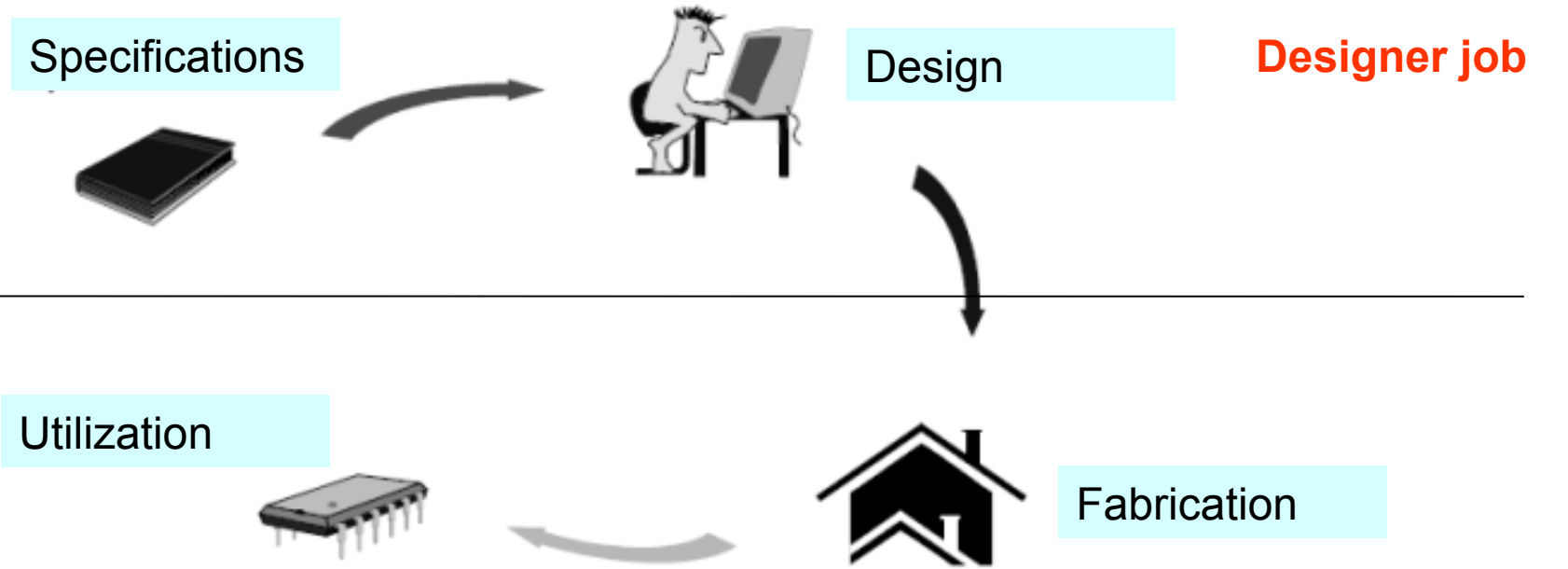
VHDL

VHDL pour les …

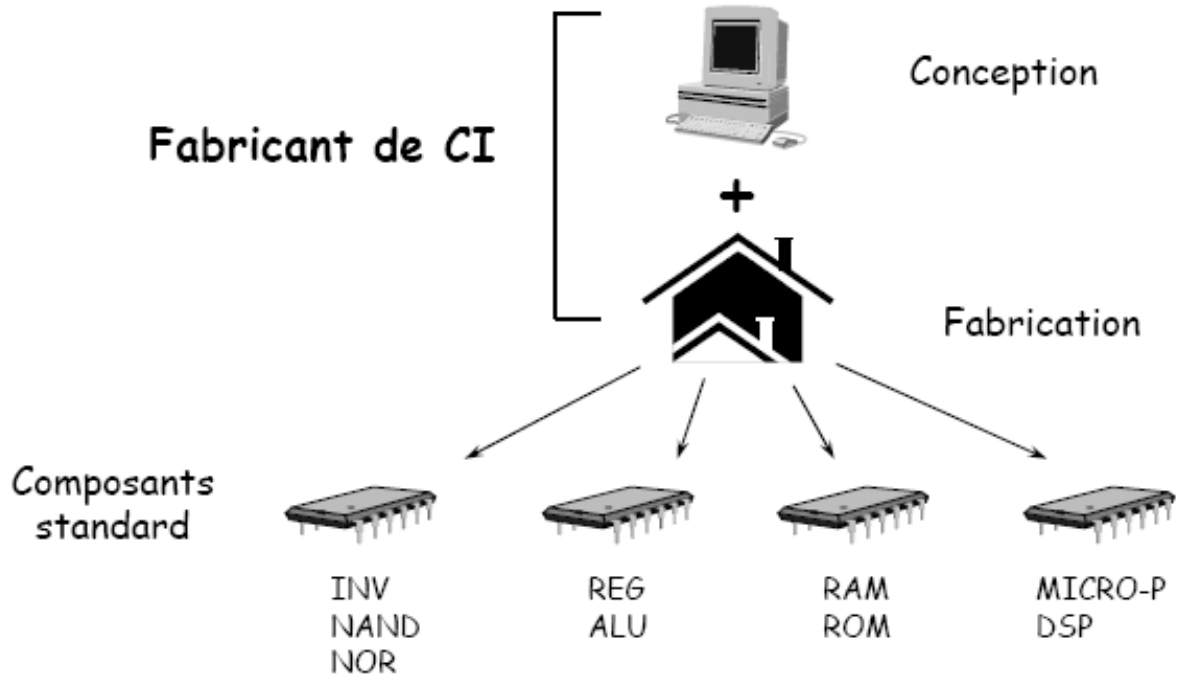
Ressources web VHDL
http://www.cs.umbc.edu/help/VHDL/
http://tams-www.informatik.uni-hamburg.de/vhdl/ **- la meilleure archive de VHDL**

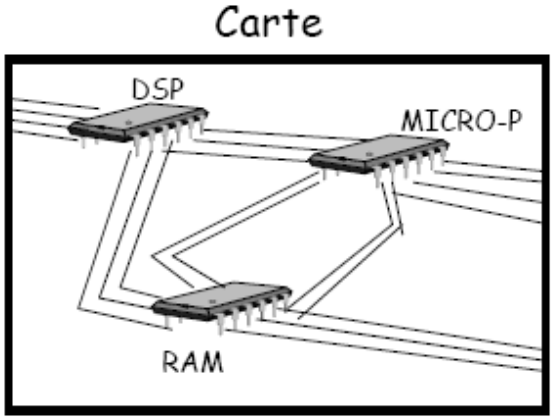
Some Concepts (how do we make an integrated digital circuit….?)



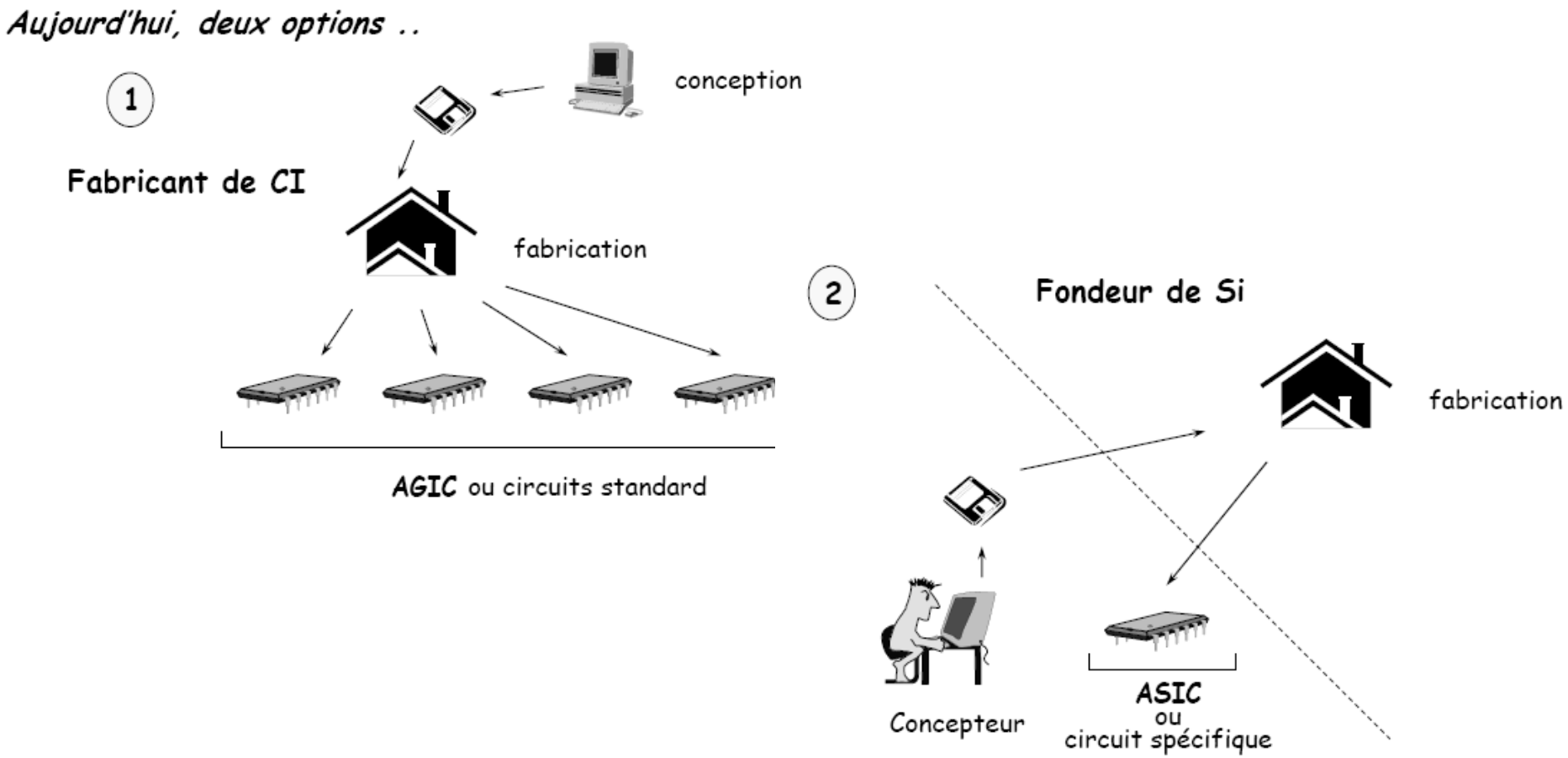
Electronic (old and new)

Autrefois, une seule option ..





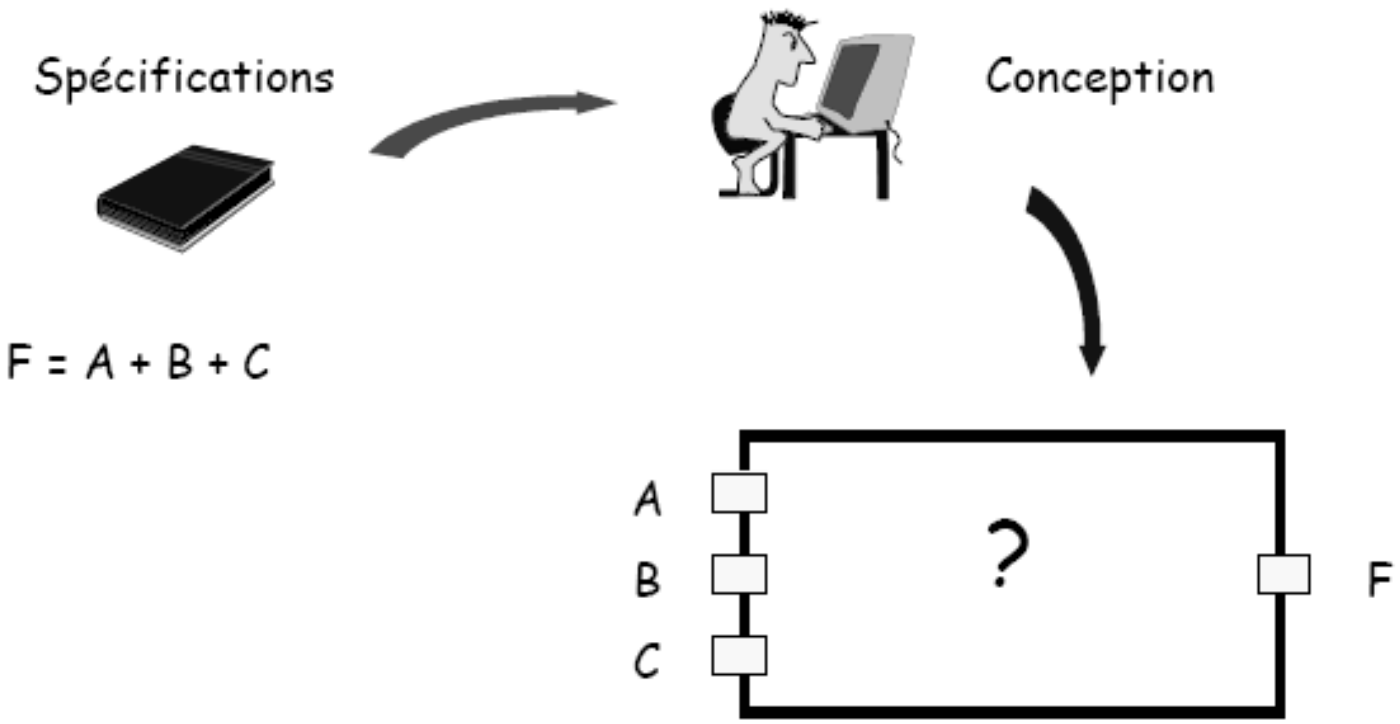
Electronic (old and new)



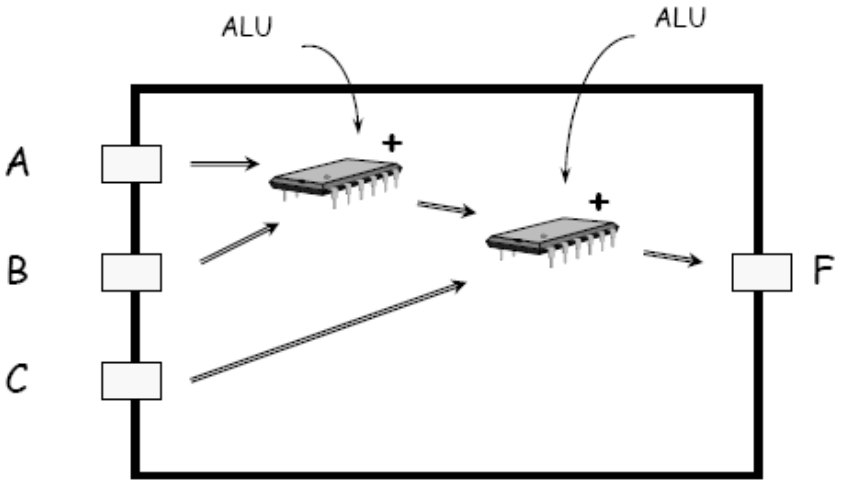
Digital Design Styles…ASIC

- ASIC = Application Specific IC
- What is in a ASIC?
  - Collection of Complex processors, memories, gates, registers, ALU, control machines, etc in a single circuit dedicated to a specific application
- Why ASIC?
  - Because designers can manage optimizations as they whish (area, power, timing), but huge cost
  - Because fabricants can make high volume products
  - We need to get to a product with lower costs in the best TTM?

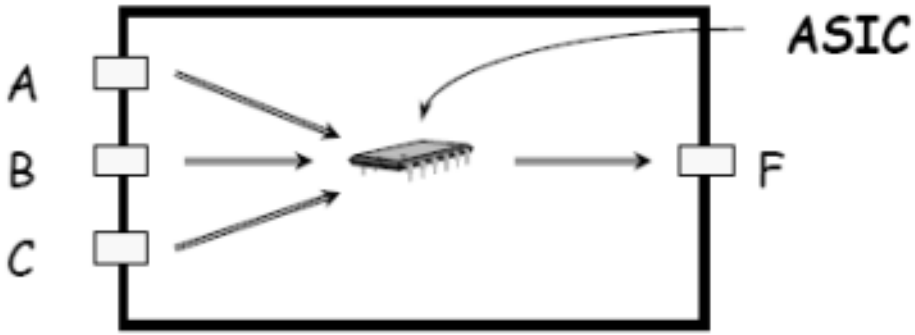
Example



- en utilisant des circuits standard



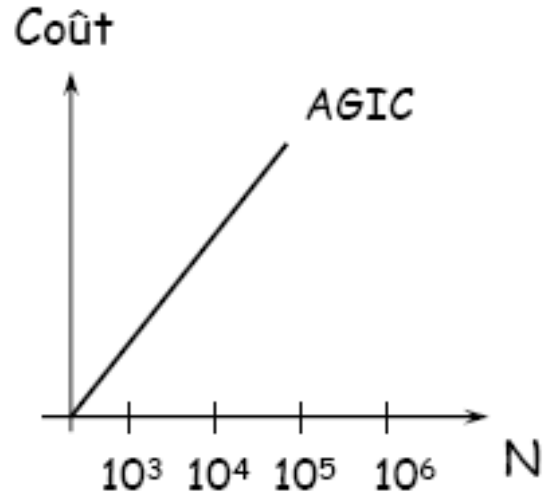
- en utilisant un ASIC



Avantages :

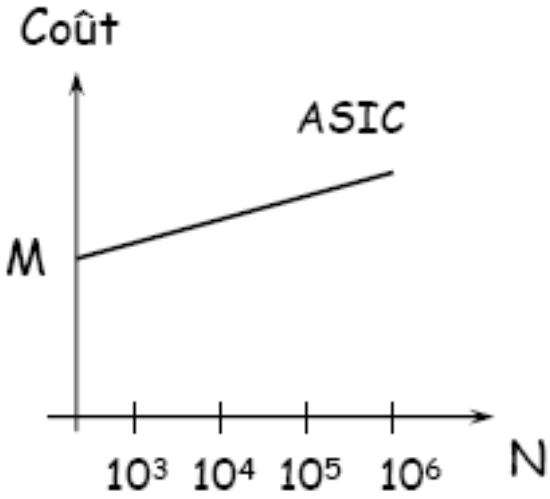
- remplacement de k circuits par un circuit unique (gain en surface)
- remplacement de k circuits par un autre plus performant

Comparison

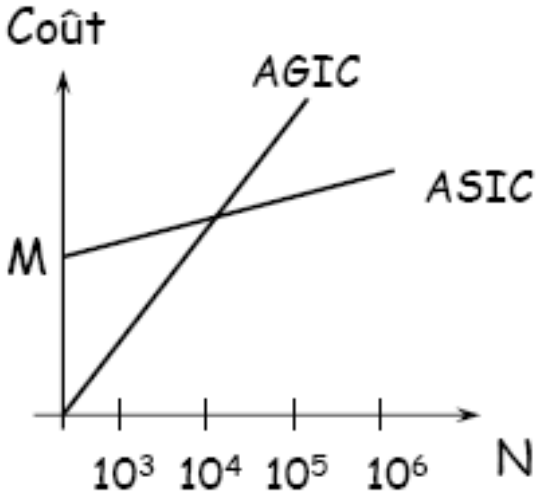


$C_{AGIC} = Cu * N$

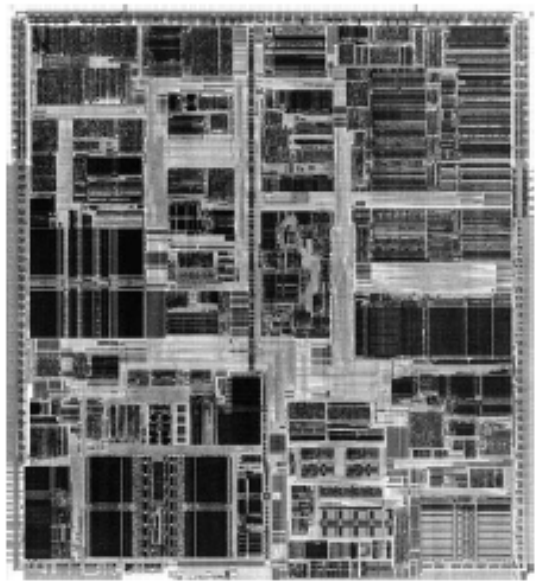
COUT



$C_{ASIC} = C'u * N + M$

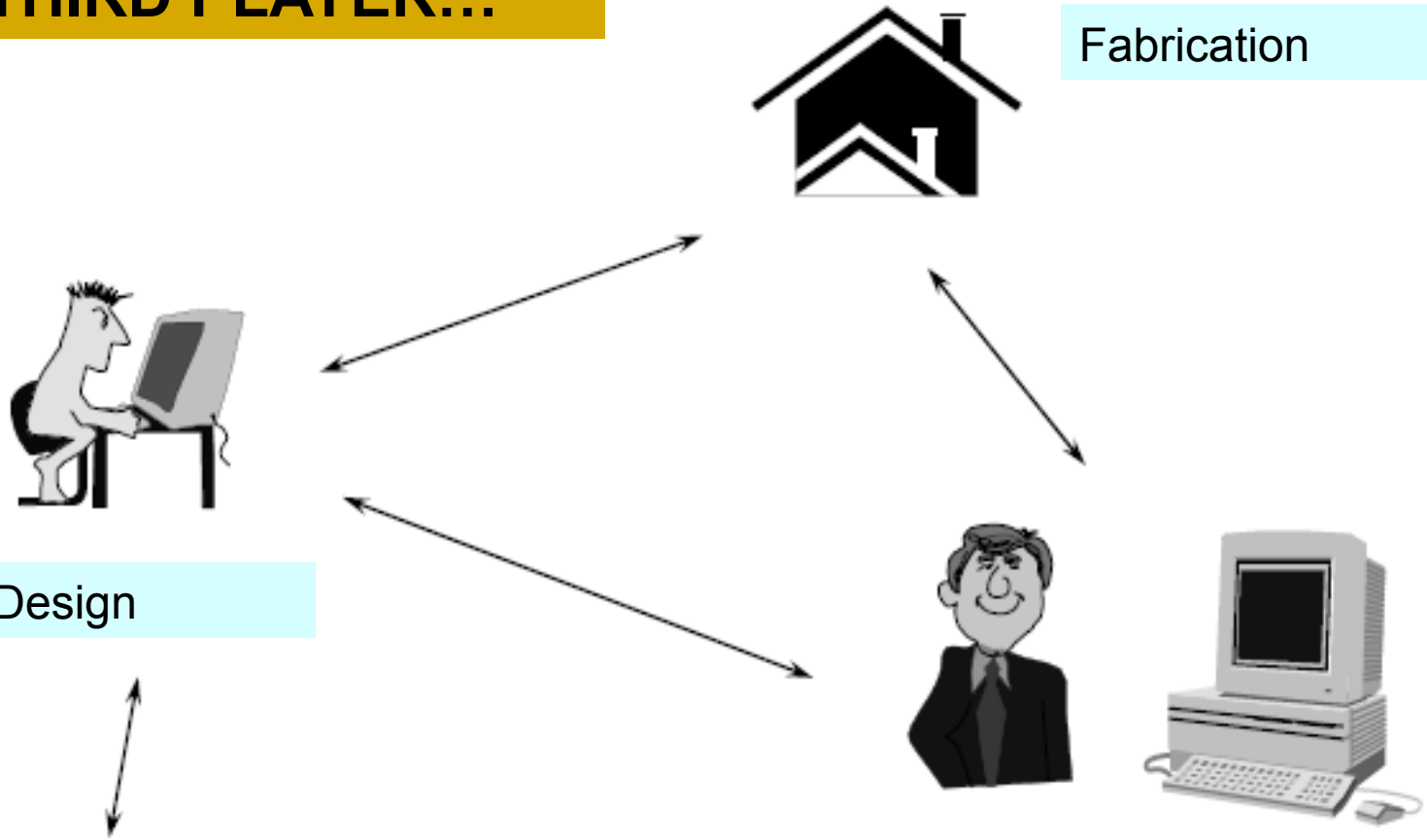


High Complexity/High Speed/G operations

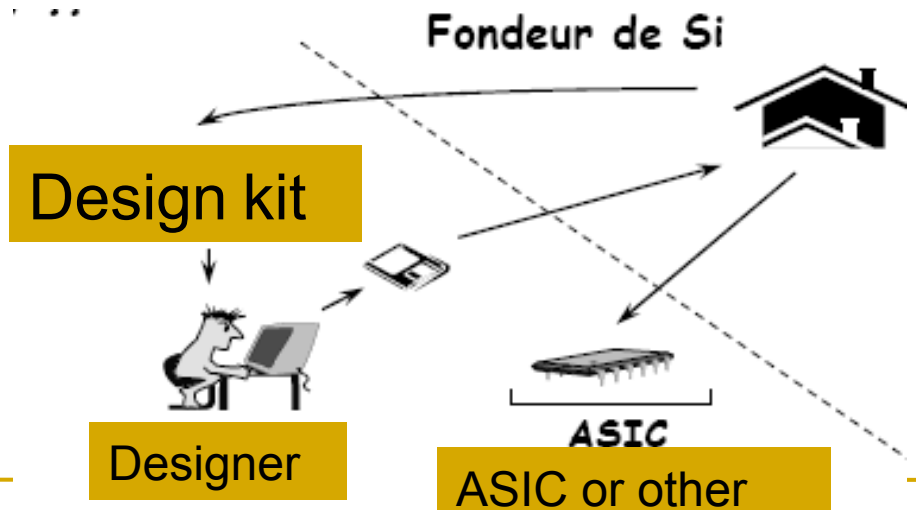


Pentium® 4 (2003)
Frequency : 3,2 GHz
Techno. : CMOS 0.13 µm
Transistors : > 55 M
Prix : 364 $

THE THIRD PLAYER…

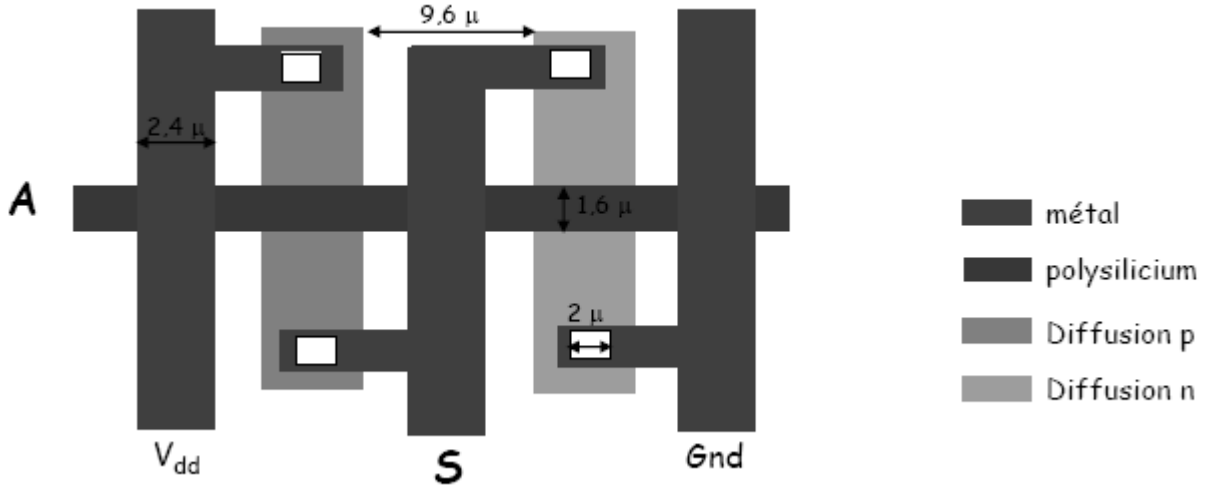


- Designer buys and uses for circuits designs:
  - Many CAD tools from CAD vendor
  - The DESIGN KIT from Fab Houses and integrate them into the tools

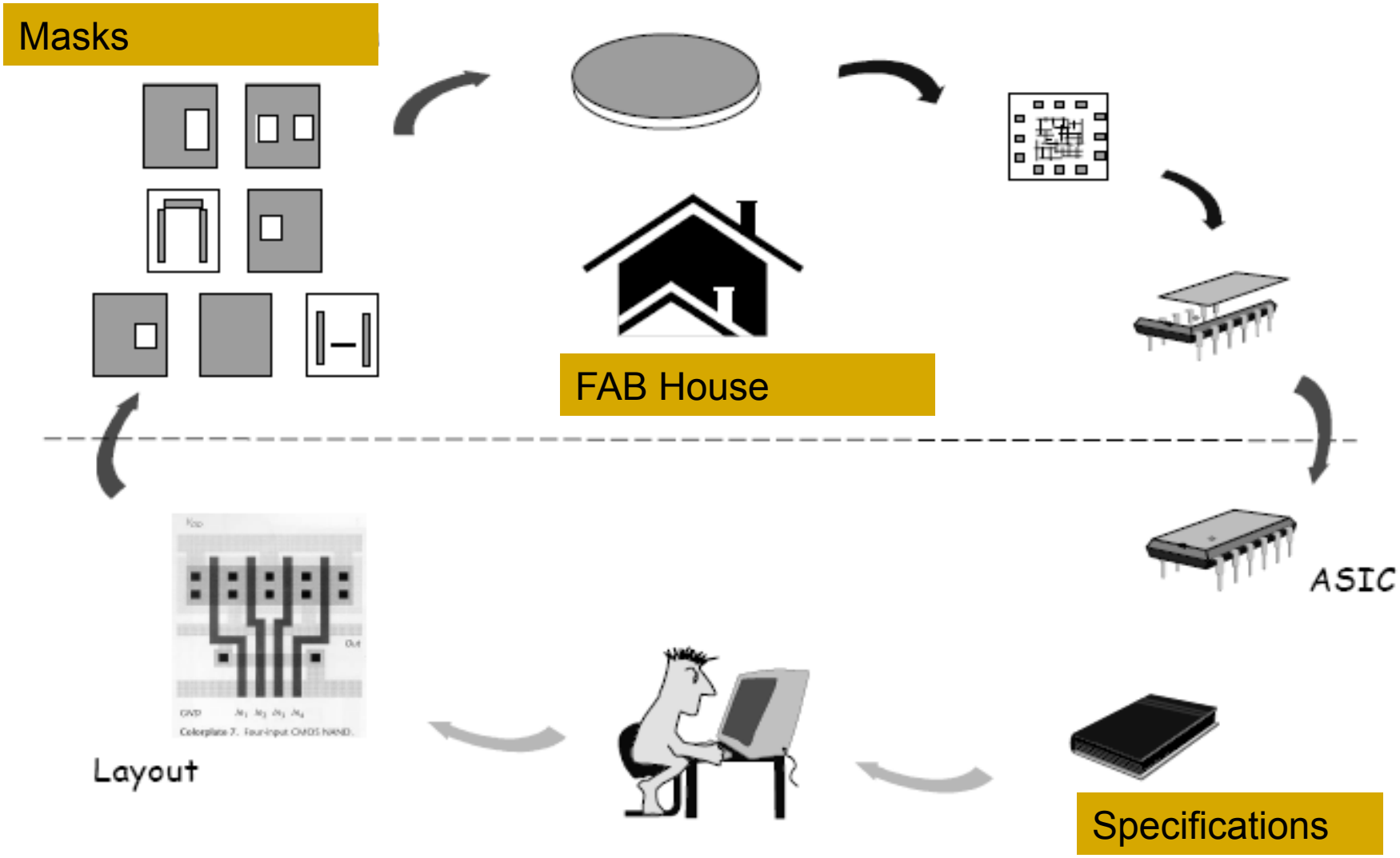


What is a Design KIT?

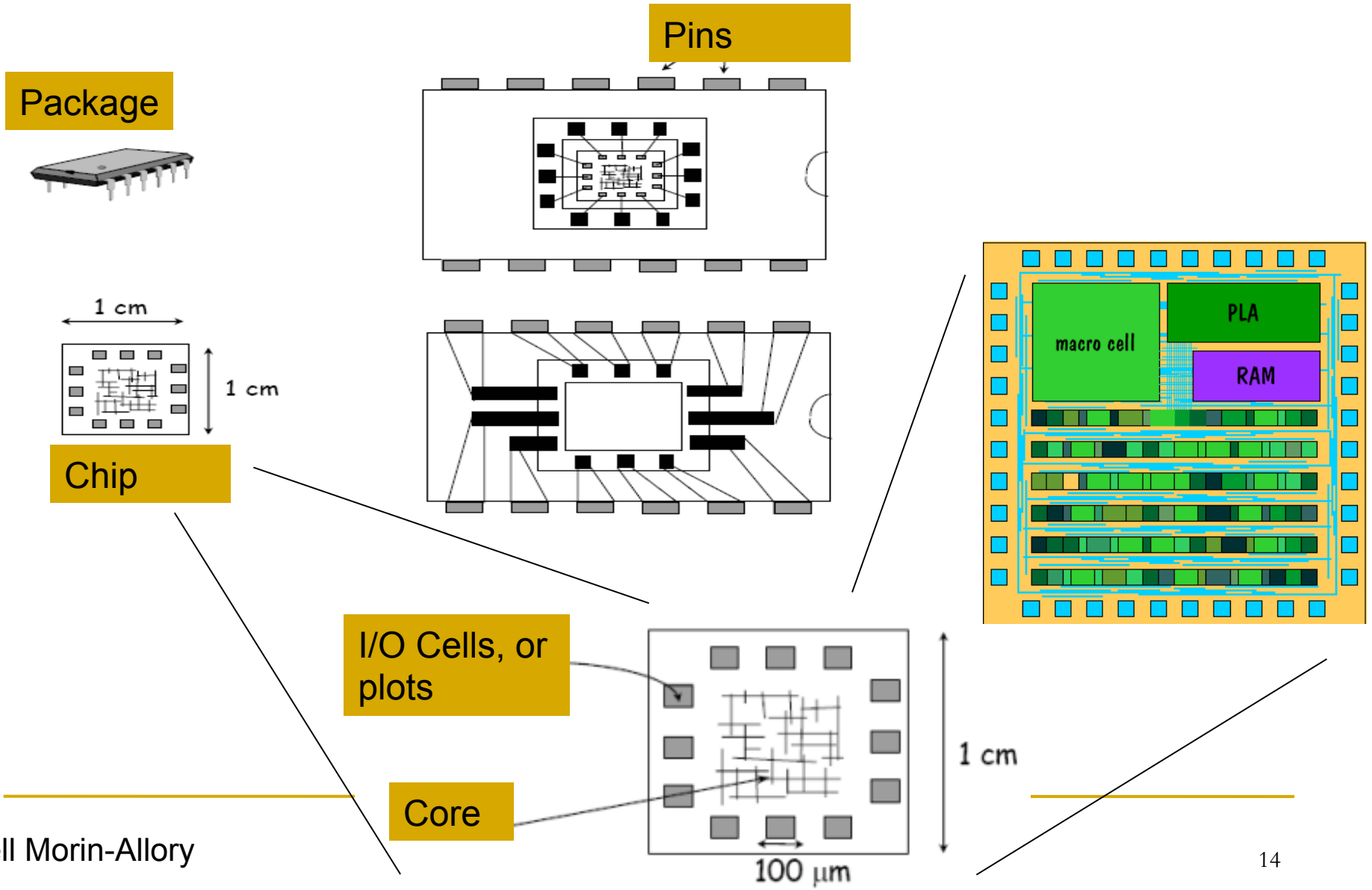
- Many things…
  - Library of logic cells and memory elements that can be fabricated by a Fab house
  - Each of these cells fully characterized wrt area, power, timing
  - Each of it with the corresponding layout
  - Rules of fabrication (and restrictions)



The whole flow

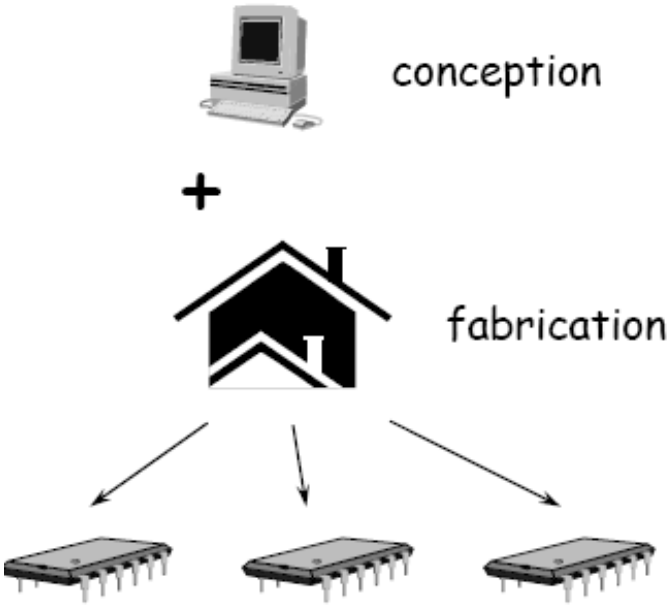


Zoom Chip and Package

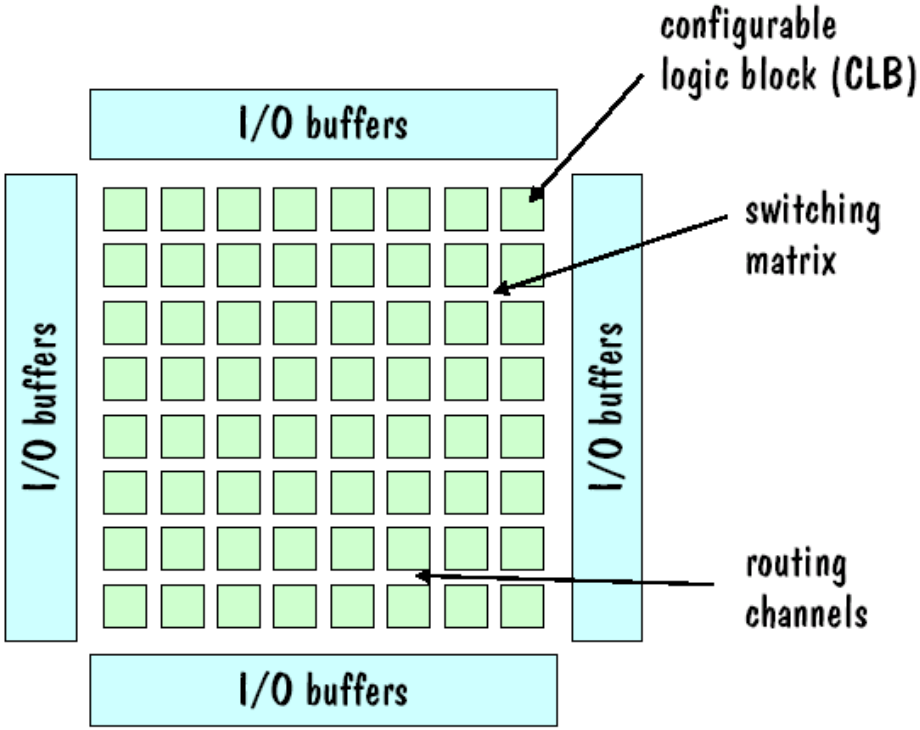


Digital Design Styles… Other circuits

- Programmable circuits



FPGA, CPLD, etc

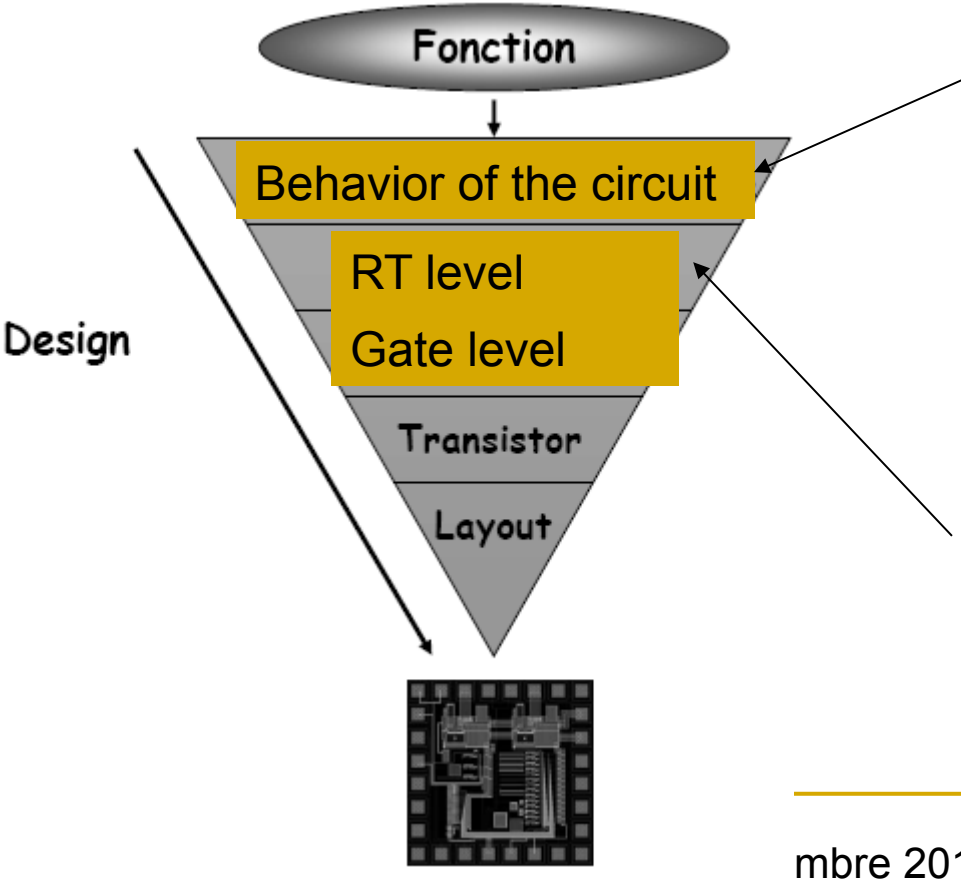


FPGA?

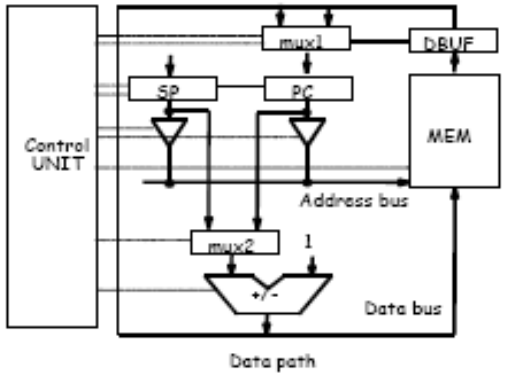
- Why?
  - Independent of application
  - Can be used thousands of times
  - Designer make his own design and program it on a empty circuit
  - No dialog with fab house
  - Very interesting in the case of few circuits volume, or application that change often (telecom)
- BUT?
  - FPGA circuit is not optimized in power, delay
  - Has a limited capacity
  - Not fully characterized
  - Too expensive for high volume…

Back to ASIC design (abstraction levels)

- By using CAD and design kit

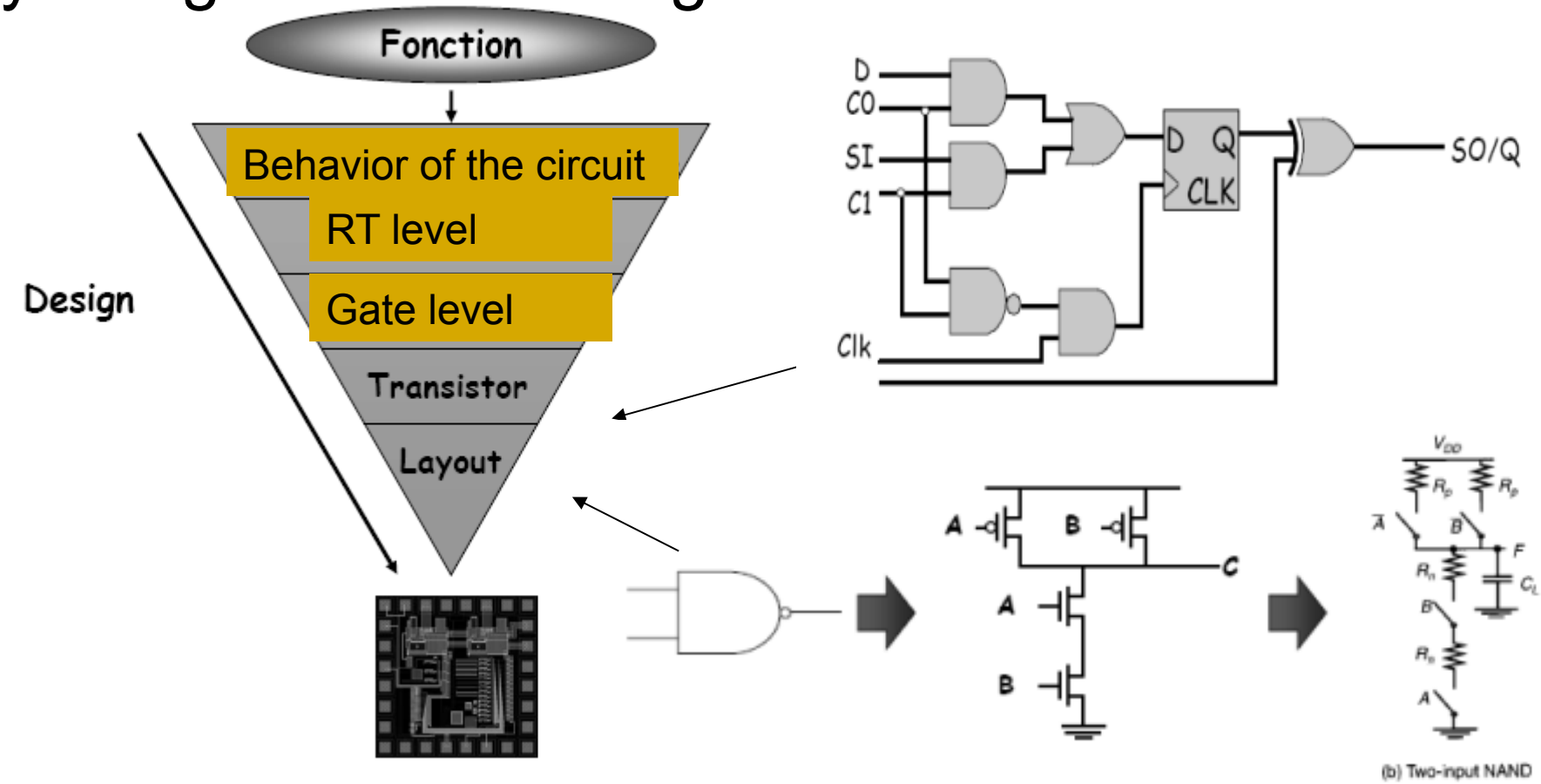



```
...
architecture behavioral of counter is
...
if clk'event and clk='1' then
res <= s2;
s2 <= s1;
s1 <= a + b;
end if
...
if reset ='0' then in_state <= "00";
case in_state is
        when "00" => in_state <= "01";
        when "01" => in_state <= "10";
```



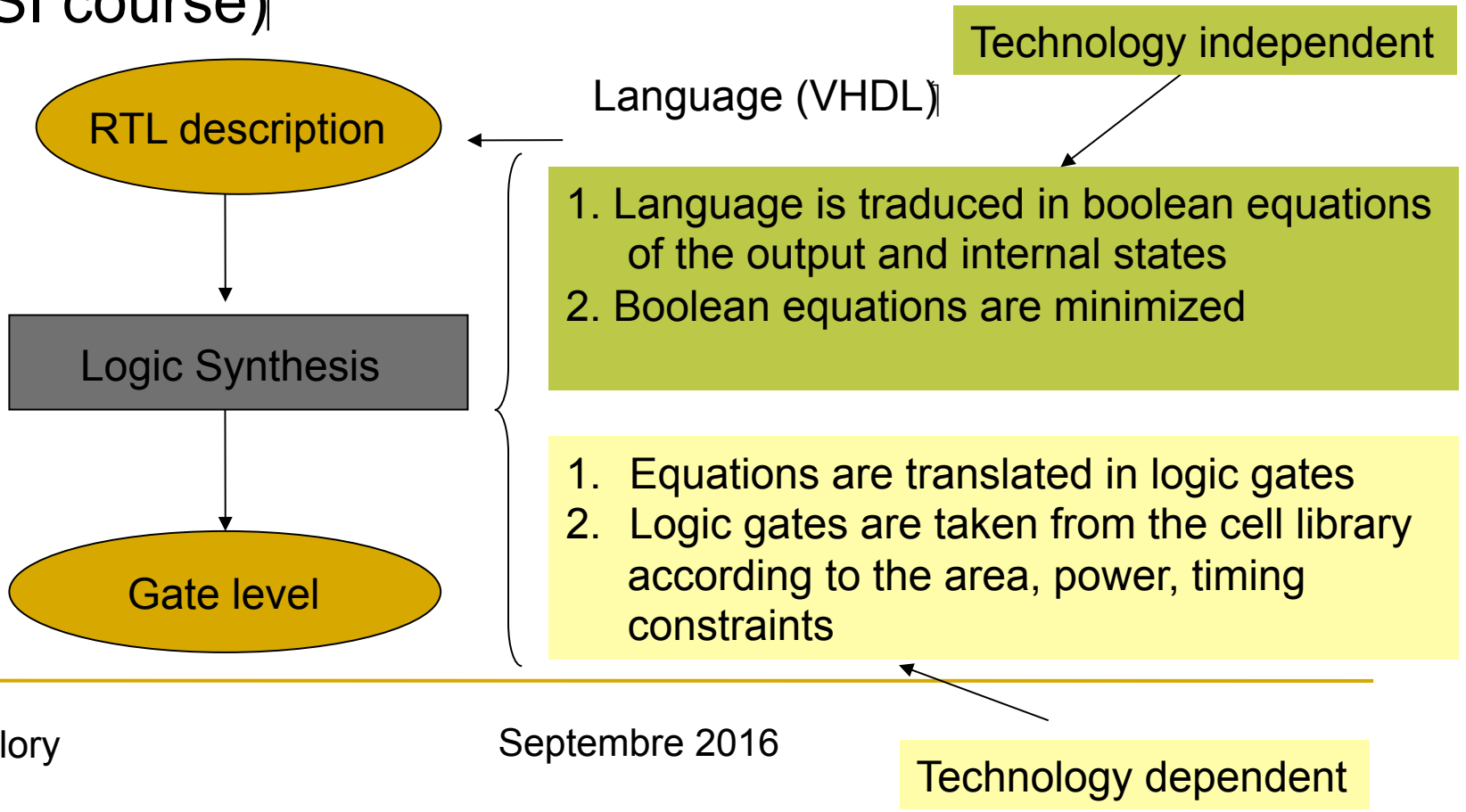
Back to ASIC design (abstraction levels)

- By using CAD and design kit



In this course …

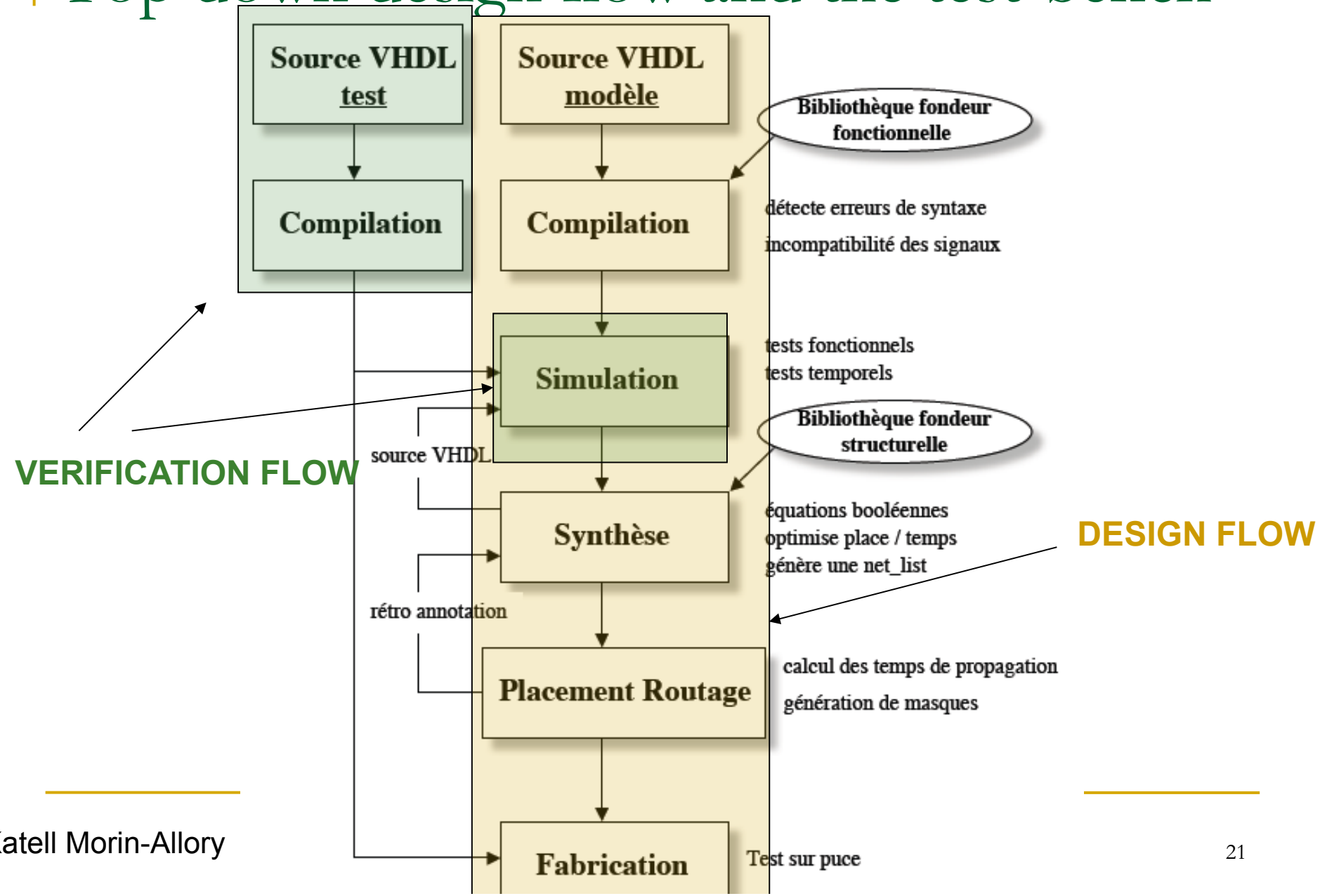
- We play modest… and stay between RTL Level and logic level…(we will see gates and transistors in VLSI course)



VHDL Overview

- VHDL hardware description language, for circuit and test-bench
- Goal: You are able to
  - **Design digital circuits with the VHDL language with behavioral, dataflow and structural modeling.**
  - **Test your circuit and verify its good/bad functionality (and correct design errors!)**
- You will be familiar with the top down design flow and the test-bench methodology.

Top down design flow and the test-bench



What is VHDL ?

V **HSIC**
(Very High Speed Integrated Circuit)

H **ardware**

D **escription**

L **anguage**

Introduction à VHDL

- **Demande des militaires US en 1980**

➠ **nécessité d'un langage de description non ambiguë des systèmes matériels**

➠ **unicité d'une telle description**

➠ **pérennité du modèle**

- **En 1987 : VHDL devient une norme IEEE**
- **Aujourd'hui c'est un standard du marché**
  - **tous les outils de conception et simulation compilent et utilisent VHDL**

Introduction à VHDL

- **Avantages**
  - **Complexe**
  - **Description structurée = travail en équipe**
  - **Adapté aux projets multi-entreprises modèles compilés -> sécurité**
  - **Indépendant de la technologie utilisée**
- **Inconvénients**
  - **Description complexe**
  - **Tout n'est pas synthétisable**

- **Pentium : 3.1 Millions de transistors, 1 an de conception**
- **PowerPC 601 : 2.8 M**

Introduction to VHDL

- Language features
  - signal data types (in, out, bidir, signal-strength ...)
  - hardware structures (memory, register-files, ...)
  - logic operators (shift, rotation, masking, ...)
  - asynchronous structures (set, reset of memories)
  - parallel or synchronous structures
- constraints (pin, technology, area, delays, ...)

RTL…

- … is Register Transfer Level
  - Means a way of modeling a circuit at architectural level (with clock cycles, registers, other components that can be combinational or sequential)
- RTL – helps automating the translation of a program into Boolean equations and then into circuits (gates, FF, tristate…)
  - Helps make a description independent of technology

VHDL vs C or other programming language

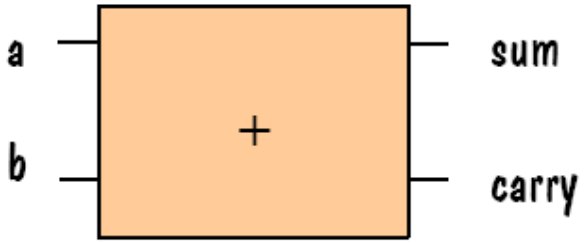
- Some specific concepts very dependent on the hardware
  - **Concurrency** (in addition with sequential execution, VHDL has concurrent/parallel instruction execution)
  - **Hierarchy** (top modules, contain modules, that contain modules, etc…)
  - **Time modeling**
  - **Signals and bus modeling**

Exercice: Full Adder et Half Adder

- Donner plusieurs architectures pour le Full Adder. Quelle est la complexite en nombre de transistors de ces architectures?
- Proposer une architecture pour le Half Adder

Design units
Entity

- the design **entity** is a primary programming abstraction in VHDL
- **entity** defines the interface of a component, without giving any information about the component behavior




```
entity HalfAdder is
   port (a,b : in bit;
         sum,carry : out bit);
end HalfAdder;
```

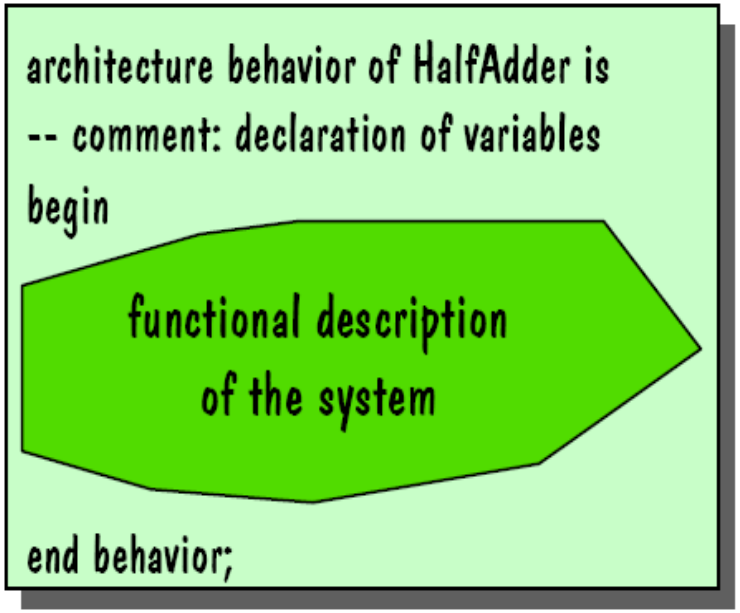
```
library IEEE;
use IEEE.std_logic_1164.all
entity HalfAdder is
   port (a,b : in std_ulogic;
         sum,carry : out std_ulogic);
end HalfAdder;
```

Entity

```
entity FULL_ADDER is
      port      (A,B,CIN : in BIT;
                 Sum, COUT : out BIT);
end FULL_ADDER;
```

Architecture

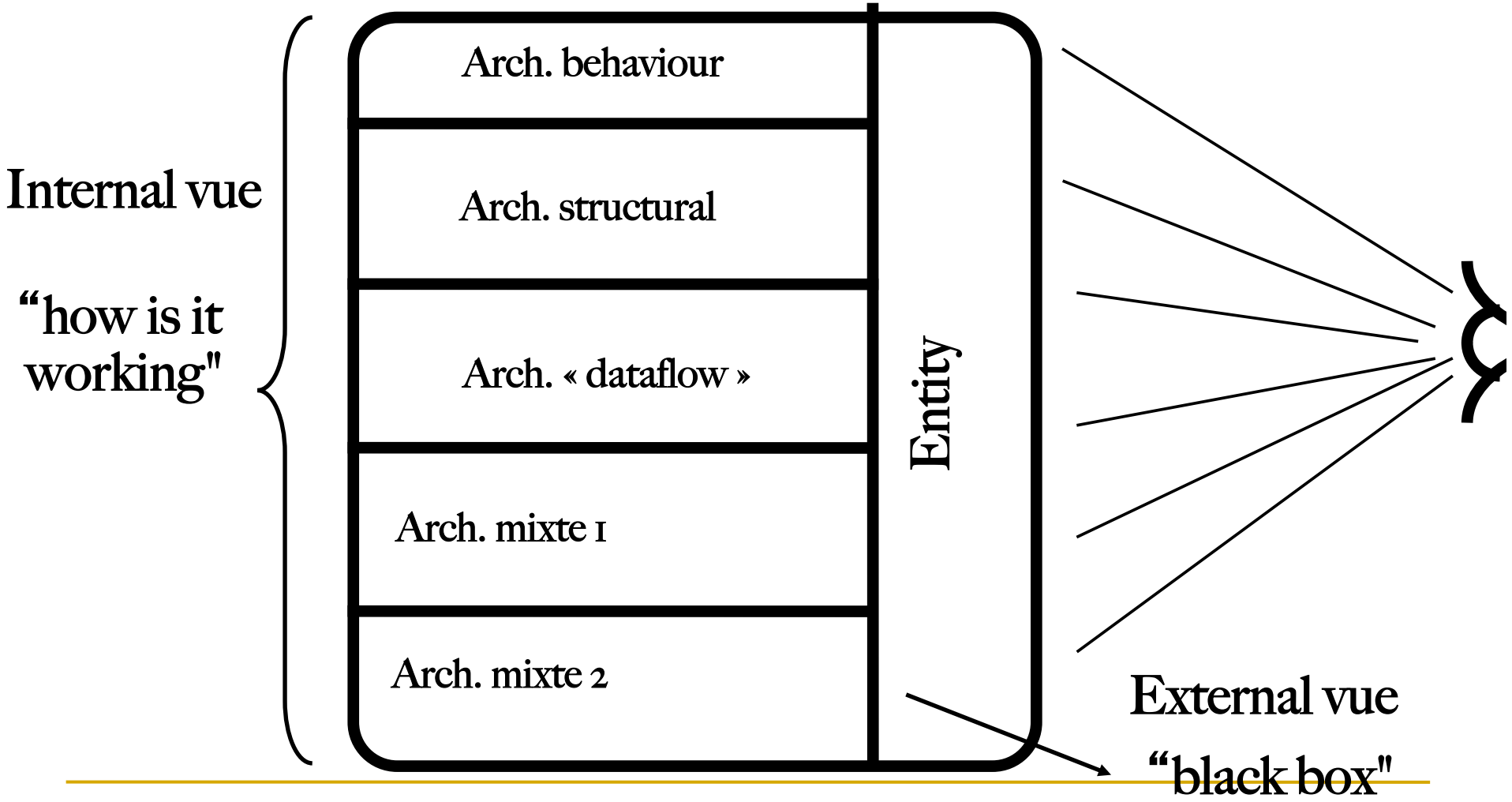
- the design **architecture** is a primary programming abstraction in VHDL
- architecture describes the internal behavior of a component, without giving any information about the component IO's
- The behavioral description can take many forms.
- These forms differ in the levels of abstraction and detail.



Syntaxe :

```
architecture name_of_architecture  of
name_of_entity is
  { here declare signals, and other }
      begin
      { suite concurrent instructions }
      end name_of_architecture ;
```

Entity/Architecture



But at the beginning let's talk: Signals and Bus

- signal values are physically associated to wires
- Carries values and has history
- VHDL language supports signal type:
  - type: **bit**, values: '0', '1'
  - type: **bit_vector**, values: "0001", etc
- VHDL package IEEE 1164 supports signal type:
  - type: **std_logic and vector std_logic_vector**
  - std_logic is a 9 value logic

value	interpretation
U	un-initialized
X	forcing unknown
0	forcing 0
1	forcing 1
Z	high impedance
W	weak unknown
L	weak 0
H	weak 1
-	don't care

Signal vs variable

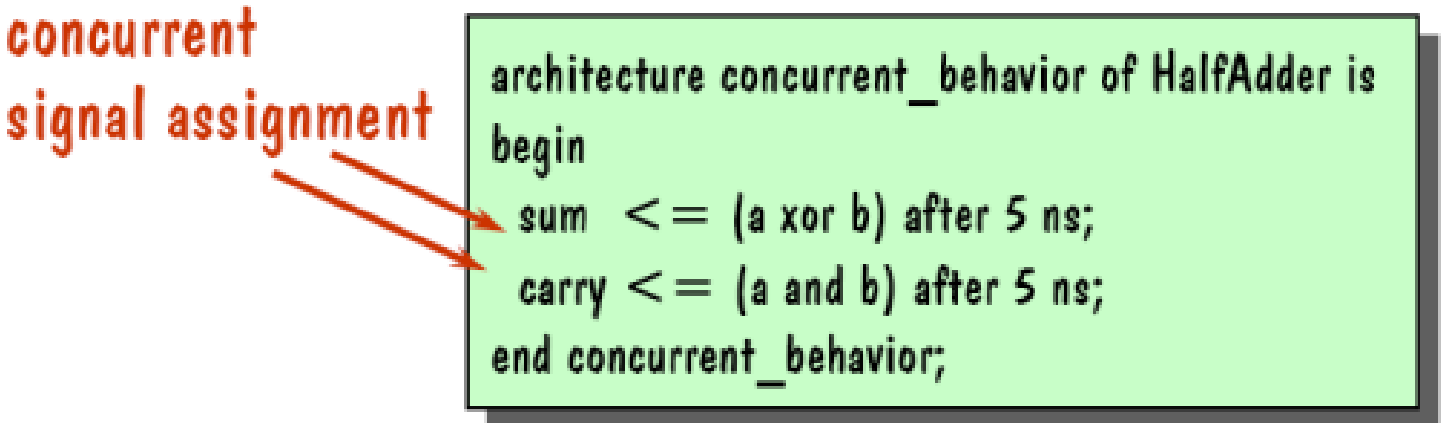
- Other objects in VHDL
  - Signal (bus)
  - Variable
  - Constant

Every object in VHDL (constante, variable or signal) has to be typed.

  - **values and operations**

Concurrency

- The operation of digital systems is inherently concurrent
- Within VHDL signals are assigned values using signal assignment statements <=
- Multiple signal assignment statements are executed concurrently



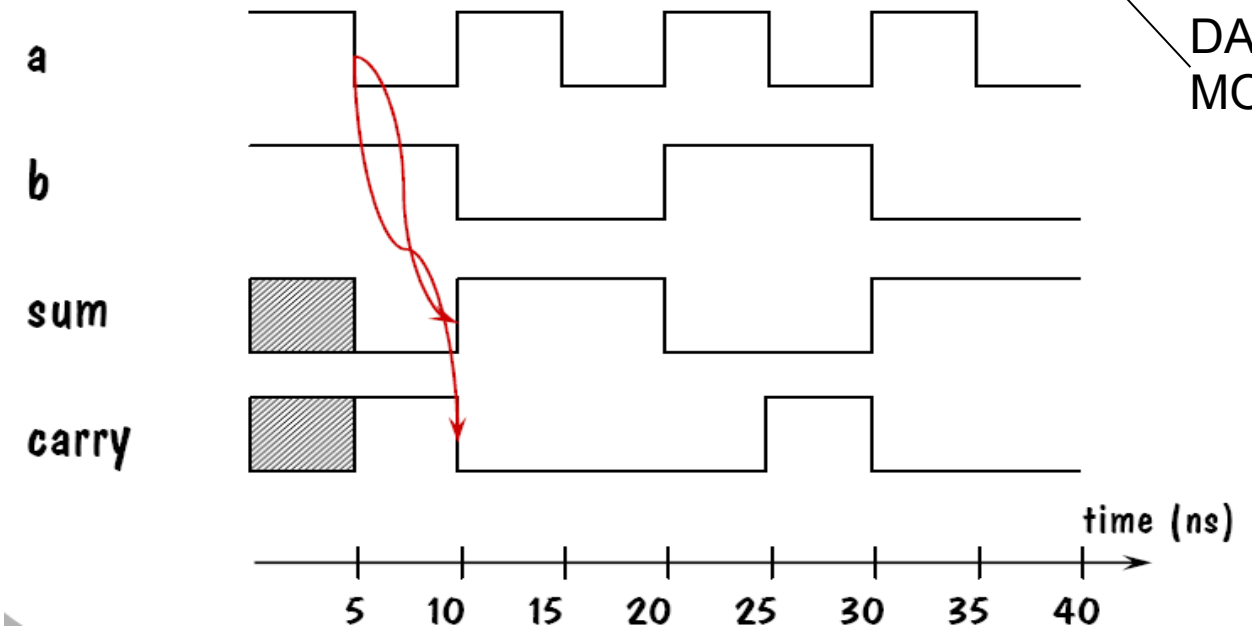
VHDL is concurrent, excepting instructions inside process and sub-programs

Half adder Dataflow architecture

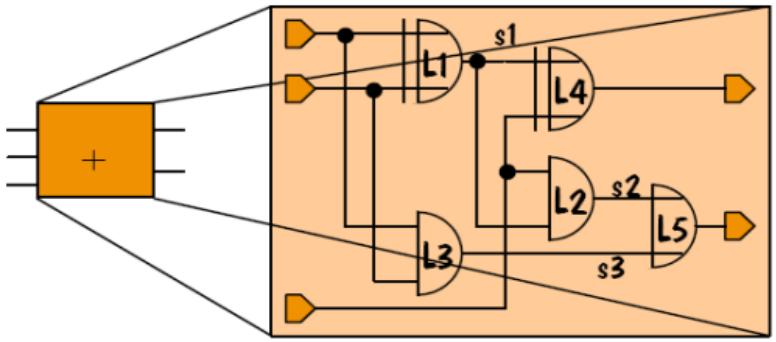
concurrent
signal assignment

```
architecture concurrent_behavior of HalfAdder is
begin
  sum   <= (a xor b) after 5 ns;
  carry <= (a and b) after 5 ns;
end concurrent_behavior;
```

DATA FLOW MODEL



Data Flow Model



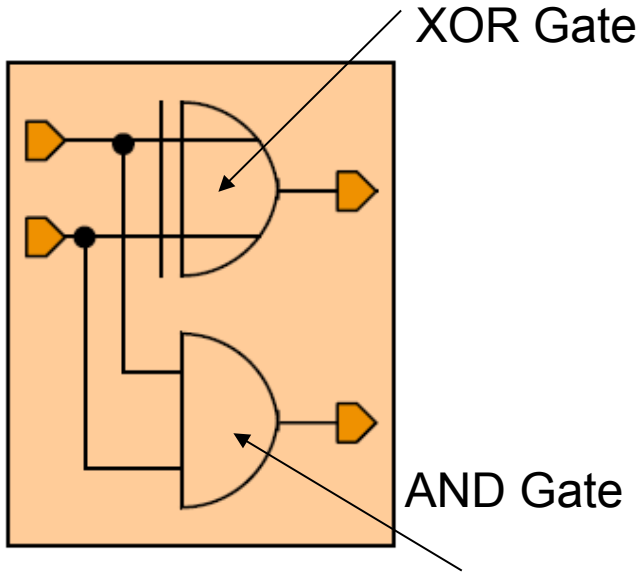
architecture declarative segment

architecture body

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

entity FullAdder is
   port (a,b,cIn: in std_logic;
         cOut,sum: out std_logic);
end FullAdder;

architecture dataflow of FullAdder is
signal s1,s2,s3 : std_logic;
constant gate_delay: Time:=5 ns;
begin
  L1:  s1  <= (a xor b) after gate_delay;
  L2:  s2 <= (cIn and s1) after gate_delay;
  L3:  s3 <= (a and b) after gate_delay;
  L4: sum <= (s1 xor cIn) after gate_delay;
  L5: cOut <= (s2 or s3) after gate_delay;
end dataflow;
```

Architecture

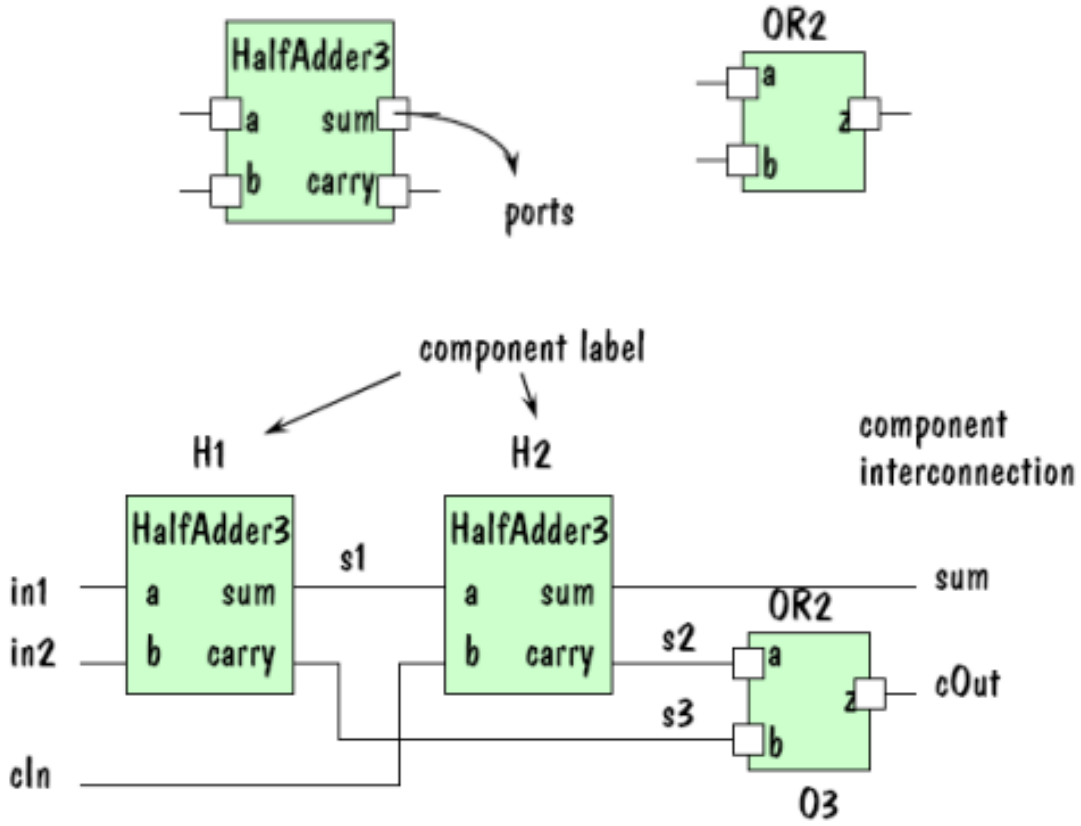
- the design **architecture** is a primary programming abstraction in VHDL
- Half adder **structural** architecture



Structural description mode

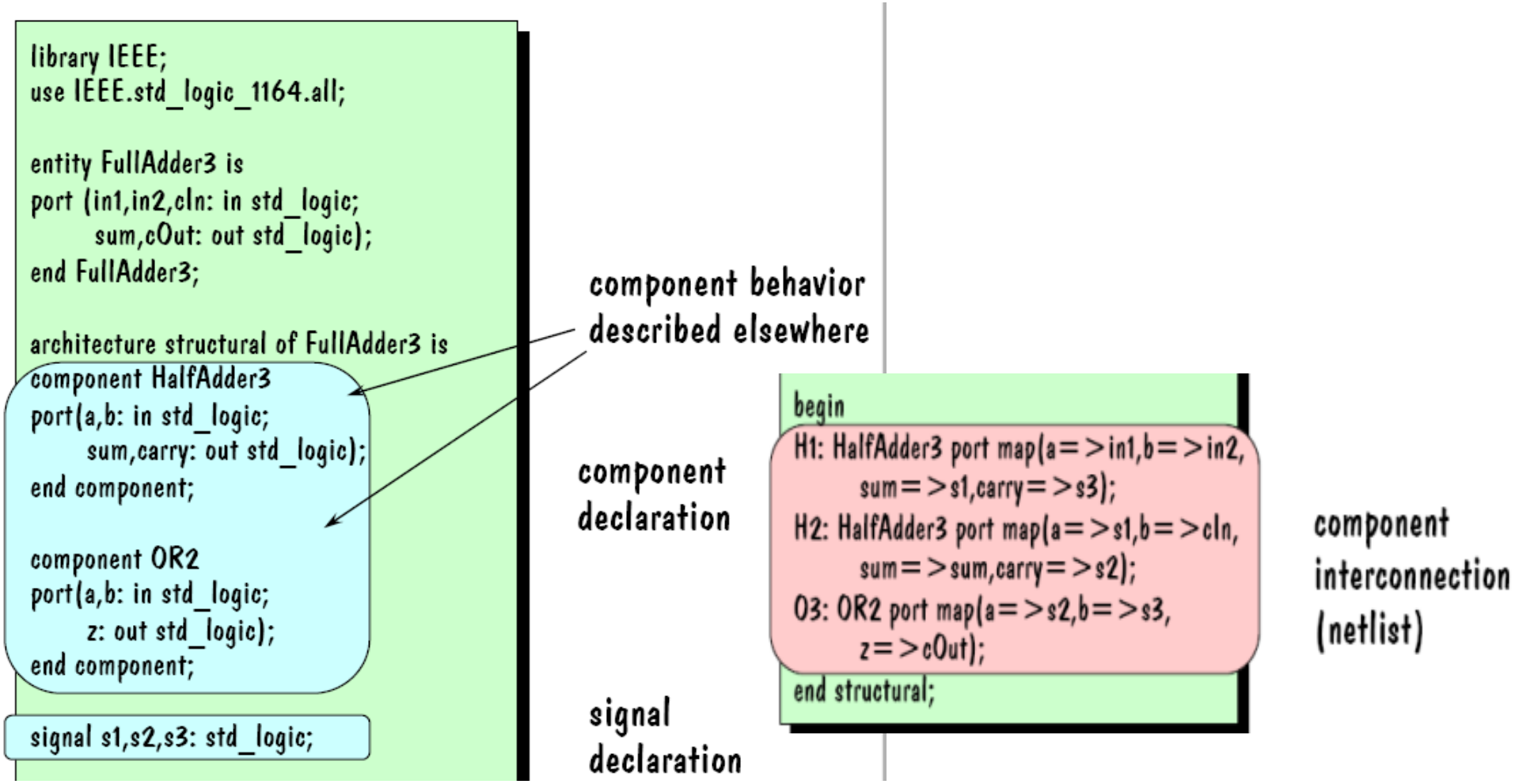
- Instead of giving an equation of the output
  - Give the structure of the module
  - a structural model of a system is described in terms of interconnection of its components
  - a structural model consists of 3 features:
    - component declaration
    - signal declaration
    - component interconnection

More complex design: Full Adder



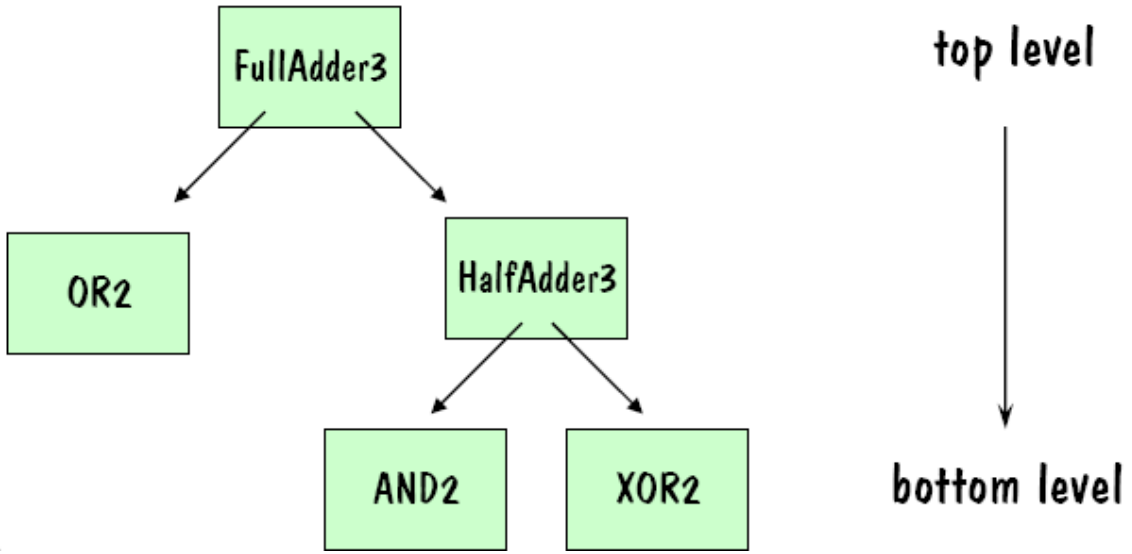
3 intermediate signals
S1, S2, S3 that have
To be declared

More complex design: Full Adder



Hierarchy

- Modern designs have several 10 millions of gates !!!
  - Structural models simply describe interconnections of gates or other components
- Structural models cannot describe all the forms of potential behavior (we cannot interconnect 10 mils gates together…)
- We do deep **hierarchy** to express different levels of detail
- Structural models are a way to manage large, complex designs at top level,
  - **where at intermediate levels we use much more dataflow description types**



Libraries

- Each user has his own **work library**
- Here we put all **successfully compiled** designs
- One user can reference
  - his own libraries,
  - or other people libraries,
  - or basic « cells libraries » (technology gates)
  - or IP libraries (DSP, memories, UAL components)

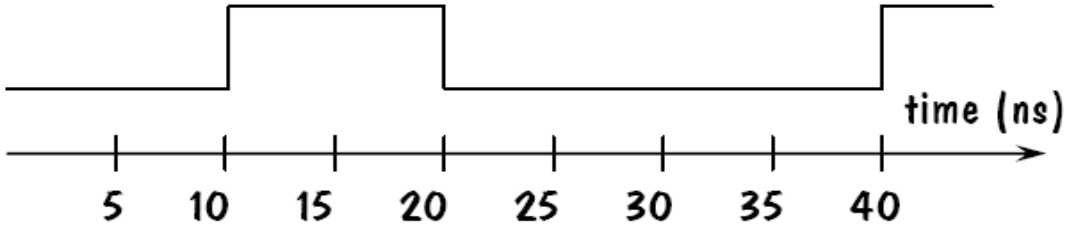
Some other relaxing stuff before going further

- **VHDL ne fait aucune différence entre majuscules et minuscules**
- **Suite de lettres, chiffres et underscore "_"**
- **Le premier caractère doit être une lettre**
  - **Tous les caractères sont significatifs**
  - **Corrects : NON_ET Bascule_JK NE555**
  - **Interdits : A#2 2A A$2 A__2**
- **Les commentaires**
  - **Ils sont liés à la ligne**
  - **Ils commencent par deux tirets "--" et finissent à la fin de la ligne**
  - **Ils sont nécessaires à la compréhension du modèle mais totalement ignorés par le compilateur**

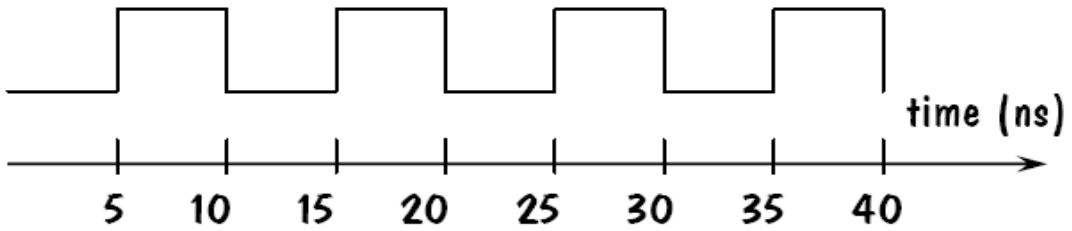
Signal Assignment

- simple signal assignments

```
sum <= (a xor b) after 5 ns, (a or b) after 10 ns,  (not a) after 15 ns;
sig <= '0', '1' after 10 ns, '0' after 20 ns, '1' after 40 ns;
```




```
clock <= '0', not(clock) after 5 ns;
```



Conditional Signal Assignment

- The right hand value is computed immediately but assigned at some points in the future using the after clause (after 5ns)

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

entity Mux4to1 is
   port (i0,i1,i2,i3: in std_logic_vector(7 downto 0);
         sel : in std_logic_vector(1 downto 0);
         z   : out std_logic_vector(7 downto 0));
end Mux4to1;

architecture dataflow of Mux4to1 is
begin
   z <= i0 after 5 ns when sel="00" else
        i1 after 5 ns when sel="01" else
        i2 after 5 ns when sel="10" else
        i3 after 5 ns when sel="11" else
        "00000000" after 5 ns;
end dataflow;
```

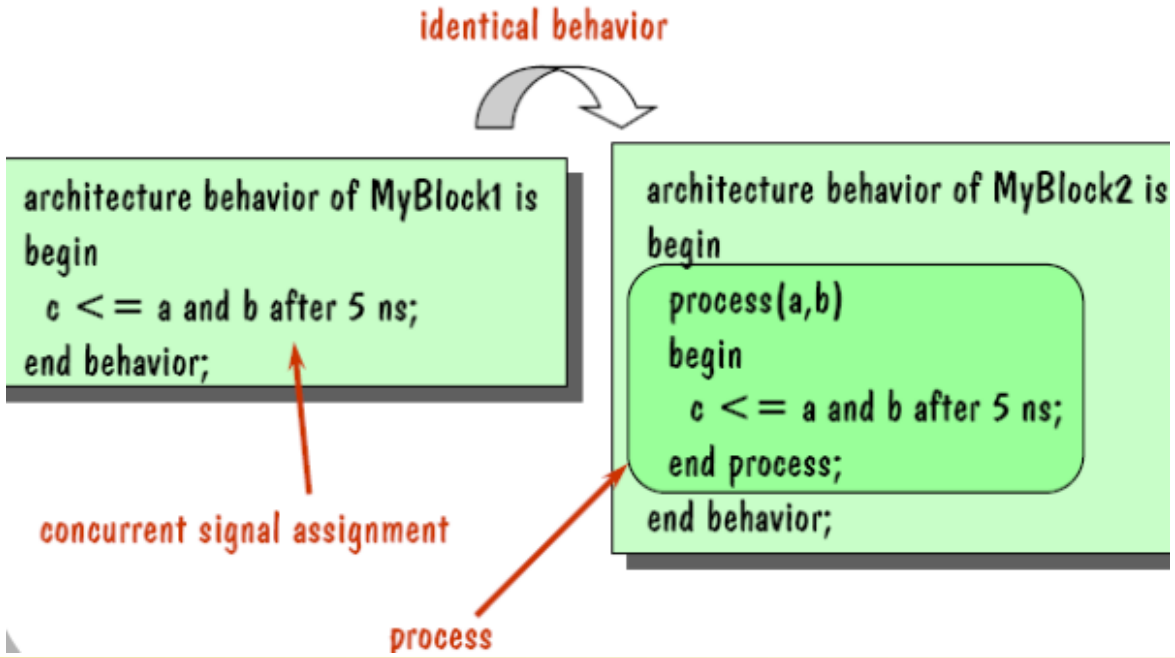
one single signal assignment

New Concept: **process**

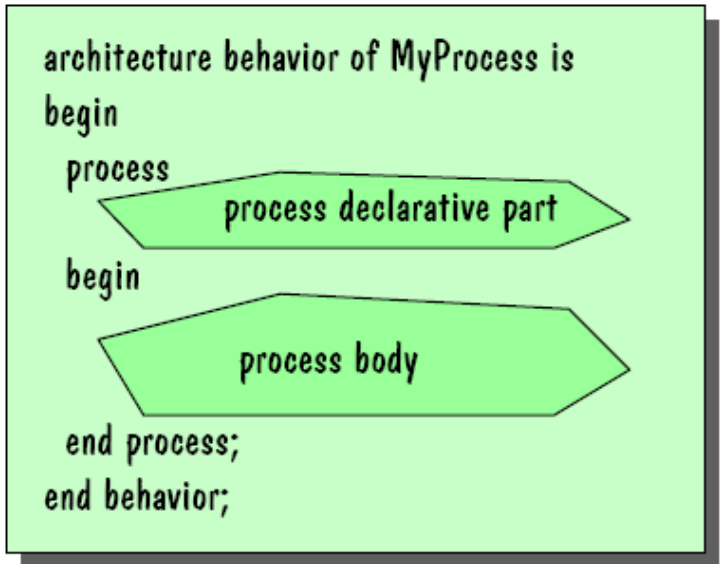
- A structure that is evaluated only when needed, not all the time
- The process construct enables the use of conventional programming language constructs (**if-then-else, loop, for, case, etc**).
- In contrast to concurrent signal assignment statements in the process are sequentially executed.
  - **Control flow within a process is strictly sequential.**
  - With respect to simulation time a process executes in zero time
  - Several **process** can be executed in parallel!

The Process Construct

- The execution of a process is initiated whenever an event occurs on any signal in the **sensitivity list**
- Processes execute concurrently with other processes and concurrent signal assignments.
- Concurrent signal assignments are in fact special cases of processes.



More on process



VHDL

- process sensitivity list
  begin statements;
  end process;
- wait on/until/for
  event; ← See later, in fact there are 2 types of process

- **Events are variable or signal changes**.
- Because…all real circuits are **event driven**.

Syntax of sequential constructions

```
if condition then ...
    {elsif autre_condition then ...
     else ...}
end if;

case expression is
     when valeur_1 => ...
     when valeur_2 => ...
     when others => ...
end case;
```

Exécution itérative

```
{label} loop ...
end loop {label};

{label} for indice in sous_type_énuméré
loop ...
end loop {label};

{label} while condition_booléenne
loop ...
end loop {label};
```

Arrêt de l'itération

next {label}; ou next {label} when COND;

Sortie de boucle

exit {label}; ou exit {label} when COND;

Sequential instructions

- If-then-else statement

```
if condition then sequential statement
[ elsif condition then sequential statement ]
[ else sequential statement ] end;
```

- case statement

```
case expression is
{when choices => sequential statements }
[ when others => sequential statements ]
end case;
```

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

entity HalfAdder is
   port (a,b: in std_logic;
         sum,carry: out std_logic);
end HalfAdder;

architecture behavioral of HalfAdder is
begin
If_Process: process(a,b)
 begin
 if (a = b) then
  sum <= '0' after 5 ns;
 else
  sum <= (a or b) after 5 ns;
 end if;
end process;
```

```
Case_Process: process(a,b)
 begin
 case a is
  when '0' => carry <= a after 5 ns;
  when '1' => carry <= b after 5 ns;
  when others => carry <= 'x' after 5 ns;
 end case;
end process;

end behavioral;
```

Multiplexer

```
entity Multiplexer4to1 is
   port (sel: in std_logic_vector (1 downto 0);
      a,b,c,d: in std_logic_vector (15 downto 0);
      z:out std_logic_vector (15 downto0));
end Multiplexer4to1;

architecture DemoExample of Multiplexer4to1 is
begin
   process (a,b,c,d,sel)
   begin
      case sel is
         when ("00") => z <= a;
         when ("01") => z <= b;
         when ("10") => z <= c;
         when ("11") => z <= d;
         when others => z<="-----
      end case;
   end process;
end DemoExample;
```

- VHDL has a special assignment problem inside a process description
  - Whenever if-then-else and case structure are not completely described => generates a memory element

4 to 1 multiplexer, with no inferred memory
All case branches should be described

Combinational circuits with PROCESS

```
process ( A, B, SEL )
begin
  if (SEL = '1') then
    OUT <= A;
  else
    OUT <= B;
  end if;
end process;
```

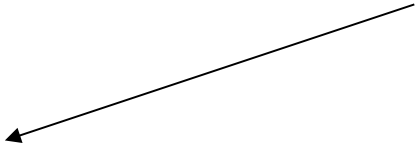
? What is happening if SEL is missing in the sensitivity list?
? What is the circuit that may be generated?

- **Do not forget important signals in the sensitivity list!**
- If you forget signal assignment, synthesis result can be really different than what you are expected.
- Or simulations scenario can be different

Wrong and good description of a combinational multiplexer!

- architecture WRONG of MUX is
  begin
    process (A, B, SEL)
    begin
      if SEL = `1` then
        Z <= A;
      end if;
    end process;
  end WRONG;

? What is the circuit that may be generated?

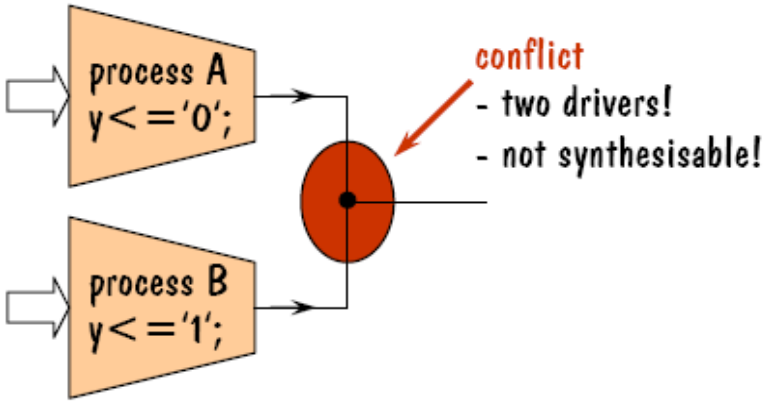


```
architecture OK_1 of MUX is
  begin
    process (A, B, SEL)
    begin
      Z <= B;
      if SEL = `1` then
        Z <= A;
      end if;
    end process;
  end OK_1;
```

```
architecture OK_2 of MUX is
  begin
    process (A, B, SEL)
    begin
      if SEL = `1` then
        Z <= A;
      else
        Z <= B;
      end if;
    end process;
  end OK_2;
```

More on process

- Never assign a value to a signal in different processes (multiple drives).



- Upon initialization all processes are executed at once.
- Thereafter processes are executed in a data-driven manner:
  - activated by events on signal list of the process or
  - by waiting on occurrences of specific events using wait statements

Syntax « process »

- **Instruction processus (à utiliser pour la synthèse): SANS WAIT!**

```
{label :} process (liste_des_signaux_surveillés)
        {partie déclarative}
begin
        {instructions séquentielles}
end process {label} ;
```

- **Le processus équivalent (à utiliser pour la simulation)**

```
{label :} process
        {partie déclarative}
begin
        {instructions séquentielles}
        wait on liste_des_signaux_surveillés;
end process {label} ;
```

Fonctionnement : exécution itérative

A l'initialisation, le processus s'exécute jusqu'à la première instruction wait rencontrée.

Lorsque l'exécution arrive sur end process, il met à jours les signaux, il se réexécute à partir de la première instruction suivant begin.

More on process

- A more general way to specify when a process executes is the **wait statement**.
- Wait statements explicitly specify the conditions under which a process may resume execution after being suspended.
- With wait statements a process can be suspended at multiple points
- **Sometimes, equivalent with the sensitivity list!**

```
wait for time expression;
        example:   wait for 20 ns;

wait on signal;
        example:   wait on clk,reset,status;

wait until condition;
        example:   wait until (a = '1');

wait;
```

Syntax

- Syntaxe générale

  wait {on liste_de_signaux} {until condition_booléenne}
  {for un_temps_donné};

- Signification

  Le processus (ou sous-programme) où se trouve l'instruction est suspendu pour au plus un temps donné (for).

  A chaque événement sur un élément de la liste des signaux (on), la condition (until) est évaluée et le processus (ou sous-programme) reprend son cours si sa valeur est vraie.

- Valeurs par défaut

  liste_des_signaux => ceux de la condition
  condition_booléenne => true
  temps (type TIME) => fin des temps ...

- Exemples

  wait;
  wait on **A, B**;
  wait on **A, B** until **TRUC(Z)**;
  wait on **A, B** until **TRUC(Z)** for **7 ns**;
  wait until **CK='1'**;

Conclusion on combinational circuits

- Use as much as possible process constructions (easy to simulate)
- BUT
  - Make sure all input signals are in the sensitivity list
    - Or in the wait statement, and only one wait statement per process!
    - Do not use process with no sensitivity list or with no wait!
  - Make sure all outputs are assigned in all branches of the program code

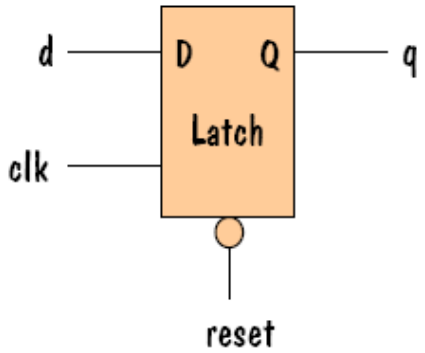
But we have also sequential circuits

- Let's see how we model a FF
  - What type of reset?
  - What is clk'event?
  - Where is the last else?

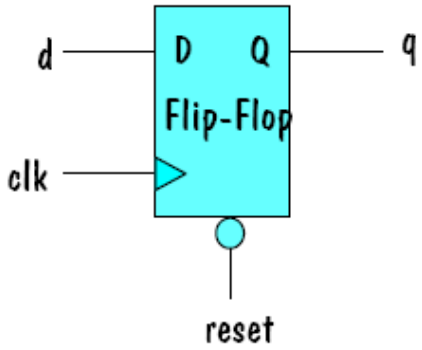
```
library IEEE;
use IEEE.std_logic_1164.all;

entity Dff2 is
   port (d,clk,rst: in std_logic;
         q,qBar: out std_logic);
end Dff2;

architecture behavioral of Dff2 is
begin
 process(clk,rst);
 begin
    if (rst='0') then
      q    <= '0' after 1 ns;
      qBar<= '1' after 1 ns;
    elsif (clk'event and clk='1') then
      q    <=d after 1 ns;
      qBar<=not d after 1 ns;
   end if;
 end process;
end behavioral;
```

Latch or FF?

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

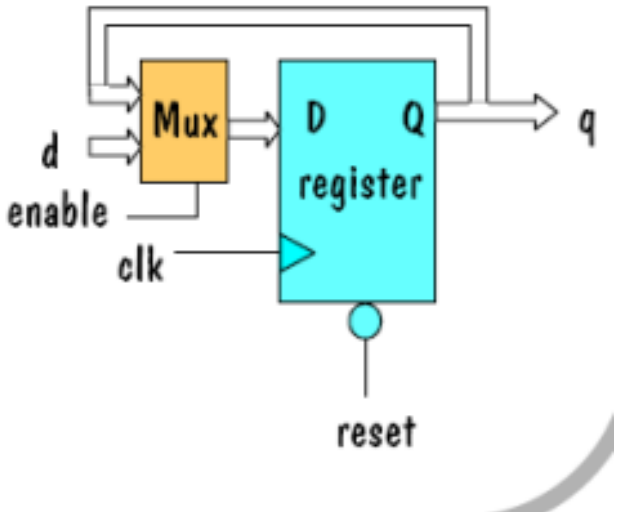



```
process(clk,reset)
begin
  if (reset = '0') then
    q <= '0';
  elsif (clk'event and clk='1') then
    q <= d;
  end if;
end process;
```



Some Synthesis

```
process(clk,reset)
begin
  if (reset = '0') then
     q <= "00000000";
  elsif rising_edge(clk) then
    if (enable = '1') then
     q <= d;
    end if;
  end if;
end process;
```



Other Sequential Circuits

- Counter

Testbench

- My first Testbench for MUX

Testbench

- A quoi sert un testbench?
- Testbench pour le MUX

```
entity Mux_TB is -- empty entity
end Mux_TB;
architecture TB of Mux_TB is -- initialize the declared signals
signal T_I3, T_I2, T_I1, T_I0: std_logic_vector(2 downto 0):="000";
signal T_O: std_logic_vector(2 downto 0);
signal T_S: std_logic_vector(1 downto 0);
component Mux port( I3, I2, I1, I0: in std_logic_vector(2 downto 0);
                    S: in std_logic_vector(1 downto 0);
                    O: out std_logic_vector(2 downto 0) );
end component;
begin
U_Mux: Mux port map (T_I3, T_I2, T_I1, T_I0, T_S, T_O );
process
    variable err_cnt: integer :=0;
    begin
      T_I3 <= "001"; T_I2 <= "010"; T_I1 <= "101"; T_I0 <= "111";
       wait for 10 ns; -- case select eqaul "00"
      T_S <= "00";
      wait for 1 ns;
      assert (T_O="111") report "Error Case 0" severity warning;
                    if (T_O/="111") then err_cnt := err_cnt+1;
                    end if;
end process;
end TB;
```

Synthesis from VHDL
How to write an FSM from A to Z

Katell Morin-Allory
katell.morin-allory@imag.fr

Septembre 2016

What are the steps to get to the phyisical implementaiton of a circuit?

1. Specification

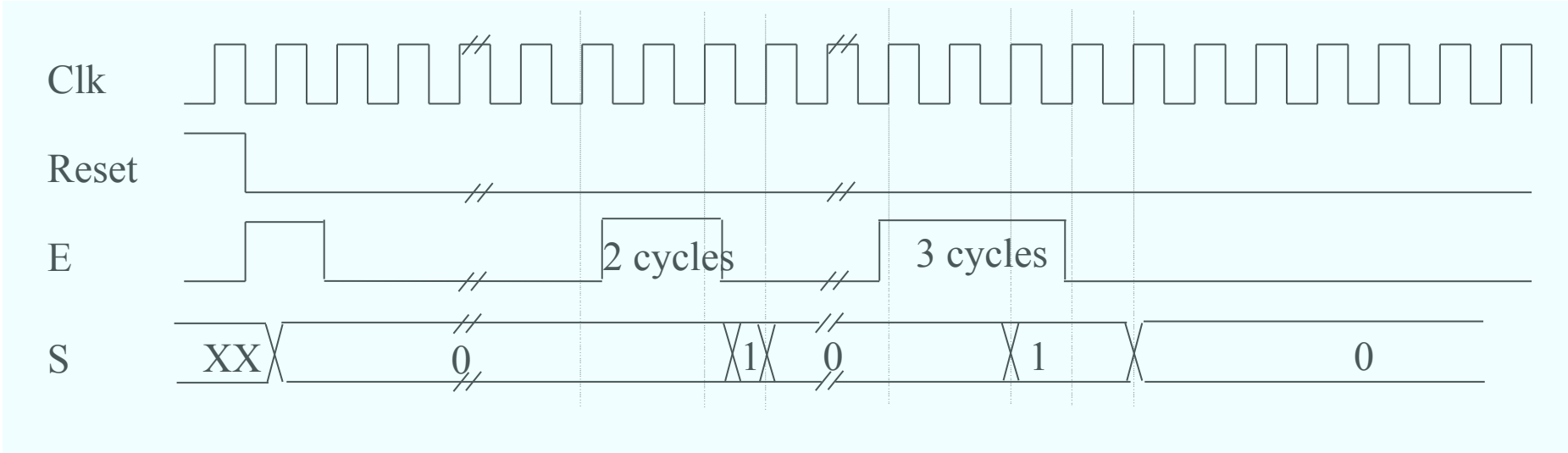
2. Modeling

3. Synthesis

1. Specification

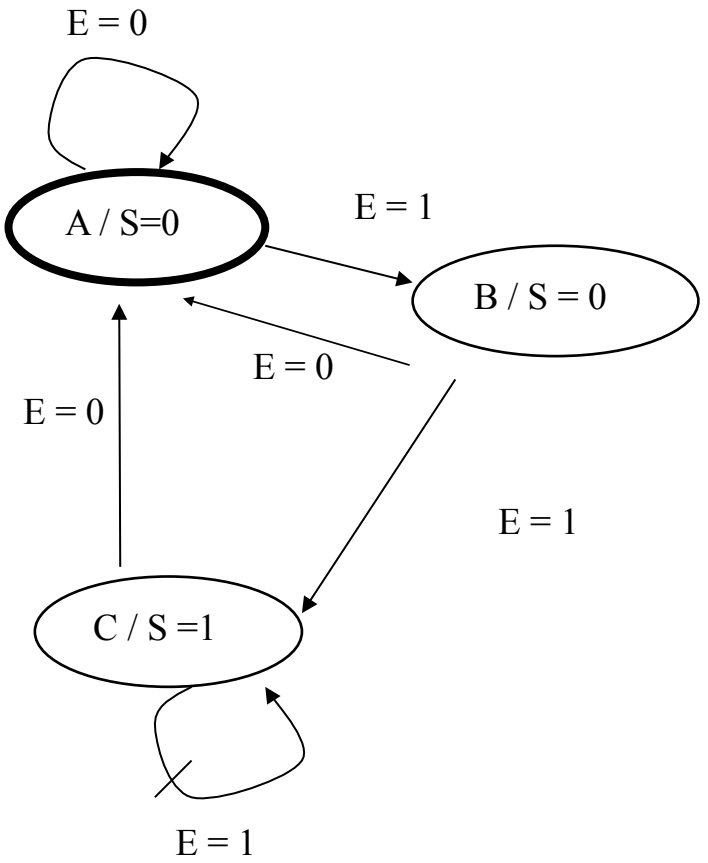
Specification (1)

- ✓ **We would like to realise an electronic system with only one input E and only one output S so that S=1 whenever E =1 for at least two succesive clock cycles**
- ✓ **Start by drawing waves and truth table**



Specification (2)

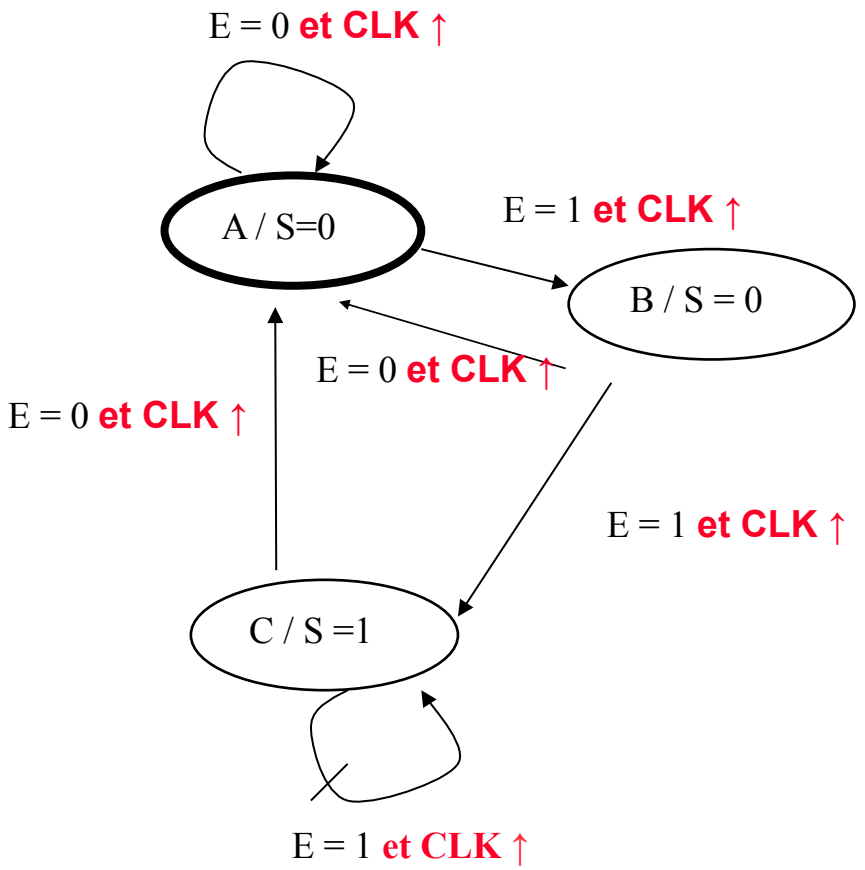
- ✓ **More formal specification form as a state transition diagram**
- ✓ **Establish with care**
  - ✓ **Condition of moving from one state to the other.**
  - ✓ **Values of the output for each state or transition**
  - ✓ **Initial state**



FSM Moore type!

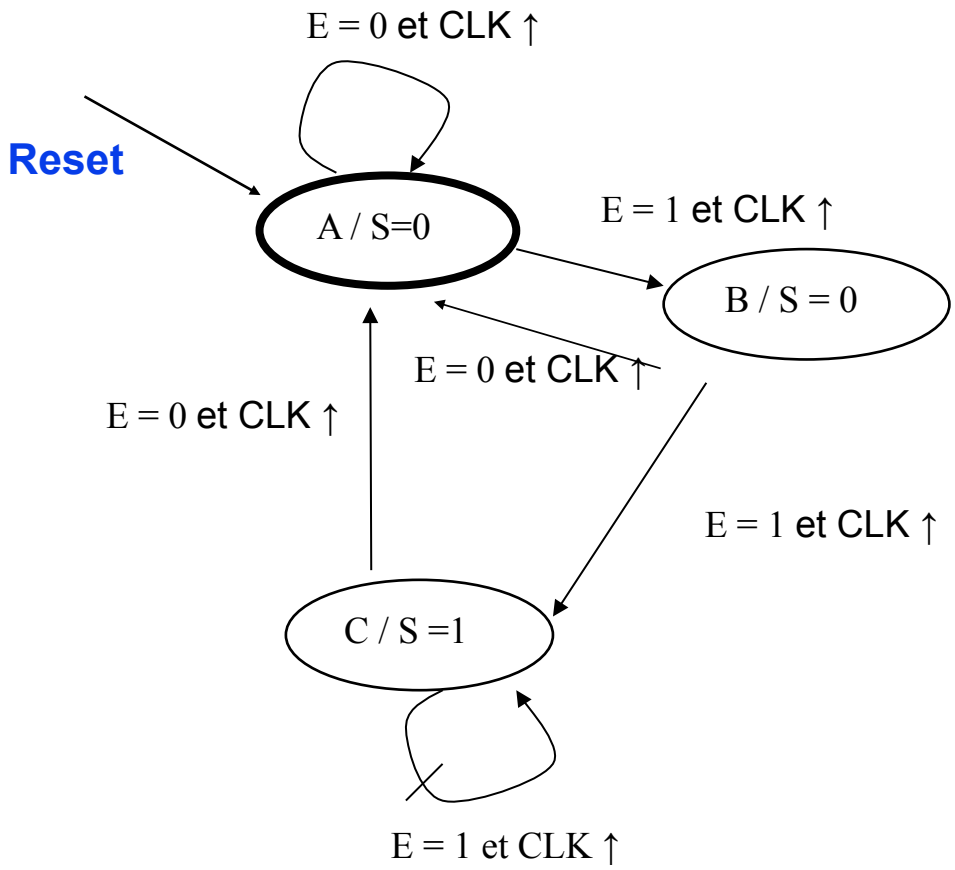
Specification (3)

- ✓ **Introduce synchronization mechanism**
- ✓ **Who is ensure the sequential functionality of the circuit (moving from one state to the other?)**
- ➢ **State diagram is modified to take into account the CLK**



Specification (3)

- **Introduction of RESET !**
- **Asynchronous or Synchronous (?)**
- **Where the circuit goes when reset arrives?**
- **Here A state!**



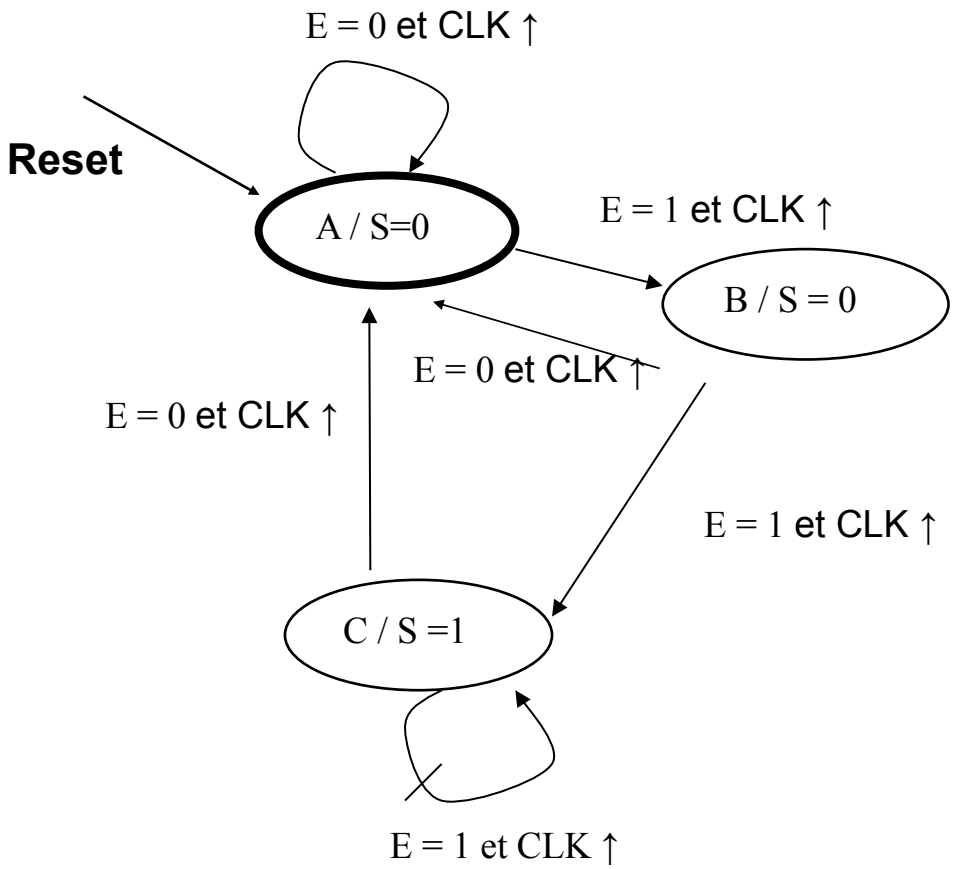
2. VHDL Modeling

Modeling in VHDL

- ✓ Several coding styles in VHDL
- ✓ Some of them are only executables (for simulation only) some of them can be used for the synthesis

Modeling (1) : behavioural
States are explicit

States are explicit



« reset »
Introduce later !

```
Entity pulse_gen is
  Port (  Clk, Reset, E: in  std_logic;
          S :  out std_logic)
End pulse_gen ;
Architecture behavior1 of pulse_gen is
Type State is (A, B, C) ;
Signal Current_State : State ;
Single_Process : process (CLK)
Begin
  If (CLK='1' and CLK'EVENT) then
     Case Current_State is
        when A =>   If E = '0' then
                        Current_State <= A ;
                    Else  Current_State <= B ;
                    End if ;
        when B=>    If  E = '0' then
                        Current_state <= A;
                    Else  Current_State <=C;
                    End if;
        when C =>     if E='0' then
                        Current_State <= A ;
                    Else  Current_state <= C;
                    End if;
      end case;
end if;
End Single Process;
S <= '1' when Current_State=C else '0';
End behaviour1;
```

Modeling (2) : …

States are explicit

Reset Asynchronous !

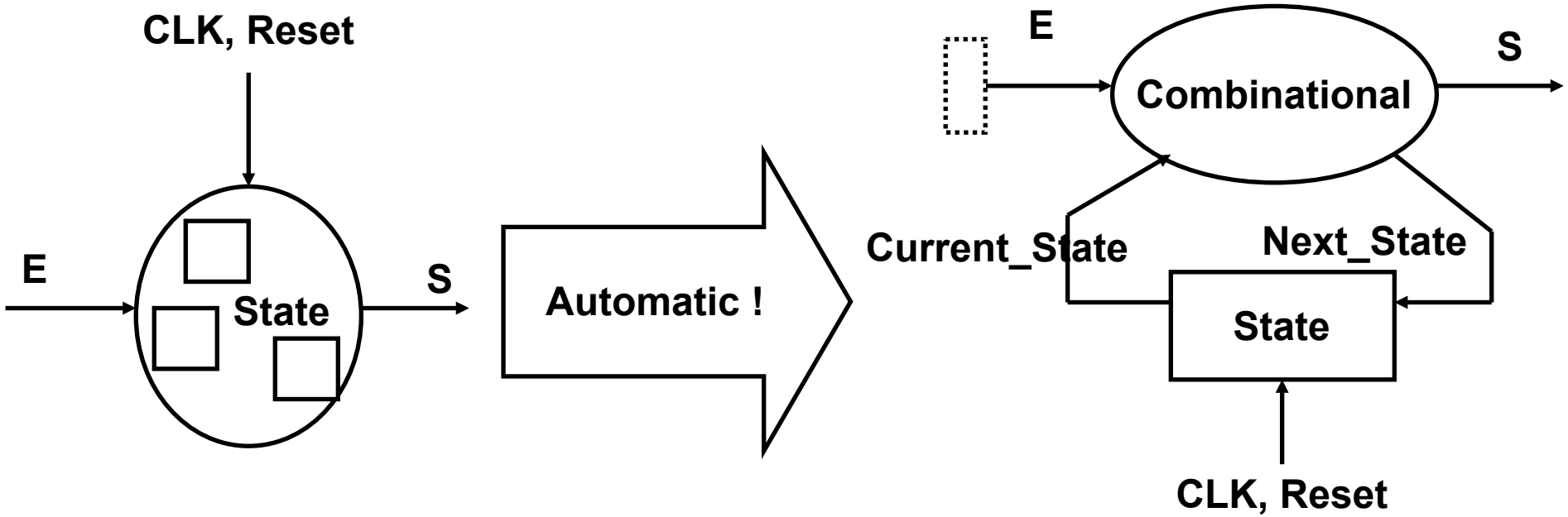
```
Single_Process : process (CLK, Reset)
Begin
    If (RESET='1') then
      Current_State <= A ;

    Elsif (CLK='1' and CLK'EVENT) then

      Case Current_State is
          when A =>   if E='0' then Current_State <=A;
                      …
      end case;
   End if;
End Process Single_Process ;
```

Modeling for Sythesis

- ✓ VHDL coding is slightly different for synthesis purposes
- ✓ **We have to split the FSM into 2 parts:**
  - **Combinational**
    - ⇨ **that depends only on the inputs and current state**
    - ⇨ **Compute the output and the next state**
  - **Sequential**
    - ⇨ **React with the clock**
    - ⇨ **Process the reset**
    - ⇨ **And transform the current state into the next state xhenever the clock…**

VHDL Modeling : RTL

- ✓ **In the VHDL split the code into memorization part and the combinational part !**

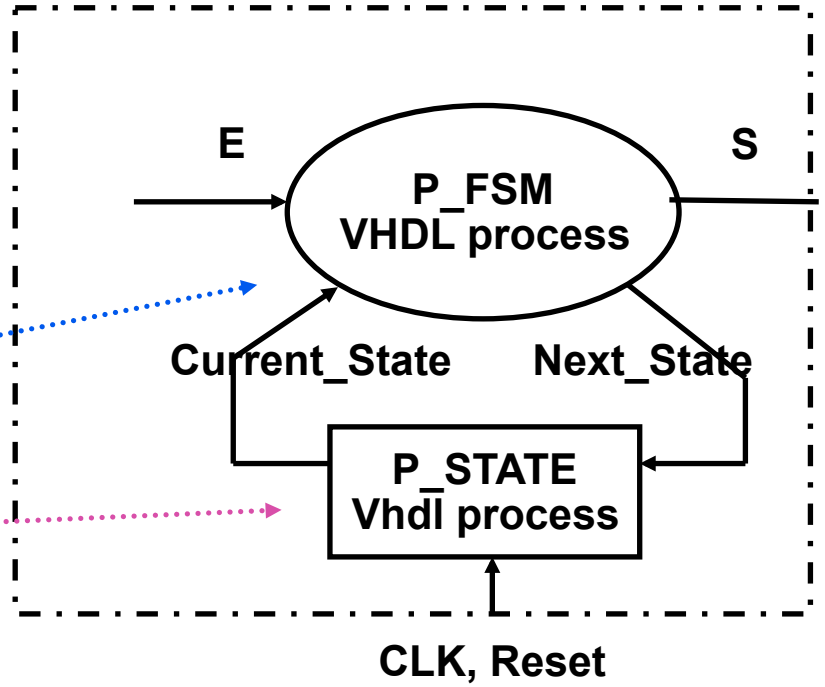


Unique Process => 2 process

RTL Modeling

✓ **Architecture !**

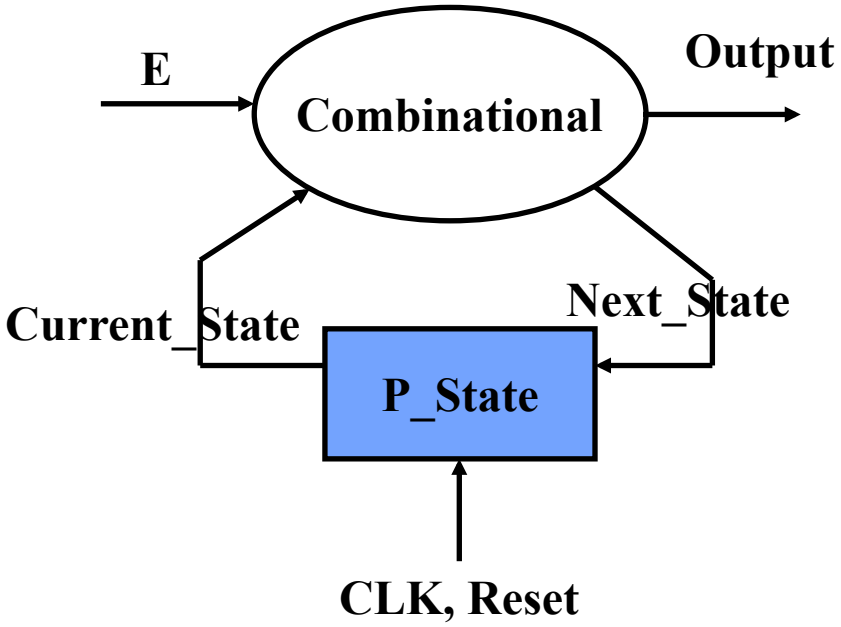
```
Entity pulse_gen is
  Port ( Clk, reset, E  :  in  std_logic;
         S :       out std_logic )
End pulse_gen ;

Architecture structural_2_process of pulse_gen is
Type State : is (A….) ;
Signal Current_State, Next_State : State ;

Begin

P_FSM : process (Current_State, E)
Begin ….. End

P_STATE : process (reset, CLK)
Begin ….. End

S<= ‘1’ when Curernt_State = C else ‘0’;

End structural_2_process ;
```



RTL Modeling

1. Memorizing process !

```
P_STATE : process (CLK, reset)
begin
    if (Reset = '1') then
        Current_State <= A ;
    elsif (CLK = '1' and CLK'event ) then
        Current_State <= Next_State ;
    end if ;
end process P_STATE ;
```

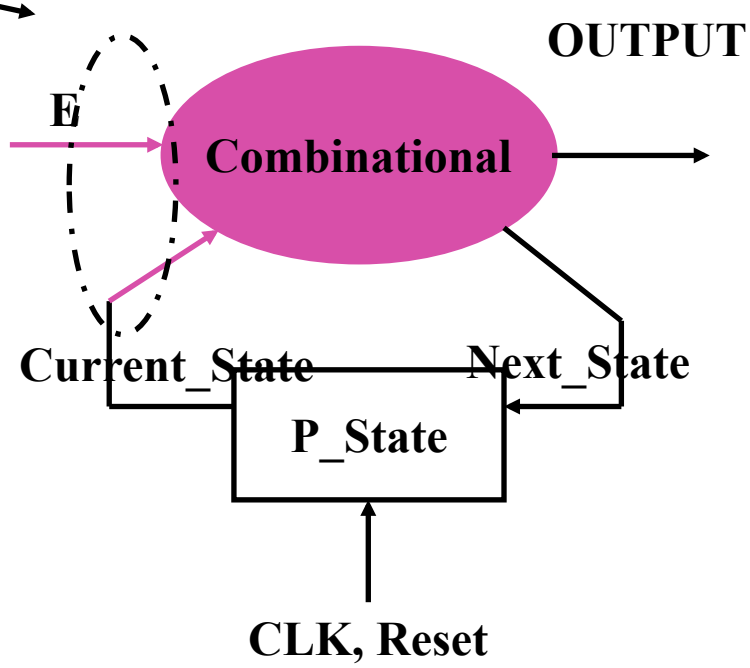


All tools can synthesize this!

RTL Modeling

✓ 2. Combinational process ! Computes only S and next state from inputs and curent state

```
P_FSM : process (Current_State, E)
begin
Case Current_State is

when A => if E='0' then Next_State <=A;
                   else Next_State <=B;
when B => if E='0' then Next_State <=A;
                   else Next_State <=C;
when C => if E='0' then Next_State <=A;
                   else Next_State <=C;

end case ;
end process P_FSM ;

S<= '1' when Curernt_State = C else '0';

End structural_2_process;
```



RTL Modeling

✓ **2. Combinational process, without memory**

```
P_FSM : process (CS, E)
Case Current_State is

when A => if E='0' then Next_State <=A;
                   else Next_State <=B;
when B => if E='0' then Next_State <=A;
                   else Next_State <=C;
when C => if E='0' then Next_State <=A;
                   else Next_State <=C;

end case ;
end process P_FSM ;

S<= '1' when Curernt_State = C else '0';

End structural_2_process;
```

Equivalent with a concurrent instruction!

```
Next_state <=

A when ( (Current_state = A and E = '0') or
(Current_State = B and E='0') or
(Current_State=C and E='0'))

else
        B when (Current_state = A and E = '1')

else
        C when ((Current_state = B and E = '1')
or (Current_state = Cand E = '1') )

else
        "--" ;
```

3. Synthesis

✓How to choose state coding

✓Binary :

values of state will be coded as follows :

- A = "00"
- B = "01"
- C = " 10"

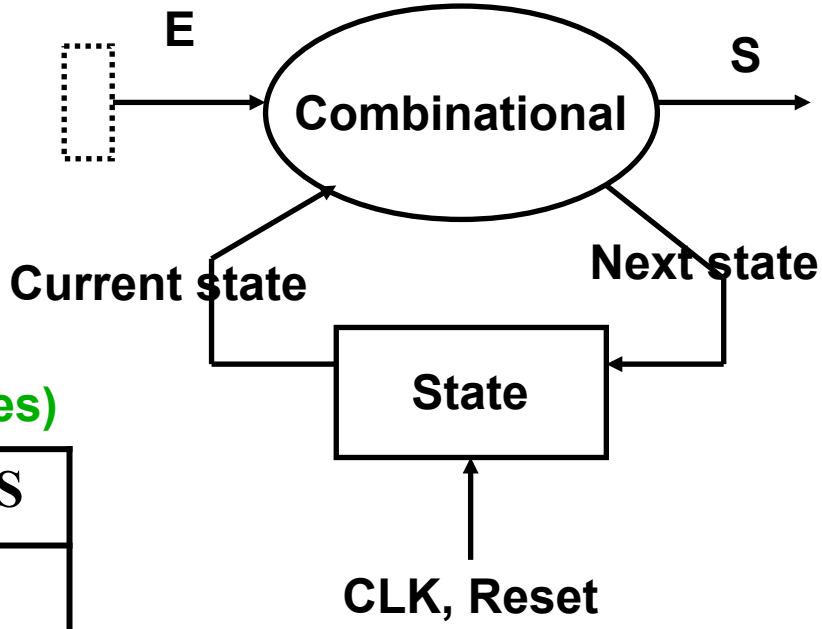
So Current_State and Next_State, will be on 2 bits represented as :

(CS(1), CS(0))

(NS(1), NS(0))

State transition table

Current state	Next State		output S
	E=0	E=1	
A	A	B	0
B	A	C	0
C	A	C	1



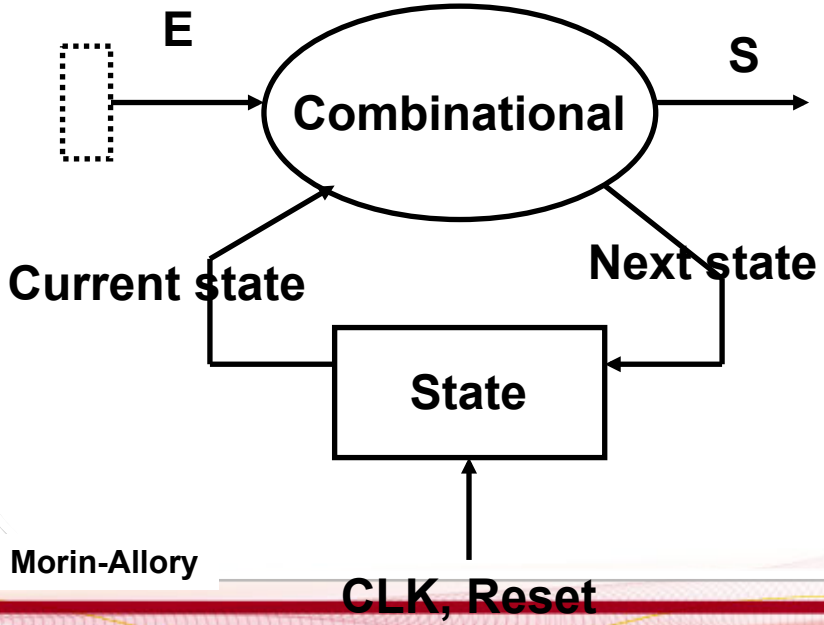
What encoding style to choose (2 variables)

Current state	Next State		Output S
	E=0	E=1	
A 00	00	01	0
B 01	00	10	0
C 10	00	10	1
11	xx	xx	x

This code is not used but should be specified with xxx!

Synthesis of state and output function

Current state y2 y1	Next State		Output S
	E=0 Y2Y1	E=1 Y2Y1	
A 00	00	01	0
B 01	00	10	0
C 10	00	10	1
11	xx	xx	x



Minimization by Karnaugh

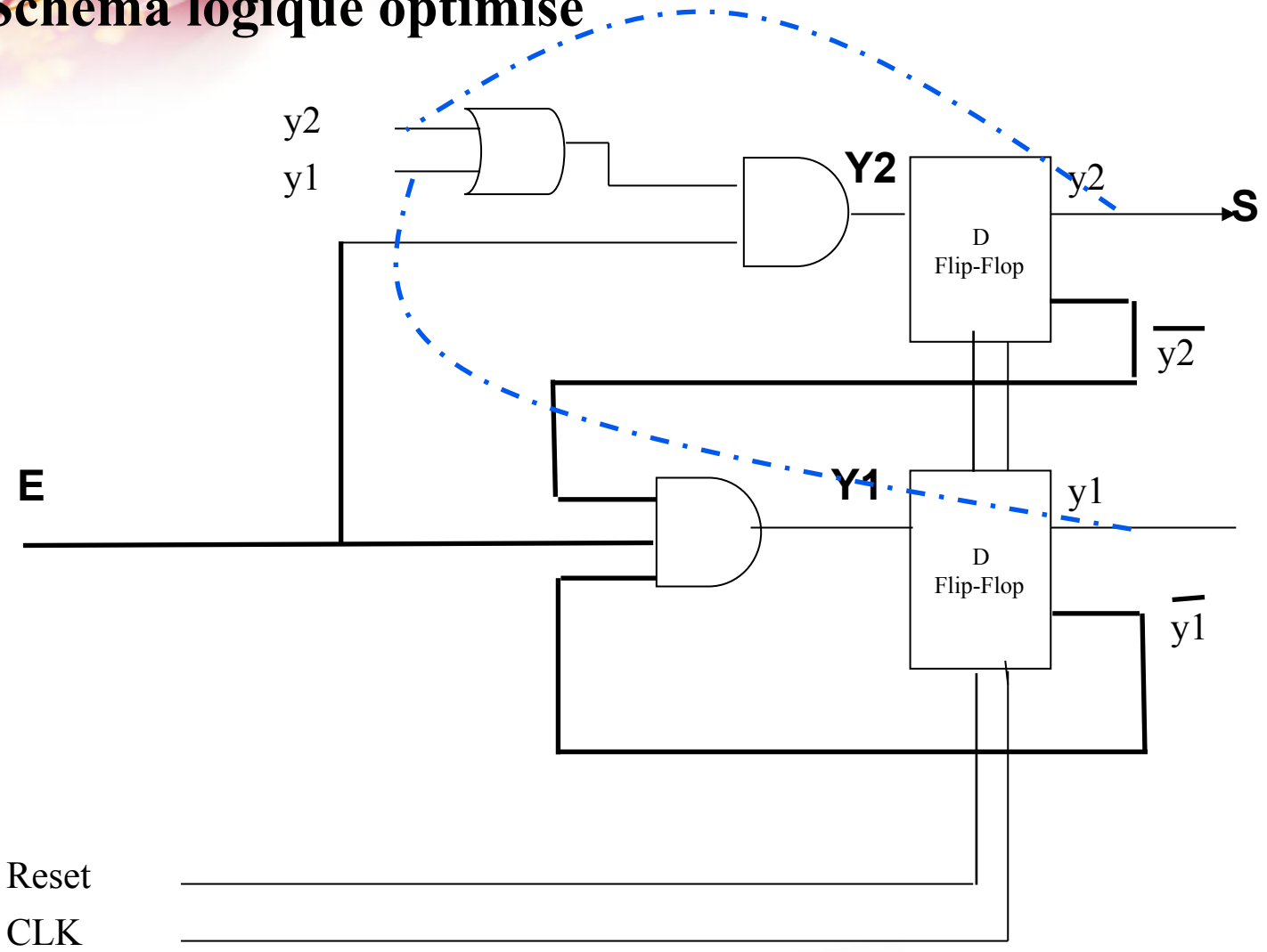
$$Y1 = E\ \overline{y1}\ \overline{y2}$$

$$Y2 = E\ (y1 + y2)$$

$$S = y2$$

Implementation

✓ **Schéma logique optimisé**



Synthesis

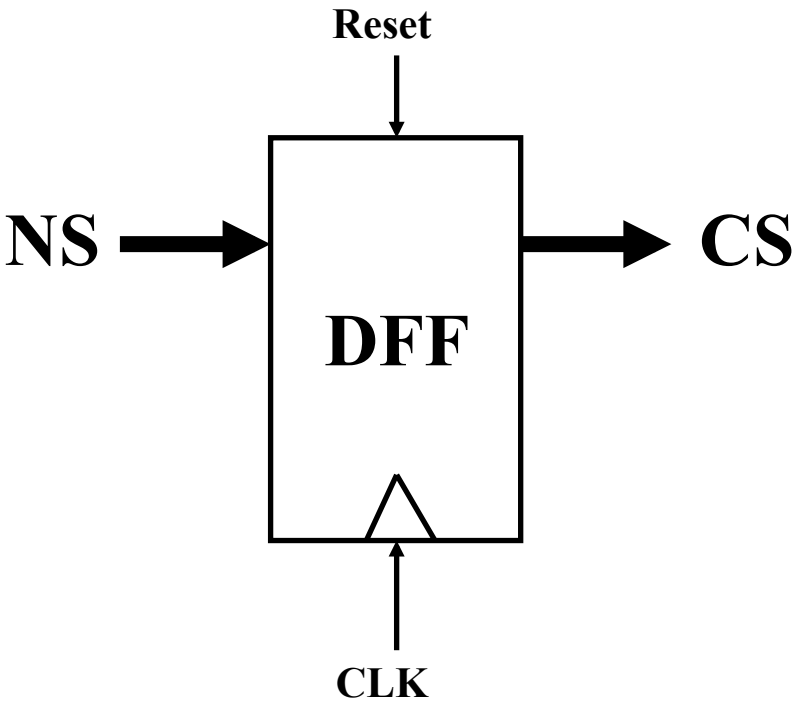
✓ Explicit assignement of states (optimized)

```
Entity pulse_gen is
  Port ( Clk, reset, E  :  in  std_logic;
         S :       out std_logic )
End pulse_gen ;

Architecture structural_2_process of pulse_gen is

Signal Current_State, Next_State : STD_logic_vector (1 downto 0);
CONSTANT A: std_logic_vector (1 downto 0): = “00”;
CONSTANT B: std_logic_vector (1 downto 0): = “01”;
CONSTANT C: std_logic_vector (1 downto 0): = “10”;

Begin

P_FSM : process (Current_State, E) --identique
Begin ….. End
P_STATE : process (reset, CLK) -- identique
Begin ….. End
S<= ‘1’ when Curernt_State = C else ‘0’;
End structural_2_process ;
```

Synthesis

Memory process !

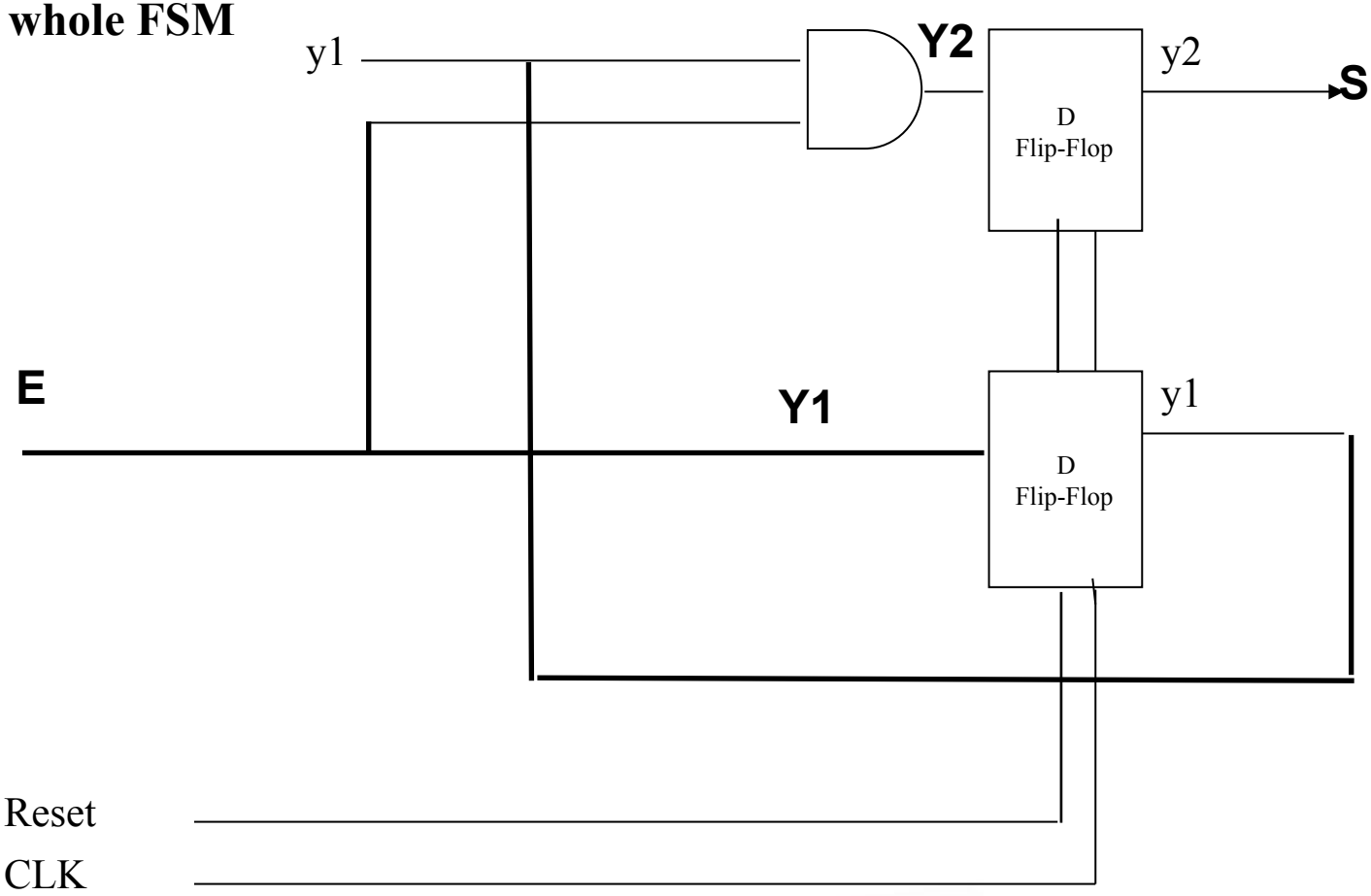
```
P_STATE : process (CLK, reset)
begin
    if (Reset =‘ 1’ ) then
        Current_State <= A ;
        elsif (CLK = ‘1’ and CLK’event) then
            Current_State <= Next_State ;
        end if ;
end process P_STATE ;
```



Bloc DFF is generic

Synthesis (2 bis)

The whole FSM



✓**Other choice for state encoding**

✓**ONE HOT : or 1 out of n (n number of states)**

- A = "001"
- B = "010"
- C = " 100"

Current_State et Next_State, will be coded on 3 bits as follows :

(CS(2), CS(1), CS(0))

(CS(2), NS(1), NS(0))

etc

Modélisation et Synthèse des Systèmes Matériels

TD n° 1
Prise en main du langage et Simulation

1. Prise en main du langage VHDL

Le but de ce premier TD est l'utilisation du langage pour
- définir une unité arithmétique et logique d'un logique d'un petit microprocesseur 8 bits
- définir le registre d'accumulation associé en sortie de l'ALU

La table de vérité ci-dessous donne les fonctions supportées par cette ALU.

Decoder			Inputs			Function	Description
S2	**S1**	**S0**	**Cin**	**A**	**B**	**Outputs F and Cout**	
0	0	0	0	A	B	F=A	Transfer A
0	0	0	1	A	B	F=A+1	Increment A
0	0	1	0	A	B	F=A+B	Add B to A
0	0	1	1	A	B	F=A+B+1	Add B to A +1
0	1	0	0	A	B	F=A-1	Decrement A
0	1	0	1	A	B	F=A-B=A+(notB) +1	Sub B from A
1	0	0	X	A	B	F=A or B	Or
1	0	1	X	A	B	F=A xor B	Xor
1	1	0	X	A	B	F=A and B	And
1	1	1	X	A	B	F=notA	Not

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


entity reg_alu is
  port (
    clk: in std_logic;
    reset_n: in std_logic;
    a: in std_logic_vector(7 downto 0);
    b: in std_logic_vector(7 downto 0);
    Cin: in std_logic;
    s2: in std_logic;
    s1: in std_logic;
    s0: in std_logic;
    f: out std_logic_vector(7 downto 0);
    cout: out std_logic
    );
end reg_alu;


architecture rtl of reg_alu is

component Full_adder_8
    port (
      Entree1    : in  std_logic_vector(7 downto 0);
      Entree2    : in  std_logic_vector(7 downto 0);
      Retenue_in : in  std_logic;
      S          : out std_logic_vector(7 downto 0);
      R          : out std_logic);
  end component;
```

```
Signal combo_f : std_logic_vector(7 downto 0);
Signal combo_cout : std_logic;
….
       process(a, b, s2, s1, s0, Cin)
….
       process(reset_n, clk)
….
end rtl;
```

- Compléter le code.
- Ecrire le fichier de configuration pour le composant Full_adder_8.
- Ecrire l'entité et l'architecture de Full_adder_8.

2. Etude et simulation d'une description en VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

Entity compteur is
        Generic(P : positive ; N : natural) ;
        Port(           CLK :   in std_ulogic ;
                        RESET: in std_ulogic ;
                        S :     out unsigned(P-1 downto
0)) ;
End compteur ;

Architecture var_arch of compteur is
Begin
        P_compteur : process
                variable C : unsigned(S'range) ;
        Begin
                Wait until CLK'event and CLK = '1' ;
                C := C + 1 ;
                If (C = N - 1)  or (RESET = '1') then
                        C := to_unsigned(0, C'length) ;
                End if ;
                S <= C;
        End process P_compteur ;
End var_arch ;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

Entity compteur is
        Generic(P : positive ; N : natural) ;
        Port(           CLK :   in std_ulogic ;
                        RESET: in std_ulogic ;
                        S :     out unsigned(P-1 downto
0)) ;
End compteur ;

Architecture sig_arch of compteur is
        Signal C : unsigned(S'range) ;
Begin
        S <= C;
        P_compteur : process
        Begin
                Wait until CLK'event and CLK = '1' ;
                C <= C + 1 ;
                If (C = N - 1) or (RESET = '1') then
                        C <= to_unsigned(0, C'length) ;
                End if ;
        End process P_compteur ;
End sig_arch ;
```

- Comparer les 2 architectures de l'entité compteur. Quelles sont les différences ?
- Compléter le code du "bench" ci-dessous.
- Effectuer une simulation "à la main " en utilisant le "bench".

```
library IEEE ;
use IEEE.std_logic_1164.ALL ;
use IEEE.NUMERIC_STD.ALL;

entity bench is
end bench;

architecture A of bench is

component compteur
generic(P: positive; N: natural);
port(           CLK :    in std_ulogic ;
                RESET: in std_ulogic ;
                S :      out unsigned(P-1 downto 0)) ;
end  component;
```

```
constant P_val: positive := 4;
constant N_val: natural := 10;

signal Clk          : STD_ULOGIC := '0' ;
signal Reset        : STD_ULOGIC ;
signal S : unsigned(P_val - 1 downto 0 ) ;

Begin

…

-- Generation de l'horloge
clk <= not(clk) after 250 ns;

-- Initialisation par "reset"
Reset <=        '0',
                '1' after 100 ns,
                '0' after 600 ns ;
end;
```

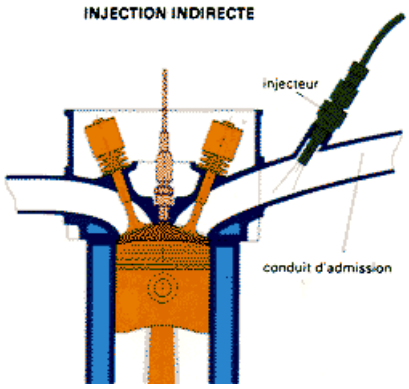
- Ecrire la configuration associée à ce "bench".

Modélisation et Synthèse des Systèmes Matériels

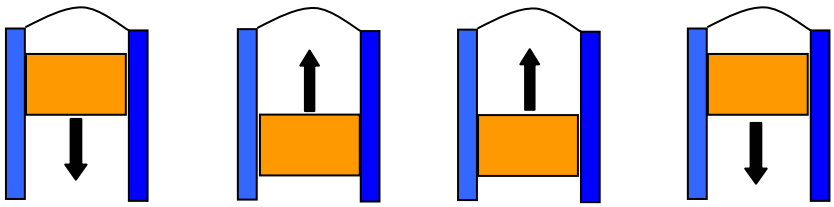
TD n° 2
Spécification et Synthèse

1. Spécification et modélisation d'un "Engine Control Unit" simplifié

Il s'agit de spécifier et de modéliser le circuit numérique (ECU) qui est chargé de la commande de l'injection d'un moteur à explosions. Dans un but de simplification, nous ne considérerons pas tous les paramètres utilisés réellement mais uniquement l'information issue de l'accélérateur et d'un capteur de position qui indique la demi-rotation du vilebrequin.



Notre étude considère l'injection indirecte monopoint pour un moteur 4 cylindres à explosion commandée (L'injection s'effectue dans la tubulure d'admission commune aux quatre cylindres).
Le fonctionnement du moteur est donné sur la figure ci-dessous. Chaque cylindre suit le cycle suivant : admission du mélange, compression, explosion et échappement. Cela signifie que le piston a fait 2 aller-retour, soit 2 tours. Notre moteur possède 4 cylindres. L'injection du mélange se fait lors de la phase d'admission par le biais d'un injecteur qui se comporte comme une électrovanne en mode ON/OFF. La quantité de mélange injectée est obtenue en faisant varier la durée d'ouverture de l'électrovanne. Par exemple, si nous souhaitons accélérer, il est nécessaire d'augmenter la durée de la commande de l'injecteur qui le maintient en position ON. Lorsqu'on relâche l'accélération la durée de l'ouverture de l'électrovanne diminue jusqu'à une limite inférieure que l'on appelle ralenti.



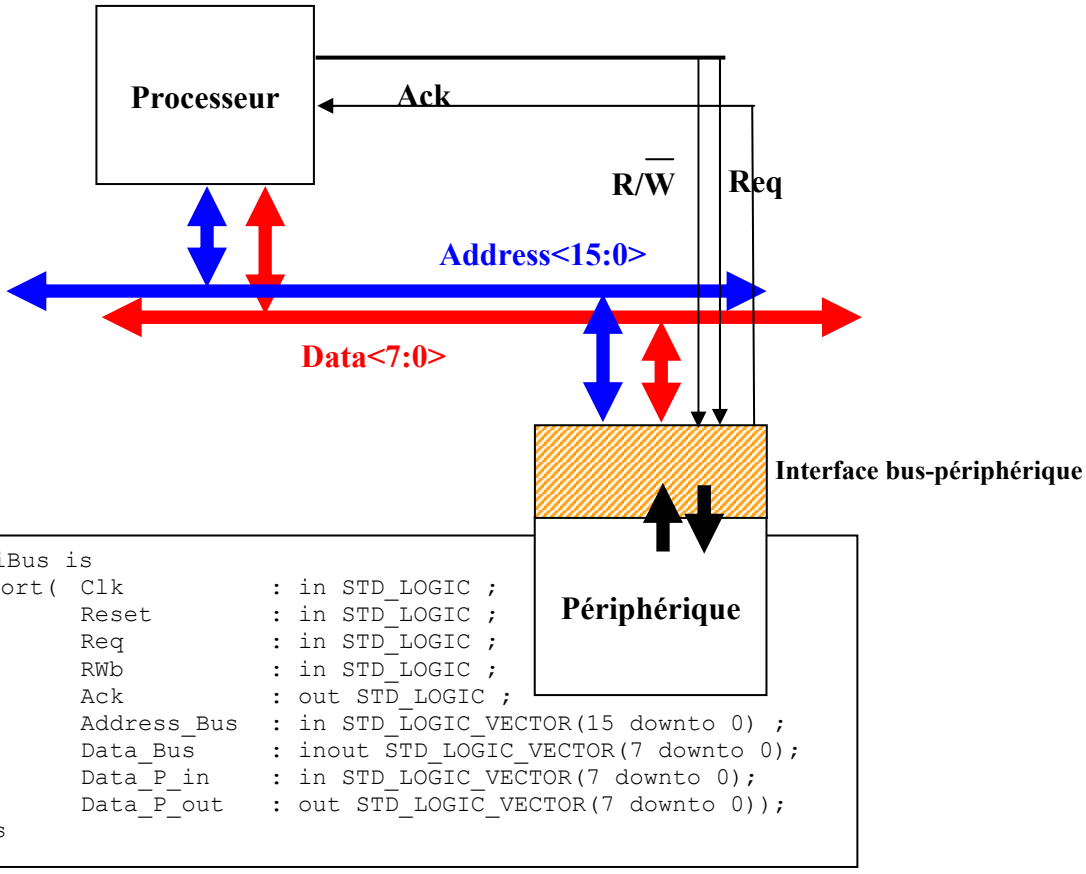
1	**Adm**	**Comp**	**Echap**	**Expl**
2	**Comp**	**Expl**	**Adm**	**Echap**
3	**Expl**	**Echap**	**Comp**	**Adm**
4	**Echap**	**Adm**	**Expl**	**Comp**

- Spécifier par un graphe d'état le circuit de commande de l'injecteur en considérant comme paramètres d'entrée de l'ECU un signal qui indique que le moteur a fait un demi-tour et une valeur numérisée de la position de l'accélérateur.
- Ecrire le modèle VHDL correspondant à l'ECU.

2. Implémentation d'un protocole de communication pour un bus de microprocesseur

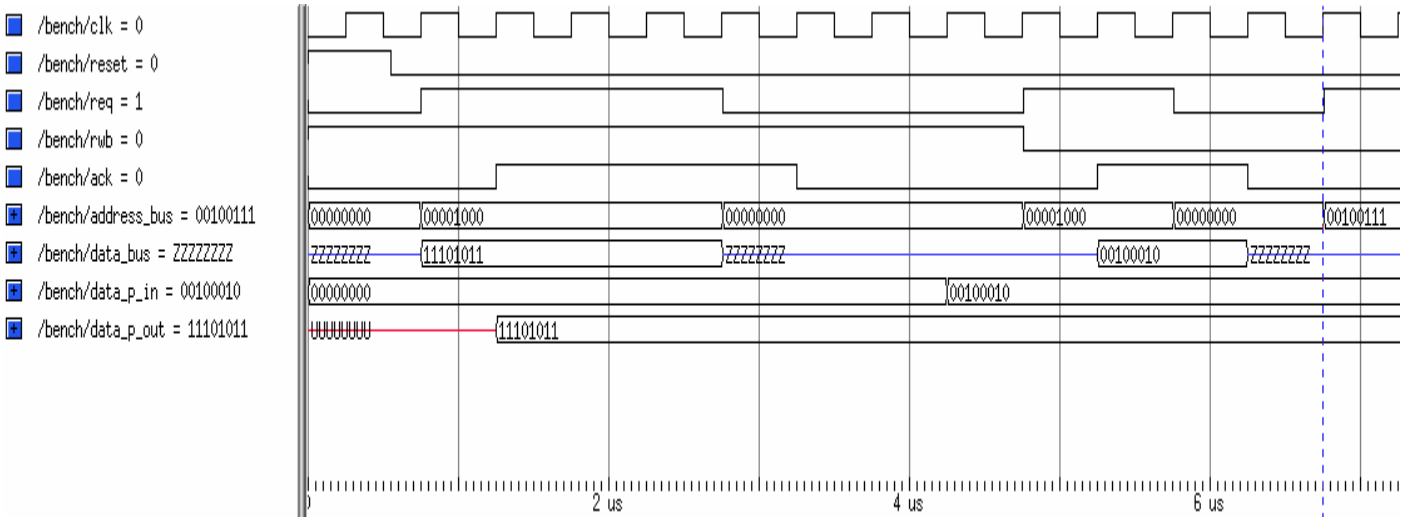
Un microprocesseur utilise un bus pour échanger des informations avec de la mémoire et des périphériques. Dans l'architecture proposée le processeur est toujours maître du bus. Cela

signifie qu'il déclenche aussi bien un ordre d'écriture qu'un ordre de lecture sur le périphérique.




```
entity iBus is
      port(  Clk          : in STD_LOGIC ;
             Reset        : in STD_LOGIC ;
             Req          : in STD_LOGIC ;
             RWb          : in STD_LOGIC ;
             Ack          : out STD_LOGIC ;
             Address_Bus  : in STD_LOGIC_VECTOR(15 downto 0) ;
             Data_Bus     : inout STD_LOGIC_VECTOR(7 downto 0);
             Data_P_in    : in STD_LOGIC_VECTOR(7 downto 0);
             Data_P_out   : out STD_LOGIC_VECTOR(7 downto 0));
end iBus
```

On se propose donc d'implémenter une interface bus-périphérique qui suit le protocole suivant :



- Ecrire l'architecture de l'entité iBus implémentant cette interface