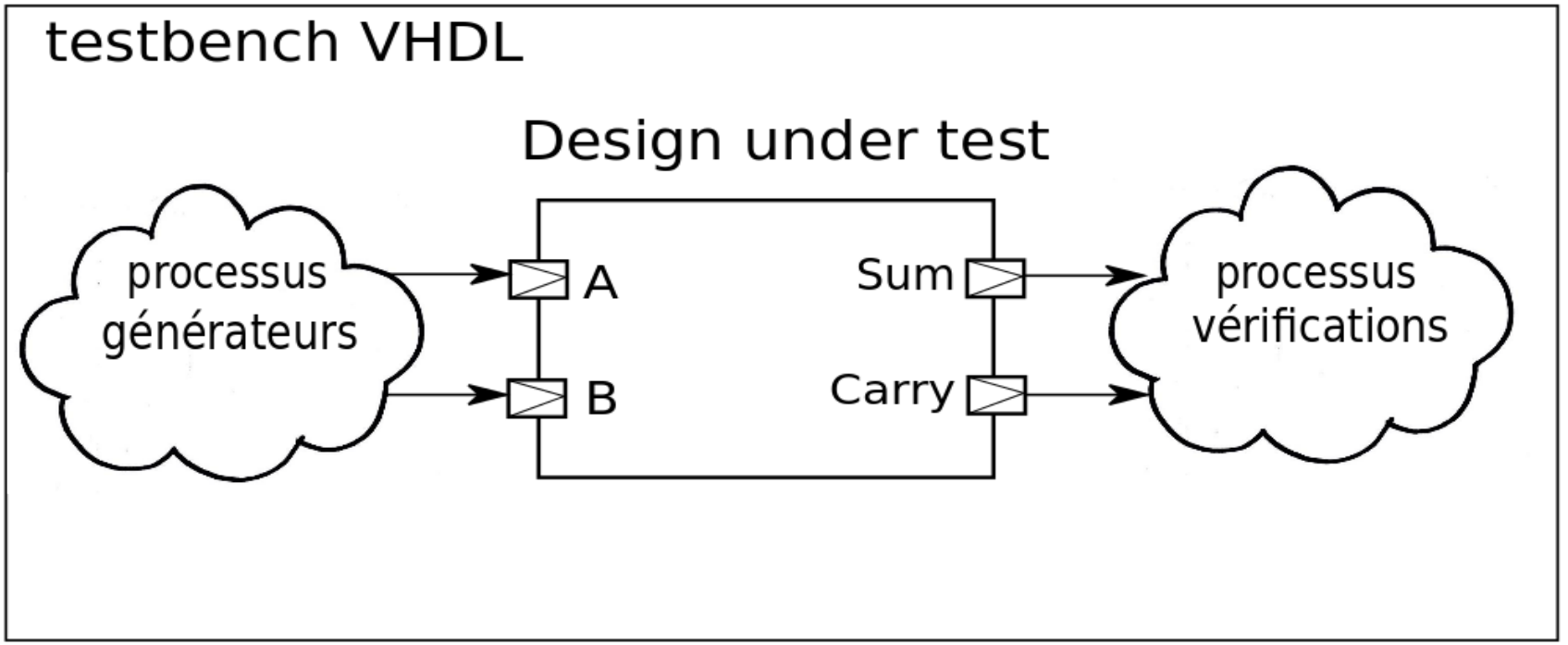
```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity HALFADD is
  port (
    A, B       : in  std_logic;
    SUM, CARRY : out std_logic
    );
end HALFADD;

architecture RTL of HALFADD is
  -- Declarations prealables eventuelles des signaux internes
  signal inutile_ici : std_logic;
  -- puis corps :
begin
  SUM   <= A xor B;
  CARRY <= A and B;
end RTL
```



Notion de testbench

- « Banc de test »
  - Allusion aux expérimentations sur table
- But :
  - Alimenter le circuit « sous test » (under test) en :
    - Générateurs classiques : clk, reset
    - Données significatives du point de vue de l'application
  - Vérifier les données sorties
- *Normalement* un testbench évite de regarder les chronogrammes, assez difficiles à analyser...

Testbench : suite



Avant goût du TE3

- Application de la technique classique de synthèse logique d'une FSM
  - Enumérer les entrées et les états
  - Calculer les sorties possibles
  - Déduire les équations logiques
    - des entrées D des bascules D
    - Des sorties
    - K-map si nécessaires
- Codage en VHDL !
  - Puis simulation avec GHDL

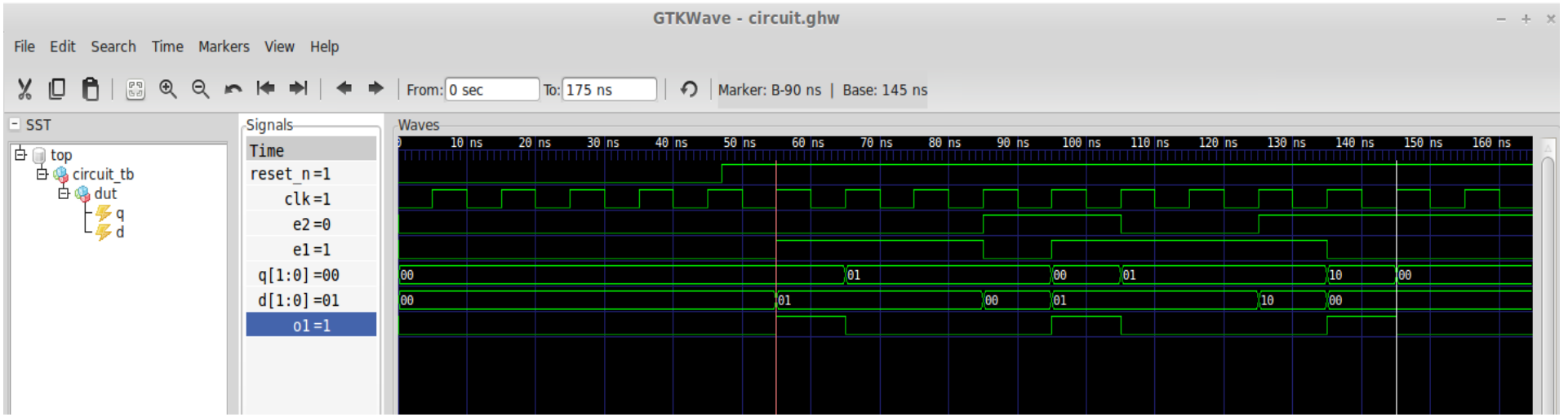
GHDL

- Permet la simulation (pas la synthèse!) de circuits numériques
- A partir de fichiers VHDL :
  - Analyse de tous les fichiers (y compris tb)
    - détecte des erreurs
    - génère un binaire
    - **ghdl -a toto.vhd**
  - Elaboration du testbench
    - **ghdl -e toto_tb**
      - Sans le .vhd
  - Lancement du simulateur
    - **ghdl -r toto_tb - - wave=toto.ghw**

Analyse des waveforms

gtkwave

Gtkwave wave.ghw =>



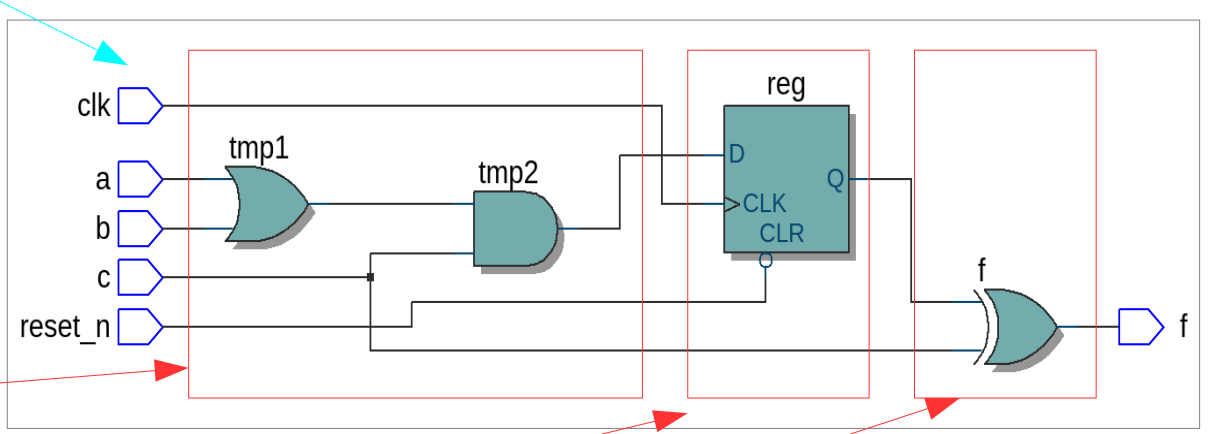
Du VHDL au circuit

Passage en revue de quelques exemples

Exemple 1 : équations et bascule D

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

entity example_1 is
  port(
    reset_n : in  std_logic;
    clk     : in  std_logic;
    a, b, c : in  std_logic;
    f       : out std_logic
    );
end entity;

architecture logic of example_1 is

  signal tmp1, tmp2 : std_logic;
  signal reg        : std_logic;

begin

  tmp1 <= a or b;
  tmp2 <= tmp1 and c;

  dff : process(reset_n, clk)
  begin
    if reset_n = '0' then
      reg <= '0';
    elsif rising_edge(clk) then
      reg <= tmp2;
    end if;
  end process;

  f <= reg xor c;

end logic;
```

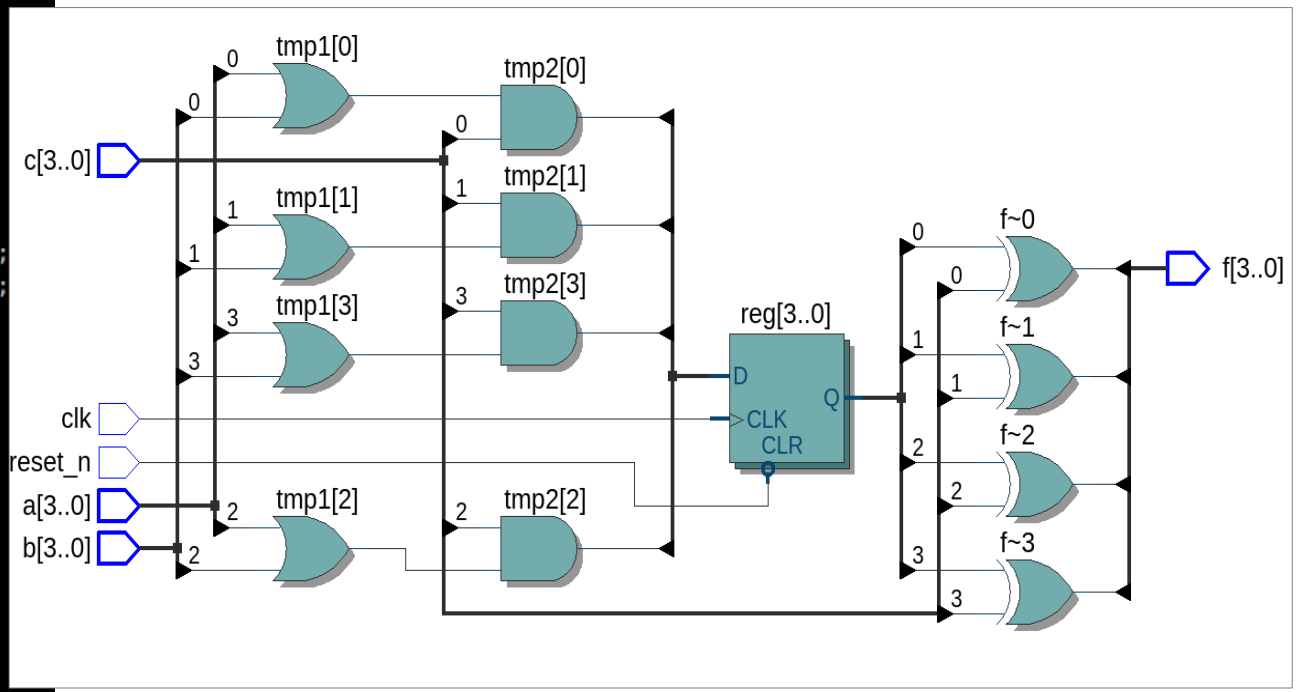
Synthèse automatique (Altera QUARTUS)



Exemple 2 : bit vector et bascules D

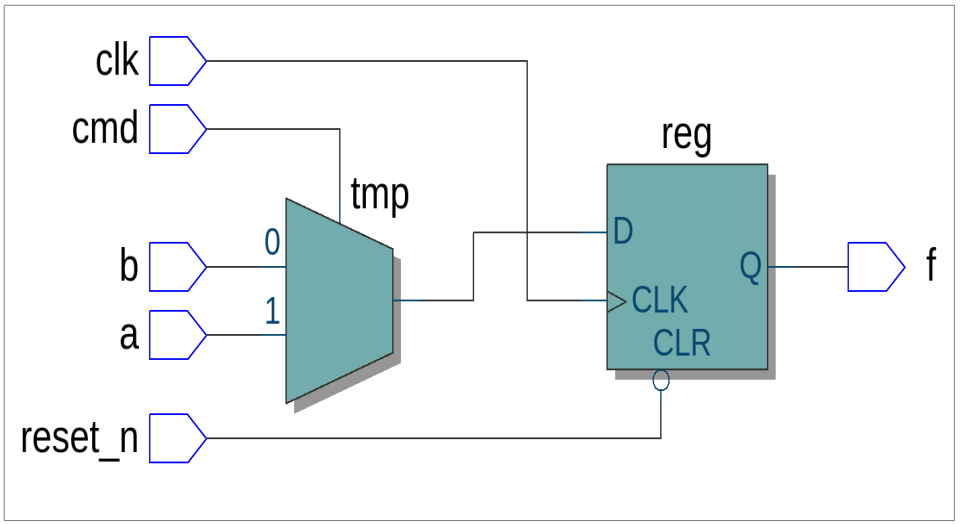
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity example_2 is
  port(
    reset_n : in  std_logic;
    clk     : in  std_logic;
    a, b, c : in  std_logic_vector(3 downto 0);
    f       : out std_logic_vector(3 downto 0)
    );
end entity;

architecture logic of example_2 is

  signal tmp1, tmp2 : std_logic_vector(3 downto 0);
  signal reg        : std_logic_vector(3 downto 0);

begin

  tmp1 <= a or b;
  tmp2 <= tmp1 and c;

  dff : process(reset_n, clk)
  begin
    if reset_n = '0' then
      reg <= "0000";
    elsif rising_edge(clk) then
      reg <= tmp2;
    end if;
  end process;

  f <= reg xor c;

end logic;
```

Vecteurs de bits



Exemple 3 : multiplexeur

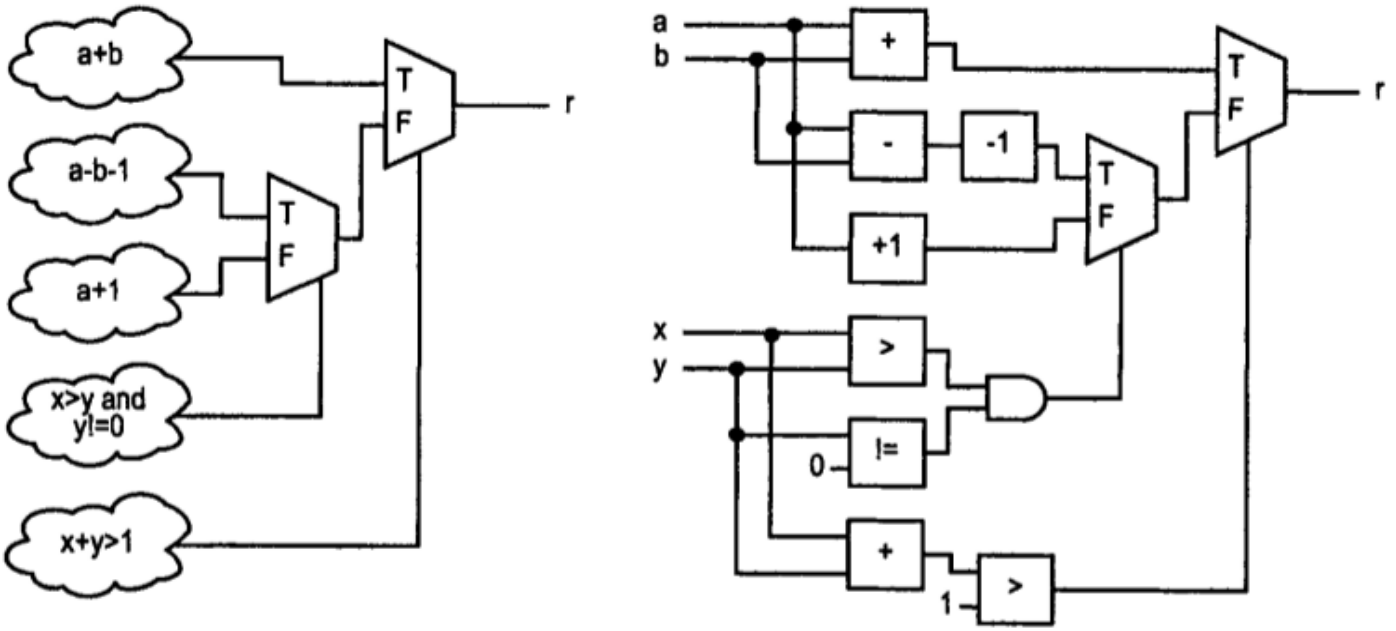
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity example_3 is
 port(
   reset_n : in std_logic;
        clk     : in std_logic;
        a,b     : in std_logic;
        cmd     : in std_logic;
        f       : out std_logic
 );
end entity;

architecture logic of example_3 is
 signal tmp : std_logic;
 signal reg : std_logic;
begin

  --multiplexer
  tmp <= a when cmd='1' else b;

  process(reset_n,clk)
  begin
    if reset_n='0' then
           reg <= '0';
         elsif rising_edge(clk) then
           reg <= tmp;
         end if;
  end process;

  f <= reg;

end logic;
```



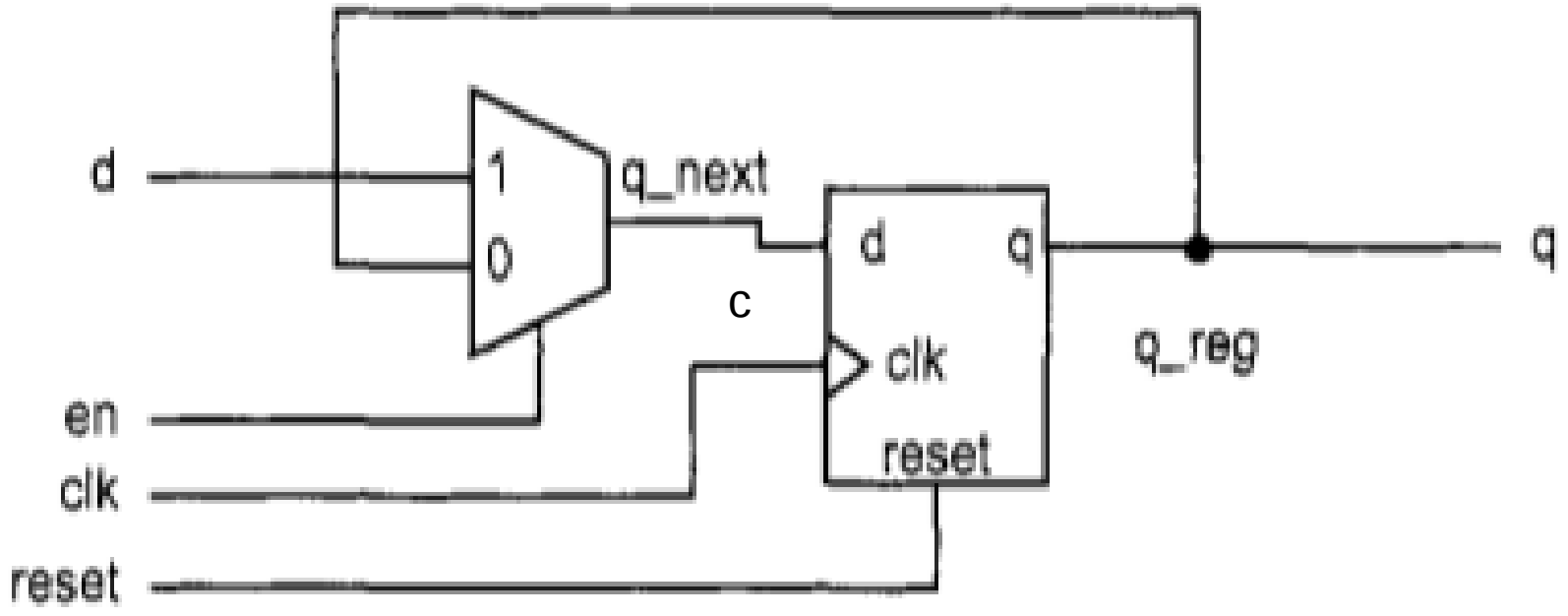
Operator	Description	Data type of operand a	Data type of operand b	Data type of result
a ** b	exponentiation	integer	integer	integer
abs a	absolute value	integer		integer
not a	negation	boolean, bit, bit_vector		boolean, bit, bit_vector
a * b	multiplication	integer	integer	integer
a / b	division			
a mod b	modulo			
a rem b	remainder			
+ a	identity	integer		integer
- a	negation			
a + b	addition	integer	integer	integer
a - b	subtraction			
a & b	concatenation	1-D array, element	1-D array, element	1-D array
a sll b	shift-left logical	bit_vector	integer	bit_vector
a srl b	shift-right logical			
a sla b	shift-left arithmetic			
a srl b	shift-right arithmetic			
a rol b	rotate left			
a ror b	rotate right			
a = b	equal to	any	same as a	boolean
a /= b	not equal to			
a < b	less than	scalar or 1-D array	same as a	boolean
a <= b	less than or equal to			
a > b	greater than			
a >= b	greater than or equal to			
a and b	and	boolean, bit, bit_vector	same as a	same as a
a or b	or			
a xor b	xor			
a nand b	nand			
a nor b	nor			
a xnor b	xnor			

Descriptions flot de données

```
. . .
signal a,b,r: unsigned(7 downto 0);
signal x,y: unsigned(3 downto 0);
. . .
r <= a+b when x+y>1 else
     a-b-1 when x>y and y!=0 else
     a+1;
```

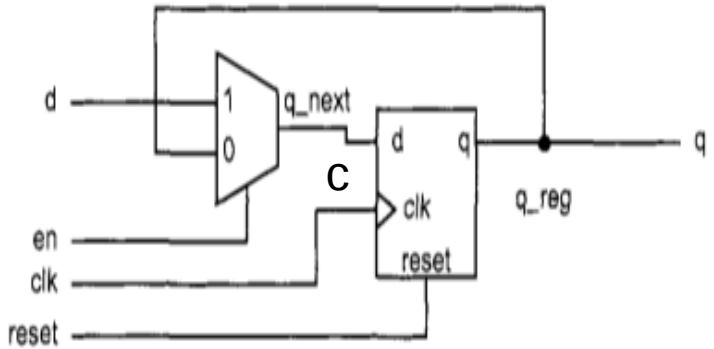


Registre avec enable



Registre avec enable

```
library ieee;
use ieee.std_logic_1164.all;
entity d_ff_en is
    port(
        clk, reset: in std_logic;
        en: in std_logic;
        d: in std_logic;
        q: out std_logic
    );
end d_ff_en;

architecture arch of d_ff_en is
begin
    process(clk,reset)
    begin
        if (reset='1') then
            q <='0';
        elsif (clk'event and clk='1') then
            if (en='1') then
                q <= d;
            end if;
        end if;
    end process;
end arch;
```



Une seule bascule D
(avec enable) !

Boucle combinatoire ?

```
process(a,b,tmp)
begin
    tmp <= tmp or b;
    y <= tmp;
end process;
```

Autorisée dans le langage, mais à proscrire en conception !