# VHDL – Logique programmable

## Partie 1 – Introduction

**Denis Giacona**

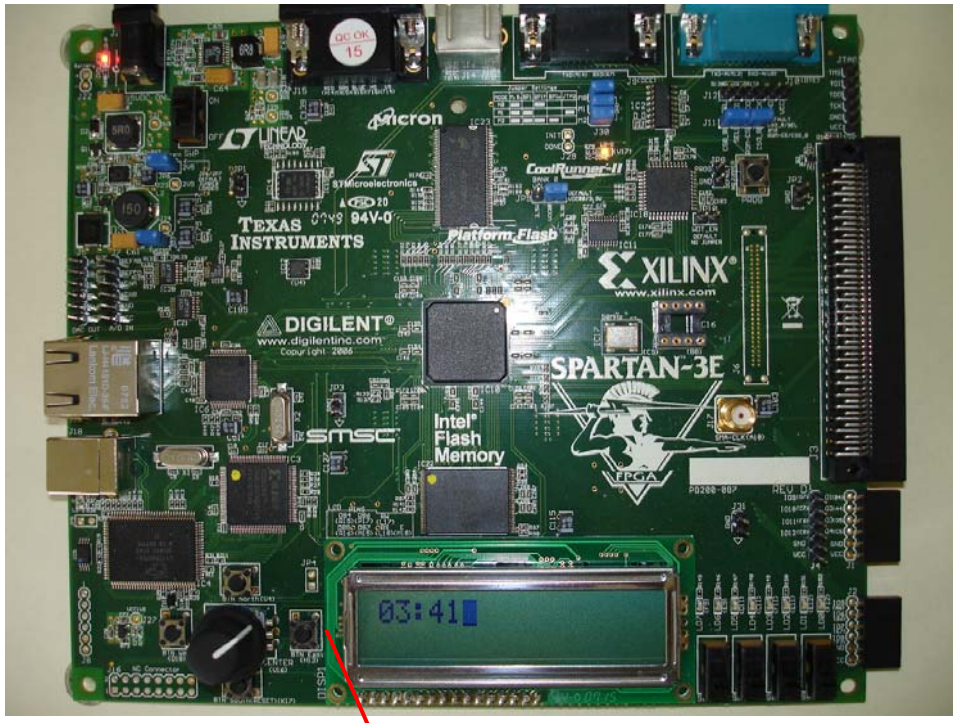
**ENSISA**
École Nationale Supérieure d'Ingénieur Sud Alsace
12, rue des frères Lumière
68 093 MULHOUSE CEDEX
FRANCE

Tél. 33 (0)3 89 33 69 00

# 1. Circuits logiques programmables CPLD et FPGA

## 1.1. Propriétés des circuits

- CPLD : Complex Programmable Logic Device
- FPGA : Field Programmable Gate Array
- (Re)programmables par l'utilisateur
- Capacité : plusieurs millions de portes logiques, plusieurs milliers de flip-flops
- Tensions d'entrées-sorties standard *low voltage* : 3,3V (LVTTL et LVCMOS); 2,5V - 1,8V - 1,5V - 1,2V (LVCMOS)
- Fréquence de fonctionnement → 500 MHz
- $t_{cko}$ flip-flop (*clock to output time*) → 0,6ns
- $t_{su}$ flip-flop (*set up time*) → 0,4ns
- Opérations possibles en un seul cycle d'horloge (quelques ns)




Carte de développement Spartan-3E

## 1.2. Domaines d'application

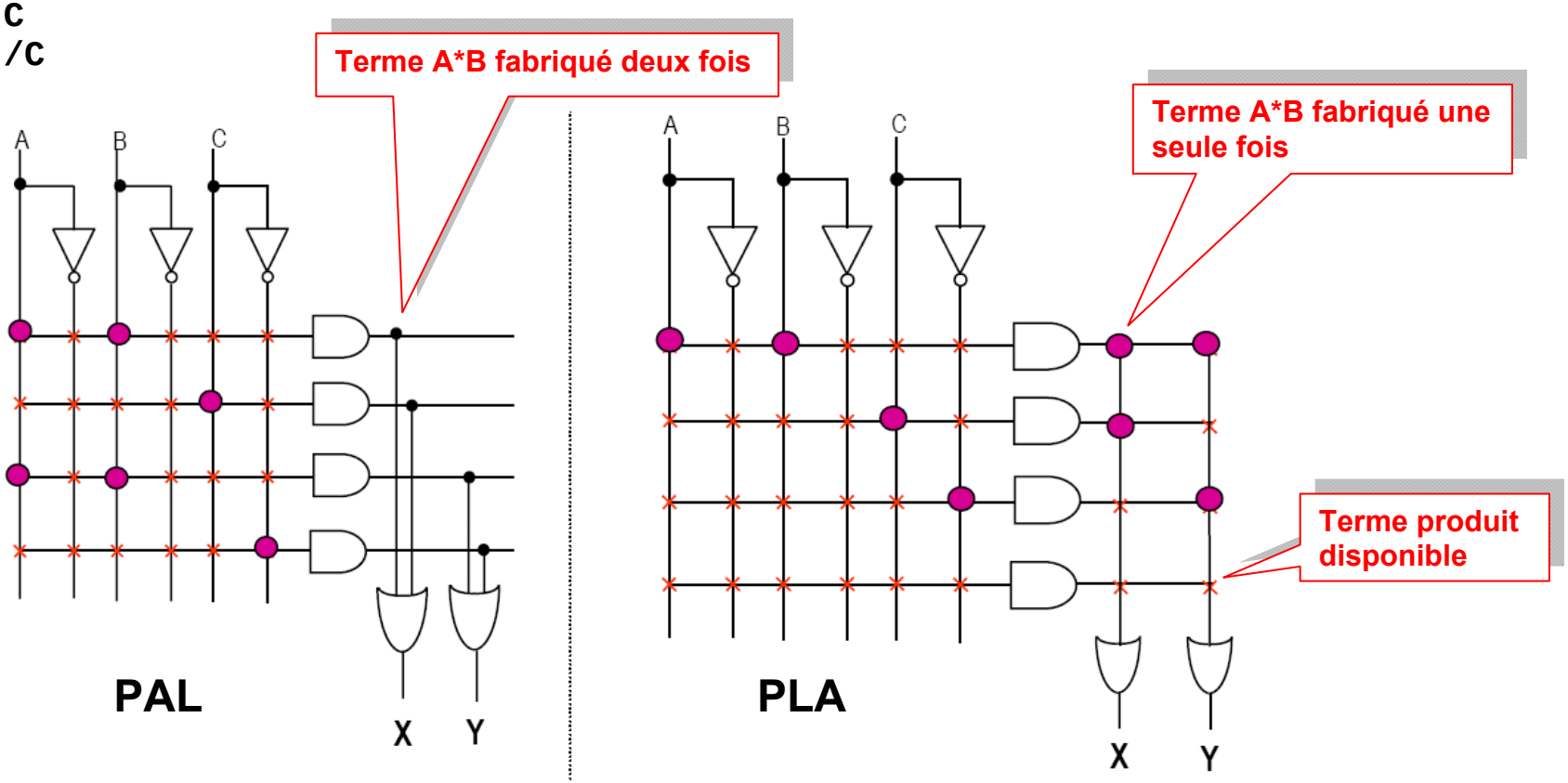
- Médical
- Electronique grand public
- Militaire/aérospatial
- Contrôle industriel
- Automobile
- Télécommunications
- Communications de données (réseaux)
- Stockage de données
- Téléphones mobiles
- Combinés multimédias
- Boîtiers décodeurs pour télévision
- Microprocesseurs
- Processeurs graphiques

# 1.3. Architectures des circuits logiques programmables (PLD)

## 1.3.1. PAL / PLA

**Deux réseaux, AND et OR, pour fabriquer des « somme de produits logiques » à l'aide de points de connexion programmables.**

```
X = A * B + C
Y = A * B + /C
```

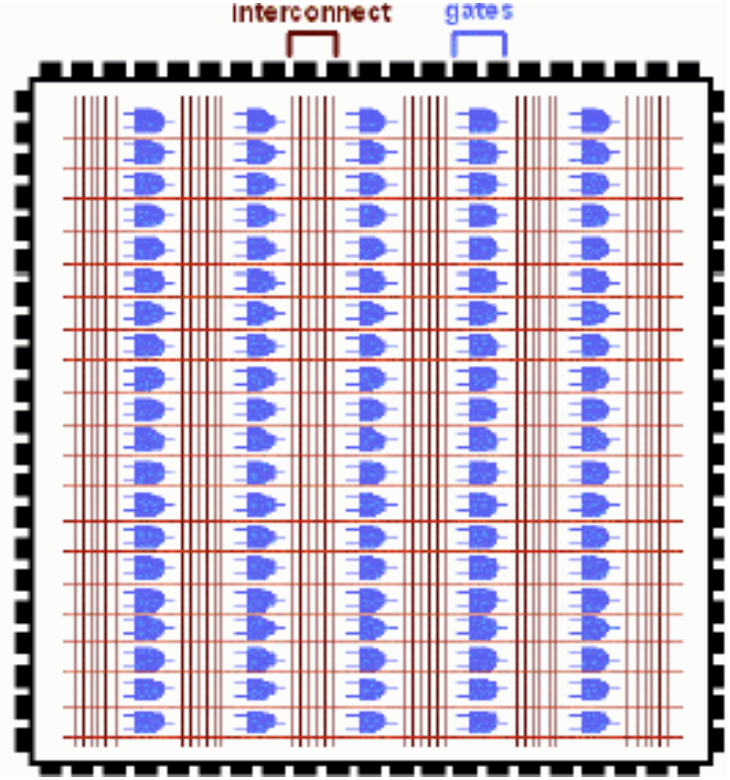



*Source : Xilinx*

## 1.3.2. Architecture FPGA

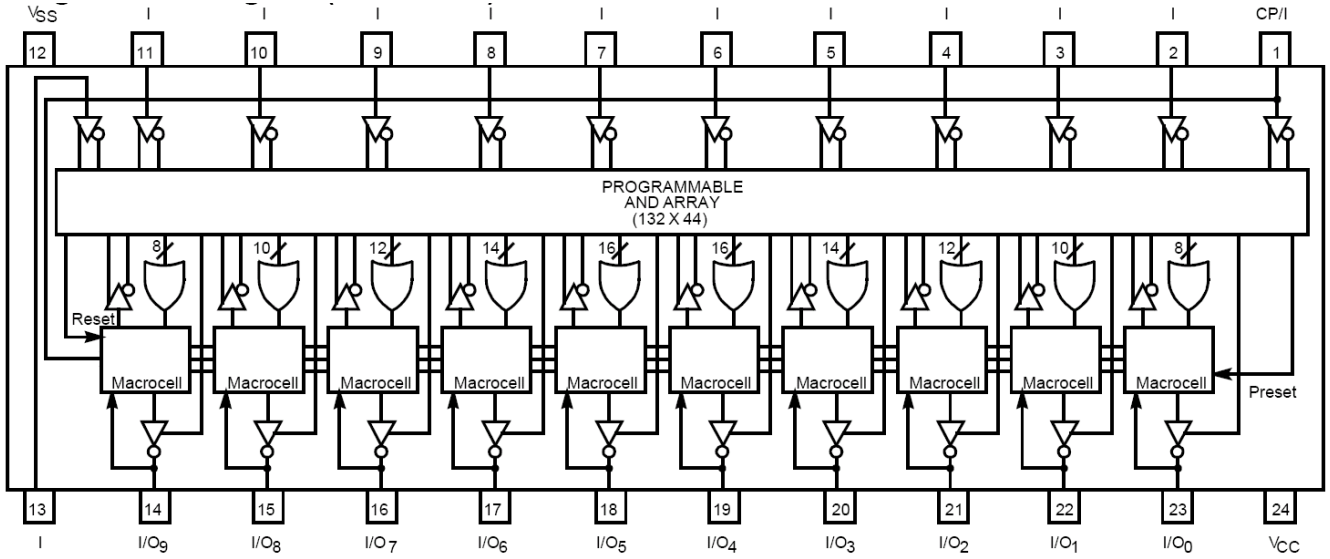
**Une « mer de portes » et un réseau d'interconnexions très souple.**

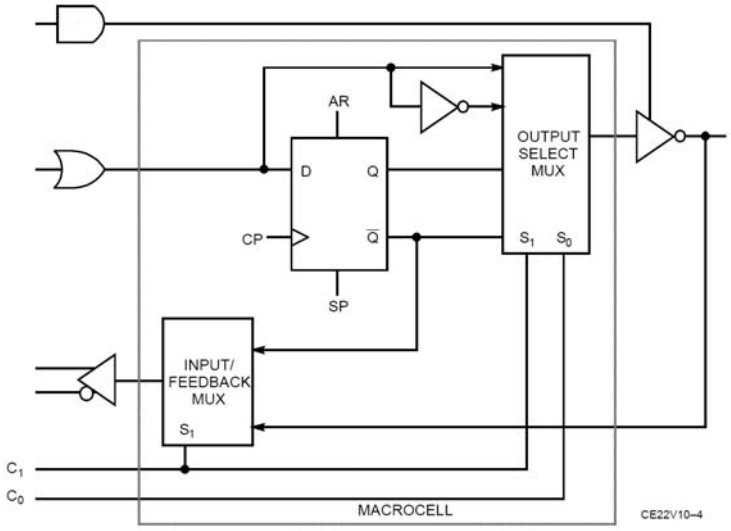
**FPGA (*Xilinx, 1985*)**



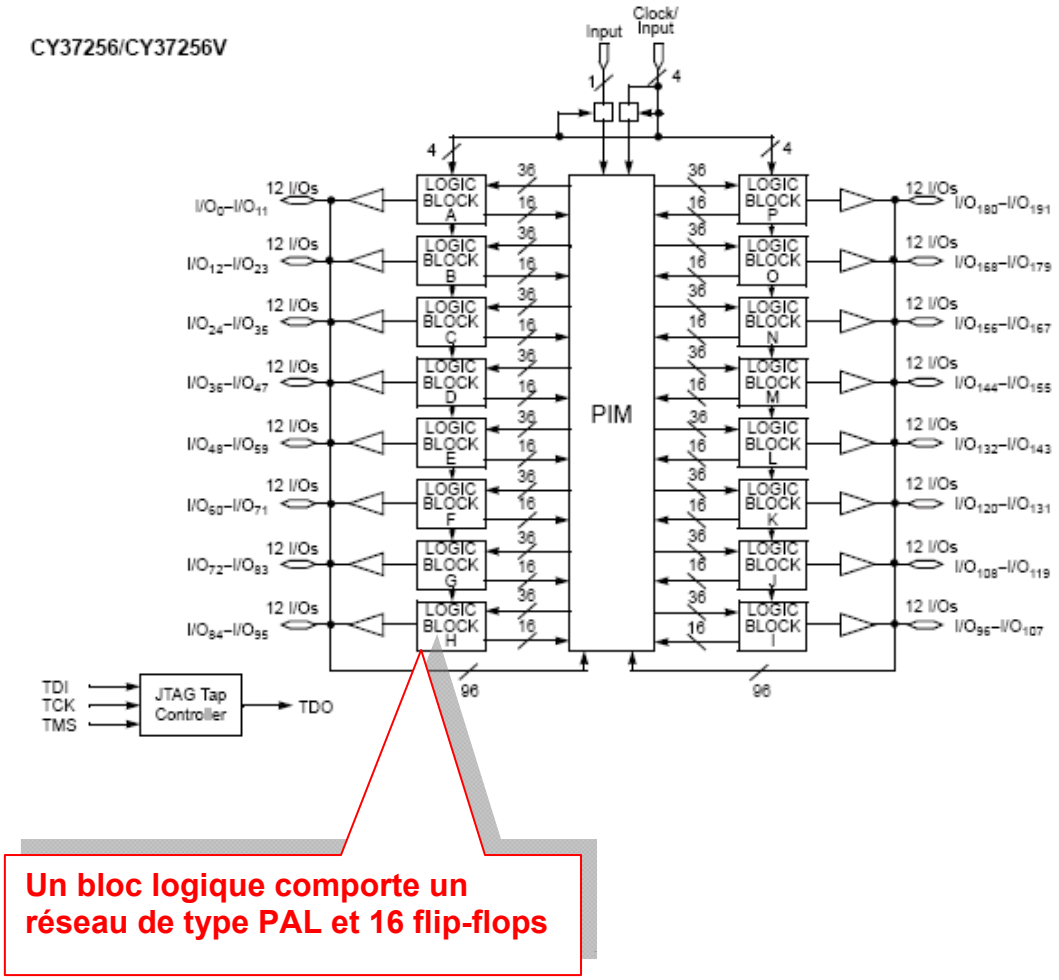
# 1.4. Exemples de circuits

## 1.4.1. Le circuit SPLD « référence » PALCE22V10 (Cypress)



# 1.4.2. Le circuit CPLD CY37256 (Cypress)

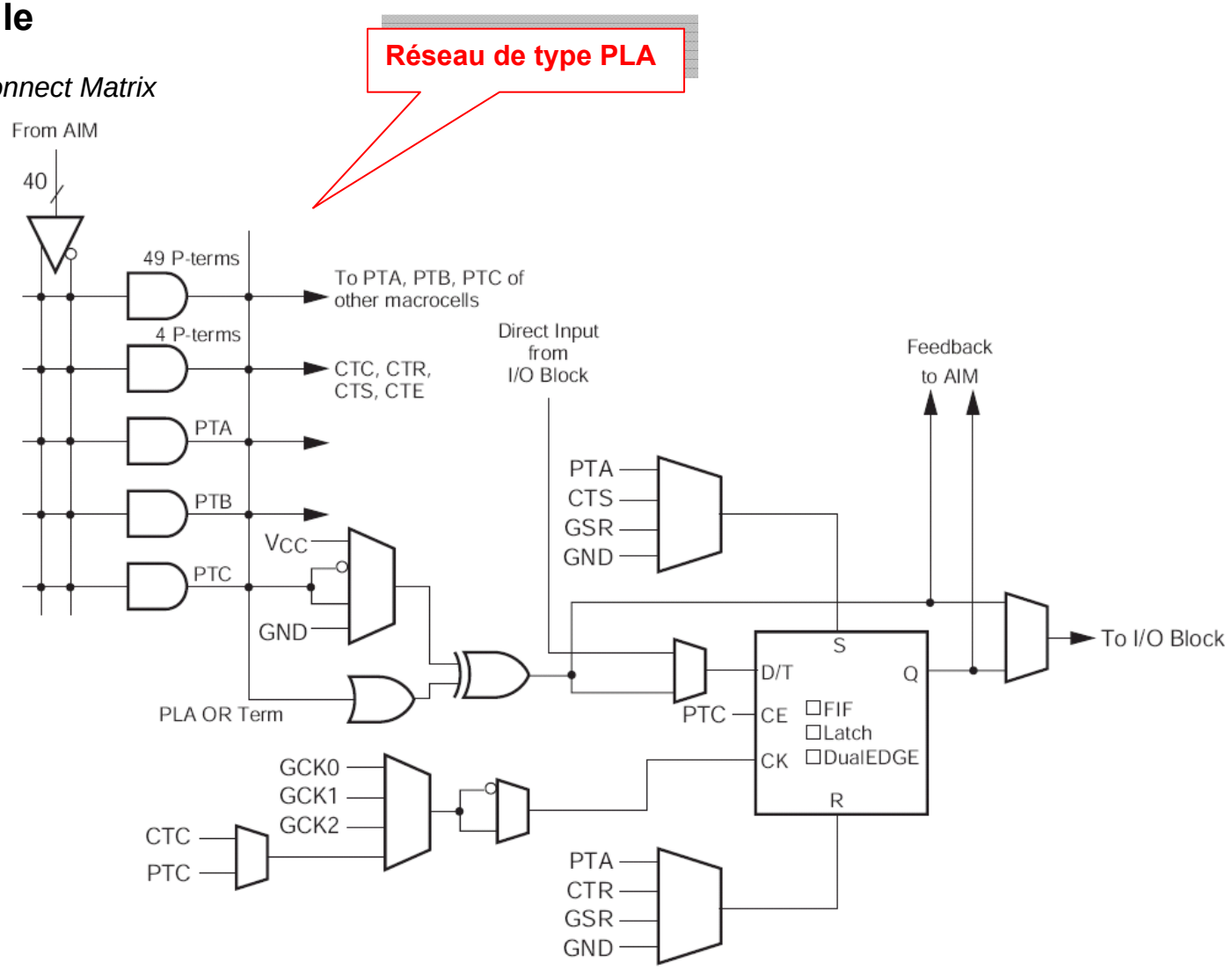



**Un bloc logique comporte un réseau de type PAL et 16 flip-flops**

# 1.4.3. Le circuit CPLD CoolRunner-II (Xilinx)

## Une macro-cellule

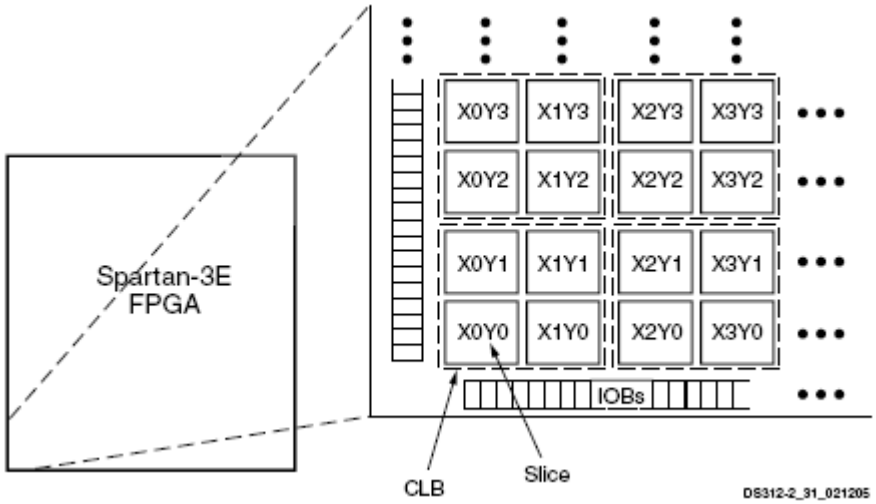
*AIM : Advanced Interconnect Matrix*



# Quelques caractéristiques données par le constructeur

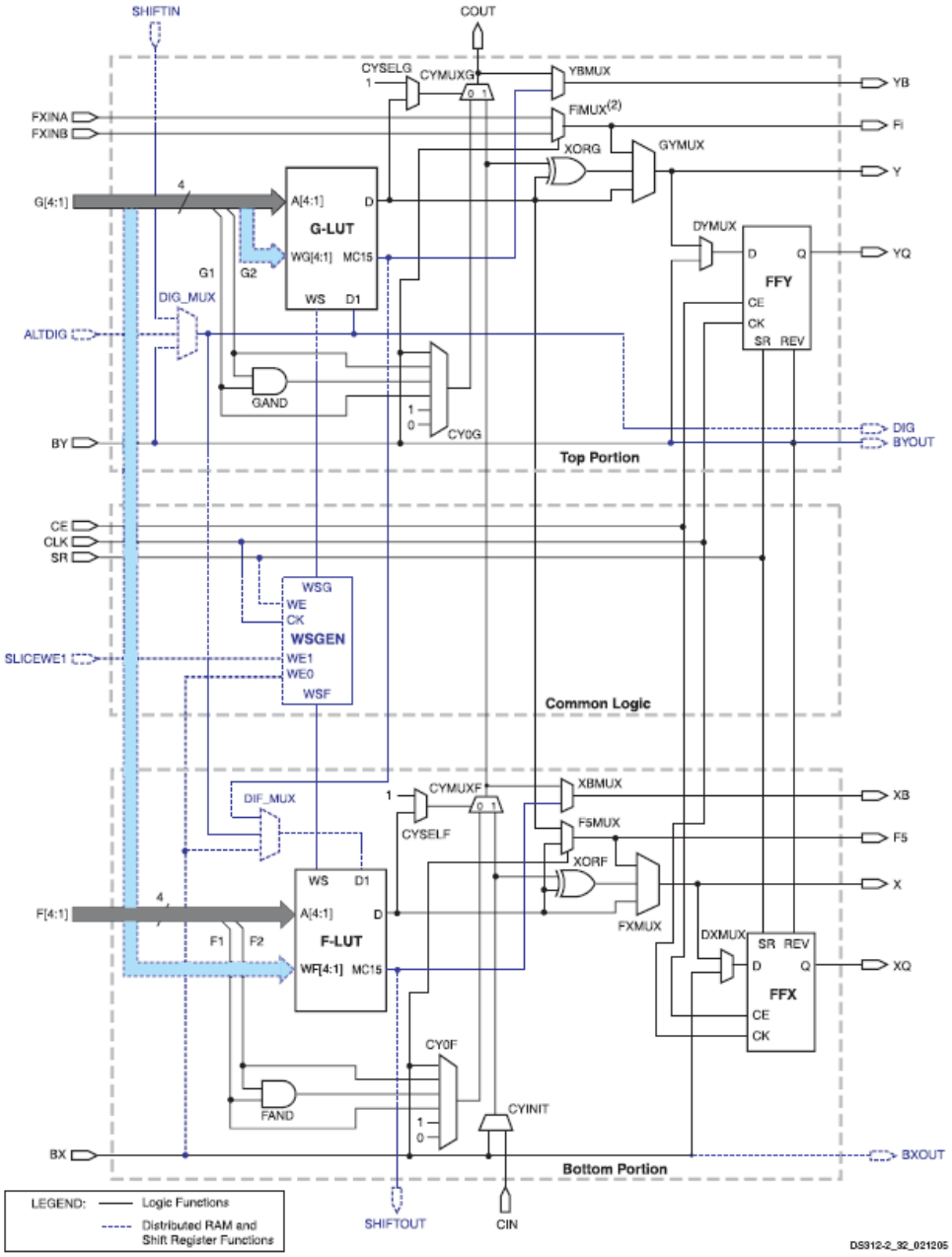
- 3.8 ns pin-to-pin logic delays
- 12 µA quiescent current
- In system programming 1.8V ISP using IEEE 1532 (JTAG) interface
- Optional Schmitt-trigger input (per pin)
- Multiple global clocks with phase selection per macrocell
- Global set/reset
- Open-drain output option for Wired-OR and LED drive
- Optional configurable grounds on unused I/Os
- Optional bus-hold, 3-state or weak pullup on selected I/O pins
- Mixed I/O voltages compatible with 1.5V, 1.8V, 2.5V, and 3.3V logic levels
- PLA architecture

## 1.4.4. Circuit FPGA Spartan-3E (Xilinx)




- ***CLB : Configurable Logic Block***
- ***IOB : Input Output Block***

# Un slice du circuit Spartan-3E



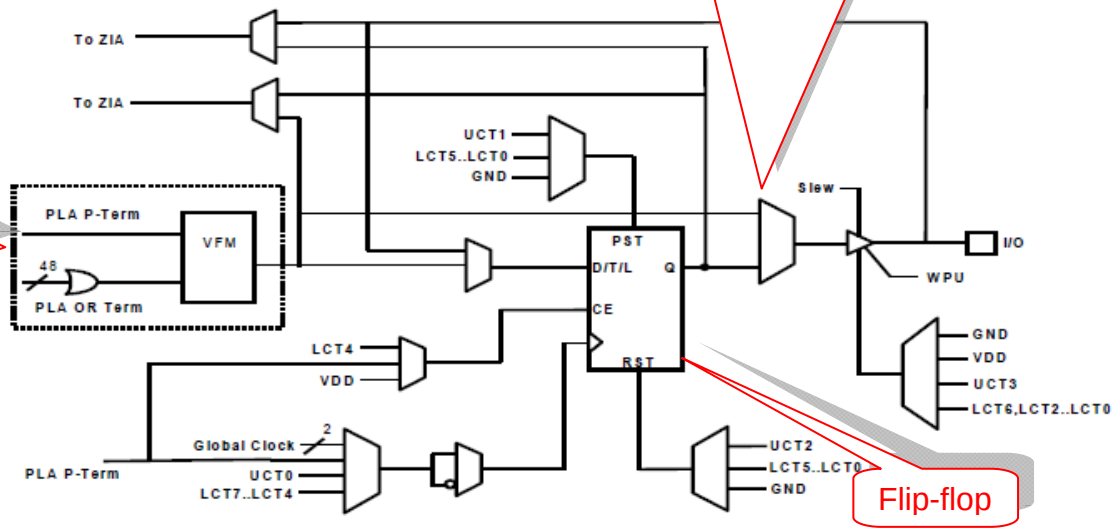
# 1.5. CPLD / FPGA

## 1.5.1. Architecture CPLD

**Réseau de type PLA**

- Fusible, permanent (*OTP : One Time Programming*)
- Anti-fusible (*OTP*)
- Transistor à grille flottante, reprogrammable in situ

Sélection programmable : choix entre la sortie combinatoire et la sortie séquentielle

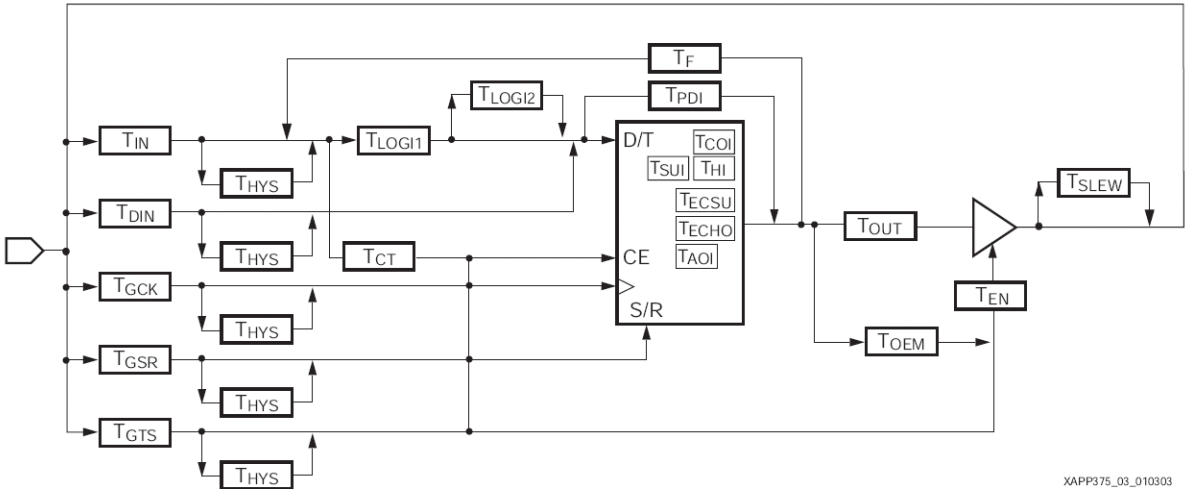



*Xilinx : XPLA3 Macrocell*

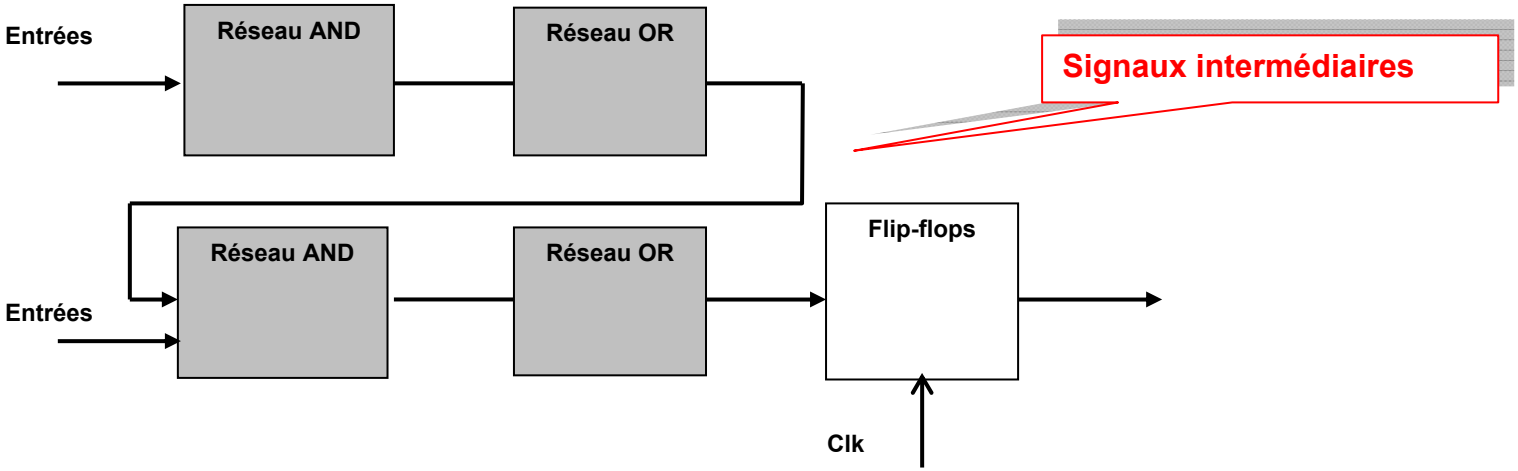
- **Utilisation des ressources non optimale : tout terme non utilisé dans une macrocellule entraîne une perte de portes logiques**

- **Temps de propagation prédictible : on connaît à l'avance la longueur des chemins de connexion**

# Modèle temporel pour le circuit CPLD CoolRunner-II (Xilinx)




# Extension d'une fonction combinatoire



## 1.5.2. Architecture FPGA

### 1.5.2.1. FPGA programmable une seule fois (OTP)

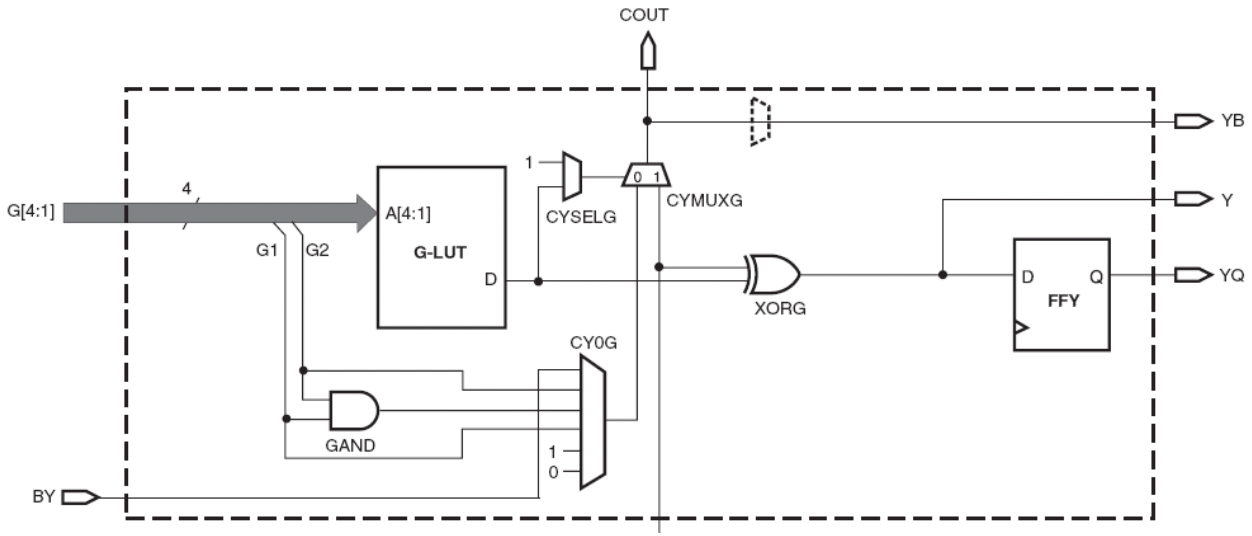
- **Logique combinatoire : portes logiques traditionnelles AND et OR**
- **Logique séquentielle : flip-flops**
- **Interconnexions : type antifusible**

### 1.5.2.2. FPGA reprogrammable (SRAM)

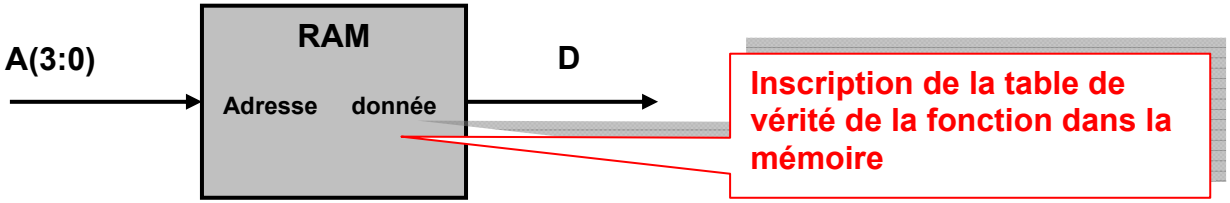
- **Logique combinatoire dans des *look up tables* (LUT) : type SRAM**
- **Logique séquentielle : flip-flops**
- **Interconnexions entre les cellules logiques dans mémoire de configuration : type SRAM**

**➔ Il faut une mémoire externe (par ex. PROM) pour la sauvegarde de la configuration (hors tension)**

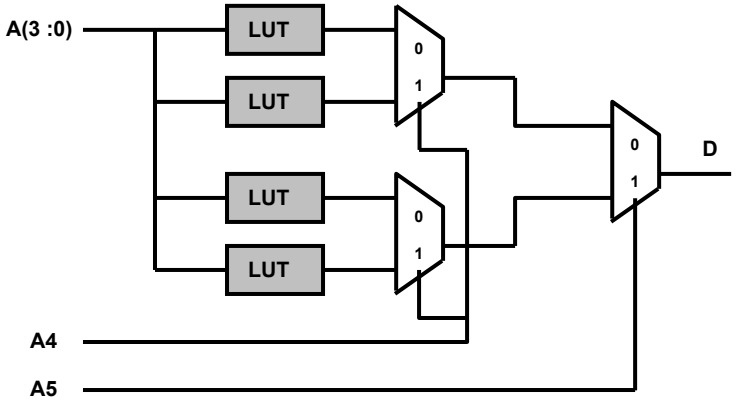
# Cellule FPGA SRAM (Xilinx)



- **LUT de type SRAM 16x1**
  - **Avantage : réalisation de n'importe quelle fonction à 4 variables**
  - **Inconvénient : peu efficace pour les fonctions simples (une fonction à 2 entrées utilise la LUT entière)**



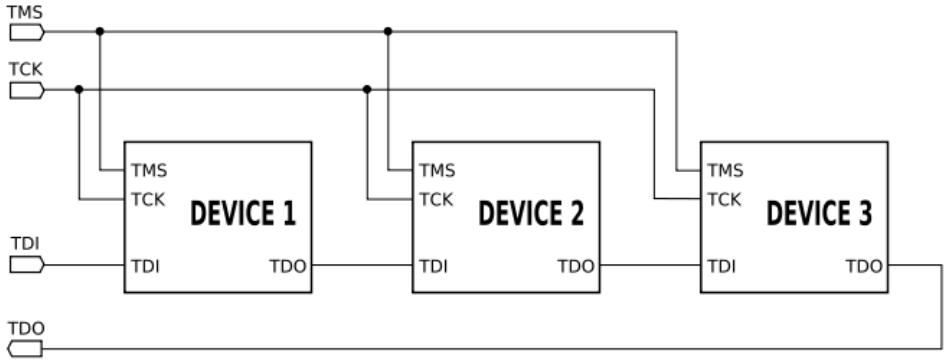

- **Extension d'une fonction combinatoire : D = f(A5,A4,A3,A2,A1,A0)**




- **Blocs IP (propriété intellectuelle)**
  - **Les constructeurs intègrent certaines cellules (pré-configurées) dédiées aux opérations arithmétiques (multiplicateurs, *MAC : Multiply-Add-Cumulate*, …)**

# 1.6. Programmation des circuits CPLD

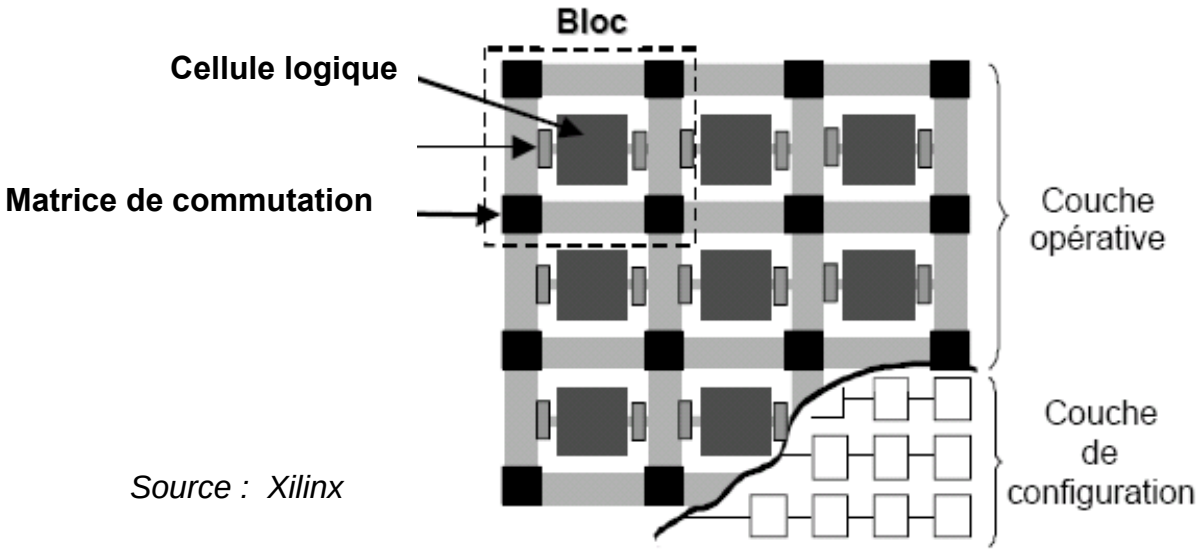
- **Programmation in situ (*ISP : In System Programming*) grâce à la technique BOUNDARY SCAN (norme conçue par le groupe de travail JTAG) ; particularité : les circuits peuvent être programmés à la chaîne**




*Source : Xilinx*

# 1.7. Programmation des circuits FPGA (technologie SRAM)

## 1.7.1. Les deux couches du circuit




*Source : Xilinx*

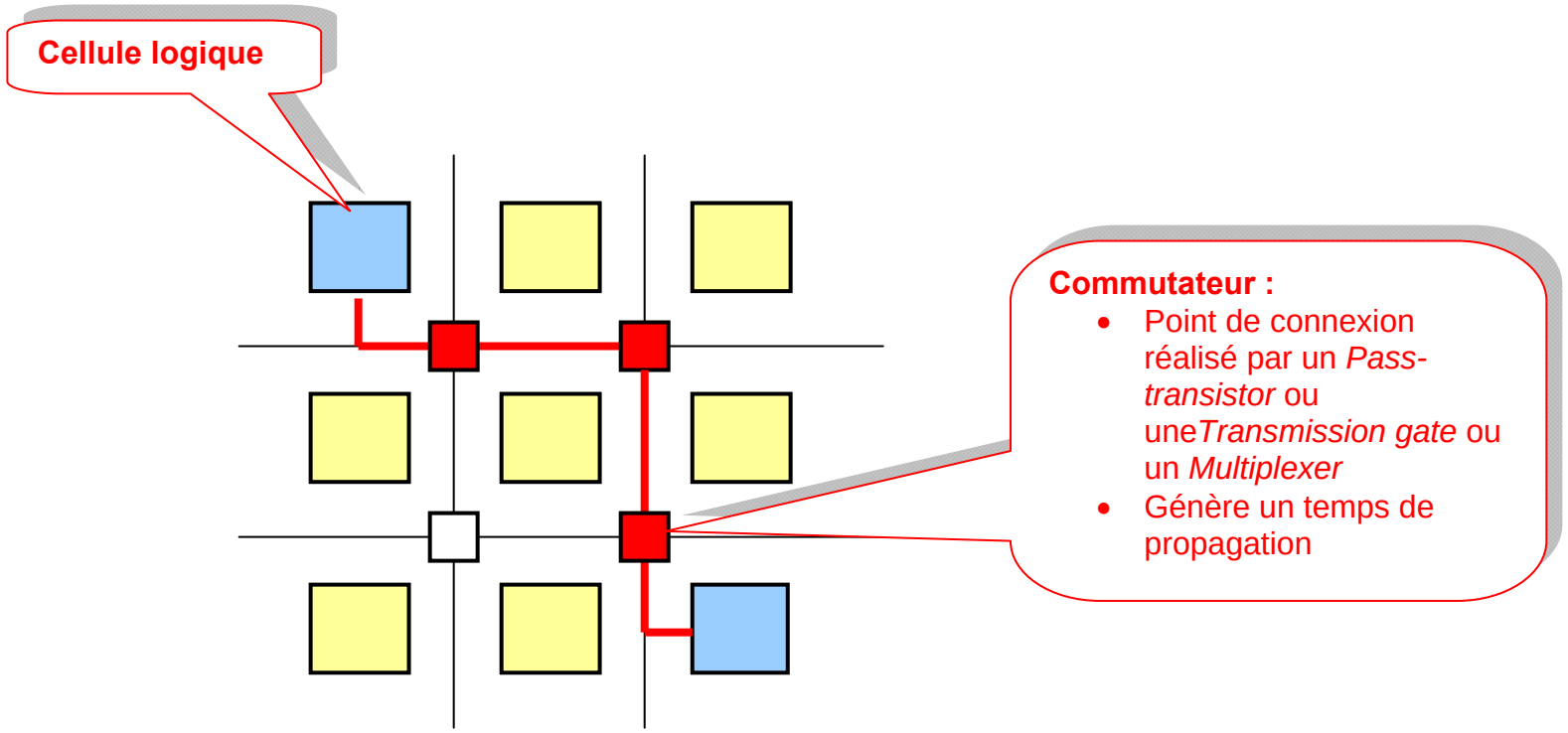
- **La couche opérative comporte une structure régulière de cellules logiques identiques entourées d'un réseau d'interconnexions**

- **La couche de configuration est un ensemble de cellules mémoires statiques (SRAM) qui, d'une part, configurent les cellules logiques de la couche opérative et, d'autre part, établissent leurs interconnexions (chemins de routage)**

## 1.7.2. Le routage

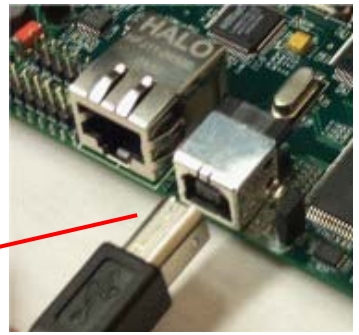
- **Les algorithmes de placement et de routage font un compromis entre l'utilisation des ressources disponibles et la vitesse**

- **Les délais de propagation des signaux dépendent du routage (non prédictible)**



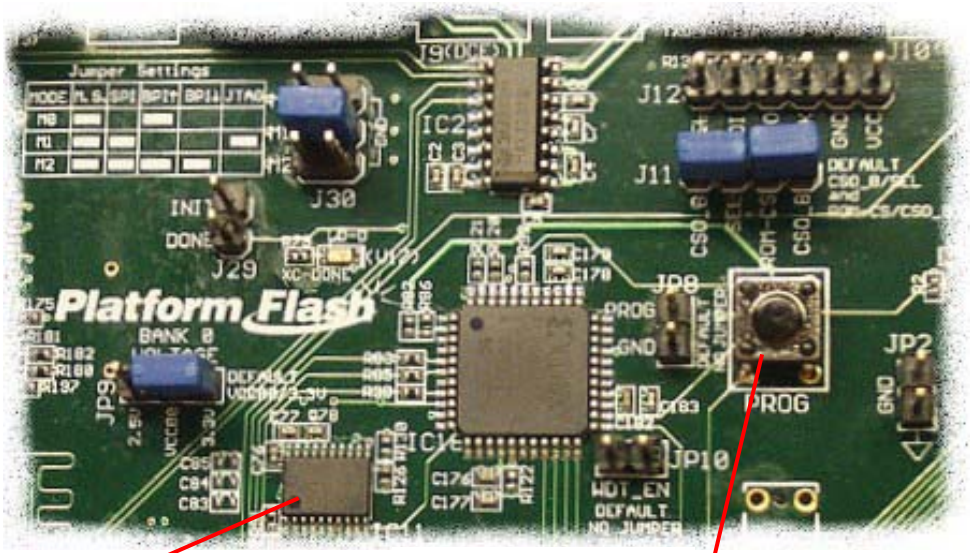
## 1.7.3. Le processus de configuration (Ex. : carte SPARTAN-3E)

❶ **Tout d'abord, la configuration (*bitstream*) est enregistrée dans un circuit mémoire *flash PROM* externe, pour permettre la conservation des informations lorsque le système est hors tension**



**Téléchargement, via un câble USB, à l'aide du logiciel iMPACT**

❷ **Après la mise sous tension, ou en appuyant sur le bouton *configuration restart*, le *bitstream* est copié de manière sérielle dans la SRAM du circuit FPGA**

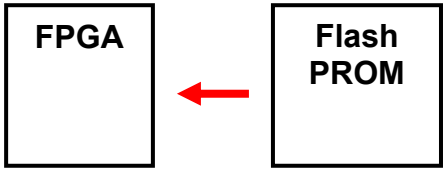


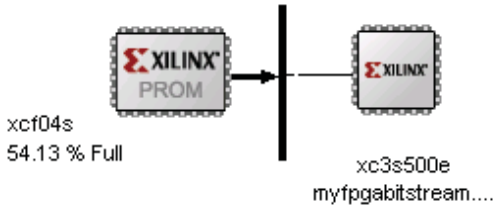

**Flash PROM**

**Bouton « *configuration restart* »**

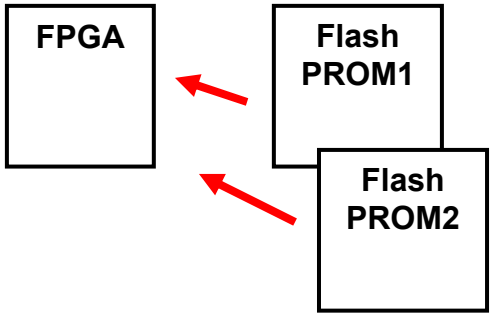
## Avantages du système : grande flexibilité

- **Configuration statique**
  - **Si une seule architecture suffit par application : chargement à la mise sous tension**




- **Configuration semi-dynamique**
  - **Plusieurs configurations possibles, pour des applications différentes**




- **Configuration dynamique (auto-reconfiguration)**
  - **Si l'architecture doit évoluer en cours de traitement : chargement continu dans certaines parties de la SRAM, sans arrêter le fonctionnement des sections inchangées)**

# 2. Les niveaux d'entrée d'une description

SystemC

## *UTF (UnTimed Functional)*

**Interface et fonctionnalité ; modèle abstrait, algorithmique, orienté logiciel et matériel**

- **Pas de notion de durée d'exécution**
- **Seul compte l'ordonnancement des tâches**

SystemC,
HDL,
FBD,
FSM

## *RTL (Register Transfer Level)*

**Interface et fonctionnalité ; modèle orienté matériel**

- **Chaque bit, chaque cycle, chaque registre du système est modélisé**

**Connexion de portes logiques, latches et flip-flops**

## 2.1. Conception au niveau UTF (UnTimed Functional)

- **Les transitions internes sont asynchrones (on ne mentionne ni horloge système ni délais)**
- **Les processus communiquent entre eux et se synchronisent avec les événements d'entrées-sorties**

### 2.1.1. Modèle de description : SystemC

- **Ensemble de classes du langage C++**
- **Permet de modéliser des systèmes matériels et logiciels**
- **Standardisé en décembre 2005**

# 2.2. Conception au niveau RTL (Register Transfer Level)

## 2.2.1. Principes

### ❒ Le concepteur définit la structure du système avec

- **des fonctions combinatoires (AND, OR, ADD, MUX, ALU, DEC, …)**
- **des registres standard (D, COUNT, SHIFT, …)**
- **des machines d'état spécifiques**

### ❒ Le concepteur respecte les règles suivantes

- **la synchronisation interne est explicite ; les cycles d'horloge ont une durée suffisante pour permettre la stabilisation des signaux avant le prochain front actif**
- **si des registres doivent changer d'état, ils sont tous mis à jour en même temps, sur le même front d'horloge système**
- **la description est indépendante des valeurs des données**

### ❒ Le concepteur choisit

- **le modèle de description le mieux adapté ou simplement celui qu'il préfère**
  - **textuel (*HDL : Hardware Description Language*)**
  - **diagramme de blocs fonctionnels (*schematic, FBD : Function Block Diagram*)**
  - **graphe d'état (*FSM : Finite State Machine*)**
- **le niveau d'abstraction**
- **la méthode de traitement**

## 2.2.2. Outils de développement

**ISE (Xilinx)**
- version WebPack, licence gratuite à durée illimitée
- synthèse des circuits du fondeur Xilinx (CoolRunner CPLD, Virtex FPGA, Spartan FPGA)

**HDL Author + ModelSim PE (Mentor Graphics) ou ModelSim Designer (Mentor Graphics)**
- passerelle vers les fondeurs Xilinx, Altera, Actel et Lattice
- ModelSim existe en version *student* gratuite

### 2.2.3. Objectifs

- **Conception**
  - **Si possible, construction d'entités à partir d'autres entités déjà crées et testées (composants logiques réutilisables archivés dans des bibliothèques)**
  - **Cœurs (noyaux) de propriété intellectuelle (*IP cores*)**
    - **éléments de circuits (blocs fonctionnels) prêts à l'emploi**
    - **fonctions standard (le concepteur ne part plus de zéro ; il développe les parties spécifiques) ➔ conception plus rapide**
    - **code écrit en HDL niveau RTL**
    - **disponibles dans les outils de développement, ou achetés séparément**
    - **exemples : MPEG4 Video Codec, FFT pour DSP, interfaces de bus PCI, interface de réseau Ethernet, …**
  - **« Cœurs libres » (*Open cores*)**
    - **gratuits (agence spatiale européenne ESA)**
    - **exemples : cœurs de processeurs RISC 32 bits, ALU, Multiplication**

- **Tests**
  - **Simulation fonctionnelle (indépendante du circuit cible)**
  - **Simulation temporelle (post routage)**

- **Implémentation (phase finale de la réalisation)**
  - **Configuration d'un composant matériel qui réalise les fonctions désirées**
    - **placement et routage des ressources (LUT, MUX pour les FPGA)**

# 3. Les modèles de description RTL

## 3.1. Langages de description de matériel (HDL)

### 3.1.1. VHDL et Verilog (deux langages normalisés quasi-équivalents)

- **VHDL : Very High Speed Integrated Circuit Hardware Description Language**
  - **Syntaxe dérivée du langage ADA**
  - **Mots clés adaptés à la conception matérielle (`architecture, port, signal, in, out, …`)**
  - **Standard en 1987**
- **Verilog**
  - **Conçu en 1985 par Gateway Design System**
  - **Syntaxe dérivée du langage C**

- **VHDL vs Verilog**
  - **VHDL plus strict que Verilog pour les types**
  - **Texte plus abondant dans VHDL**

### 3.1.2. VHDL-AMS

- **Extension aux circuits analogiques (VHDL-Analog & Mixed Systems)**

## 3.1.3. Description d'une porte logique AND en SystemC

```
#include "systemc.h"

SC_MODULE(and3)
{
  sc_in<bool> i0;
  sc_in<bool> i1;
  sc_in<bool> i2;
  sc_out<bool> x;

  void compute_and()
  {
    x = i0 & i1 & i2;
  };

  SC_CTOR(and3)
  {
    SC_METHOD(compute_and);
    sensitive(i0);
    sensitive(i1);
    sensitive(i2);
  }
};
```

Déclaration des ports

Déclaration d'un processus

Déclaration de la liste de sensibilité du processus

## 3.1.4. Description d'une porte logique AND en Verilog

```
module and3 (i0, i1, i2);

  input i0;
  input i1;
  input i2;

  output x;

  assign x = i0 & i1 & i2;

endmmodule
```

### 3.1.5. Description d'une porte logique AND en VHDL

```
entity and3 is port(

 i0 : in std_logic;
 i1 : in std_logic;
 i2 : in std_logic;

 x : out std_logic);

end entity ;


architecture arch_and3 of and3 is

begin

 x <= i0 and i1 and i2;

end arch_and3
```

## 3.1.6. Description du comportement d'un compteur en VHDL

```
process (CLK, ar)
begin
if ar = '1' then
    qint <= (others => '0');
elsif CLK'event and CLK='1' then
        if qint < 9 then
            qint <= qint + 1;
        else
            qint <= (others => '0');
        end if;
end if;
end process;

q <= qint;
max <= '1' when qint = 9 else '0';

end cnt4bits;
```

## 3.1.7. Propriétés du style comportemental VHDL

- **Les systèmes séquentiels synchrones sont toujours décrits en mentionnant l'horloge**

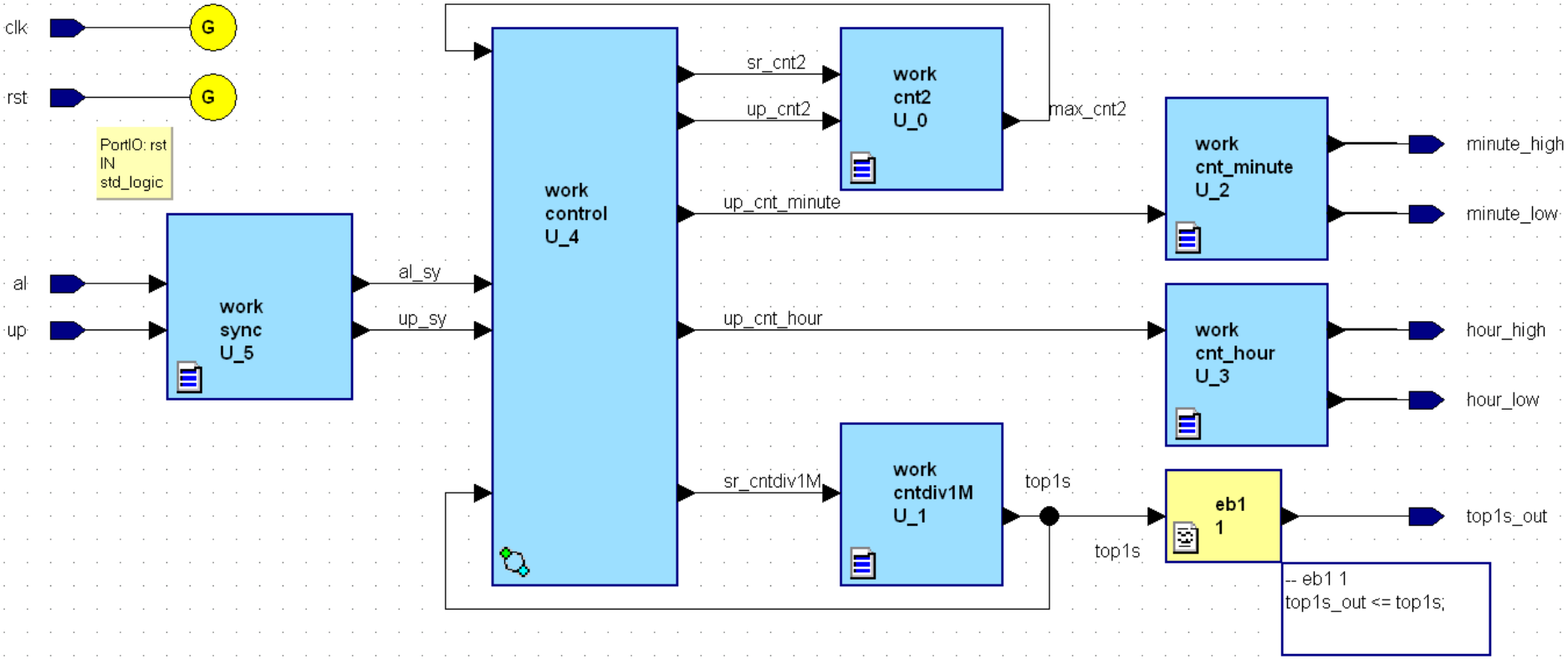
```
process (clk)
begin
if clk'event and clk = '1' then
  ...
```

- **Les bornes des variables de boucle sont statiques, fixées à la compilation (donc indépendantes des valeurs des signaux d'entrée)**

```
for i in 1 to 5 loop
  if product_reg(0)='1' then
    product_reg(9 downto 5) := product_reg(9 downto 5)
        + num1_reg(4 downto 0);
  end if;
  product_reg(9 downto 0) := '0' & product_reg(9 downto 1);
end loop;
```

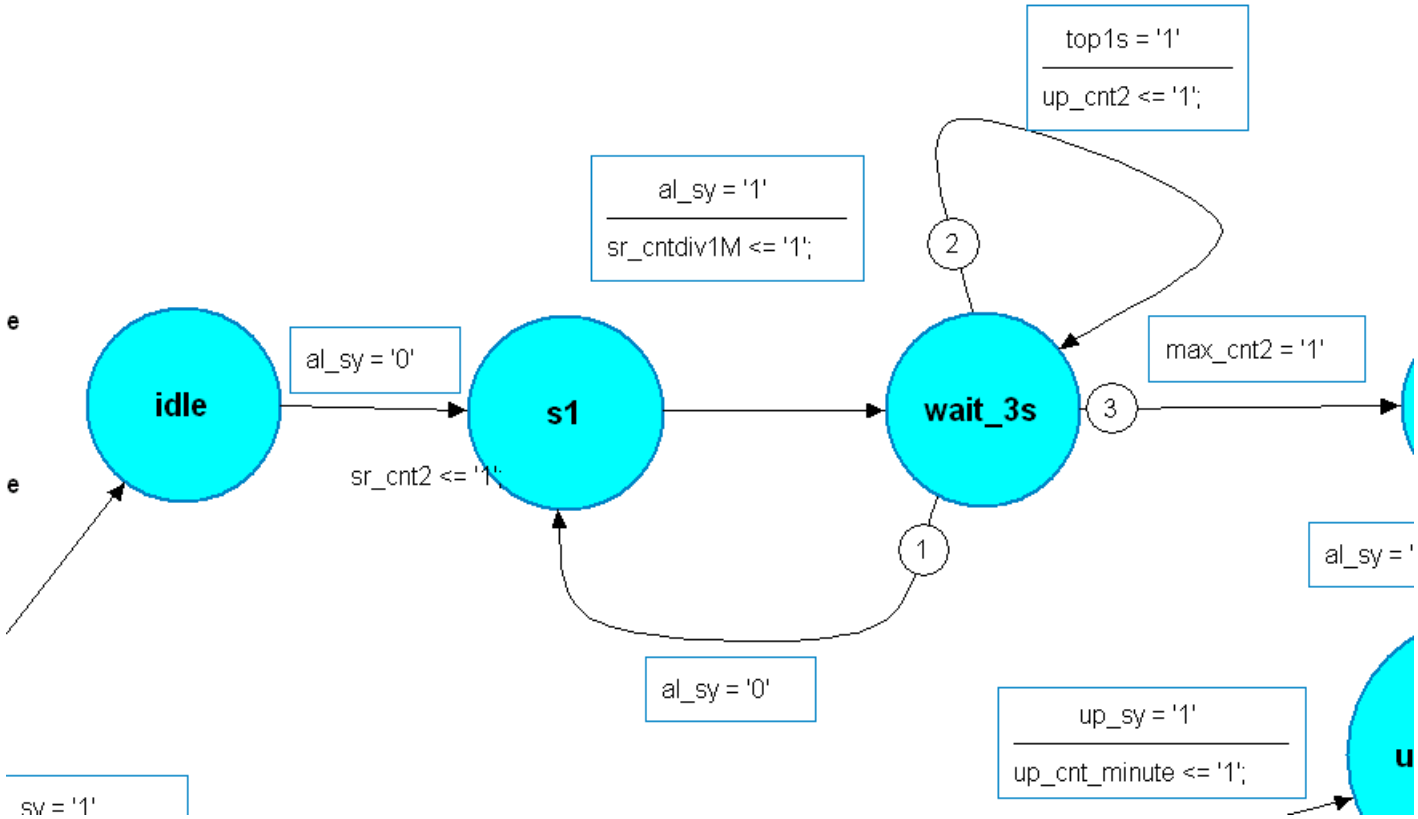
## 3.2. Description schématique (FBD : Function Block Diagram)

### Avec l'outil *ModelSim Designer (Mentor Graphics)*



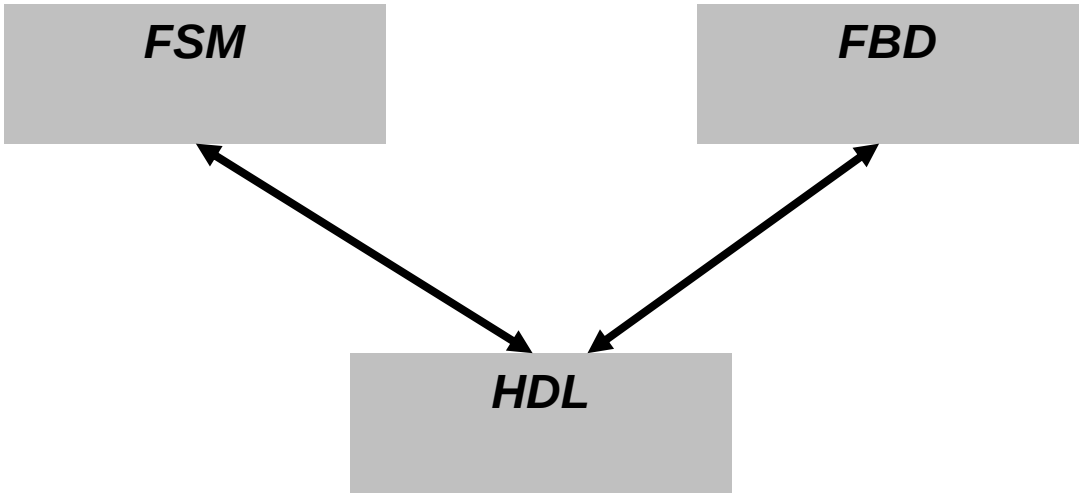
## 3.3. Description par graphe d'état (FSM : Finite State Machine)

### Avec l'outil *ModelSim Designer (Mentor Graphics)*



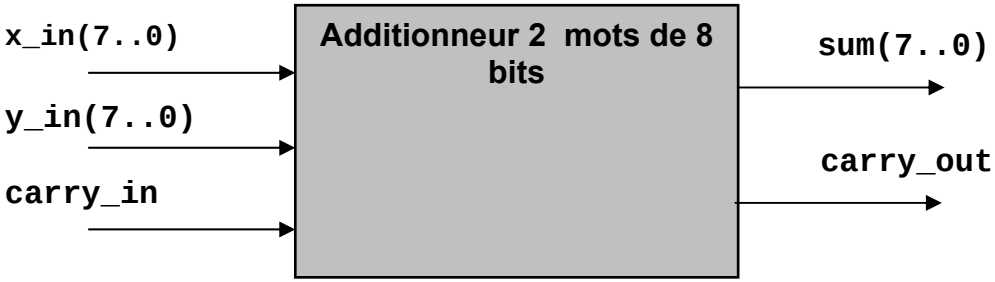
## 3.4. Conversion des modèles

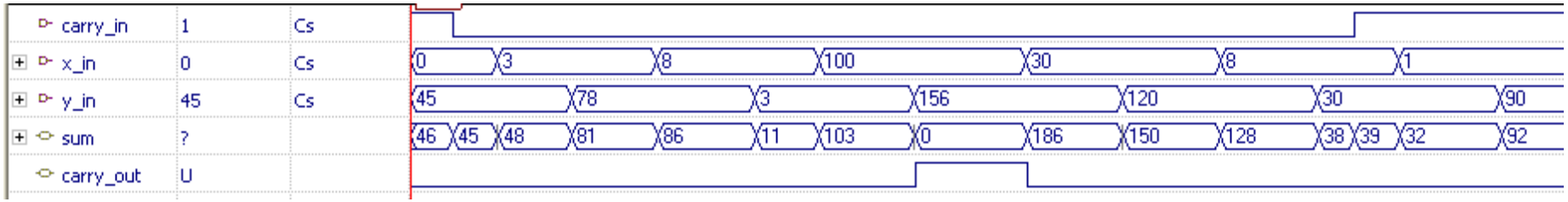
**Les conversions sont prises en charge par la plupart des outils de développement.**



# 4. Choix du modèle de description, du niveau d'abstraction et de la méthode de traitement

## 4.1. Exemple 1 : un additionneur




**Deux méthodes de traitement :**

- Propagation de la retenue (*ripple carry*)
- Calcul anticipé de la retenue (*carry look-ahead*)

## 4.1.1. Outil Warp (Cypress)

### 4.1.1.1. Description HDL flot de données

**Application de la méthode *ripple carry***

**Utilisation de l'opérateur arithmétique +**

```
(carry_out,sum)  <= ("00000000" & carry_in) + ('0' & x_in) + ('0' & y_in);
```

**Après optimisation automatique par le compilateur pour un circuit intégré particulier; par exemple le cy37256p160 (Cypress)**

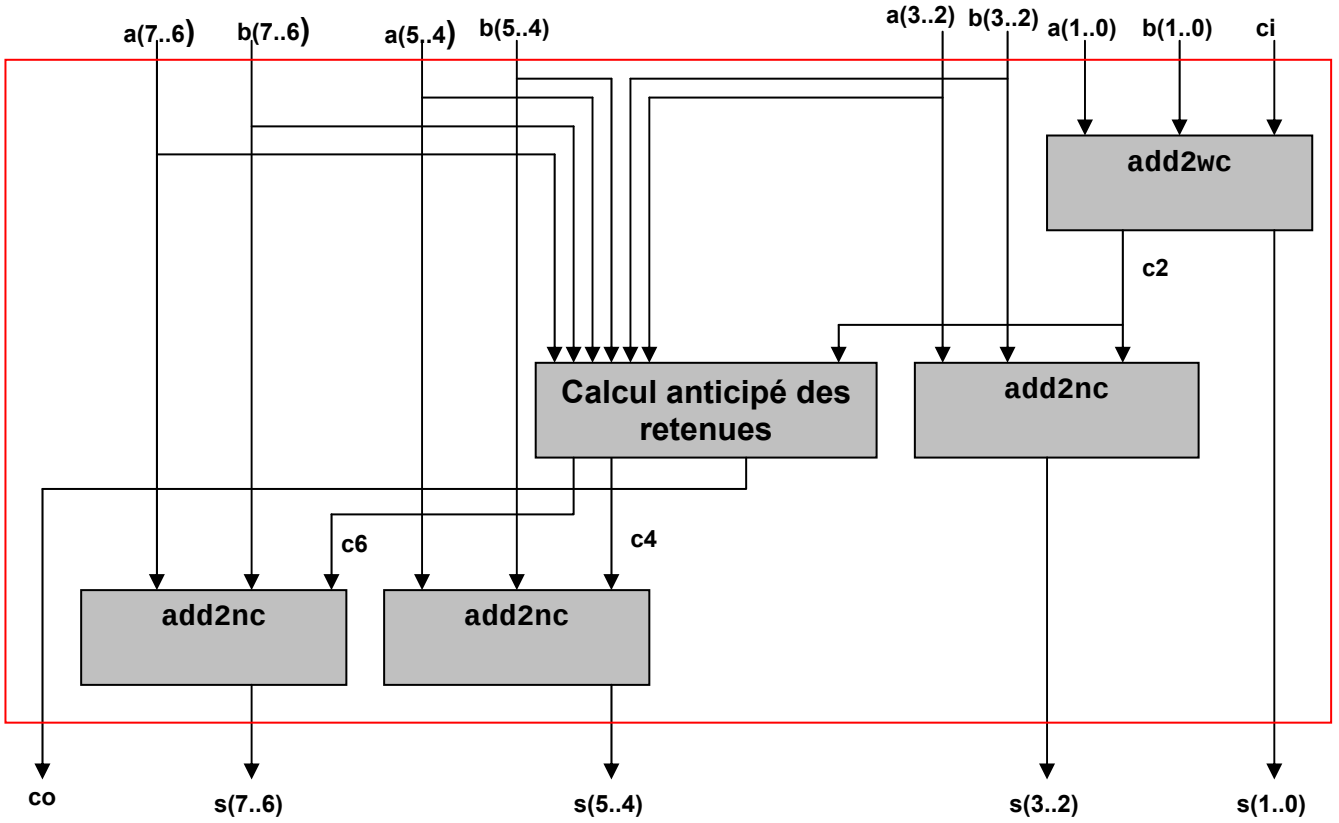
- **temps de propagation tPD = 59.0 ns pour co**
- **taux d'utilisation des ressources 8%**

**➔ Avantage : concision de la description**

## 4.1.1.2. Description HDL structurelle

**Application de la méthode *carry look-ahead***

**Principe (schéma-bloc) :**



```
i0 : c_add2wc port map(a(1), a(0),b(1), b(0), ci, sum(1), sum(0), c2);
i1 : c_add2nc port map(a(3), a(2),b(3), b(2), c2, sum(3), sum(2));
i2 : c_add2nc port map(a(5), a(4),b(5), b(4), c4, sum(5), sum(4));
i3 : c_add2nc port map(a(7), a(6),b(7), b(6), c6, sum(7), sum(6));

e1 <= (a(3) and b(3)) or ((a(3) or b(3)) and (a(2) and b(2)));
r1 <= (a(3) or b(3)) and (a(2) or b(2));

c4 <= e1 or (c2 and r1);


e2 <= (a(5) and b(5)) or ((a(5) or b(5)) and (a(4) and b(4)));
r2 <= (a(5) or b(5)) and (a(4) or b(4));

c6 <= e2 or ((e1 or (c2 and r1)) and r2);


e3 <= (a(7) and b(7)) or ((a(7) or b(7)) and (a(6) and b(6)));
r3 <= (a(7) or b(7)) and (a(6) or b(6));

co <= e3 or ((e2 or ((e1 or (c2 and r1)) and r2)) and r3);
```

**Pour le même circuit intégré cy37256p160 (Cypress) :**

- **temps de propagation tPD = 37.0 ns pour sum(7)**
- **taux d'utilisation des ressources 7%**

**➔ Avantage : performance**

### 4.1.1.3. Description HDL comportementale

**Application de la méthode *carry look-ahead* dans un algorithme**

**Principe mathématique (la retenue est calculée en même temps que la somme) :**

Somme et retenue de l'étage n d'un additionneur :

```
sum_n     =  x_in_n xor y_in_n xor carry_n
carry_n+1 = (x_in_n and y_in_n) or (x_in_n and carry_n) or (y_in_n and carry_n)
```

Terme de génération ($G_n$) et de propagation ($P_n$) d'une retenue :

```
G_n   = x_in_n and y_in_n
P_n   = x_in_n or y_in_n
```

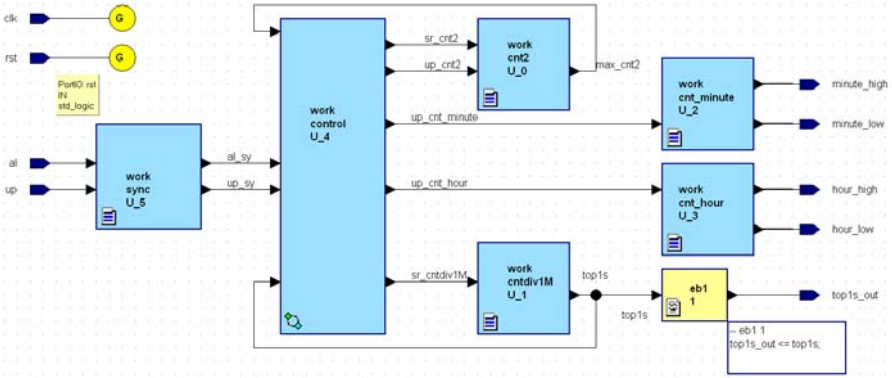
➔

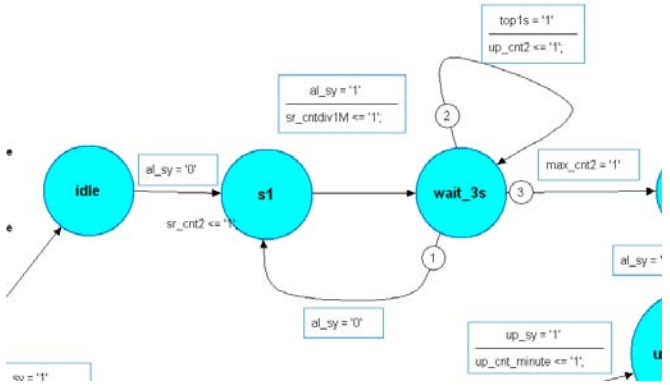
```
carry_n+1     = G_n   or (P_n and carry_n)
carry_n       = G_n-1 or (P_n-1 and carry_n-1)
```

```
h_sum <= x_in XOR y_in;
carry_generate <= x_in AND y_in;
carry_propagate <= x_in OR y_in;

PROCESS (carry_in,carry_generate,carry_propagate,carry_in_internal)
BEGIN
  carry_in_internal(1) <= carry_generate(0) OR (carry_propagate(0) AND carry_in);
  inst: FOR i IN 1 TO 6 LOOP
          carry_in_internal(i+1) <= carry_generate(i) OR (carry_propagate(i) AND
carry_in_internal(i));
              END LOOP;
          carry_out <= carry_generate(7) OR (carry_propagate(7) AND
carry_in_internal(7));
END PROCESS;

sum(0) <= h_sum(0) XOR carry_in;
sum(7 DOWNTO 1) <= h_sum(7 DOWNTO 1) XOR carry_in_internal(7 DOWNTO 1);
```

# 5. De la conception à la programmation du circuit

## 5.1. Processus standard



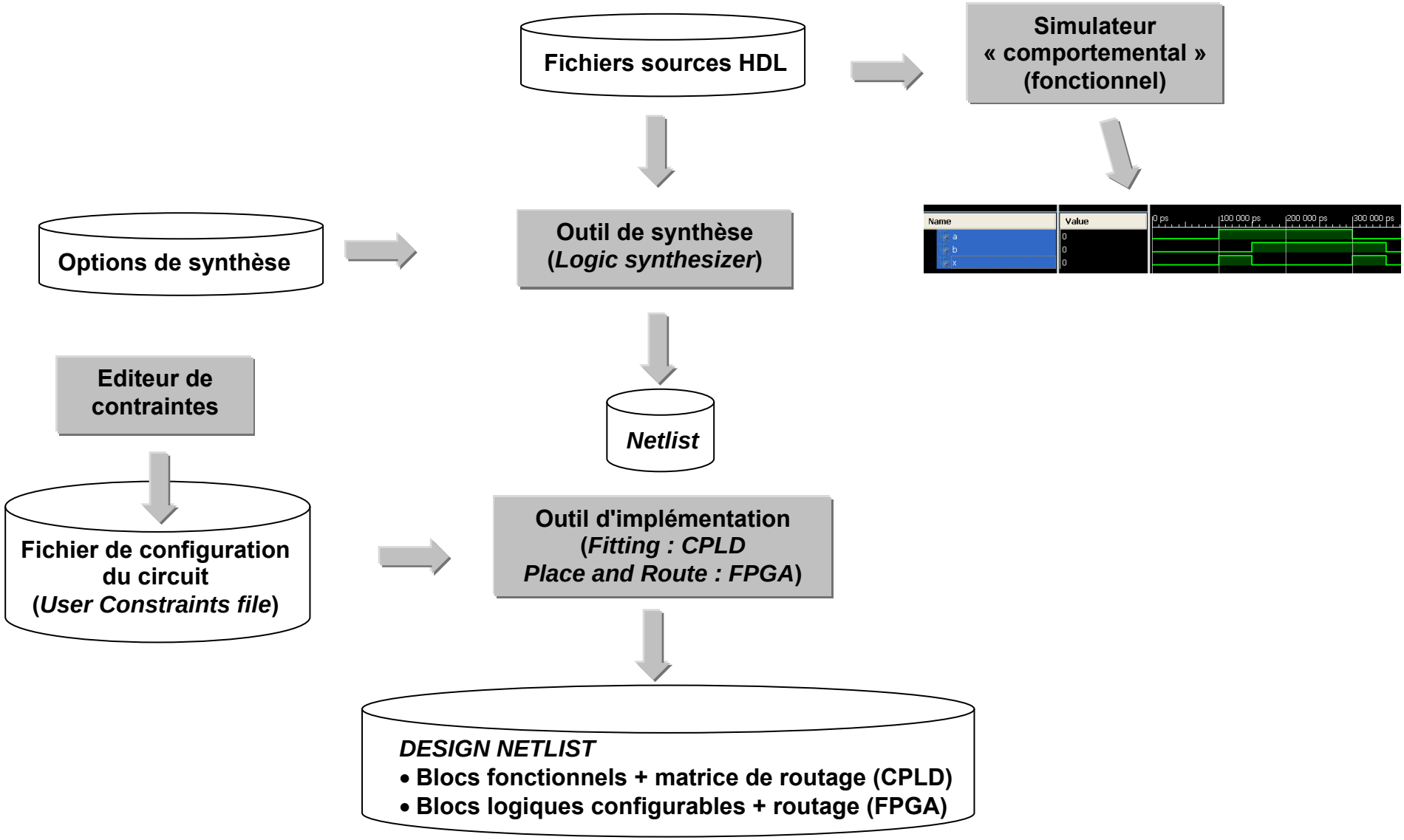

**Traducteur FBD → HDL**

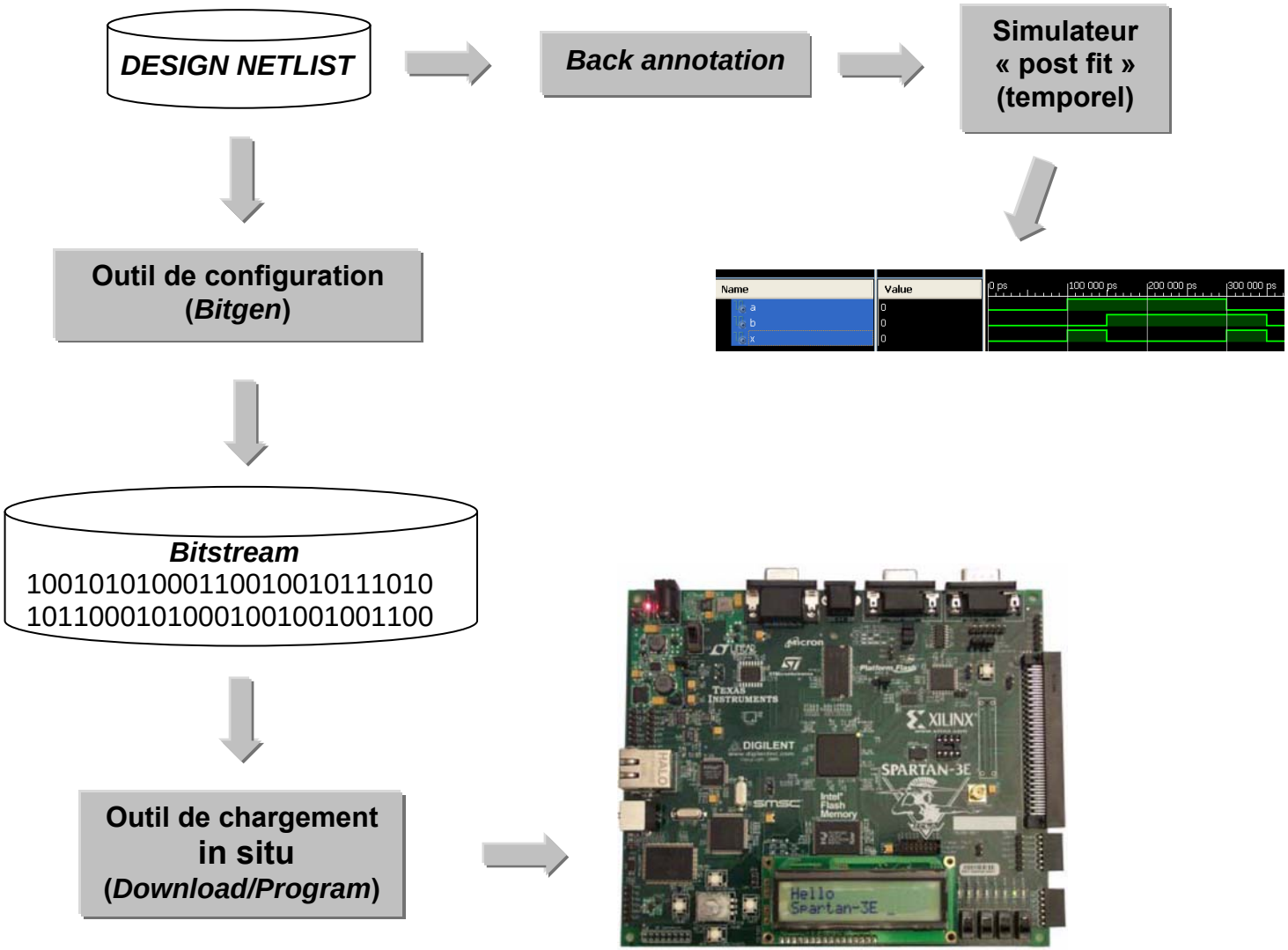
**Traducteur FSM → HDL**

```
-- code VHDL (comportement)

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

**Fichiers sources HDL**

```
-- code VHDL (stimuli)

ar <= '0' after 150ns;
```


Fichiers sources HDL
Simulateur « comportemental » (fonctionnel)
Name
Value
0 ps
100 000 ps
200 000 ps
300 000 ps
a
b
x
0
0
0
Options de synthèse
Outil de synthèse (Logic synthesizer)
Editeur de contraintes
Netlist
Fichier de configuration du circuit (User Constraints file)
Outil d'implémentation (Fitting : CPLD Place and Route : FPGA)
DESIGN NETLIST
• Blocs fonctionnels + matrice de routage (CPLD)
• Blocs logiques configurables + routage (FPGA)


DESIGN NETLIST
Back annotation
Simulateur « post fit » (temporel)
Outil de configuration (Bitgen)
Name
Value
a
b
x
0 ps
100 000 ps
200 000 ps
300 000 ps
Bitstream
10010101000110010010111010
10110001010001001001001100
Outil de chargement in situ (Download/Program)
XILINX
SPARTAN-3E
DIGILENT
Hello
Spartan-3E

## 5.2. Lexique

- ***Synthesis***

  **Transformation d'une description HDL en *netlist*, c.-à-d. en un fichier accepté par un outil d'implémentation**

- ***Netlist***

  **Description standard des portes logiques, des flips-flops, des interconnexions et des noms des entrées-sorties.**

  **Le standard industriel est le format *EDIF : Electronic Design Interchange Format***

- ***User constraints***

  **Numéros de broche (*pin-out*), niveaux de tension, *slew rate* (pente des signaux de sortie), …**

- ***Fitting (CPLD)***

  **Implémentation dans un circuit CPLD particulier : sélection des ressources logiques du circuit et des chemins d'interconnexion en tenant compte des broches d'entrées-sorties choisies par le concepteur**

- ***Place and Route (FPGA)***

  **Implémentation dans un circuit FPGA particulier : choix de la meilleure localisation (par ex. le chemin le plus court) des blocs logiques du circuit (*Place*) et détermination des interconnexions des blocs (*Route*)**

- ***Behavioral simulation***

  **Simulation fonctionnelle par interprétation directe du code HDL (sans délais de propagation de signaux)**

- ***Back-annotation***

  **Procédé post-implémentation permettant de connaître avec exactitude les délais de propagation des signaux dans le circuit choisi (délais induits par les LUTs, les flip-flops et le routage).**

- ***Post fit/route simulation (gate level simulation)***

  **Simulation temporelle (avec indication des délais de propagation de signaux)**

- ***Bitstream***

  **Fichier (par ex. ASCII) contenant des '1' et des '0' pour la configuration du circuit cible**

- ***Programming***

  **Programmation des antifusibles des circuits OTP (CPLD ou FPGA).**

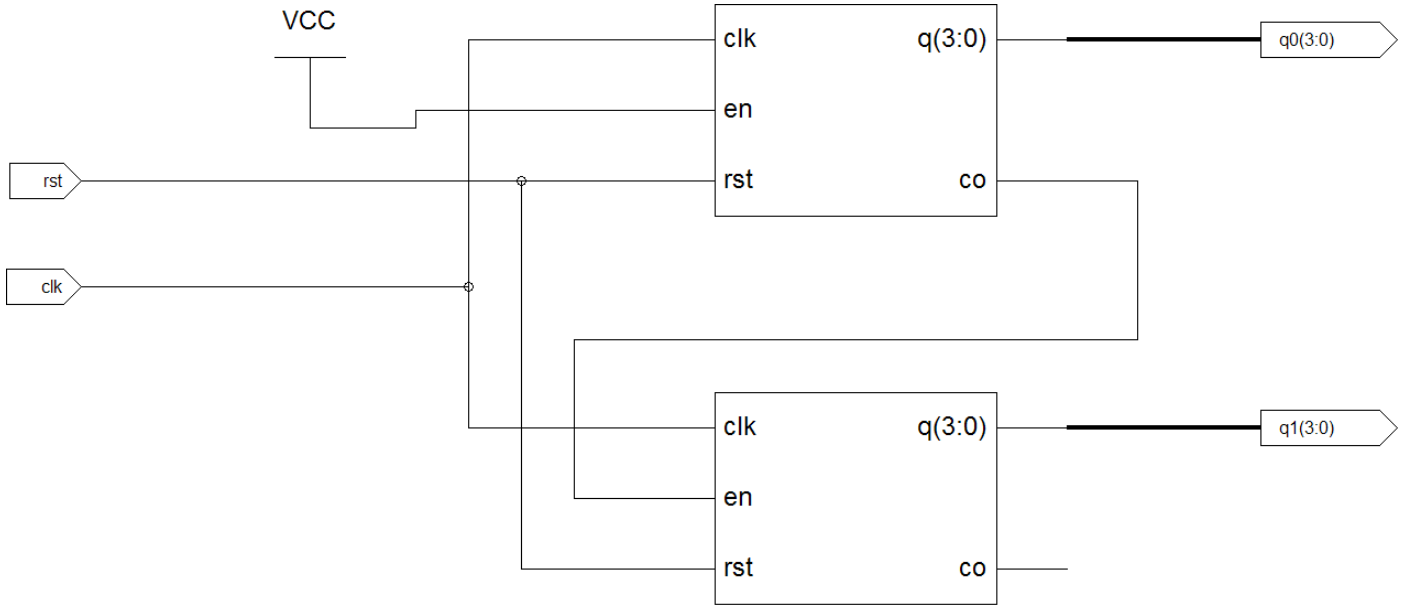
  **Chargement de la configuration dans un circuit non volatile avec conservation de l'information à la mise hors tension du circuit (circuit CPLD reprogrammable, ou mémoire annexe PROM d'un circuit FPGA SRAM)**

- ***Downloading***

  **Chargement de la configuration dans un circuit volatile (directement dans la SRAM d'un circuit FPGA)**

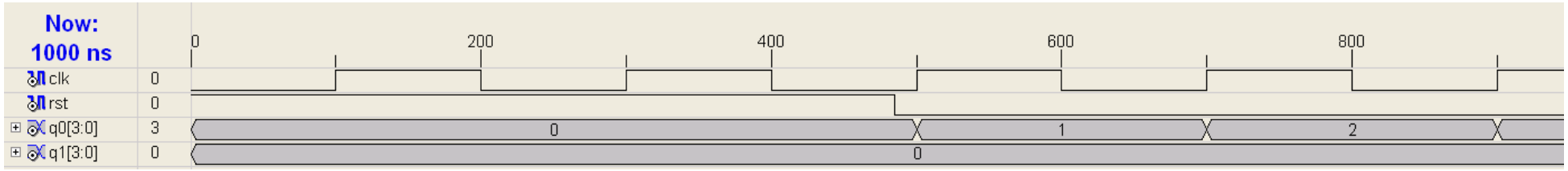
# 5.3. La simulation

## ❑ Exemple : un compteur BCD 2 digits



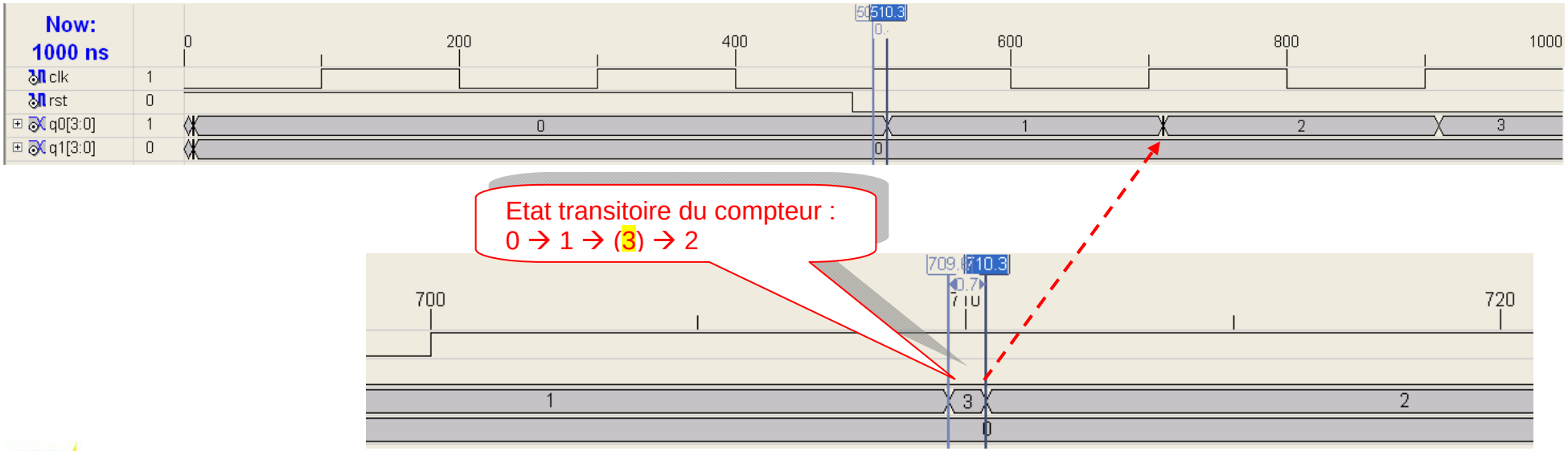
### 5.3.1. Simulation comportementale (fonctionnelle)

- Elle montre le comportement théorique (pas de temps de propagation)




### 5.3.2. Simulation « post route »

- Les délais de propagation sont ceux du circuit FPGA Spartan 3A XC3S700A (Xilinx).



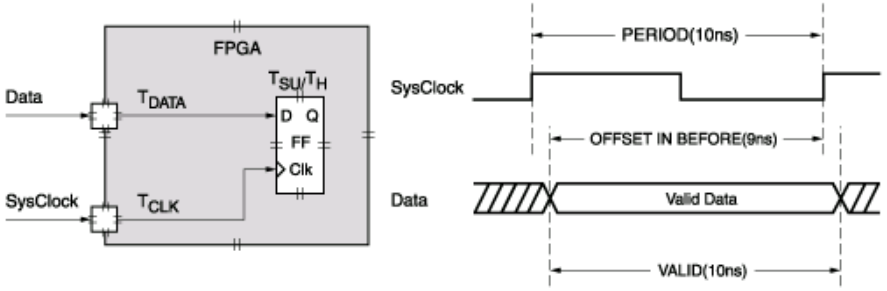
# 5.4. La configuration

## 5.4.1. Options de synthèse

- **Mode d'optimisation**
  - **espace / vitesse**
- **Codage des machines d'états**
  - **Compact, sequential, one-hot, Gray, Johnson, user-encoded, …**
  - **Pour les FPGA : one-hot => un flip-flop par état => logique état futur réduite => gain de temps**

## 5.4.2. Le fichier des contraintes

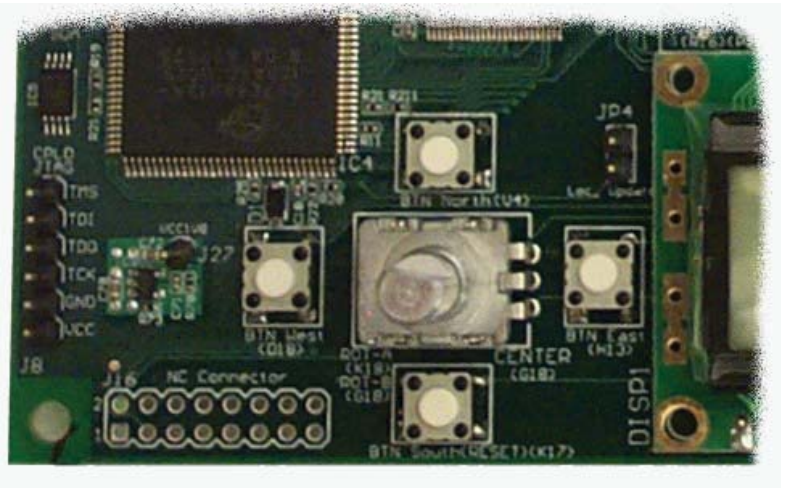
- **Contraintes temporelles d'entrée et de sortie (uniquement pour les FPGA)**
  - **set up time, clock period, …**




- **Propriétés des broches d'entrée-sortie**
  - **Assignation des broches**
  - **Niveau de tension (ex. LVTTL 3.3V)**
  - **Terminaison (ex. PULLDOWN ou PULLUP)**
  - **Temps de montée, *Slew rate* (ex. FAST ou SLOW)**

**Le circuit FPGA `Xilinx XC3S500E SPARTAN-3E` comprend 230 broches d'entrées-sorties**

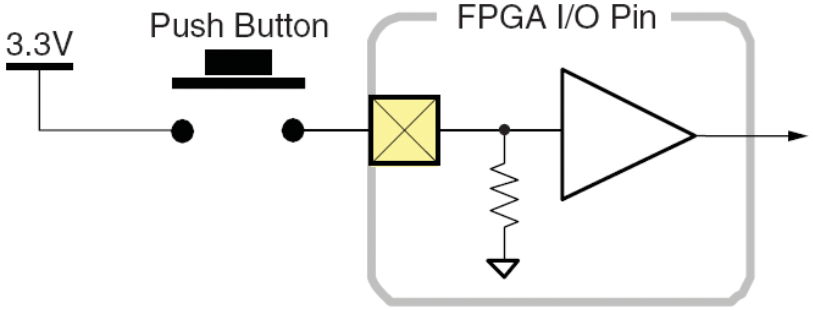
**Les 4 boutons poussoirs (ci-contre) doivent être connectés sur les entrées référencées `H13, V4, K17 et D18`**




## Instructions de configuration

```
NET "BTN_EAST" LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "BTN_NORTH" LOC = "V4" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "BTN_SOUTH" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "BTN_WEST" LOC = "D18" | IOSTANDARD = LVTTL | PULLDOWN ;
```

Permet de configurer la connexion de la résistance de « pulldown » »

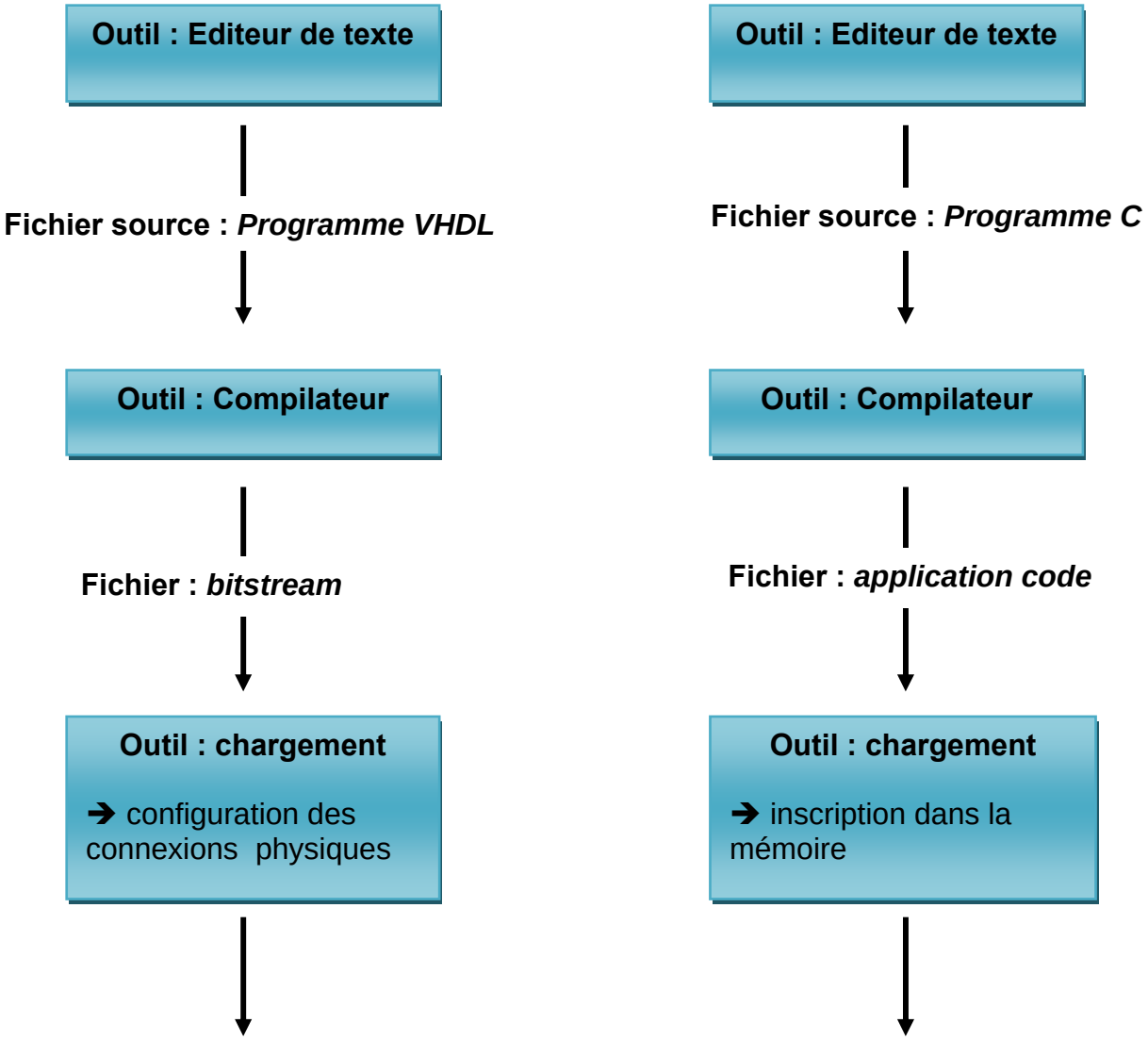


## 5.5. Programmation CPLD / programmation microcontrôleur

- **Le code compilé d'un programme écrit en langage HDL (tel VHDL ou Verilog) est destiné à configurer les interconnexions des ressources physiques des circuits programmables (CPLD, FPGA)**

**A titre de comparaison :**

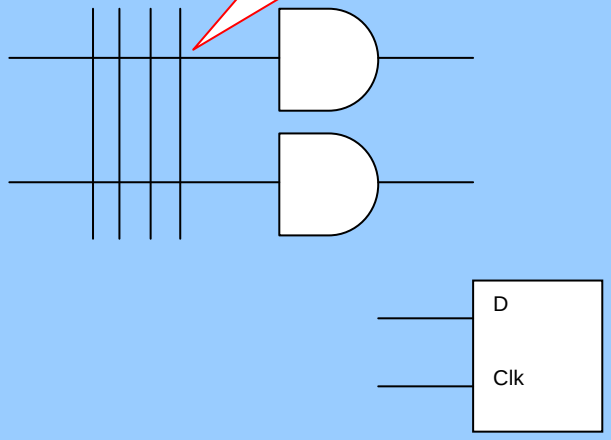
- **Le code compilé d'un programme écrit dans un langage de programmation procédural (tel que C) est destiné à être exécuté sur une machine programmée (ordinateur, microprocesseur, microcontrôleur)**


Outil : Editeur de texte
Fichier source : Programme VHDL
Outil : Compilateur
Fichier : bitstream
Outil : chargement
➔ configuration des connexions physiques
Outil : Editeur de texte
Fichier source : Programme C
Outil : Compilateur
Fichier : application code
Outil : chargement
➔ inscription dans la mémoire

Le fichier *bistream* définit les connexions physiques entre les portes logiques et les flip-flops.

➔ **C'est l'image d'un "câblage"**

**Circuit intégré : CPLD**




Le fichier code contient des instructions codées (8bits, 16 bits, …)

➔ **C'est le programme qui sera exécuté par l'unité centrale**

**Circuit intégré : Microcontrôleur**