TELECOM PARIS
école nationale supérieure des télécommunications

VHDL language, from specification to model

Very high speed integrated circuits Hardware Description Language

Agenda

TELECOM PARIS
école nationale supérieure des télécommunications

- Introduction
- Principles of Event Driven Simulation
- Practical Organization of Files and Projects
- Compilation Units
- Syntax
  - Sequential VHDL
  - Concurrent VHDL
- Standardized Packages
- Logic Synthesis
- Advices

History of the VHDL Language

TELECOM PARIS
école nationale supérieure des télécommunications

- VHDL (IEEE 1076-1987) was born in 1987 from the joint efforts of:
  - IEEE ->
    Computer Society ->
    Design Automation Technical Committee ->
    Design Automation Standards Subcommittee ->
    VHDL Analysis and Standardization Group
  - CAD Language Systems Inc.
- VHDL (IEEE 1076-2002) is the most recent revision
- Some companion standards:
  - VITAL
  - Synthesis
  - etc.

Warning

TELECOM PARIS
école nationale supérieure des télécommunications

- This course presents VHDL revision 1993
  - Some modifications in the following revisions of the standard may be contradictory with this course
  - Please read the IEEE Language Reference Manuals (LRMs) for more information
- The most important reason for this choice is that some tools still don't implement 100% of VHDL-2002
- This course presents VHDL for synthesis. A lot of features that are very important for modeling are omitted

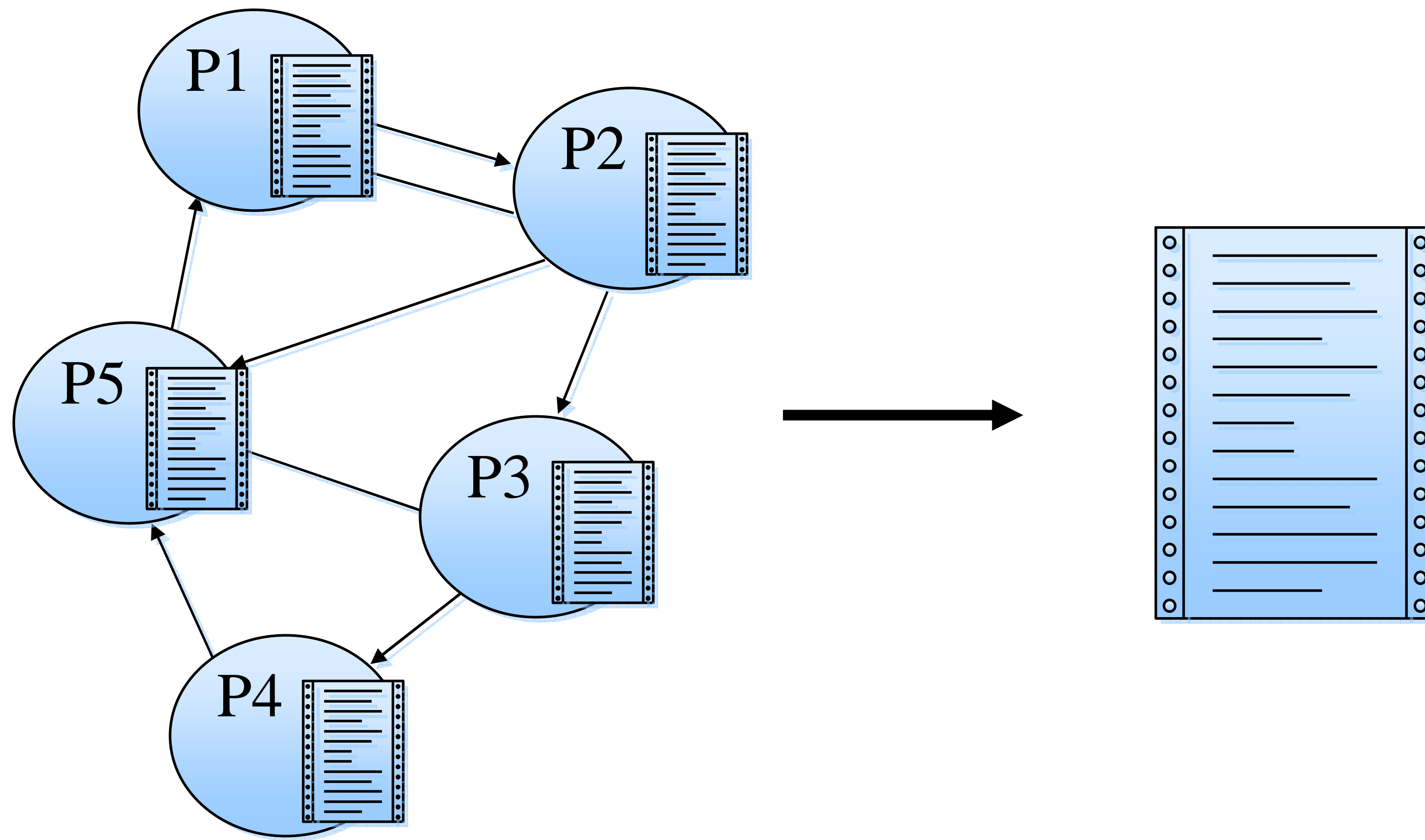
Agenda

TELECOM PARIS
école nationale supérieure des télécommunications

- Introduction
- Principles of Event Driven Simulation
- Practical Organization of Files and Projects
- Compilation Units
- Syntax
  - Sequential VHDL
  - Concurrent VHDL
- Standardized Packages
- Logic Synthesis
- Advices

Simulating parallel systems on sequential computers

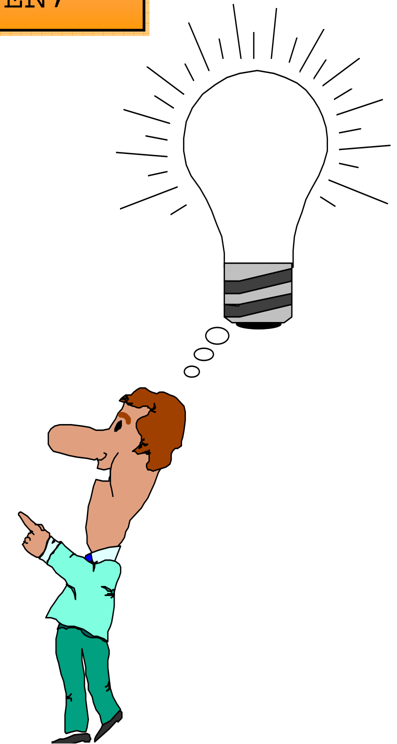
TELECOM PARIS
école nationale supérieure des télécommunications



Simulating parallel systems on sequential computers

- Parallelism is needed
- Non-determinism is an issue:
  - Let’s introduce a new king of variable, dedicated to communication between sequential programs (processes): the signal
  - When executing an assignment statement the value of the signal is not affected
  - The value of the signals is modified once every process was executed (after 1 Δ)

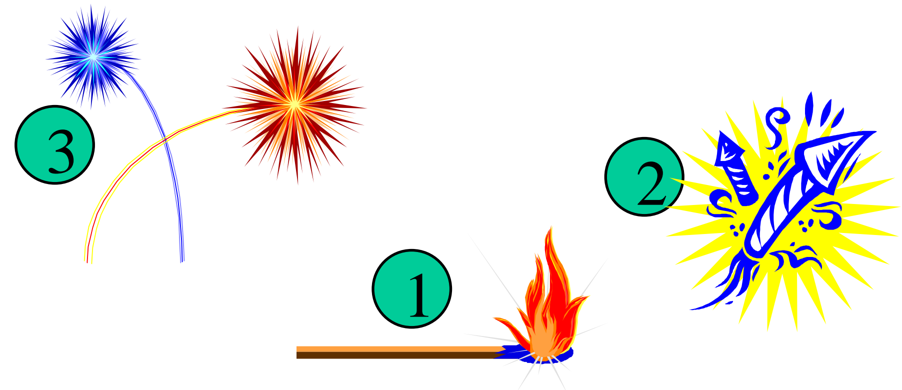
```
SIG1 <= 1;
SIG2 <= RED;
SIG3 <= TEN;
```



TELECOM PARIS
école nationale supérieure des télécommunications

The symbolic time

- Used to specify dependencies between events, that is, to order events
- Needed to distinguish the cause and the effect
- Is the only one logic synthesizers support




```
wait until A = 25;
B <= 3;   -- (after Δ)
```

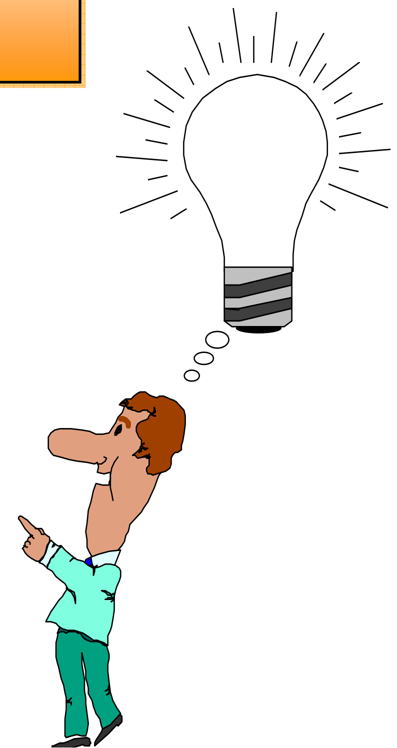
```
wait until B = 3;
C <= 12;  -- (after Δ)
```

Simulating Parallel Systems on Sequential Computers

TELECOM PARIS
école nationale supérieure des télécommunications

- Sequential programming is still needed:
- Classical variables still exist inside the processes
- Variable assignment is immediate
- Processes run in a very classical way

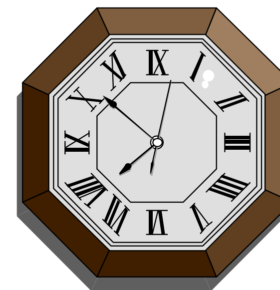
```
VAR1 := 1;
VAR2 := RED;
VAR3 := TEN;
```



Physical time

TELECOM PARIS
école nationale supérieure des télécommunications

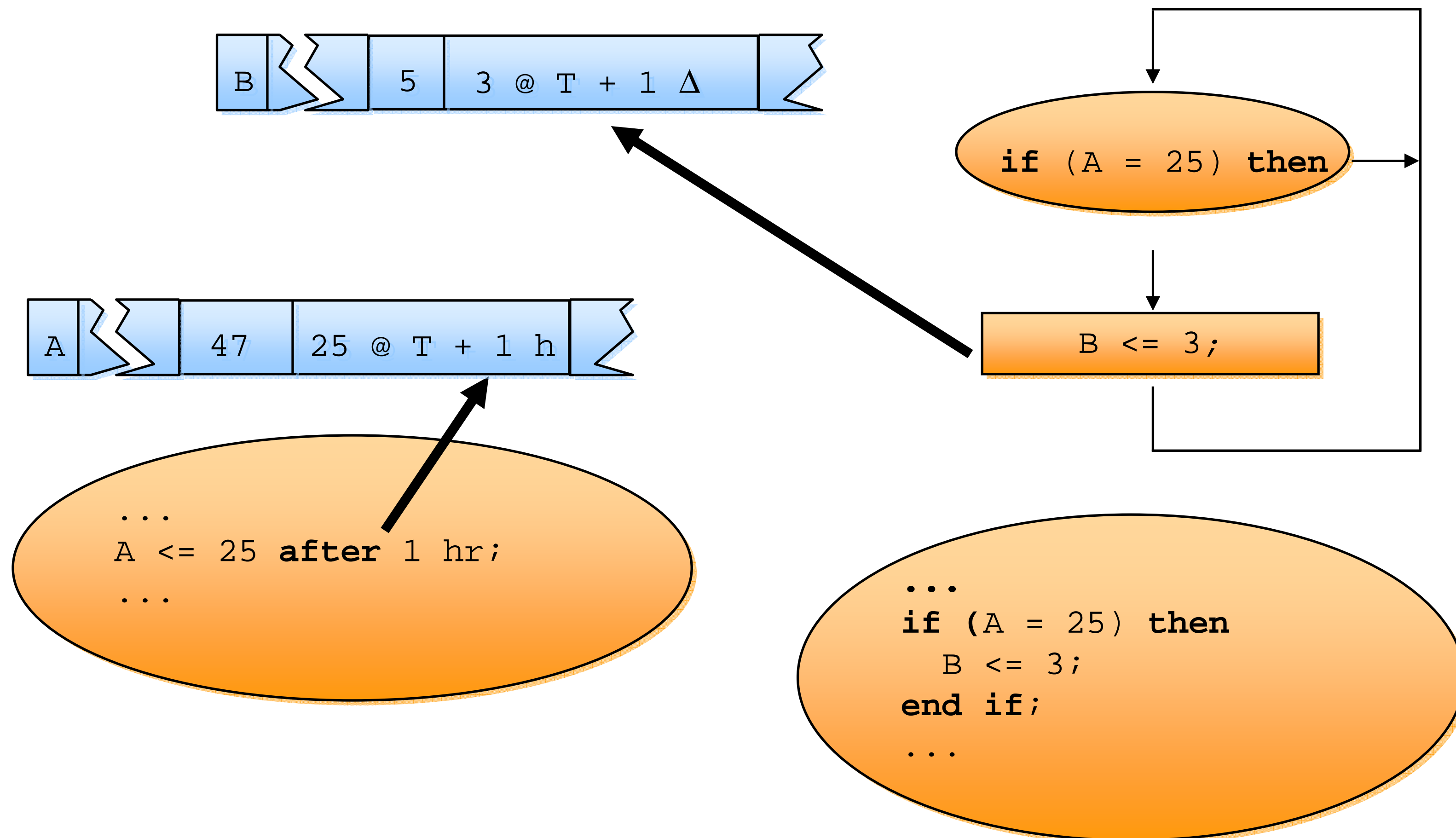
- Physical events occur at a physical time
- We want to model physical events too
- The physical time must be modeled



```
A <= 25 after 1 hr;
```

The signal and its driver

TELECOM PARIS
école nationale supérieure des télécommunications

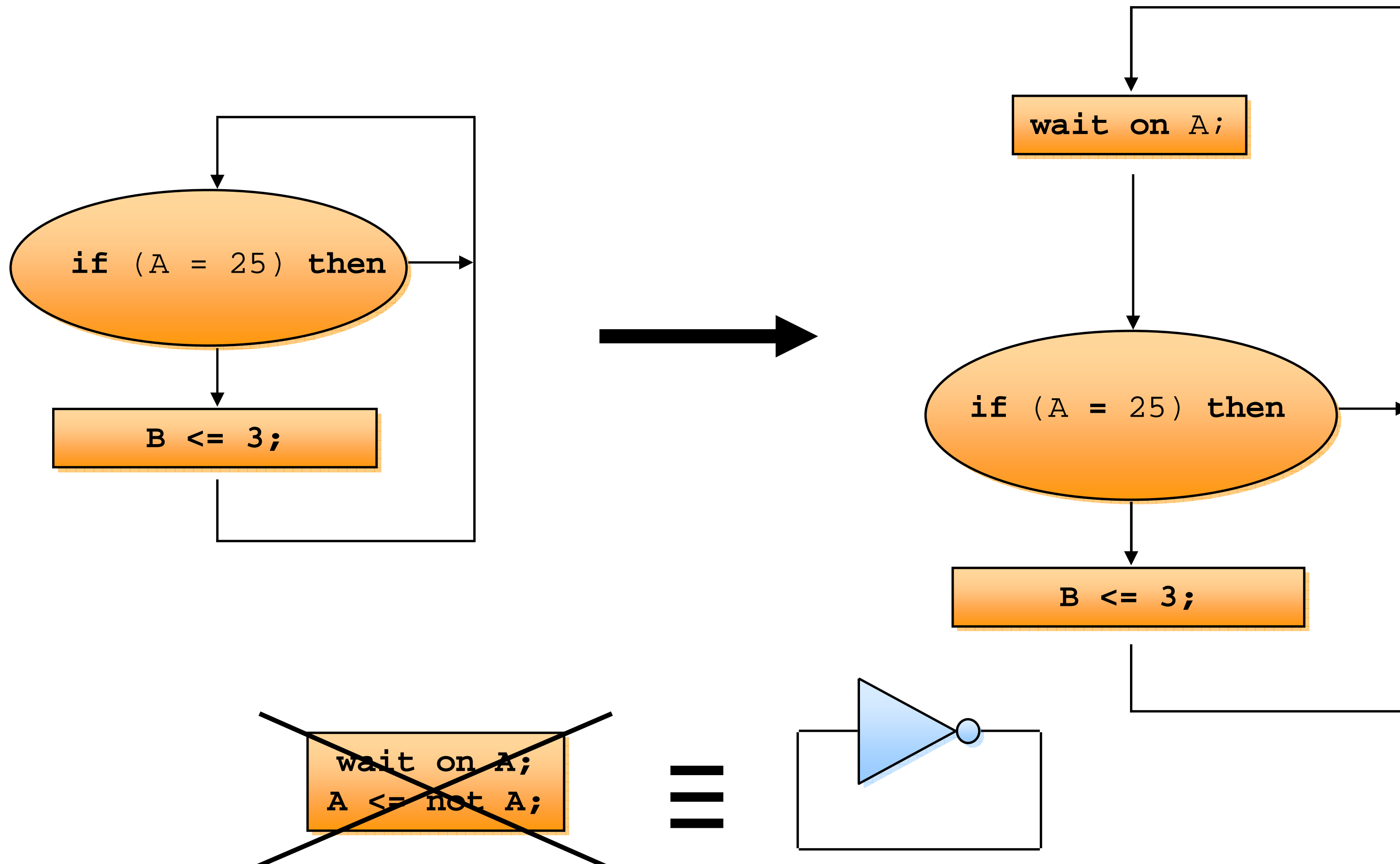


Synchronization between processes

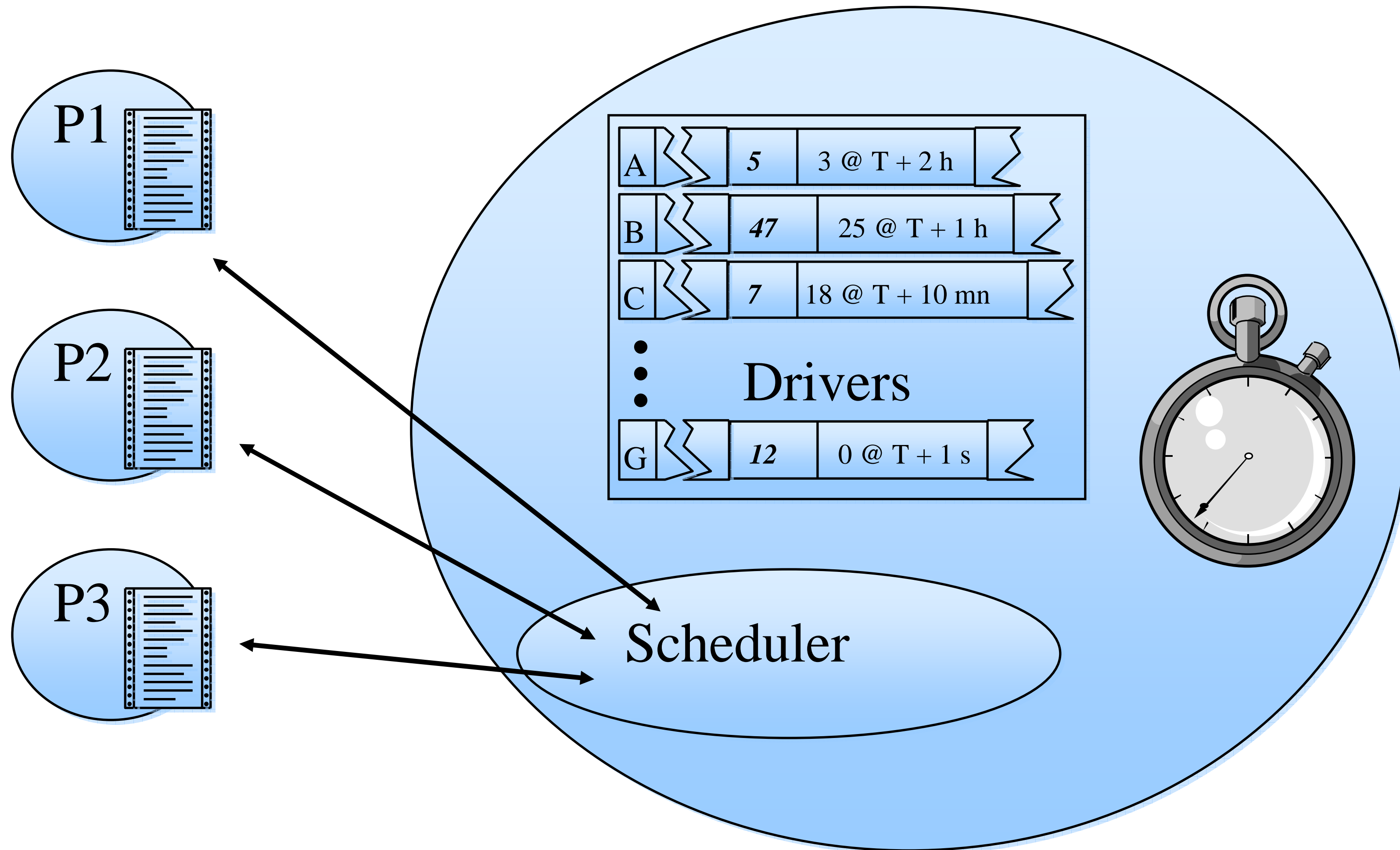
TELECOM PARIS
école nationale supérieure des télécommunications

- Delta cycles have no physical duration
  - So the physical time cannot increase during simulation!
- For most processes incremental step by step execution is very inefficient
  - How to run processes when and only when it's needed
- At each simulation step the simulator resumes only those processes which inputs changed
  - So it must be able to identify what signals are an input of any particular process…
  - … and decide whether they changed since the last execution of the process (or since the last simulation step)

Synchronization between processes



The simulation engine



The process and the time

TELECOM PARIS
école nationale supérieure des télécommunications

- The time evolves when the process is suspended on one of its synchronization points
- Between 2 synchronization points the time is constant
- This is the “Zero-Time” execution
- Signal assignment is delayed
- The process is an infinite loop

```
process
begin
  S <= A + B;
  wait until RISING_EDGE(CLK);
  R := S;
  A <= I0;
  B <= I1;
  wait until RISING_EDGE(CLK);
  if (R > 1023) then
    ALARM <= TRUE;
    COUNTER <= 127;
  else
    COUNTER <= COUNTER - 1;
  end if;
  wait until RISING_EDGE(CLK);
  ISO <= COUNTER * 7;
end process;
```

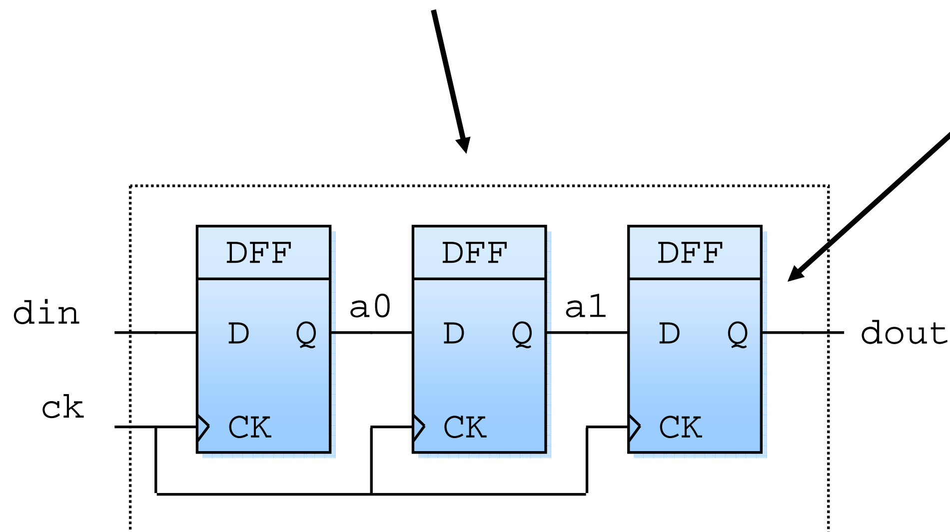
Example

TELECOM PARIS
école nationale supérieure des télécommunications

```
entity REGS is

  port (CK, DIN : in BIT;
        DOUT : out BIT);

end entity REGS;
```




```
architecture ARC of REGS is

  signal A0, A1 : BIT;

begin

  REGS_PR : process(CK)

  begin

    if (CK = '1') then
      A0   <= DIN;
      A1   <= A0;
      DOUT <= A1;
    end if;

  end process REGS_PR;

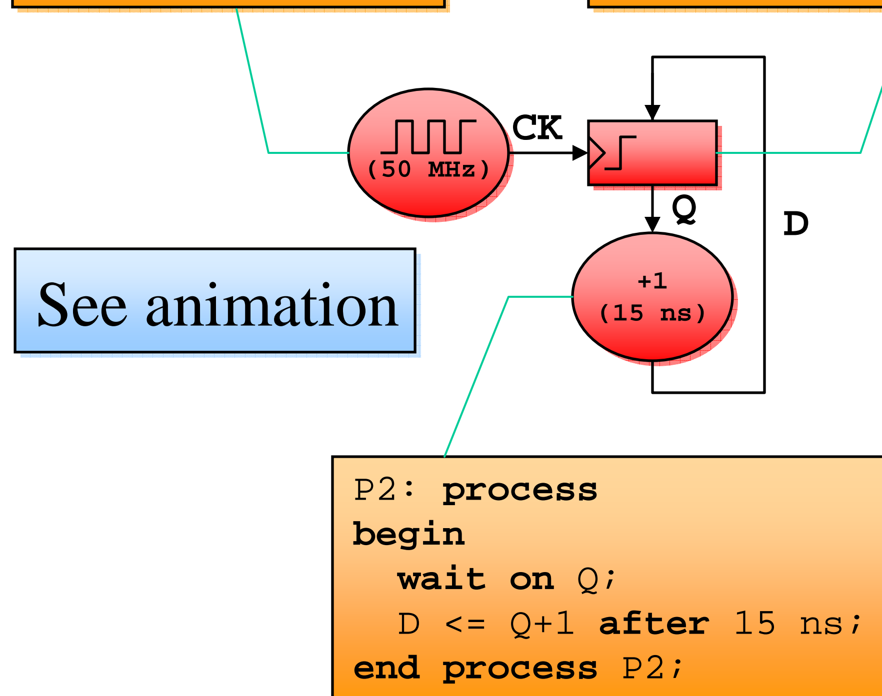
end architecture ARC;
```

Event-driven simulation step by step

```
architecture SIM of INC is
  signal CK: Bit;
  signal D, Q: Natural;
begin
  P1: process
  begin
    CK <= '0';
    wait for 10 ns;
    CK <= '0';
    wait for 10 ns;
  end process P1;
  P2: process
  begin
    wait on Q;
    D <= Q+1 after 15 ns;
  end process P2;
  P3: process
  begin
    wait on CK;
    if (CK = '1') then
      Q <= D;
    end if;
  end process P3;
end architecture SIM;
```

```
P1: process
begin
  CK <= '0';
  wait for 10 ns;
  CK <= '1';
  wait for 10 ns;
end process P1;
```

```
P3: process
begin
  wait on CK;
  if (CK = '1') then
    Q <= D;
  end if;
end process P3;
```



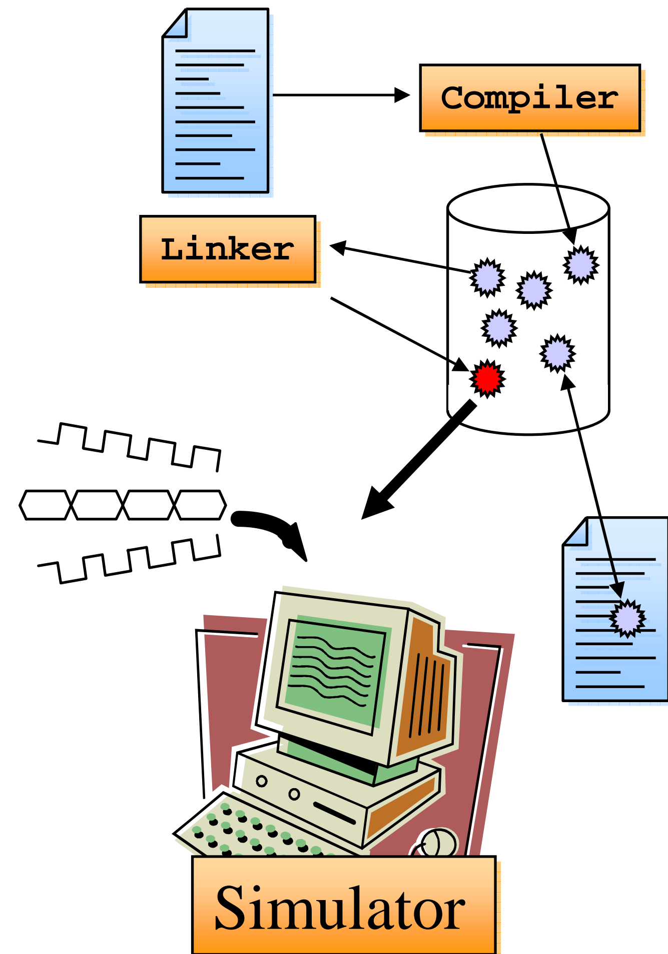
Agenda

TELECOM PARIS
école nationale supérieure des télécommunications

- Introduction
- Principles of Event Driven Simulation
- **Practical Organization of Files and Projects**
- Compilation Units
- Syntax
  - Sequential VHDL
  - Concurrent VHDL
- Standardized Packages
- Logic Synthesis
- Advices

Structure of the language

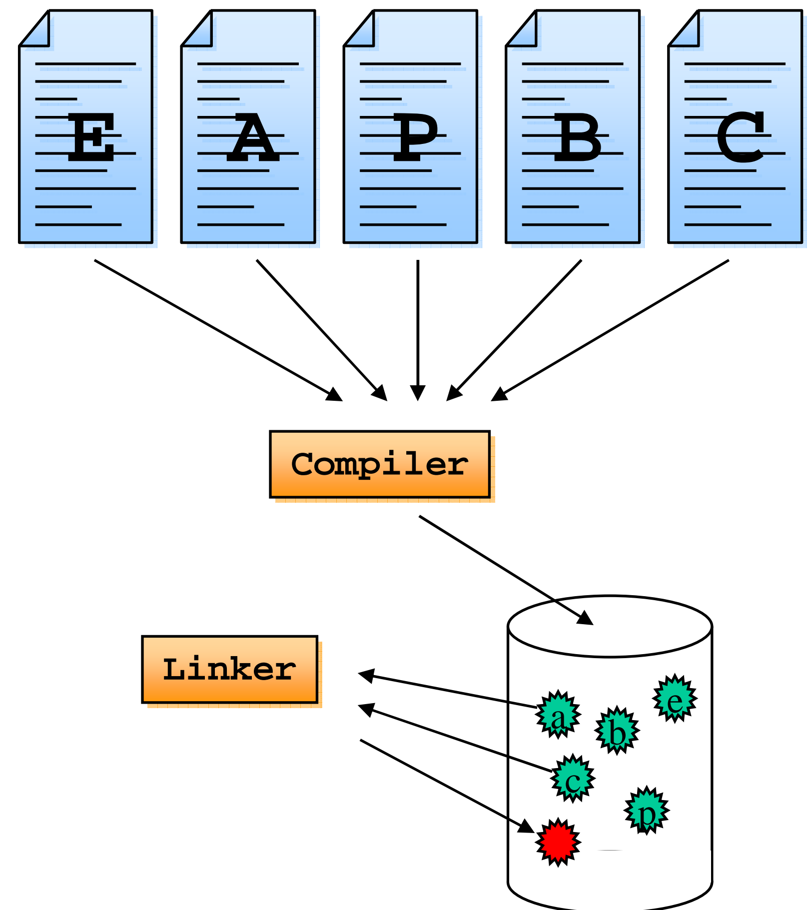
- To simulate we need:
  - Analysis (compilation) of source files
  - Elaboration (link) of compilation results
  - We always simulate the result of an elaboration
- The result of an analysis or an elaboration is stored in a library
- The content of an existing library may be used in another program (after the proper declaration)



Structure of the language

TELECOM PARIS
école nationale supérieure des télécommunications

- 5 compilation units:
  - Entity
  - Architecture
  - Package declaration
  - Package body
  - Configuration
- Only the 5 compilation units can be compiled (analyzed)
- Only the result of the compilation of an architecture or a configuration can be elaborated (linked)



Libraries

TELECOM PARIS
école nationale supérieure des télécommunications

- The symbolic name `WORK` designates the target library of a compilation (the library in which the result of the compilation will be stored)
- To access a library it must first be declared:
  - ```
    library LIB;
    use LIB.PAQ.OBJ;
    ```
- Creation and management of the libraries are not defined in the standard, they are tool-dependant. Every environment has its own solutions
- Libraries may be shared between users
    ```

Agenda

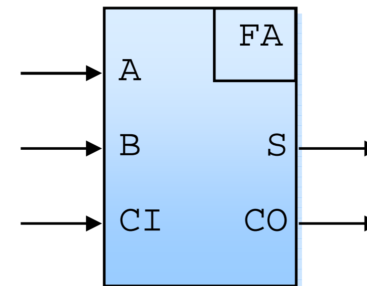
TELECOM PARIS
école nationale supérieure des télécommunications

- Introduction
- Principles of Event Driven Simulation
- Practical Organization of Files and Projects
- Compilation Units
- Syntax
  - Sequential VHDL
  - Concurrent VHDL
- Standardized Packages
- Logic Synthesis
- Advices

The entity

TELECOM PARIS
école nationale supérieure des télécommunications

- It's the interface specification. It provides:
  - The module name (FA)
  - Its input-output ports:
    - Name
    - Direction (in, out, inout, ...)
    - Type (BIT, BIT_VECTOR, BOOLEAN, INTEGER, ...)
- The ports are visible and usable as signals inside the associated architecture. They must not be re-declared in the architecture

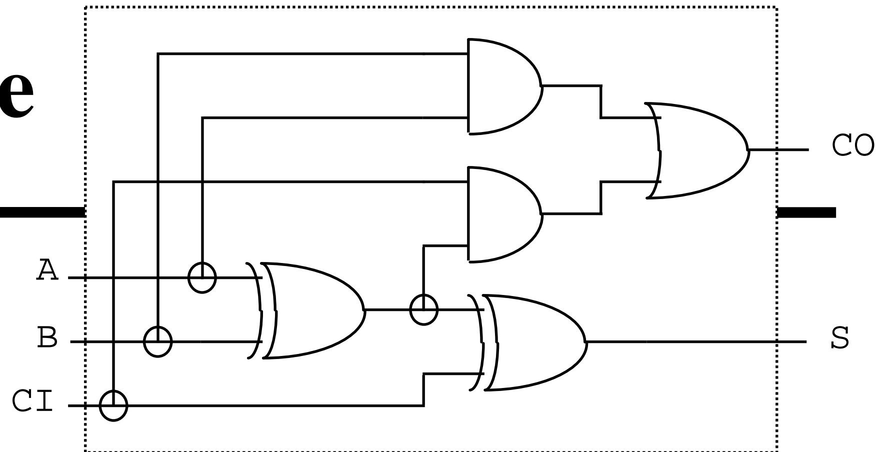



```
entity FA is

  port(A, B, CI: in  BIT;
          S, CO:     out
BIT);

end entity FA;
```

The architecture



- It's the internal description. It's always associated with its entity
- A single entity may be associated with several architectures

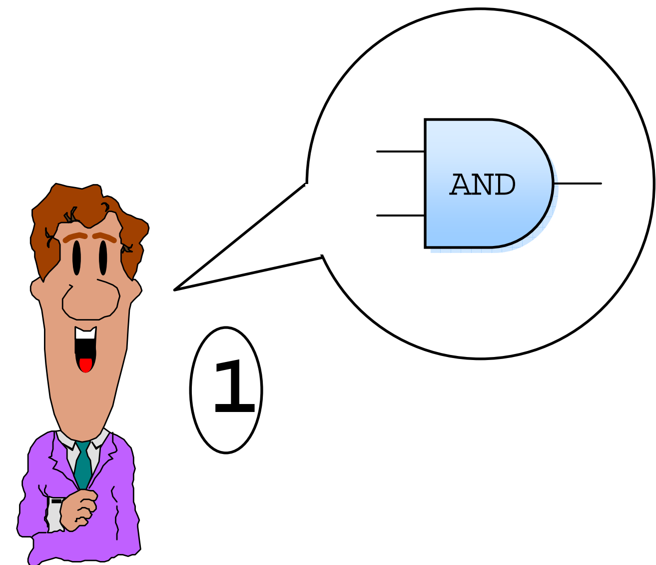
```
architecture DF of FA is
  signal I0, I1: BIT;
begin
  P1: process(A, B)
  begin
    I0 <= A xor B;
    I1 <= A and B;
  end process P1;
  P2: process(I0, I1, CI)
  begin
    S <= I0 xor CI;
    CO <= (I0 and CI) or I1;
  end process P2;
end architecture DF;
```

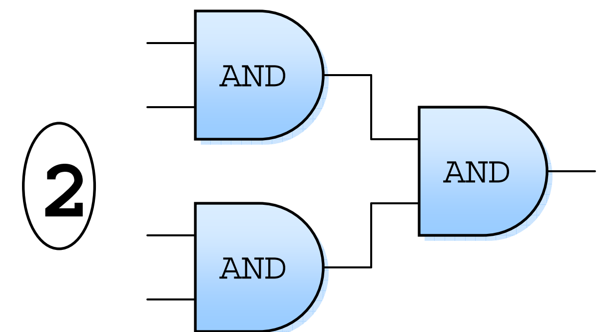
```
architecture BEV of FA is
begin
  PR: process(A, B, CI)
  begin
    S  <= A xor B xor CI;
    CO <= (A and B) or (A and CI) or
          (B and CI);
  end process PR;
end architecture BEV;
```

Structure of the language

TELECOM PARIS
école nationale supérieure des télécommunications

- VHDL is a heavily declarative language:
  - Every object must be declared before usage:
    - Variable
    - Signal
    - Constant
    - Function
    - Procedure
    - Component
    - ...
  - There are dedicated declaration area
  - One cannot declare anything anywhere…





The architecture

TELECOM PARIS
école nationale supérieure des télécommunications

- Declaration areas
- Body (concurrent instructions)
  - Process
  - Concurrent signal assignment
  - Component instantiation...
- Parallel execution, file order not relevant

```
architecture DF of FA is

  signal SI: BIT;
  component MAJ
    port(X, Y, Z: in BIT; M: out BIT);
  end component;

begin

  PP: process(SI, CI)
  begin
    S  <= SI xor CI;
  end process PP;

  SI <= A  xor B;

  RET: MAJ port map(X => A, Y => B,
                    Z => CI, M => CO);

end architecture DF;
```

The architecture

TELECOM PARIS
école nationale supérieure des télécommunications

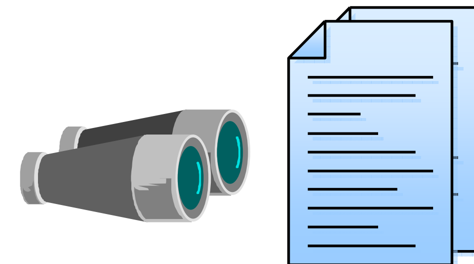
- The ports of the associated entity are visible and usable as signals inside the associated architecture. They must not be re-declared in the architecture
- Input ports (`in`) are read-only
- Output ports (`out`) are write-only

```
entity FA is
  port(A, B, CI: in  BIT;
       S, CO:    out BIT);
end entity FA;

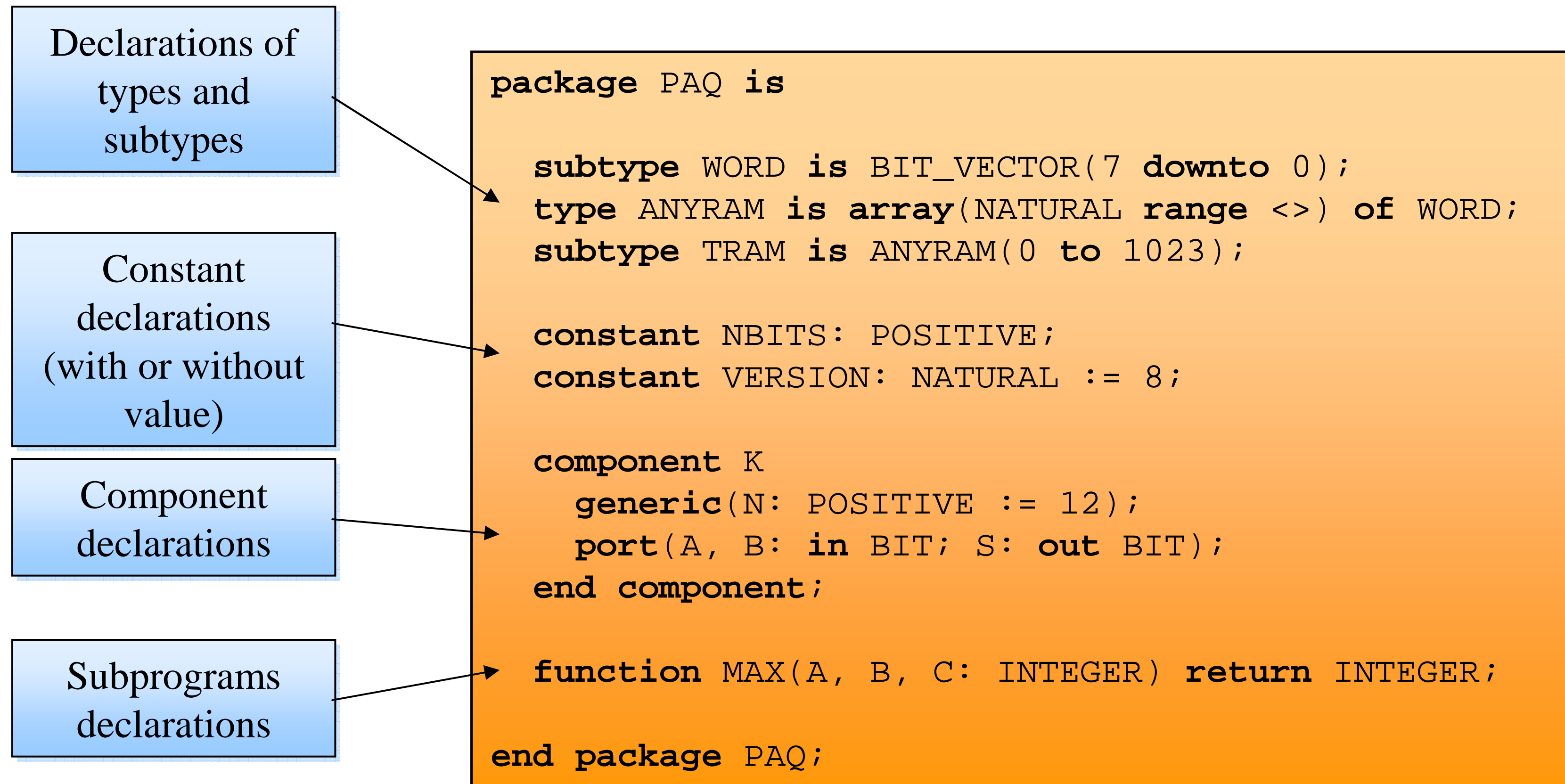
architecture BUG of FA is
  signal A, B, CI, S, CO: BIT;
begin
  A  <= B or CI;
  CO <= (A and B) or (A and CI) or
        (B and CI);
  S <= (A or B or CI) and not CO or
       A and B and CI;
end architecture BUG;
```

The package

- It’s a collection of reusable things
- It’s made of two compilation units:
  - Package declaration
  - Package body
- The content of the package declaration is “visible” from another compilation unit if it declared its use (public part)
- The package body is “invisible” from the other compilation units (private part)



The package declaration



The package body

TELECOM PARIS
école nationale supérieure des télécommunications

Constant declarations

Public or private subprogram bodies. Here with name overloading

```
package body PAQ is

  constant NBITS: POSITIVE := 32;

  function MAX(A, B: INTEGER) return INTEGER is
  begin
    if(A>B) then
      return A;
    else
      return B;
    end if;
  end MAX;

  function MAX(A, B, C: INTEGER) return INTEGER is
  begin
    return MAX(A, MAX(B,C));
  end MAX;

end package body PAQ;
```

Components

TELECOM PARIS
école nationale supérieure des télécommunications

- Structural description = assembly of simpler devices
- Based on the declaration and instantiation of a component
- The component is not a compilation unit it's a prototype for an entity
- Components allow for a top-down design strategy

```
entity AND3 is
  port(A0, A1, A2: in BIT;
           Z: out BIT);
end entity AND3;
architecture STR of AND3 is
  signal TMP: BIT;
  component AND2
    port(A, B: in BIT;
         C:    out BIT);
  end component;
begin
  I0: AND2 port map(A => A0,
                    B => A1,
                    C => TMP);
  I1: AND2 port map(A => A2,
                    B => TMP,
                    C => Z);
end architecture STR;
```

The configuration

TELECOM PARIS
école nationale supérieure des télécommunications

- It binds component instances on entity / architectures pairs
- If it exists it is the compilation unit to elaborate before simulation
- The simulator always needs a kind of configuration
- Logic synthesizers usually implement a default configuration scheme. Some don't even support configurations

```
configuration CF1 of AND2 is
  for CMP
  end for;
end configuration CF1;

library BIB;

configuration CF2 of AND3 is
  for STR
    for I0: AND2
      use configuration BIB.CF1;
    end for;
    for I1: AND2
      use entity BIB.AND2(CMP);
    end for;
  end for;
end configuration CF2;
```

The configuration

TELECOM PARIS
école nationale supérieure des télécommunications

- It may be flat or hierarchical (in this case the top level configuration is the one to elaborate)
- The `for all` statement simplifies its source code
- Even when empty it gives useful information (an entity name and an associated architecture name)

```
configuration C3 of E3 is
  for A3
    for I2: K2
      use entity BIB.E2(A2);
      for A2
        for all: K1
          use entity BIB.E1(A1);
        end for;
      end for;
    end for;
  end for;
end configuration C3;
```

Immediate configuration

TELECOM PARIS
école nationale supérieure des télécommunications

- It's possible to immediately bind instantiated components to entity / architecture pairs with an immediate configuration statement. Syntax:
  - ```
    for all: COMPONENT_NAME use entity
    ENTITY(ARCHITECTURE);
    ```
- or:
  - ```
    for LABEL1, LABEL2: COMPONENT_NAME use
    entity ENTITY(ARCHITECTURE);
    ```
- Immediate configuration statements must appear just after the local declarations and before the architecture body

Instantiation of entities or configurations

TELECOM PARIS
école nationale supérieure des télécommunications

- One can avoid components and directly instantiate:
  - An entity – architecture pair
  - A configuration

```
entity AND3 is
  port(A0, A1, A2: in BIT;
           Z: out BIT);
end entity AND3;
architecture STR of AND3 is
  signal TMP: BIT;
begin
  I0: entity WORK.AND2(CMP)
      port map(A => A0, B => A1, C => TMP);
  I1: configuration WORK.CF2
      port map(A => A2, B => TMP, C => Z);
end architecture STR;
```

Hierarchical design

TELECOM PARIS
école nationale supérieure des télécommunications

- To build a design from sub-designs
  - Instantiate entity – architecture pair
  - Wire them together
  - It's the structural description style (vs. behavioral)

```
entity AND3 is
  port(A0, A1, A2: in BIT;
        Z: out BIT);
end entity AND3;
architecture STR of AND3 is
  signal TMP: BIT;
begin
  I0: entity WORK.AND2(CMP)
      port map(A => A0, B => A1, C => TMP);
  I1: entity WORK.AND2(CMP)
      port map(A => A2, B => TMP, C => Z);
end architecture STR;
```

Agenda

TELECOM PARIS
école nationale supérieure des télécommunications

- Introduction
- Principles of Event Driven Simulation
- Practical Organization of Files and Projects
- Compilation Units
- Syntax
  - Sequential VHDL
  - Concurrent VHDL
- Standardized Packages
- Logic Synthesis
- Advices

The process

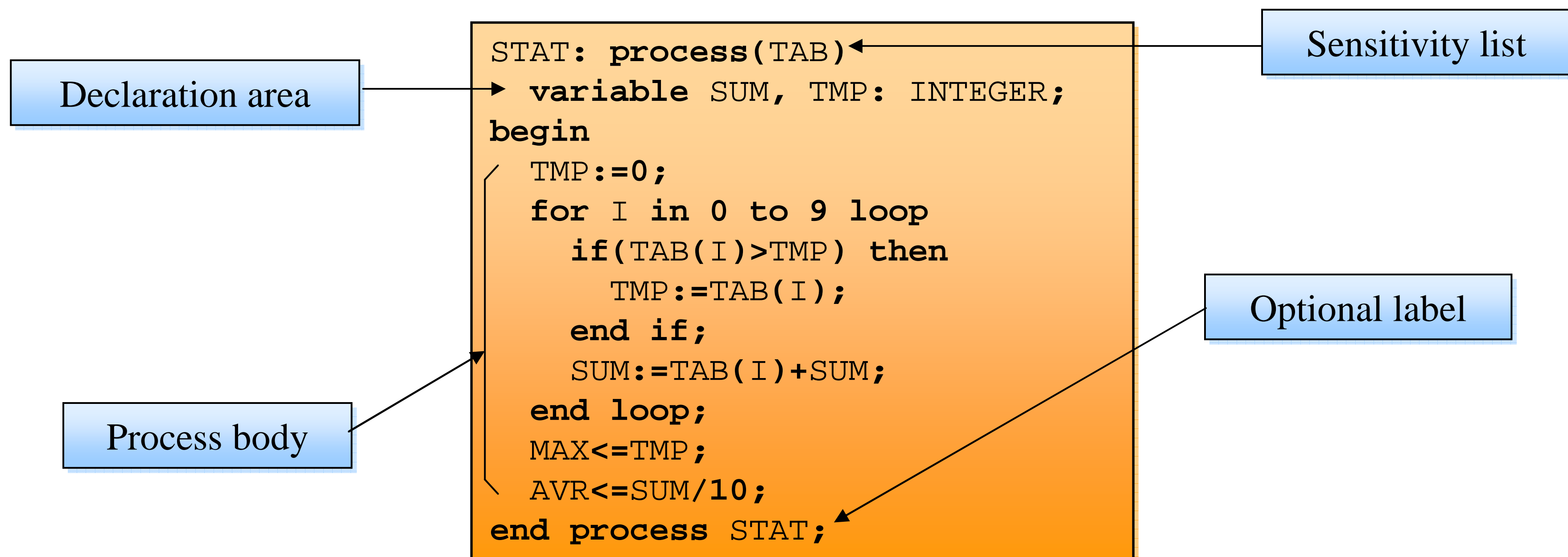
- A process is a sequential program
- Every object it manipulates has a type
- It manipulates objects with operators
- Its control flow is specified by control structures

```
int max, i, sum, avr;
int tab[10];
...
max=0;
for(i=0; i<10; i++) {
  if(tab[i]>max)
    max=tab[i];
  sum+=tab[i];
  }
avr=sum/10;
```

```
variable MAX, SUM, AVR: INTEGER;
type T is array(0 to 9) of INTEGER;
variable TAB: T;
...
MAX:=0;
for I in 0 to 9 loop
  if(TAB(I)>MAX) then
    MAX:=TAB(I);
  end if;
  SUM:=TAB(I)+SUM;
end loop;
AVR:=SUM/10;
```

Processes with sensitivity lists

- A process may have a sensitivity list
- The sensitivity list is a list of signals
- It's the only synchronization point of the process



Exercise #1: examples of processes

TELECOM PARIS
école nationale supérieure des télécommunications

- This process models a combinatorial function of signals X and Y. What function? As soon as X or/and Y changes the function is re-executed. Write a combinatorial process implementing the majority function of 3 signals. Same question with the full adder

```
signal X, Y, Z: BIT;
P1: process(X, Y)
begin
  if X = '1' then
    Z <= '1';
  elsif Y = '1' then
    Z <= '1';
  else
    Z <= '0';
  end if;
end process P1;
```

- This process is a synchronous one. It models the behavior of a D-flip-flop (DFF).What signal is the clock? The input? The output? Explain the behavior of this process. Write a process modeling a DFF on rising edge of its clock and with asynchronous, active low, reset.

```
signal X, Y, Z: BIT;
P2: process(Z)
begin
  if Z = '1' then
    Y <= X;
  end if;
end process P2;
```

Comments, identifiers, literals, ...

TELECOM PARIS
école nationale supérieure des télécommunications

- **Comments start with a double dash (`--`) and extend until end of line (no multi-line comment `/* ... */`)**
- **Identifiers are sequences of letters, digits and underscores (_). They must start with a letter. VHDL is case insensitive**
- **Literals are constant explicit values:**
  - `45` **and** `7.89` **are numeric literals**
  - `"this is a string of characters "`
  - `'C'` **is a character literal**
  - `"000111010110"`, `B"000111010110"`, `O"726"` **and** `X"1E6"` **are bit-string literals**
  - `null` **is an access (pointer) literal**
- **Expressions are terminated by a semicolon (`;`)**

Kinds of value containers

TELECOM PARIS
école nationale supérieure des télécommunications

- Value containers can be one of three kinds:
  - Variables, very similar to variables in any other programming language, they are dedicated to classical sequential programming (inside processes)
  - Constants, similar too to what is found in other languages
  - Signals, the VHDL originality, dedicated to parallel programming and, more precisely, to the exchanges between several programs running in parallel
- In order to avoid common mistakes assignments are denoted in different ways depending on the kind of container:
  - `A := 178` for variables and constants
  - `S <= 178` for signals

Initialization of variables and signals

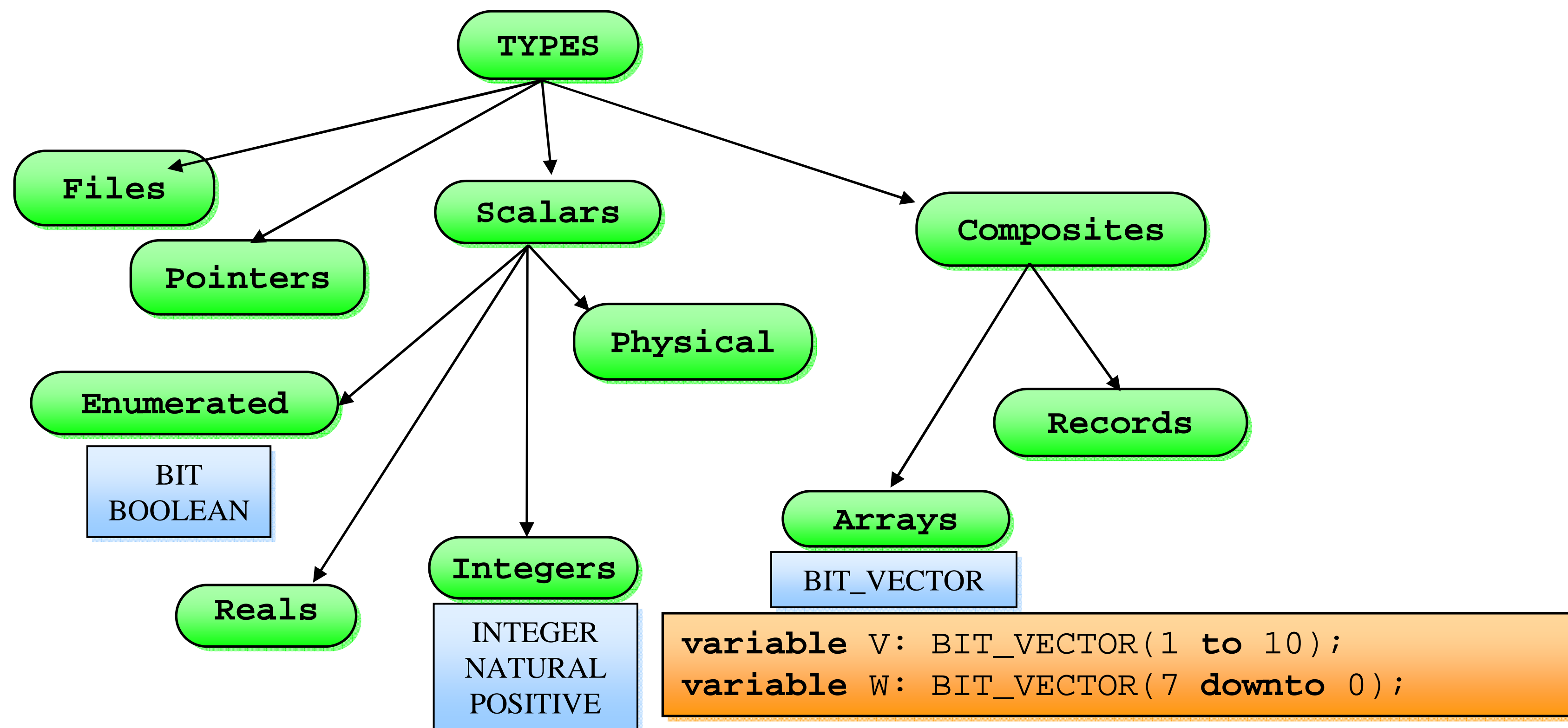
TELECOM PARIS
école nationale supérieure des télécommunications

- A variable or a signal is initialized at the beginning of the simulation (time zero). Its default initialization value is the leftmost value of the declaration of its type:
  - ```
    type T is (RED, GREEN, BLUE);
    ...
    variable V: T; -- Initialization value of V is RED
    ```
- It is possible to declare another initialization value when declaring a variable or a signal:
  - ```
    signal S: INTEGER := 0;
    ...
    variable V: BOOLEAN := TRUE;
    ```
- Very often it's a bad idea because it may hide real "reset" defaults

The types

TELECOM PARIS
école nationale supérieure des télécommunications

- Signals, variables and constants always have a type



The integer types

TELECOM PARIS
école nationale supérieure des télécommunications

- The type Integer is build from the integer type of the host CPU. The LRM requires that its length is larger or equal to 32 bits. Subtypes my be defined.
  - `type INTEGER is range CPU_DEPENDENT;`
- The types `NATURAL` and `POSITIVE` are range subtypes, (`range`) of the same base type Integer
  - `subtype NATURAL is INTEGER range 0 to INTEGER'HIGH`
  - `subtype POSITIVE is INTEGER range 1 to INTEGER'HIGH;`
- A subtypes inherits the properties of its base types. Compatibility errors may occur during assignment
- The type attribute `INTEGER'HIGH` represents the largest element of type Integer. Its value is CPU-dependent. `'HIGH` is a type attribute, as in ADA

The integer types

TELECOM PARIS
école nationale supérieure des télécommunications

- Integer objects are used to represent array indices, loop indices, data, …
- They are defined from a base type by giving the range (bounds and direction):
  - `subtype ONE_TO_TEN is NATURAL range (1 to 10);`
  - `subtype TEN_TO_ONE is NATURAL range (10 downto 1);`
- Warning there is a difference between types and subtypes:
  - `type ONE_TO_TEN is range 1 to 10;`
  - `type TEN_TO_ONE is range 10 downto 1;`
- Warning: the bounds must be compatible with the base type
- Warning: a type for which bounds order and direction are different is an empty type

The real types

TELECOM PARIS
école nationale supérieure des télécommunications

- **The type `REAL` is based on the architecture of the host CPU. The LRM says that it emulates the mathematical behavior of real numbers and requires it's dynamic to allow the representation of numbers from `-1.0E+38` to `1.0E+38`**
- **In VHDL a real number is written:**
  - `+/-number. number{E+/- number}`
- **Examples:**
  - `A := 1.0;`
  - `A := 1.0E10;`
  - `A := 1.5E-20;`

The physical types

TELECOM PARIS
école nationale supérieure des télécommunications

- **VHDL allows the definition of physical types. They are dedicated to representing physical values such as time, voltage, etc. A physical type is a combination of an integer type and a units system…**
- **The type `TIME` is the only predefined physical type:**

```
type TIME is range CPU_DEPENDENT units
    fs;
    ps = 1000 fs;
    ns = 1000 ps;
    us = 1000 ns;
    ms = 1000 us;
    sec = 1000 ms;
    min = 60 sec;
    hr = 60 min;
end units;
```

The enumerated types

TELECOM PARIS
école nationale supérieure des télécommunications

- **An enumerated type is a type with an exhaustive definition by enumeration:**
  - `type COLORS is (RED, YELLOW, BLUE, GREEN, ORANGE);`
  - `type FOUR_STATES is ('X', '0', '1', 'Z');`
  - `type STD_ULOGIC is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');`
- **The order in which an enumerated type is declared is meaningful. Example: a signal or a variable of enumerated type `T` takes `T'LEFT` as default initialization value**

Predefined enumerated types

TELECOM PARIS
école nationale supérieure des télécommunications

```
type BOOLEAN is (FALSE, TRUE);
type BIT is ('0', '1');
type SEVERITY_LEVEL is (NOTE, WARNING, ERROR, FAILURE);
type CHARACTER is
  (NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL,
   BS,  HT,  LF,  VT,  FF,  CR,  SO,  SI,
   DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB,
   CAN, EM,  SUB, ESC, FSP, GSP, RSP, USP,
   ' ', '!', '"', '#', '$', '%', '&', ''',
   '(', ')', '*', '+', ',', '-', '.', '/',
   '0', '1', '2', '3', '4', '5', '6', '7',
   '8', '9', ':', ';', '<', '=', '>', '?',
   '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
   'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
   'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
   'X', 'Y', 'Z', '[', '\', ']', '^', '_',
   '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g',
   'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
   'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
   'x', 'y', 'z', '{', '|', '}', '~', 'DEL', etc.);
```

The array types

TELECOM PARIS
école nationale supérieure des télécommunications

- Array types are collections of identical objects indexed by ranges of integer or enumerated types
- Example :
  - `type BUS is array (0 to 31) of BIT;`
  - `type RAM is array (0 to 1024, 0 to 31) of BIT;`
  - ```
    type PRICE is range 0 to INTEGER'HIGH units
      cent;
      nickel = 5 cent;
      dime = 2 nickel;
      dollar = 10 dime;
    end units;
    ```
  - ```
    type COLORS is (WHITE, BLUE, GREEN, RED, YELLOW,
    BLACK, RAINBOW);
    ```
  - ```
    type PAINTINGS_PRICES is array (COLOR range WHITE
    to BLACK) of PRICE;
    ```
    ```

Unconstrained arrays

TELECOM PARIS
école nationale supérieure des télécommunications

- An array type may be declared with an unknown range:
  - `type BIT_VECTOR is array (NATURAL range <>) of BIT;`
- Unconstrained array types may be used to model parameters of subprograms. But of course variable or even infinite size objects cannot exist. In order to create an object of type `BIT_VECTOR` its actual size must be declared:
  - `subtype TYPE_BUS is BIT_VECTOR(0 to 31);`
  - `variable VARIABLE_BUS1: TYPE_BUS;`
  - `variable VARIABLE_BUS2: BIT_VECTOR(0 to 31);`

The STRING type

TELECOM PARIS
école nationale supérieure des télécommunications

- VHDL defines character strings:
  - `type STRING is array (POSITIVE range <>) of CHARACTER;`
  - `“This is a string"   -- STRING`
- Some literals are ambiguous and cannot be typed only by evaluating the context:
  - `'1'      -- BIT or CHARACTER ?`
  - `B"01010101"   -- BIT_VECTOR in binary form`
  - `O"0120768" -- BIT_VECTOR in octal form`
  - `X"0134DF54"   -- BIT_VECTOR in hexadecimal form`
  - `"01010101" -- BIT_VECTOR or STRING ?`
- Qualification may be used to solve the ambiguity:
  - `BIT_VECTOR'("01010101")`
- Warning: qualification is not a conversion

The records

TELECOM PARIS
école nationale supérieure des télécommunications

- A record is an object which elements are heterogeneous
- Example:
  - `type OPTYPE is (ADD, SUB, MPY, DIV, JMP);`
  - ```
    type INSTRUCTION is record
       OPCODE: OPTYPE;
       SRC: INTEGER;
       DST: INTEGER;
    end record;
    ```
- VHDL has no records with variants (the C unions)
    ```

The access types (pointers)

TELECOM PARIS
école nationale supérieure des télécommunications

- The concept of pointers is very far from hardware but one can create pointer types in VHDL to reference dynamic data. Pointer types may be useful to very abstract high level hardware descriptions not intended for logic synthesis
- Dynamic objects are allocated with the statement `new`
- Destruction is done with the statement `deallocate` which is implicitly self-declared when the `access` type is declared
- Example :
  - `type FIFO_ELEMENT is array(0 to 3) of STD_LOGIC;`
  - `type FIFO_ACCESS is access FIFO_ELEMENT;`
  - `variable FIFO_PTR: FIFO_ACCESS;`
  - `FIFO_PTR := new FIFO_ELEMENT;`
  - `dealocate(FIFO_PTR);`

The access types (pointers)

TELECOM PARIS
école nationale supérieure des télécommunications

- A dynamic object is created and elaborated with the standard rules of VHDL. To create linked lists where objects point to objects of the same type an "incomplete declaration" is used:

```
type T;
type T_PTR is access T;
type T is
record
  VALUE: INTEGER;
  NEXT: T_PTR;
end record;
type PI is access INTEGER;
```

```
variable V, W: T_PTR;
variable VI: PI;
...
V := new T'(1, null);
V.NEXT := new T'(2, new T'(3, null));
V. NEXT. NEXT. NEXT := new T;
W := V. NEXT. NEXT. NEXT;
W.all := (4, null);
VI.all := W.VALUE = 4;
```

The file types

TELECOM PARIS
école nationale supérieure des télécommunications

- A file is an object allowing data exchanges with the outside. It is external to the VHDL system
- A file is a sequence of records of the same base type (scalar, record or array). It is readable, writable or appendable
- A file type is declared:
  - `type FT is file of TM;`
- As soon as a file type is declared several associated subprograms are implicitly self-declared:
  - Opening and closing procedures, end of file test function:
    - `procedure FILE_OPEN (file F: FT; External_Name: in STRING; Open_Kind: in FILE_OPEN_KIND := READ_MODE);`
    - `procedure FILE_OPEN (Status: out FILE_OPEN_STATUS; file F: FT; External_Name: in STRING; Open_Kind: in FILE_OPEN_KIND := READ_MODE);`
    - `procedure FILE_CLOSE (file F: FT);`
    - `function ENDFILE (file F: FT) return BOOLEAN;`

The file types

TELECOM PARIS
école nationale supérieure des télécommunications

- As soon as a file type is declared several associated subprograms are implicitly self-declared:
  - Read and write procedures:
    - `procedure READ (file F: FT; VALUE: out TM);`
    - `procedure WRITE (file F: FT; VALUE: in TM);`
- Declaration of an object `F1` of type `FT` :
  - `file F1: FT; FILE_OPEN(F1, "foo.txt"); -- read mode`
  - `file F1: FT is "foo.txt"; -- read mode`
  - `file F1: FT open WRITE_MODE is "foo.txt"; -- write mode`
- A file must be opened either in read mode (`READ_MODE`), in write mode (`WRITE_MODE`) or in append mode (`APPEND_MODE`)
- Note: when `TM` is an unconstrained array type the `READ` procedure is declared:
  - `procedure READ (file F: FT; VALUE: out TM; LENGTH: out NATURAL);`

The text files

TELECOM PARIS
école nationale supérieure des télécommunications

- The package `TEXTIO` from the library `STD` contains subprograms and declarations for text I/O
- One file type `TEXT`
- Two predefined `TEXT` files: `INPUT` and `OUTPUT`

```
use STD.TEXTIO.all;
```

```
type TEXT is file of STRING;
```

```
file INPUT: TEXT open READ_MODE
  is "STD_INPUT";
file OUTPUT: TEXT open WRITE_MODE
  is "STD_OUTPUT";
```

```
file FOO: TEXT;
FILE_OPEN(FOO, "foo.txt");
```

```
file BAR: TEXT;
FILE_OPEN(BAR, "bar.txt", WRITE_MODE);
```

The text files

TELECOM PARIS
école nationale supérieure des télécommunications

- Besides the implicitly declared functions and procedures `TEXT` files may be accessed line by line through `LINE` objects
- The `READLINE` and `WRITELINE` procedures read and write one entire line of a text file
- The `READ` and `WRITE` procedures read and write inside the line; they are defined for the types `BIT`, `BIT_VECTOR`, `BOOLEAN`, `CHARACTER`, `INTEGER`, `REAL`, `STRING` and `TIME`

```
type LINE is access STRING;
```

```
procedure READLINE(F: in TEXT;
  L: out LINE);
procedure WRITELINE(F: out TEXT;
  L: out LINE);
```

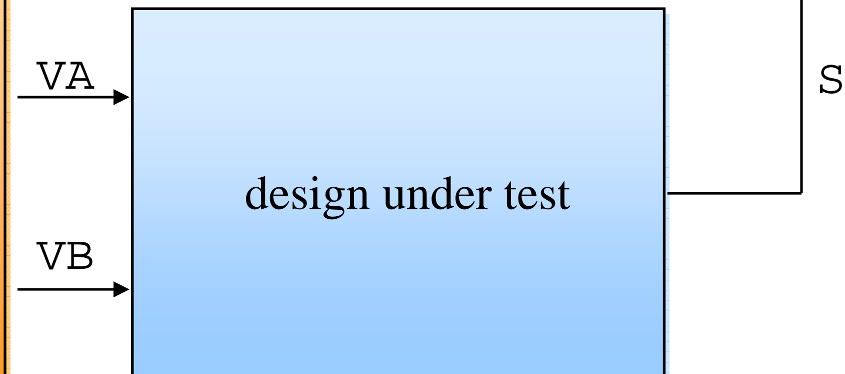
```
procedure READ(L: inout LINE;
  VALUE: out BIT;
  GOOD: out BOOLEAN);
procedure READ(L: inout LINE;
  VALUE: out BIT);
procedure WRITE(L: inout LINE;
  VALUE: in BIT;
  JUSTIFIED: in SIDE := RIGHT;
  FIELD: in WIDTH := 0);
```

Files

- Examples of use: simulation environments (reading of input patterns in a file, writing of output results in another file)

```
WRITING: process(S)
  variable L: LINE;
  file OUTPUTS: TEXT open WRITE_MODE
    is « out.dat";
begin
  WRITE(L, S);
  WRITE(L, STRING'(" at time "));
  WRITE(L, NOW);
  WRITELINE(OUPUTS, L);
end process WRITING;
```

```
READING: process
  variable L: LINE;
  file INPUTS: TEXT is « in.dat";
  variable A: BIT_VECTOR(7 downto 0);
  variable B: NATURAL range 0 to 11;
begin
  READLINE(INPUTS, L);
  READ(L, A);
  VA <= A;
  READ(L, B);
  VB <= B;
  wait for 20 ns;
end process READING;
```



The type attributes

TELECOM PARIS
école nationale supérieure des télécommunications

- The type attributes are used to examine already declared types
  - `type COLORS is (RED, YELLOW, BLUE, GREEN, ORANGE);`
  - `type FOUR_STATES is ('X', '0', '1', 'Z');`
- Some attributes are implicitly self-declared at type declaration. Examples:
  - `T’BASE – returns the base type of type T`
  - `COLORS'LEFT = RED`
  - `COLORS'RIGHT = ORANGE`
  - `FOUR_STATES'HIGH = 'Z‘`
  - `FOUR_STATES'LOW = 'X'`
- Exercise #2: what is the returned value of these 5 attributes for this subtype?
  - `subtype REVERSE_COLORS is COLORS range ORANGE downto RED;`

The type attributes

TELECOM PARIS
école nationale supérieure des télécommunications

- **Let variable** `A` **be of discrete type** `T`
  - `T'POS(A) -- returns the position of A in the type`
  - `T'VAL(N) -- returns the Nth value in the type`
  - ```
    T'SUCC(A) -- returns the successor of A
              -- T'SUCC(A) = T'VAL(T'POS(A) + 1)
    ```
  - ```
    T'PRED(A) -- returns the predecessor of A
              -- T'PRED(A) = T'VAL(T'POS(A) - 1)
    ```
  - ```
    T'LEFTOF(A) -- returns the element at the left of A
                -- in the declaration of the type
    ```
  - ```
    T'RIGHTOF(A) -- returns the element at the right of A
                 -- in the declaration of the type
    ```
- **VHDL is a strongly typed language. Any call to one of these attributes issues an error when the result is out of bounds of the type. The error will be issued at compile, elaboration or run time depending on the context**

The array types attributes

TELECOM PARIS
école nationale supérieure des télécommunications

```
type T is array (0 to 3, 7 downto 0) of BIT;
variable TAB: T;
TAB'LEFT(1)           -- returns 0
TAB'LEFT(2)           -- returns 7
TAB'RIGHT(1)          -- returns 3
TAB'RIGHT(2)          -- returns 0
TAB'HIGH(1)           -- returns 3
TAB'HIGH(2)           -- returns 7
TAB'LOW(1)            -- returns 0
TAB'LOW(2)            -- returns 0
TAB'RANGE(1)          -- returns 0 to 3
TAB'RANGE(2)          -- returns 7 downto 0
TAB'REVERSE_RANGE(2)  -- returns 0 to 7
TAB'REVERSE_RANGE(1)  -- returns 3 downto 0
TAB'LENGTH(1)         -- returns 4
TAB'LENGTH(2)         -- returns 8
```

Aggregate

TELECOM PARIS
école nationale supérieure des télécommunications

```
type OPTYPE is (ADD, SUB, MPY, DIV, JMP)
type T is array (1 to 5) of OPTYPE;
type U is
  record
    R1, R2, R3: INTEGER range 0 to 31;
    OP: OPTYPE;
  end record;
variable A: T; variable B: U;
...
A := (ADD, SUB, MPY, DIV, JMP);
A := (ADD, SUB, MPY, 5 => JMP, 4 => DIV);
A := (3 => ADD, SUB, MPY, JMP, DIV);
A := (ADD, 2 | 4 => MPY, others => DIV);
A := (SUB, 2 to 4 => DIV, 5 => JMP);
B := (0, 1, 2, ADD);
B := (OP => JMP, others => 0);
```

Operators

- **Logical:** **and**, **or**, **nand**, **nor**, **xor**, **not**
- **Relational:** `=`, `/=`, `<`, `<=`, `>`, `>=`
- **Addition:** `+`, `-`, `&` (concaténation)
- **Sign:** `+`, `-`
- **Multipliers:** `*`, `/`, **mod**, **rem**
  - A = (A / B) * B + (A **rem** B)
  - sign(A **rem** B) = sign(A)
  - **abs**(A **rem** B) < **abs**(B)
  - (-A) / B = -(A / B) = A / (-B)
  - ∃N, A = B * N + (A **mod** B)
  - sign(A **mod** B) = sign(B)
  - abs(A **mod** B) < **abs**(B)
- **Miscellaneous:** `**` (exponentiation), **abs** (absolute value)

Control structures

TELECOM PARIS
école nationale supérieure des télécommunications

```
if C1 = -65 then
  A := 10;
  B := '0';
elsif C1 = -64 then
  A := 20;
  B := '1';
elsif C1 >= -63 and C1 <= -60
then
  A := 20;
  B := '0';
elsif C1 = -59 or C1 = 187 then
  A := 30;
  B := '0';
else
  A := 30;
  B := '1';
end if;
```

```
case C1 is
  when -65 => A := 10;
              B := '0';
  when -64 => A := 20;
              B := '1';
  when -63 to -60 => A := 20;
                     B :=
'0';
  when -61 | 187 => A := 30;
                    B := '0';
  when others => A := 30;
                 B := '1';
end case;
```

Control structures

TELECOM PARIS
école nationale supérieure des télécommunications

- Loop indices must not be declared
- Loop indices are considered as constants inside the loop body
- Loop labels are optional
- Control flow may be modified by `next` and `exit` statements

```
L1: for I in 0 to 13 loop
  ...
  L2: loop
    ...
    L3: while NON_STOP_L3 loop
      ...
      exit L2 when STOP_L2;
      next L3 when CONT_L3;
      if STOP_L1 then
        exit L1;
      end if;
    end loop L3;
  end loop L2;
end loop L1;
```

The `wait` instruction

- A process must contain at least one synchronization point
  - Either implicit: a list of signals, named sensitivity list, is declared in the process header. The equivalent `wait` instruction (`wait on list`) is the last instruction of the process body. It is forbidden to put other `wait` statements in the process
  - Or explicit: no signal list in the header. The process may then contain several `wait` statements
- The complete form of the `wait` instruction:
  - `wait [on S1, S2, ...] [until CONDITION] [for DURATION];`
  - `S1`, `S2` must be signals
  - `CONDITION` is an expression that evaluates as a boolean
  - `DURATION` is a timeout

`wait` : examples

TELECOM PARIS
école nationale supérieure des télécommunications

- Eternal:
  - `wait;`
- No condition, no timeout:
  - `wait on S1, S2;`
- No list, no timeout:
  - `wait until (S1 = '0') and (S2 > ORANGE);`
- Equivalent loop:
  - ```
    loop
      wait on S1, S2;
      exit when (S1 = '0') and (S2 > ORANGE);
    end loop;
    ```
- Warning: if the condition contains no signals the `wait` becomes eternal. Classical example:
  - `wait until NOW > 10 s;`
    ```

`wait` : examples

TELECOM PARIS
école nationale supérieure des télécommunications

- These two processes are equivalent

```
signal X, Y, Z: BIT;

PA: process(X, Y)

begin

  if X = '1' then
    Z <= '1';
  elsif Y = '1' then
    Z <= '1';
  else
    Z <= '0';
  end if;

end process PA;
```

```
signal X, Y, Z: BIT;

PA: process

begin

  if X = '1' then
    Z <= '1';
  elsif Y = '1' then
    Z <= '1';
  else
    Z <= '0';
  end if;
  wait on X, Y;

end process PA;
```

Assert

TELECOM PARIS
école nationale supérieure des télécommunications

- Check a property and issues a message if it is not verified. Can also stop the simulation. Used to check the proper use of a model
  - ```
    assert CONDITION
      [report MESSAGE]
      [severity LEVEL];
    ```
  - `CONDITION`: **Boolean condition asserted true**
  - `MESSAGE`: **message (string) to print in case of violation**
  - `LEVEL`: **assertion level from predefined type :** `NOTE`, `WARNING`, `ERROR`, `FAILURE`
- **An assertion which level is** `ERROR` **or** `FAILURE` **usually stops the simulation (usually a parameter in the simulator).** `ERROR` **is the default level**
    ```

Signal assignments

TELECOM PARIS
école nationale supérieure des télécommunications

- **Signal assignments are of two types:**
  - **Inertial**
    - `SIG1 <= reject 2 ns inertial 3 after 2 ns, 5 after 6 ns, 0 after 10 ns;`
    - `SIG1 <= inertial 3 after 2 ns, 5 after 6 ns, 0 after 10 ns;`
    - `SIG1 <= 3 after 2 ns, 5 after 6 ns, 0 after 10 ns;`
  - **Transport**
    - `SIG1 <= transport 3 after 2 ns, 5 after 6 ns, 0 after 10 ns;`
- **The rejection delay is:**
  - **Always less than the delay of the first transaction in the waveform**
  - **By default equal to the delay of the first transaction in the waveform**
  - **Zero for transport**
- **Algorithm to update the driver of the signal (transport type assignment):**
  - **1) Delete old transactions which date is equal or greater than the date of the first transaction of the new waveform**
  - **2) Add new transactions at the end of the driver**

Signal assignments

- Algorithm to update the driver of the signal (inertial type assignment):
  - 1) Delete old transactions which date is equal or greater than the date of the first transaction of the new waveform
  - 2) Add new transactions at the end of the driver
  - 3) Mark the new transactions
  - 4) Old transactions which date is before the date of the first transaction of the new waveform minus the rejection limit are marked too
  - 5) Every transaction preceding a marked transaction of same value is marked
  - 6) The transaction presently driving the signal is marked
  - 7) Unmarked transactions are deleted

Signal assignments

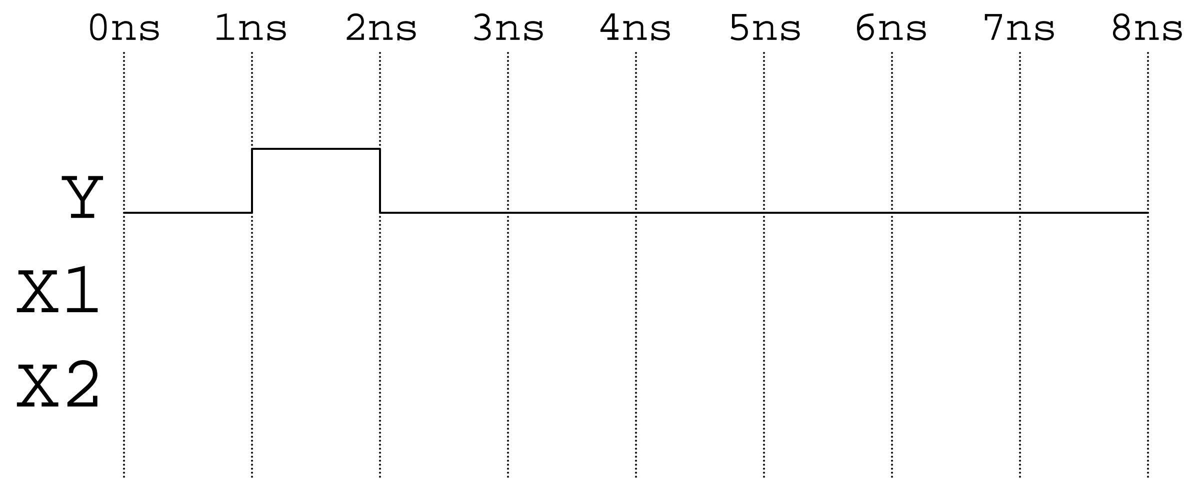
TELECOM PARIS
école nationale supérieure des télécommunications

- Let `A` be a signal which driver at date `1 ns` is:
  - `[0@0ns][5@3ns][1@5ns][3@6ns][8@12ns]`
- Let's execute the assignment:
  - `A <= transport 1 after 5 ns, 2 after 10 ns, 3 after 15 ns;`
- The driver becomes:
  - `R1: [0@0ns][5@3ns][1@5ns]`
  - `R2: [0@0ns][5@3ns][1@5ns][1@6ns][2@11ns][3@16ns]`
- If the assignment had been:
  - `A <= 1 after 5 ns, 2 after 10 ns, 3 after 15 ns; -- rejection = 5 ns`
- The driver would be:
  - `R1: [0@0ns][5@3ns][1@5ns]`
  - `R2: [0@0ns][5@3ns][1@5ns][1@6ns][2@11ns][3@16ns]`
  - `R3: [0@0ns][5@3ns][1@5ns][1@6ns][2@11ns][3@16ns]`
  - `R4: [0@0ns][5@3ns][1@5ns][1@6ns][2@11ns][3@16ns]`
  - `R5: [0@0ns][5@3ns][1@5ns][1@6ns][2@11ns][3@16ns]`
  - `R6: [0@0ns][5@3ns][1@5ns][1@6ns][2@11ns][3@16ns]`
  - `R7: [0@0ns][1@5ns][1@6ns][2@11ns][3@16ns]`

Exercise #3

- Draw the waveforms of signals `X1` and `X2`

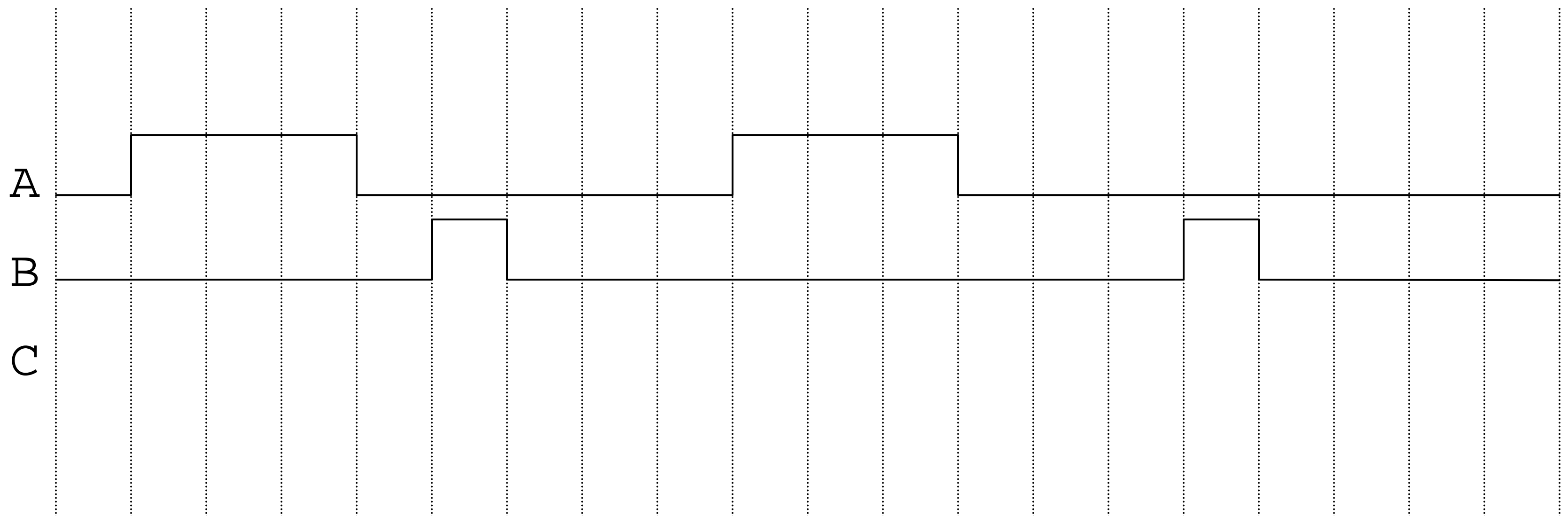
```
process(Y)
begin
  X1 <= Y after 2 ns;
  X2 <= transport Y after 2 ns;
end process;
```



Exercise #4

- Draw the waveforms of signal `C`

```
process(A, B)
begin
  C <= A or B after 2 ns;
end process;
```



Signal attributes

TELECOM PARIS
école nationale supérieure des télécommunications

- **If `S` is a signal, VHDL defines several attributes to investigate the signal's status:**
  - **The attribute `S'EVENT` is a function returning a `BOOLEAN`. It returns `TRUE` if the signal changed during the current simulation step. Example: to detect the rising edge of a clock:**
    - ```
      if (CLK = '1') and CLK'EVENT then
         Q <= D;
      end if;
      ```
  - **The attribute `S'LAST_EVENT` is a function returning a `TIME`. It returns the time elapsed since the last event on signal `S`.**
  - **The attribute `S'LAST_VALUE` is a function returning a `S`. It returns the value the signal `S` had before the last event.**
  - **The attribute `S'STABLE(T)` is a signal of type `BOOLEAN`. Its value is `TRUE` if there wasn't any event of `S` for the duration `T`.**
      ```

Signal attributes

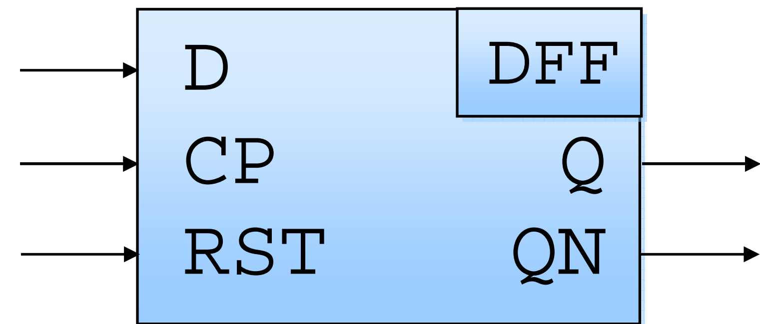
TELECOM PARIS
école nationale supérieure des télécommunications

- **The following attributes are about transactions and not events:**
  - **The attribute** `S'ACTIVE` **is a function returning a** `BOOLEAN`**. It returns** `TRUE` **if the signal had a transaction during the current simulation step**
  - **The attribute** `S'LAST_ACTIVE` **is a function returning a** `TIME`**. It returns the elapsed time since the last transaction on** `S`**.**
  - **The attribute** `S'QUIET(T)` **is a signal of type type** `BOOLEAN`**. Its value is** `TRUE` **if there was no transaction on** `S` **during** `T`**.**
  - **The attribute** `S'TRANSACTION` **is a signal of type** `BIT` **that toggles at every transaction on** `S`**.**
  - **The attribute** `S'DELAYED(T)` **is a signal of type of** `S`**. Its behavior is the behavior of** `R` **in:**
    - `R <= transport S after T;`

Exercise #5

TELECOM PARIS
école nationale supérieure des télécommunications

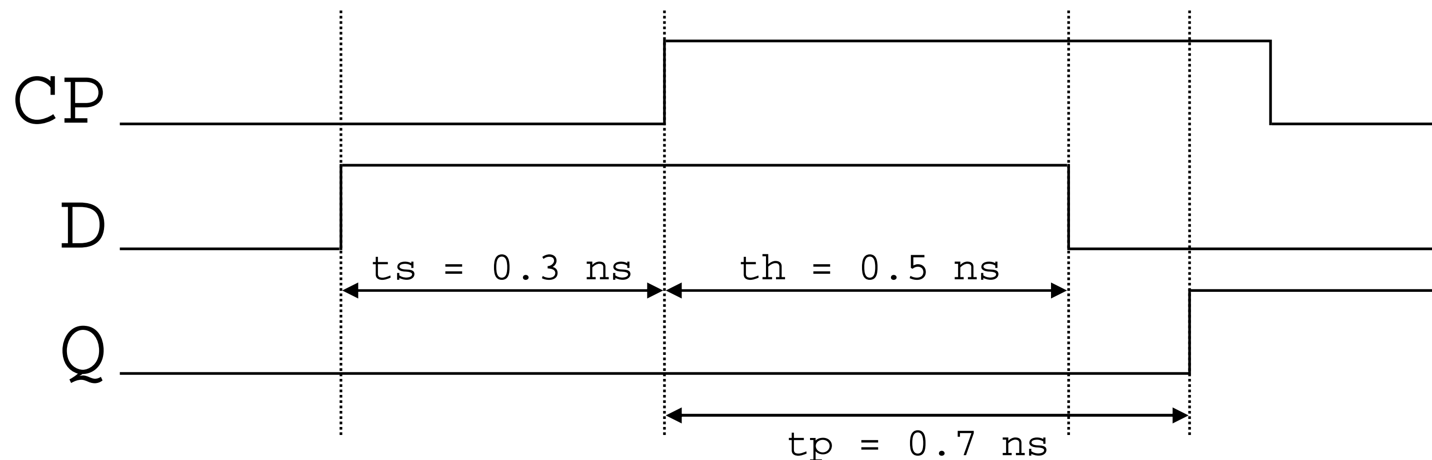
- **Design the process(es) modeling a DFF which ports are:**
  - `RST` **is an active low (`'0'`) asynchronous reset**
  - `CP` **is the clock; the `DFF` is synchronized on the rising edge of `CP`**
  - `D` **is the input**
  - `Q` **is the output, `QN` is the inverted output**

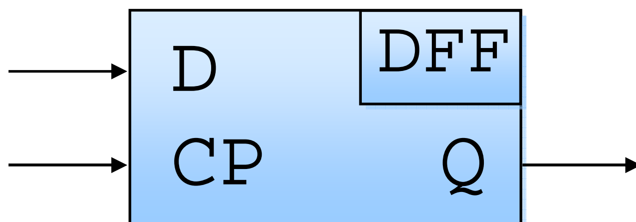


Exercise #6

TELECOM PARIS
école nationale supérieure des télécommunications

- Design the process(es) modeling a **`DFF`** which setup, hold and propagation times are represented on the following waveforms:





The subprograms

TELECOM PARIS
école nationale supérieure des télécommunications

- 2 types of subprograms :
  - Functions
    - Read only parameters
    - Return a value
  - Procedures
    - Writable parameters
    - No value returned
- Both pertain to the sequential domain
- Same usage as in every programming language

```
function F(A: BIT_VECTOR)
  return BIT is
  ...
begin
  ...
end function F;
```

```
procedure P(A: in BIT_VECTOR;
            S: out BIT) is
  ...
begin
  ...
end procedure P;
```

Name overloading

TELECOM PARIS
école nationale supérieure des télécommunications

- In VHDL, as in ADA, several subprograms may share the same name
- The compiler identifies the right subprogram depending on:
  - The name used for the call
  - The shape of the parameters:
    - Number and type of parameters
    - Type of the returned value (for functions)
- The compiler issues an error if there are more than one or zero candidates. It usually provides the list of the considered candidates
- Operators may also be overloaded by using their functional representation:
  - `function "+"(A, B: bit) return bit;`

Subprogram example

TELECOM PARIS
école nationale supérieure des télécommunications

```
function NAT2VEC(VAL: NATURAL; SIZE: POSITIVE) return BIT_VECTOR;
```

```
function NAT2VEC(VAL: NATURAL; SIZE: POSITIVE) return BIT_VECTOR is
  variable RES: BIT_VECTOR(SIZE - 1 downto 0) := (others => '0');
  variable TMP: NATURAL := VAL;
begin
  for I in 0 to SIZE - 1 loop
    exit when TMP = 0;
    if (TMP mod 2 = 1) then RES(I) := '1'; end if;
    TMP := TMP / 2;
  end loop;
  assert (TMP = 0) report "NAT2VEC: overflow"
    severity WARNING;
  return RES;
end function NAT2VEC;
```

Exercise #7

TELECOM PARIS école nationale supérieure des télécommunications

- Imagine the function `VEC2NAT`
  - `VEC2NAT("001011100") = 92`
  - `VEC2NAT("1001001")   = 73`
- Write the functions `PP` and `PG`
  - `PP(12, 18)         = 12`
  - `PP("001", "110") = "001"`
  - `PG(7, 0)           = 7`
  - `PG("001", "110") = "110"`

Combinational processes: warning

TELECOM PARIS
école nationale supérieure des télécommunications

- The sensitivity list must be complete
- All the outputs must receive a value in every execution of the process
- The best logic synthesizers issue warnings
- The violation of one of this rules will probably lead to different behaviors before and after logic synthesis
- Unwanted memory units inferred by the synthesizer are usually the indicator that one of this rules is violated

NO!

```
process(A, B)
begin
  if(A='1') then
    S<='1';
  elsif(B='1') then
    S<='1';
  end if;
end process;
```

YES

```
process(A, B)
begin
  if(A='1') then
    S<='1';
  elsif(B='1') then
    S<='1';
  else
    S<='0';
  end if;
end process;
```

Warning: the process without synchronization Point

TELECOM PARIS
école nationale supérieure des télécommunications

- It's the most frequent error. The process has no sensitivity list and no `wait` statements. Warning: `wait` statements may be present but masked by control structures (`if`, `case`, `loop`, `...`)
- Effect : the simulation time (symbolic and physical) is stuck at `0`. Nothing happens, the simulator executes the same process forever and the other processes are never executed
- Solution: stop the simulation, identify what process was running and that time and fix it

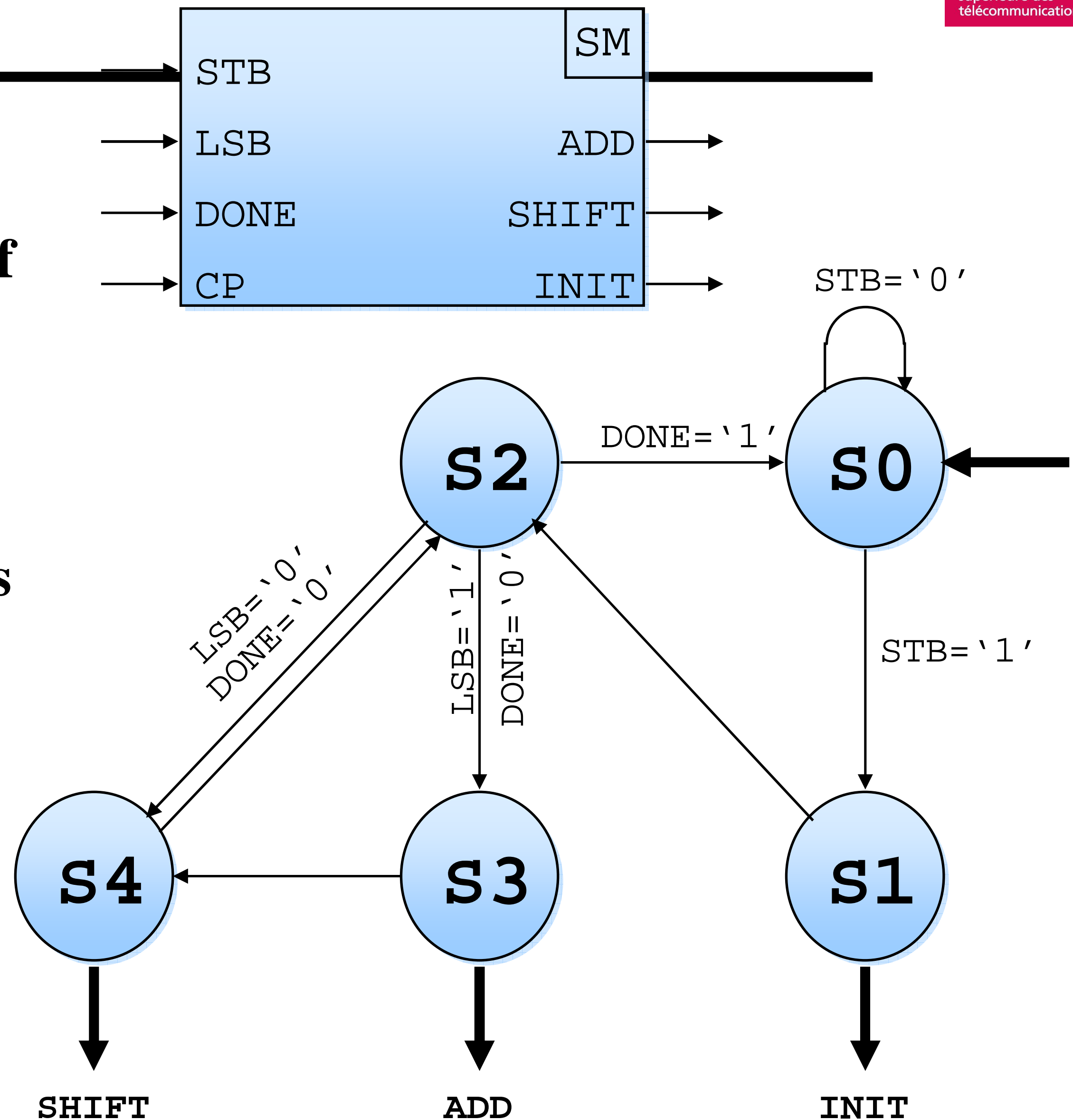
Warning: the combinational loop

TELECOM PARIS
école nationale supérieure des télécommunications

- It's a process or a collection of processes equivalent to a combinational loop with a zero physical propagation time
- Effect: the symbolic time increases very fast while the physical time is stuck. Nothing happens
- Solution: stop the simulation, identify what process was running and that time and fix it
- Note: several processes may be involved in a combinational loop

Exercise #8

- This is the interface and state diagram of a Moore finite state machine. It is synchronized on the rising edge of the clock `CP`. The inputs and outputs are active high (`'1'`). Design the process(es) needed to model this state machine



Agenda

TELECOM PARIS
école nationale supérieure des télécommunications

- Introduction
- Principals of Event Driven Simulation
- Practical Organization of Files and Projects
- Compilation Units
- Syntax
  - Sequential VHDL
  - Concurrent VHDL
- Standardized Packages
- Logic Synthesis
- Advices

Concurrent VHDL

TELECOM PARIS
école nationale supérieure des télécommunications

- VHDL models are made of concurrent instructions
- The 5 concurrent instructions are:
  - Processes
  - Entity (or component) instantiations
  - Concurrent procedure calls
  - Concurrent assertions
  - Concurrent signal assignments
- Concurrent instructions execute in pseudo-parallelism, their order of appearance in the file has no impact on the behavior

Concurrent VHDL

TELECOM PARIS
école nationale supérieure des télécommunications

- The body of an architecture is a set of concurrent instructions
- A process is ONE concurrent instructions. And it is made of several sequential instructions

```
architecture ARC of FOO is
  signal I0, I1: INTEGER;
begin

  P1: process(A, B)
  begin
    I0 <= A+B;
    I1 <= A*B;
  end process P1;

  P2: process(I0, I1)
  begin
    if(I0 /= 0) then
      S <= I1/I0;
      ALARM <= FALSE;
    else
      S <= 0;
      ALARM <= TRUE;
    end if;
  end process P2;

end architecture ARC;
```

Concurrent VHDL

TELECOM PARIS
école nationale supérieure des télécommunications

- In order to make some models simpler some alternate forms of the process are provided:
  - Concurrent procedure calls
  - Concurrent assertions
  - Concurrent signal assignments
- These concurrent instructions look like sequential instructions but they are not
- It is very important to remember that they are simplified versions of processes (short hands)

Short hands

TELECOM PARIS
école nationale supérieure des télécommunications

```
architecture AR of SUM is
begin
  PR: process(A, B, CI)
  begin
    S <= A xor B xor CI;
  end process PR;
end architecture AR;
```

```
architecture AR of SUM is
begin
  S <= A xor B xor CI;
end architecture AR;
```

Short hands

TELECOM PARIS
école nationale supérieure des télécommunications

```
architecture AR of SUM is

begin

  PR: process(A, B, CI)

  begin

    if (A = '0') then
      S <= B xor CI;
    elsif (B = '0') then
      S <= not CI;
    else
      S <= CI;
    end if;

  end process PR;

end architecture AR;
```

```
architecture AR of SUM is

begin

  S <= B xor CI when (A = '0') else
       not CI when (B = '0') else
       CI;

end architecture AR;
```

Short hands

TELECOM PARIS
école nationale supérieure des télécommunications

```
architecture AR of SUM is

begin

  PR: process(STATE)

  begin

    case STATE is
      when INIT =>
        NEXT_STATE <= RUN;
      when RUN =>
        NEXT_STATE <= WAIT;
      when WAIT =>
        NEXT_STATE <= INIT;
      when others =>
        NEXT_STATE <= INIT;
    end case;

  end process PR;

end architecture AR;
```

```
architecture AR of SUM is

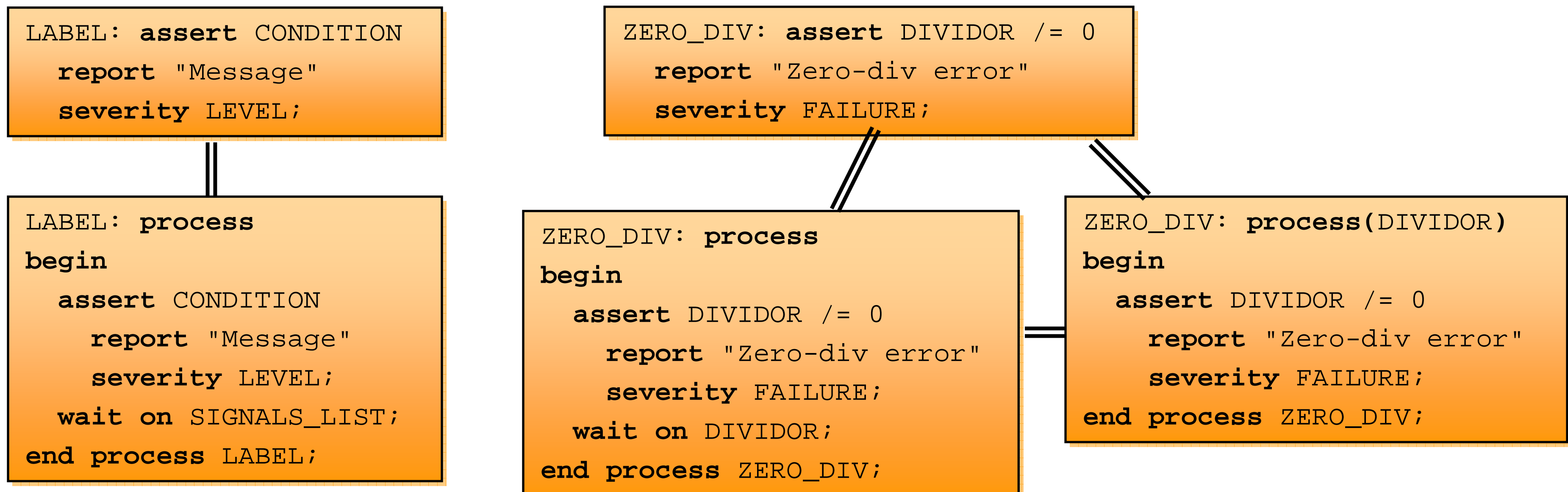
begin

  with STATE select
    NEXT_STATE <= RUN when INIT,
                  WAIT when RUN,
                  INIT when WAIT,
                  INIT when others;

end architecture AR;
```

Concurrent assertions

- The concurrent assertion is equivalent to a regular process containing a single sequential assertion where `SIGNALS_LIST` is the list of signals appearing in `CONDITION`



Concurrent procedures

TELECOM PARIS
école nationale supérieure des télécommunications

- **Concurrent procedure calls are supported. They can, for instance, be used to monitor signals. It is equivalent to a regular process where** `SIGNALS_LIST` **is the list of signals in** `PARAMETERS_LIST` **that are declared** `in` **or** `inout`

```
LABEL: PROCEDURE_NAME(PARAMETER_LISTS);
```

```
LABEL: processus
begin
 PROCEDURE_NAME(PARAMETER_LISTS);
  wait on SIGNALS_LIST;
end processus LABEL;
```

Structural VHDL

- VHDL supports hierarchical descriptions. Two mechanisms can be used:
  - Entity instantiations. Simple and fast but the less flexible. Does not allow top-down design flows. Every instantiated entity must be compiled prior compilation of embedding architecture
    - ```
      ADD16: entity WORK.ADD(DTW)
      generic map(N => 16) port map(A, B, S);
      ```
  - Component instantiations. More complex because every component instance must be bound to an actual entity (configuration). More flexible too because components are declarations. Allows compilation of top-level first (top-down design flows)
    - ```
      component ADD is generic(N: Positive);
                       port(X, Y: in Unsigned(15 downto 0);
                            Z: out Unsigned(15 downto 0));
      …
      ADD16: ADD generic map(N => 16) port map(A, B, S);
      ```

Structural VHDL

TELECOM PARIS
école nationale supérieure des télécommunications

- When compiling the compiler checks the compatibility between component or entity interface and port mapping to actual signals
- If configurations are used they can be flat or hierarchical and they allow specifying actual circuits versions
- Configurations are a powerful tool but may lead to complex descriptions

Component declaration

TELECOM PARIS
école nationale supérieure des télécommunications

- The syntax is very similar to the syntax of an entity declaration
  - ```
    component COMPONENT_NAME is
      port(PORTS_DECLARATION);
    end component COMPONENT_NAME;
    ```
- A component declaration my be found in:
  - The declarative part of an architecture
  - In a package declaration (reusable)
- Example :
  - ```
    component AND2
      port(A, B: in STD_LOGIC; C: out STD_LOGIC);
    end component;
    ```

Component instantiation

- A component instantiation (in an architecture body) has the following syntax:
  - ```
    INSTANCE_NAME: COMPONENT_NAME
      port map(PORTS_BINDING);
    ```
- Where `INSTANCE_NAME` is a unique name for this instance
- The `port map` statement binds formal input/output ports to actual signals
- Ports – signals associations may be:
  - Positional:
    - `FA: FULL_ADDER(A, B, CI, S, CO);`
  - Named:
    - `FA: FULL_ADDER(A => X, B => Y, CI => Z, S => U, CO => V);`
  - As with aggregates both syntaxes may be mixed
    ```

Component instantiation: example

```
entity AND2 is
  port(A, B: in BIT;
       C: out BIT);
end entity AND2;

architecture ARC of AND2 is
begin
  C <= A and B;
end architecture ARC;

configuration AND2_CFG of AND2 is
  for ARC
  end for;
end configuration AND2_CFG;
```

```
entity AND3 is
  port(I1, I2, I3: in BIT;
       S: out BIT);
end entity AND3;
```

```
architecture STR of AND3 is
  component AND2
    port(A, B: in BIT; C: out BIT);
  end component;
  signal TMP: BIT;
begin
  A0: AND2 port map(A => I1, B => I2,
                    C => TMP);
  A1: AND2 port map(A => I3, B => TMP,
                    C => S);
end architecture STR;

configuration AND3_CFG of AND3 is
  for STR
    for all: AND2
      use configuration WORK.AND2_CFG;
    end for;
  end for;
end configuration AND3_CFG;
```

Component-entity binding rules

TELECOM PARIS
école nationale supérieure des télécommunications

- When binding a component instance to an actual entity the exact one to one association between component ports and entity ports must be known
- If the entity has a different interface (port names, order) the configuration must also associate component ports and entity ports (with a `port map` statement):
  - ```
    for all: NAND2 use entity WORK.AND2(BEHAVE)
      port map(I1 => A, I2 => B, S => C);
    ```
    ```

Generic parameters

TELECOM PARIS
école nationale supérieure des télécommunications

- VHDL offers several mechanisms to build generic descriptions. Generic parameters are one of them
- Inside the associated architecture a generic parameter is considered as a constant

```
entity ADD is
  generic(N: POSITIVE range 1 to 32 := 8);
  port(A, B: in  BIT_VECTOR(N - 1 downto 0);
       CI:   in  BIT;
       S:    out BIT_VECTOR(N - 1 downto 0);
       CO:   out BIT);
end entity ADD;
```

Generic parameters

TELECOM PARIS
école nationale supérieure des télécommunications

- Generic parameters may have a default value
- The actual value of a generic parameter may be given by:
  - The component or entity instantiation statement (`generic map`)
  - De default value in component declaration
  - The default value in associated entity declaration

Generic parameters, example of use

```
architecture ARC of MUL is
  component ADD
    generic(N: POSITIVE range 1 to 32 := 8);
    port(A, B: in  BIT_VECTOR(N - 1 downto 0);
         S:    out BIT_VECTOR(N - 1 downto 0));
  end component;
  signal X1, X2, S: BIT_VECTOR(16 downto 0);
. . .
begin
. . .
  I_ADD: ADD generic map(N => 17);
             port map(A => X0, B => X1, S => Z);
. . .
end architecture ARC;
```

The `generate` statement

- The `generate` statement are another mechanism to build generic descriptions
- They are the concurrent equivalent of the sequential for loops and if statements

```
architecture RTL of ADD is
  component ADD1
    port(A, B, CI: in  BIT;
         S, CO:    out BIT);
  end component;
  signal C: BIT_VECTOR(N downto 0);
begin
  G: for I in 0 to N - 1 generate
    IA: ADD1 port map(A(I), B(I), C(I), S(I), C(I + 1));
  end generate G;
  C(0) <= CI;
  CO <= C(N);
end architecture RTL;
```

The `generate` statement

```
entity FA
  port(X, Y, Z: in BIT;
       I, J:    out BIT);
end entity FA;

architecture BEV of FA is
begin
  I <= X xor Y xor Z;
  J <= (X and (Y or Z)) or
       (Y and Z);
end architecture BEV;
```

```
configuration CFG of ADD is
  for RTL
    for G
      for IA: ADD1
        use entity BIB.FA(BEV);
        port map(X => A, Y => B,
                 Z => CI, I => S,
                 J => CO);
      end for;
    end for;
  end for;
end configuration CFG;
```

Resolution functions

TELECOM PARIS
école nationale supérieure des télécommunications

- When a signal is driven by multiple processes it has as many drivers as source processes. In order to compute its actual value a function is needed; the resolution function. This function is associated to the type of the signal which is said to be a resolved signal:

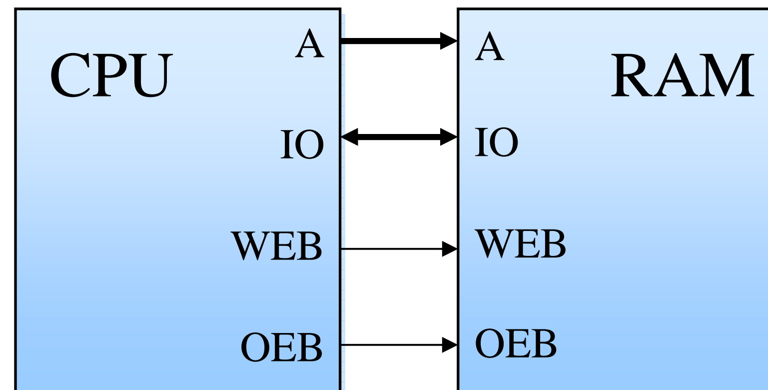
```
function OU_CABLE(VAL: BIT_VECTOR)
  return BIT is
begin
  if (VAL'LENGTH = 0) then
    return '0';
  end if;
  for I in VAL'RANGE loop
    if (VAL(I) = '1') then
      return '1';
    end if;
  end loop;
  return '0';
end function OU_CABLE;
```

```
subtype RESOLVED_BIT is
  OU_CABLE BIT;

signal S1: RESOLVED_BIT;
signal S2: OU_CABLE BIT;
```

Resolution functions: example of use

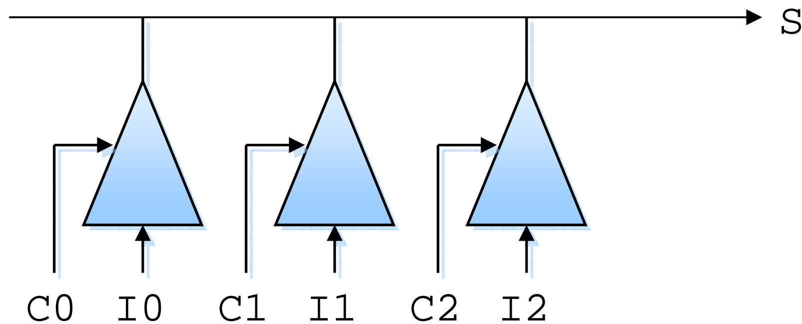
TELECOM PARIS
école nationale supérieure des télécommunications




```
entity CPU is
  port(A: out STD_ULOGIC_VECTOR(7 downto 0);
       IO: inout STD_LOGIC_VECTOR(15 downto 0);
       WEB, OEB: out STD_ULOGIC);
end entity CPU;
architecture ARC of CPU is
. . .
  IO <= "0100110001101111"; -- ecriture
. . .
  IO <= "ZZZZZZZZZZZZZZZ"; -- lecture
. . .
end architecture ARC;
```

Resolution functions: example of use

TELECOM PARIS
école nationale supérieure des télécommunications




```
signal S: STD_LOGIC;
. . .
S <= I0 when C0 = '1' else
      'Z' when others;
. . .
S <= I1 when C1 = '1' else
      'Z' when others;
. . .
S <= I2 when C2 = '1' else
      'Z' when others;
. . .
```

Resolved type: beware

TELECOM PARIS
école nationale supérieure des télécommunications

- Never use a resolved type if it's not needed
- Compiler and linker would not help detecting unwanted shortcuts
- Systematically using resolved types (`STD_LOGIC` instead of `STD_ULOGIC`) is thus dangerous
- Systematically using resolved types (`STD_LOGIC` instead of `STD_ULOGIC`) also slows down the simulations (resolving a conflict takes time)

Agenda

TELECOM PARIS
école nationale supérieure des télécommunications

- Introduction
- Principals of Event Driven Simulation
- Practical Organization of Files and Projects
- Compilation Units
- Syntax
  - Sequential VHDL
  - Concurrent VHDL
- Standardized Packages
- Logic Synthesis
- Advices

Library STD, package STANDARD

TELECOM PARIS
école nationale supérieure des télécommunications

- The STD package is a standard one that comes with any VHDL design environment
- The STD library is implicitly declared in each source file. It is never necessary to re-declare it
- The STANDARD package defines the base types:
  - **Enumerated:** BOOLEAN, BIT, CHARACTER, SEVERITY_LEVEL
  - **Numeric:** INTEGER, NATURAL, POSITIVE, REAL, TIME
  - **Composite:** STRING, BIT_VECTOR
- And the NOW function
- The STANDARD package is implicitly declared in each source file. It is never necessary to re-declare it

Library STD, package TEXTIO

- **This package is dedicated to ASCII files I/O**
- **Unfortunately it is very poor**
- **It defines:**
  - **Types** `LINE`, `TEXT`, `SIDE` **and** `WIDTH`
  - **Files** `INPUT` **and** `OUTPUT`
  - **Procedures** `READLINE`, `READ`, `WRITELINE` **and** `WRITE`
  - **Function** `ENDLINE`
- **It has to be explicitly declared:**
  - `use STD.TEXTIO.all;`

Library `IEEE`, package `STD_LOGIC_1164`

TELECOM PARIS
école nationale supérieure des télécommunications

- **Defines a multi-valued logic as an enumerated type:**
  - ```
    type STD_ULOGIC is (
        'U',  -- Uninitialized
        'X',  -- Forcing Unknown
        '0',  -- Forcing  0
        '1',  -- Forcing  1
        'Z',  -- High Impedance
        'W',  -- Weak Unknown
        'L',  -- Weak 0
        'H',  -- Weak 1
        '-'   -- Don't care);
    ```
- **Also defines a resolution function for** `STD_ULOGIC` **and the associated resolved type:** `STD_LOGIC`
- **And also defines the corresponding vector types** `STD_ULOGIC_VECTOR` **and** `STD_LOGIC_VECTOR`
    ```

Library IEEE, package STD_LOGIC_1164

TELECOM PARIS
école nationale supérieure des télécommunications

- **Also defines:**
  - **Some resolved sub-types of** STD_ULOGIC **(**X01**,** X01Z**, etc.)**
  - **All the logic operators for** STD_ULOGIC, STD_LOGIC, STD_ULOGIC_VECTOR **and** STD_LOGIC_VECTOR
  - **Conversion functions from and to types** BIT **and** BIT_VECTOR
  - **Functions** RISING_EDGE **and** FALLING_EDGE **to detect edges on signals of type** STD_ULOGIC
  - **Functions** IS_X **to detect undefined values (**'U', 'X' , 'Z' , 'W' , '-'**)**

Library IEEE, others packages

TELECOM PARIS
école nationale supérieure des télécommunications

- The IEEE library contains some other useful packages
- Two are dedicated to arithmetic on vectors:
  - NUMERIC_BIT:
    - Defines types SIGNED and UNSIGNED (arrays of BIT)
    - Overloads the arithmetic, logic and relational operators for those types
    - Defines the conversion function from and to Integer types
    - Adds various dedicated functions (rotations, shifts, etc.)
  - NUMERIC_STD
    - Defines types SIGNED and UNSIGNED (arrays of STD_LOGIC)
    - Overloads the arithmetic, logic and relational operators for those types
    - Defines the conversion function from and to Integer types
    - Adds various dedicated functions (rotations, shifts, etc.)

Agenda

TELECOM PARIS
école nationale supérieure des télécommunications

- Introduction
- Principals of Event Driven Simulation
- Practical Organization of Files and Projects
- Compilation Units
- Syntax
  - Sequential VHDL
  - Concurrent VHDL
- Standardized Packages
- **Logic Synthesis**
- Advices

Combinational logic inference

TELECOM PARIS
école nationale supérieure des télécommunications

- Combinational logic is inferred from variables or signals that are:
  - Unconditionally assigned
  - Before being read
  - Every time the process resumes
- Or
  - Conditionally assigned
  - For every possible condition
  - Every time the process resumes

```
process (a, b, c)
begin
  z <= a+b+c;
end process;
```

```
process (a, b, c)
begin
  if b = '1' then
    z <= a;
  else
    z <= c;
  end if;
end process;
```

Latches inference

TELECOM PARIS
école nationale supérieure des télécommunications

- Latches are inferred from variables or signals that are:
  - Not affected for every process execution
  - On a particular level of a control signal
  - With or without initialization
    - Synchronous
    - Asynchronous

```
process(data_in, enable,
         set_sig, reset_sig)
begin
  if enable = '1' then
    if set_sig = '1' then
      data_out <= '1';
    elsif reset_sig = '1' then
      data_out <= '0';
    else
      data_out <= data_in;
    end if;
  end if;
end process;
```

```
process(data_in, enable)
begin
  if enable = ‘1’ then
    data_out <= data_in;
  end if;
end process;
```

Latches inference

TELECOM PARIS
école nationale supérieure des télécommunications

- Latches are inferred from variables or signals that are:
  - Not affected for every process execution
  - On a particular level of a control signal
  - With or without initialization
    - Synchronous
    - Asynchronous

```
process(data_in, enable,
        set_sig, reset_sig)
begin
  if set_sig = '1' then
    data_out <= '1';
  elsif reset_sig = '1' then
    data_out <= '0';
  elsif enable = '1' then
    data_out <= data_in;
  end if;
end process;
```

Flip-flops inference

TELECOM PARIS
école nationale supérieure des télécommunications

- D flip-flops are inferred from variables or signals that are:
  - Not affected for every process execution
  - On a particular edge of a control signal (clock)
  - With or without initialization
    - Synchronous
    - Asynchronous

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

```
process(clk)
begin
  if clk = '1' and
     clk'event then
    if set_sig = '1' then
      data_out <= '1';
    elsif reset_sig = '1' then
      data_out <= '0';
    else
      data_out <= data_in;
    end if;
  end if;
end process;
```

Flip-flops inference

TELECOM PARIS
école nationale supérieure des télécommunications

- D flip-flops are inferred from variables or signals that are:
  - Not affected for every process execution
  - On a particular edge of a control signal (clock)
  - With or without initialization
    - Synchronous
    - Asynchronous

```
process(clk, set_sig, reset_sig)
begin
  if set_sig = '1' then
    data_out <= '1';
  elsif reset_sig = '1' then
    data_out <= '0';
  elsif clk = '1' and clk'event then
    data_out <= data_in;
  end if;
end process;
```

Clock edge specification

TELECOM PARIS
école nationale supérieure des télécommunications

- Clock edge specification supported by most synthesizers:
  - `if (clk'event and clk = '1') then`
  - `wait until (clk'event and clk = '1');`
  - `if (rising_edge(clk)) then`
  - `wait until rising_edge(clk);`
  - `if (clk'event and clk =  ’0') then`
  - `wait until (clk'event and clk =  ’0');`
  - `if (falling_edge(clk)) then`
  - `wait until falling_edge(clk);`
  - ...

Unwanted latches inference

TELECOM PARIS
école nationale supérieure des télécommunications

❑ Unwanted latches inference

```
signal curr_state, next_state, modifier:
    std_logic_vector(2 downto 0);
process(curr_state, modifier)
begin
  case curr_state is
    when "000"  => next_state <= "100" or  modifier;
    when "001"  => next_state <= "110" or  modifier;
    when "010"  => next_state <= "001" and modifier;
    when "100"  => next_state <= "101" and modifier;
    when "101"  => next_state <= "010" or  modifier;
    when "110"  => next_state <= "000" and modifier;
    when others => null;
  end case;
end process;
```

Unwanted latches inference

TELECOM PARIS
école nationale supérieure des télécommunications

❑ Avoid unwanted latches inference

```
signal curr_state, next_state, modifier:
    std_logic_vector(2 downto 0);
process(curr_state, modifier)
begin
  next_state <= "100";
  case curr_state is
    when "000"  => next_state <= "100" or  modifier;
    when "001"  => next_state <= "110" or  modifier;
    when "010"  => next_state <= "001" and modifier;
    when "100"  => next_state <= "101" and modifier;
    when "101"  => next_state <= "010" or  modifier;
    when "110"  => next_state <= "000" and modifier;
    when others => null;
  end case;
end process;
```

Unwanted latches inference

TELECOM PARIS
école nationale supérieure des télécommunications

❑ Avoid unwanted latches inference

```
signal curr_state, next_state, modifier:
    std_logic_vector(2 downto 0);
process(curr_state, modifier)
begin
  case curr_state is
    when "000"  => next_state <= "100" or  modifier;
    when "001"  => next_state <= "110" or  modifier;
    when "010"  => next_state <= "001" and modifier;
    when "100"  => next_state <= "101" and modifier;
    when "101"  => next_state <= "010" or  modifier;
    when "110"  => next_state <= "000" and modifier;
    when others => next_state <= "100";
  end case;
end process;
```

Supported loops

TELECOM PARIS
école nationale supérieure des télécommunications

- **Loops**
  - **Can be synthesized**
  - **But they are unrolled first**
  - **Bounds of `for` loops must be static:**
    - **This is synthesizable: `for I in 0 to 7 loop`**
    - **Not this: `for I in F(X) to G(Y) loop` (except when `X` and `Y` are compile-time constants)**
  - **Conditions of `while` loops must be static**
    - **This is synthesizable: `while FALSE loop`**
    - **Not this: `while C(X, Y) loop` (except when `X` and `Y` are compile-time constants)**
    - **The `while` and infinite loops are usually not synthesizable**

`wait` statement

TELECOM PARIS
école nationale supérieure des télécommunications

- The `wait` statements are sometimes supported but with limitations. Examples of such limitations:
  - One single `wait` statement per process
  - Always as the first (or last) instruction
  - Only for clock edge specification
  - With clock as single signal in `on` part…
  - ...

Synthesis options

TELECOM PARIS
école nationale supérieure des télécommunications

- Synthesis options
  - May be special comments
  - VHDL attributes
  - Multiple usages:
    - `synthesis on/off`
    - `translate on/off`
    - Set and reset
    - Arithmetic architectures
    - Encoding of enumerated types
    - Wired or multiplexed logic (`case`)
    - Coding and optimization of state machines
    - Semantics of resolution functions
    - ...

Synthesis packages

TELECOM PARIS
école nationale supérieure des télécommunications

- Some packages are dedicated to logic synthesis
  - IEEE standard:
    - `IEEE.STD_LOGIC_1164`
    - `IEEE.NUMERIC_BIT`
    - `IEEE.NUMERIC_STD`
  - Proprietary packages:
    - Attributes declarations – Synthesis options
    - Proprietary arithmetic functions
    - Macro-functions VHDL models
    - VHDL models of standard cells libraries
    - ...

NUMERIC_BIT and NUMERIC_STD

TELECOM PARIS
école nationale supérieure des télécommunications

- **Standard arithmetic on** BIT **(or** STD_ULOGIC**) based types**
  - **Types** SIGNED **and** UNSIGNED
  - **Classical arithmetic operators are overloaded for** SIGNED **and** UNSIGNED
  - SIGNED **and integres or** UNSIGNED **and integers can be mixed in expressions**
  - **Integer to and from vector conversion functions are defined:**
    - TO_INTEGER
    - TO_SIGNED, TO_UNSIGNED
  - **Vector types being compatible one with the other the corresponding conversion functions all have the same name as the destination type:**
    - SIGNED
    - UNSIGNED
    - BIT_VECTOR

Dangers

- **Beware:**
  - **Unwanted registers**
    - **Flip-flops**
    - **Latches**
  - **Incomplete sensitivity lists**
  - **Loops**
  - **Combinational loops**
  - **Sign in arithmetic operations**
  - **Partitioning**
  - **Portability**
    - **The semantics for synthesis is not standard**
    - **Proprietary packages are … proprietary**

Agenda

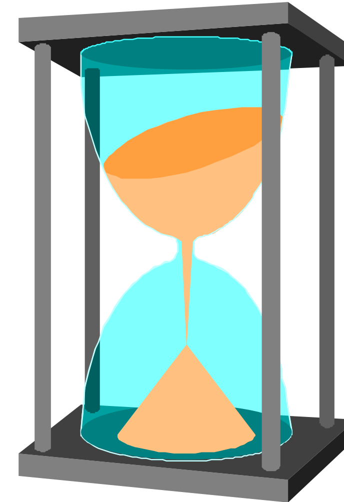
TELECOM PARIS
école nationale supérieure des télécommunications

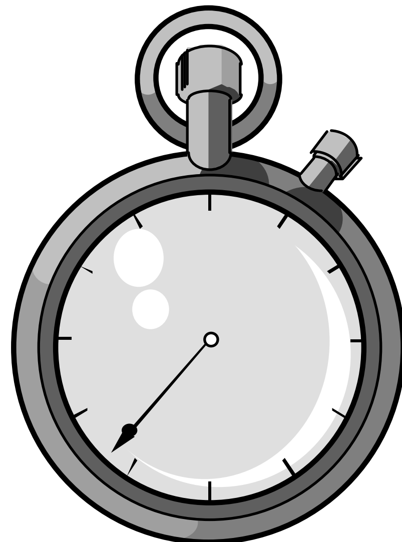
- Introduction
- Principals of Event Driven Simulation
- Practical Organization of Files and Projects
- Compilation Units
- Syntax
  - Sequential VHDL
  - Concurrent VHDL
- Standardized Packages
- Logic Synthesis
- **Advices**

Synchronize

TELECOM PARIS
école nationale supérieure des télécommunications

```
...
wait for 10 * PERIODE;
...
```





```
...
for I in 0 to 9 loop
  wait until (CK = '1');
end loop;
...
```

For Synthesis separate synchronous and combinational

TELECOM PARIS
école nationale supérieure des télécommunications

```
PR: process(CK)
begin
  if (CK = '1') then
    S <= S + 1;
  end if;
end process PR;
```

```
PRC: process(SORTIE)
begin
  E <= S + 1;
end process PRC;
```

```
PRS: process(CK)
begin
  if (CK = '1') then
    S <= E;
  end if;
end process PRS;
```

To speed up simulation, avoid signals

```
architecture ARC of REGS is
  signal A0, A1: BIT;
begin
  REGS_PR: process(CK)
  begin
    if (CK = '1') then
      A0 <= DIN;
      A1 <= A0;
      DOUT <= A1;
    end if;
  end process REGS_PR;
end ARC;
```

```
architecture ARC of REGS is
begin
  REGS_PR: process(CK)
    variable A0, A1: BIT;
  begin
    if (CK = '1') then
      DOUT <= A1;
      A1 := A0;
      A0 := DIN;
    end if;
  end process REGS_PR;
end ARC;
```

For synthesis count registers

TELECOM PARIS
école nationale supérieure des télécommunications

```
architecture ARC of REGS is

begin

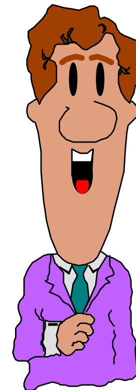
  REGS_PR: process(CK) -- 3 bits registers
    variable A0, A1: BIT;
  begin
    if (CK = '1') then
      DOUT <= A1;
      A1 := A0;
      A0 := DIN;
    end if;
  end process REGS_PR;

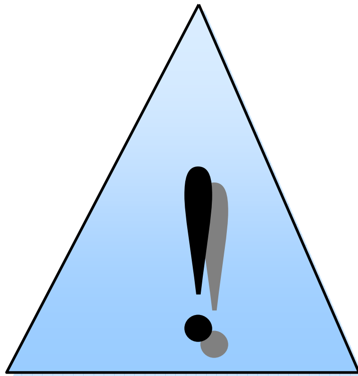
end ARC;
```

Comment a lot and don't mix models and reality

TELECOM PARIS
école nationale supérieure des télécommunications

- One line of code = 10 lines of comments
- HDL /= matériel





Master complexity

TELECOM PARIS
école nationale supérieure des télécommunications

- Very frequently the problem turns out to be much more complex than initially expected. In such situations the designer progressively piles up modifications
- Effect: il the problem is serious the code rapidly becomes a unusable piece of code, impossible to understand or maintain
- Solution: Restart from scratch, taking into account the discovered new problems. Re-design the partitioning, the data structures, rewrite everything

Hardware and software

TELECOM PARIS
école nationale supérieure des télécommunications

- VHDL is a programming language but it's main goal is to model hardware. Writing VHDL without a clear idea of the underlying hardware cannot give good results
- Effects: impossibility to refine the code into a synthesizable form, different behaviors before and after synthesis
- Solution: think hardware first. To model a hardware architecture you must have a clear idea of it