

Introducció al llenguatge VHDL

- 1. Introducció**
- 2. Elements bàsics**
- 3. Tipus de dades**
- 4. Primitives de descripció bàsiques**
- 5. Descripció de flux de dades**
- 6. Descripció de comportament**
- 7. Descripció estructural**
- 8. Organització del disseny**

1. Introducció

- **Avantatges de la utilització de llenguatges de descripció hardware (HDL – Hardware Description Language)**
 - Millor productivitat (abstracció del disseny)
 - Re-utilització flexible de components
 - Escurçament del cicle de disseny
 - Millor documentació (descripció de comportament)
 - Descripció jeràrquica (aproximació *top-down*)
 - Descripció **independent de la tecnologia**
 - Permet diferents alternatives d'implementació

- **HDLs vs Llenguatges de programació:**

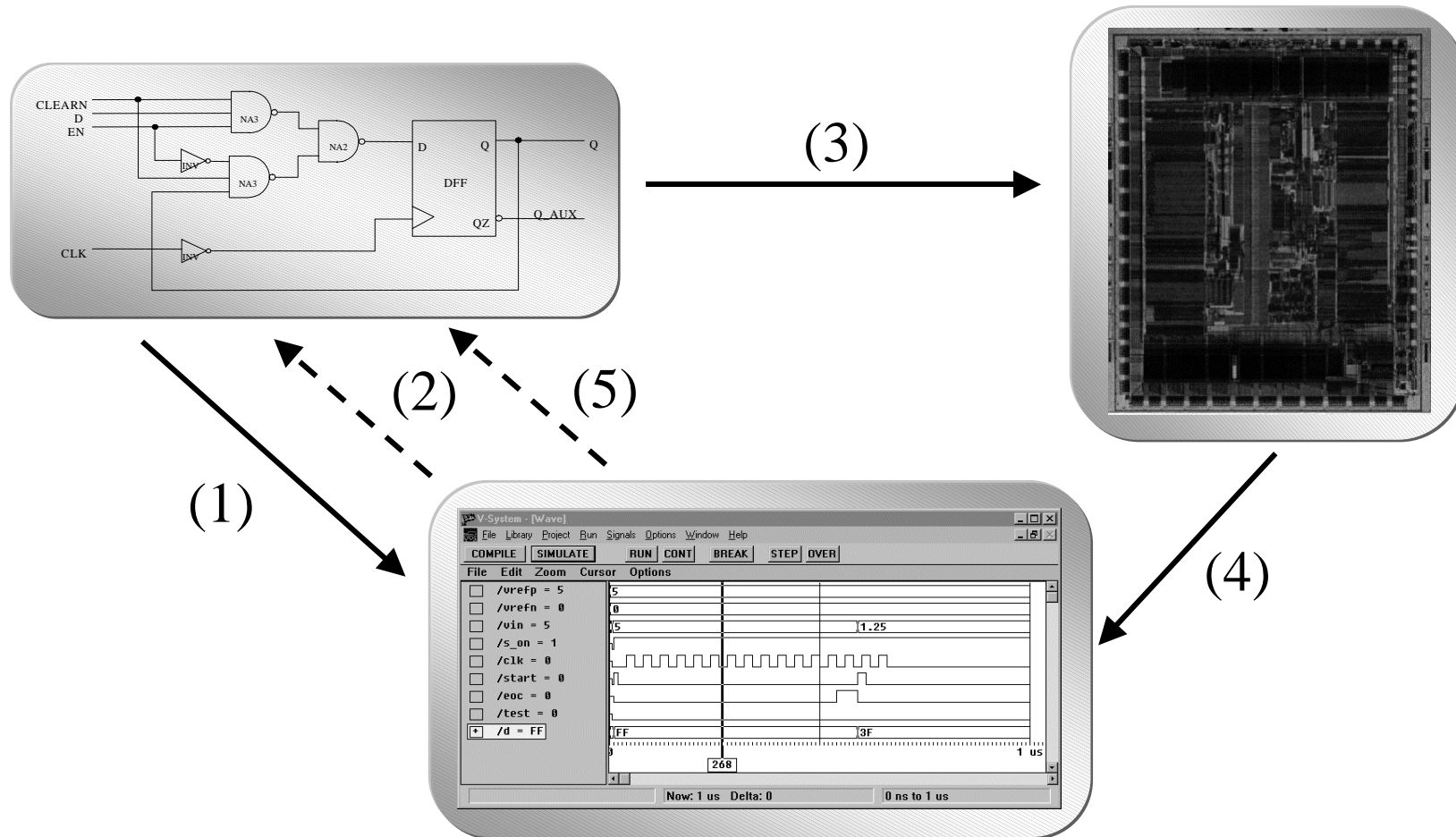
1) HDLs descriuen **sistemes concurrents**

Llenguatges de programació **normalment** especifiquen **sistemes seqüencials**

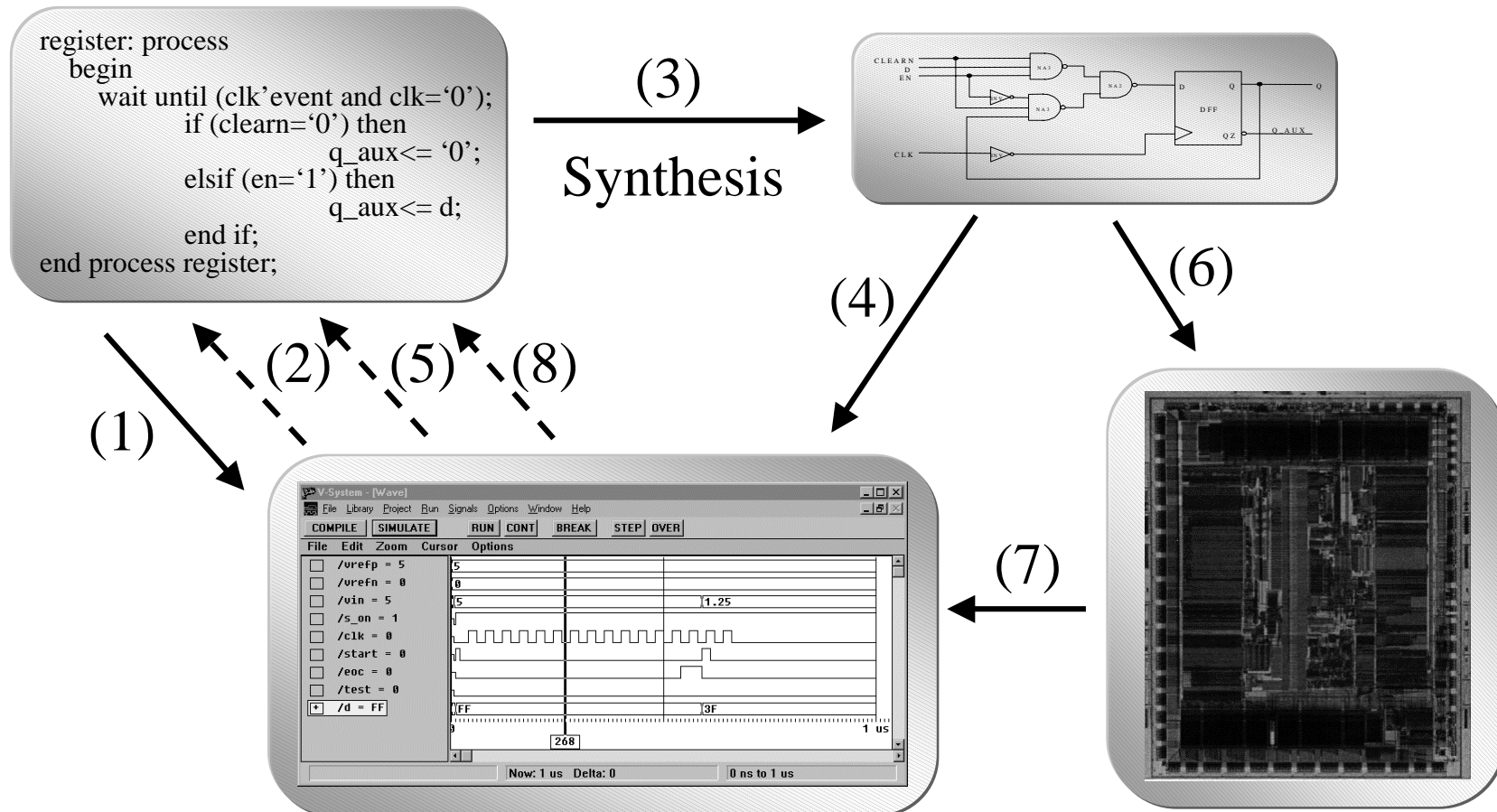
2) HDLs han de permetre descripcions estructurals y de comportament

3) HDLs han de permetre l'especificació de característiques temporals i restriccions del sistema

- **Flux de disseny tradicional:**



- Flux de disseny basat en HDLs:



- **Característiques del llenguatge VHDL:**

- Projecte VHSIC (DoD, USA)
- Objectiu inicial: facilitar la documentació
- Desenvolupat per TI, Intermetrics, IBM (1985)
- Estàndar IEEE (1076) des de 1987 (revisió 1993)
- Vinculat a tipus de dades (ADA)
- Nivells bàsics de descripció:
 - flux de dades, estructural, comportament
- Objectes bàsics:
 - constants, variables, senyals i fitxers

• Bibliografia:

➤ Digital design:

J. F. Wakerly, “Digital Design. Principles & Practices”, (3rd. Edition), Prentice-Hall, 2000.

➤ VHDL Language:

P. Ashenden, “The Designer’s Guide to VHDL”, Morgan Kaufmann, 1996.

➤ VHDL & Synthesis:

K.C. Chang, "Digital Design and Modeling with VHDL and Synthesis", IEEE Computer Society Press, 1997.

2. Elements bàsics

- **Comentaris:** --
 - Això és un comentari
- **Identificadors (etiquetes):**
 - caràcters alfabètics + digits + _
 - inici: **sempre** amb caràcter alfabètic
 - **no poden** acabar amb _
 - **no poden** contenir dos _ succesius
- **Caràcters:** 'C'
- **Strings:** "NoMés Una LíNIA"
- **Números (sencers o reals):** 23 45.2E-4 16#A4#E-7
- **Senyals:** Nodes d'un sistema

- **Bit strings:** [B|O|X]”101”
 - **Constants:**
`constant n_bits_per_word: integer:=12;`
`constant pi: real:=3.1415926535`
 - **Variables:**
`variable counter: integer:=0;`
 - **Final de sentència:** ;
 - **Assignacions:** <= (senyals) := (variables i constants)
-
- **Assert primitive:** Monitorització de condicions
`assert vdd/=5 report “VDD not connected”
severity warning;`

3. Tipus de dades (escalars)

➤ **Declaració de tipus:**

type mean_weight **is** 65 **to** 80;

type logic_level **is** ('0', '1', 'Z', 'X');

➤ **Tipus sencer:**

type word_type **is** (4, 8, 16, 32, 64);

variable opcode: word_type:=4;

➤ **Tipus real:**

type probability **is** 0.0 **to** 1.0;

➤ **Tipus físic:**

type resistance **is range** 0 **to** 1E9
units

ohm;

kohm=1000 ohm;

Mohm = 1000 kohm

end units resistance;

- **Tipus time:** Resolució temporal mínima: *fs*
- **Tipus character:** Conjunt de caràcters ISO de 8 bits
- **Tipus boolean:**
 type boolean is (false, true);
- **Tipus bit:**
 type bit is ('0', '1');
- **Declaració de subtipus:**
 subtype pointer is integer range 15 downto 0;
- **Conversió entre tipus:** **real(84) integer(36.9);**
- **Operadors:**
 **** abs not**
 *** / mod rem**
 + - &
 sll srl sla sra rol ror
 = /= < <= > >=
 and or nan nor xor xnor

Precedència

- **Tipus de dades compostos**

- **Array:** Conjunt d'elements del mateix tipus

type word **is** (0 to 15) **of** bit;

variable register_A: word:=X"A42E";

carry := register_A(4);

type curve_points **is** (0 to 3) **of** integer;

variable curve_1: curve_points:=

(1=>0, 2=>5, **others** =>0);

type RAM **is** (natural **range** <>) **of** bit_vector(7 **downto** 0);

variable control_ram: ram(0 to 1024);

type bidim_array **is** (0 to 3, 0 to 7) **of** bit;

➤ **Record:** Camps amb **tipus diferents**

type instruction **is** **record**

opcode: bit_vector(7 **downto** 0);

source_reg: bit_vector(3 **downto** 0);

destination_reg: bit_vector(3 **downto** 0);

end record instruction;

variable instruction_reg: instruction;

instruction_reg.opcode:=data_cpu(7 **downto** 0);

instruction_reg.source_reg:=data_cpu(11 **downto** 8);

instruction_reg.destination_reg:=data_cpu(15 **downto** 12);

➤ **Atributs:** Especifiquen propietats d'un element

Atributs escalars

Attribute	Type of T	Type of result	Result
T'left	any scalar type or subtype	same as T	leftmost value in T
T'right	“	“	rightmost value in T
T'low	“	“	lowest value in T
T'high	“	“	highest value in T
T'ascending	“	boolean	true if T is an ascending range, false otherwise
T'image(x)	“	string	textual representation of the value x from type T
T'value(s)	“	base type of T	value in T represented by the string s
T'pos(x)	any discrete type or subtype	universal integer	position of x in T
T'val(x)	“	base type of T	value at position x in T

Atributs escalars (cont.)

Attribute	Type of T	Type of result	Result
T'succ(x)	any discrete type or subtype	base type of T	value in T at position one greater than that of x
T'pred(x)	“	“	value in T at position one less than that of x
T'leftof(x)	“	“	value in T at position one to the left of x
T'rightof(x)	any discrete or physical type or subtype	“	value in T at position one to the right of x
T'base	any type or subtype	“	base type of type T, only allowed as a prefix of another attribute

Atributs d'array


Attribute	Result
A'left(n)	left bound of index range of dimension n of A
A'right(n)	right bound of index range of dimension n of A
A'low(n)	lower bound of index range of dimension n of A
A'high(n)	upper bound of index range of dimension n of A
A'range(n)	index range of dimension n of A
A'reverse_range(n)	reverse of index range of dimension n of A
A'length(n)	length of index range of dimension n of A
A'ascending(n)	true if index range of dimension n of A is ascending, false otherwise

Atributs de senyal

Attribute	Type of result	Result
S'delayed(t)	base type of S	a signal that takes on the same value as S but is delayed by time T
S'stable(t)	boolean	a boolean signal that is true if there has been no event on S in the time interval t up to the current time, otherwise false
S'quiet(t)	boolean	a boolean signal that is true if there has been no transaction on S in the time interval t up to the current time, otherwise false
S'transaction	bit	implicit bit signal, which changes its value each time there is a transaction on S
S'event	boolean	true if there is a transaction on S in the current simulation cycle, otherwise false
S'active	boolean	true if there is a transaction on S in the current simulation cycle, otherwise false
S'last_event	time	time interval since the last event on S
S'last_active	time	time interval since the last transaction on S
S'last_value	base type of S	value of S before the last event
S'driving	boolean	true if the current process is producing a transaction on S
S'driving value	base type of S	value assigned to S in the current process

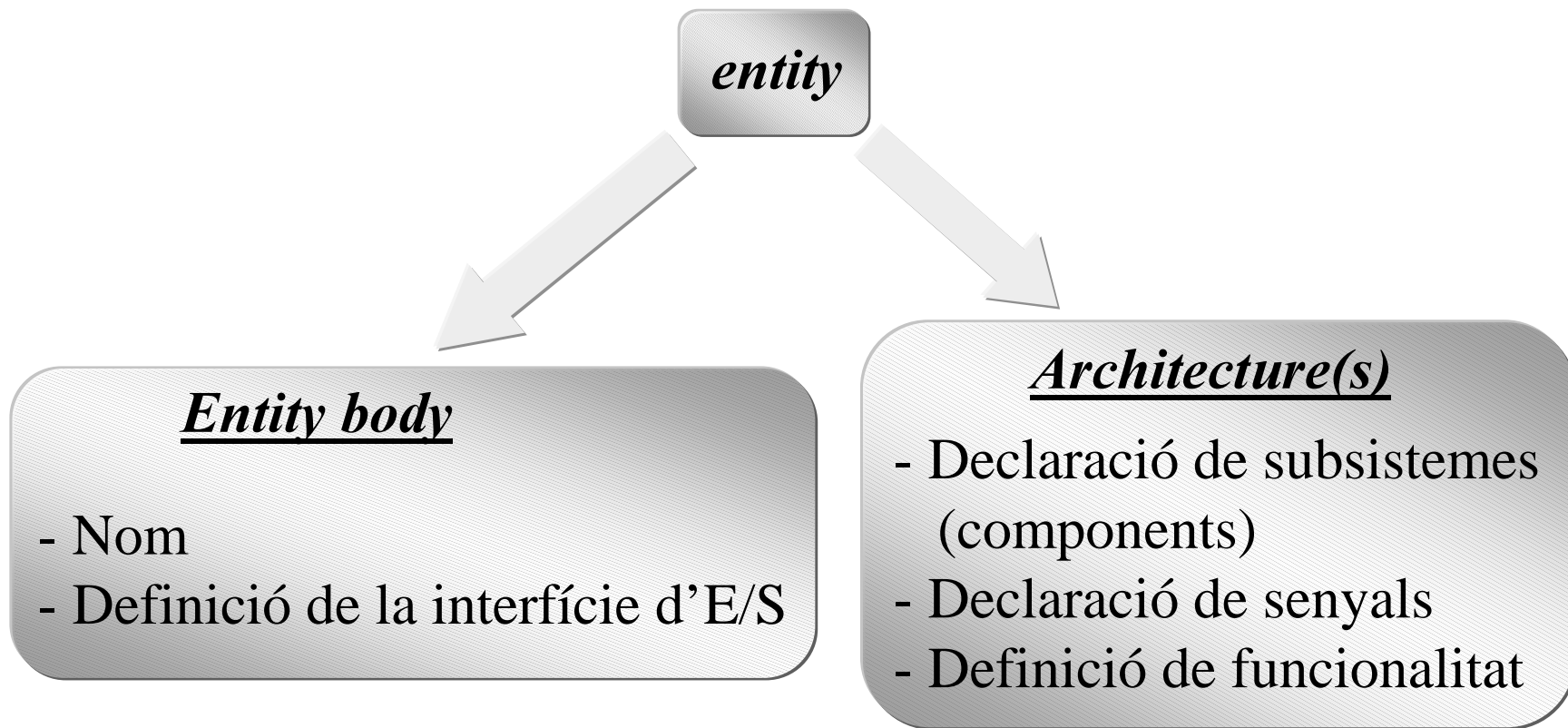
➤ **Exemple:**

```
function increment (val:bit_vector) return bit_vector is  
variable result: bit_vector(val'range);  
variable carry: bit;  
begin  
    result(0):= not val(0);  
    carry:= val(0);  
    for i in 1 to val'high loop  
        carry:= carry and val(i-1);  
        result(i):= val(i) xor carry;  
    end loop;  
    return result;  
end increment;
```

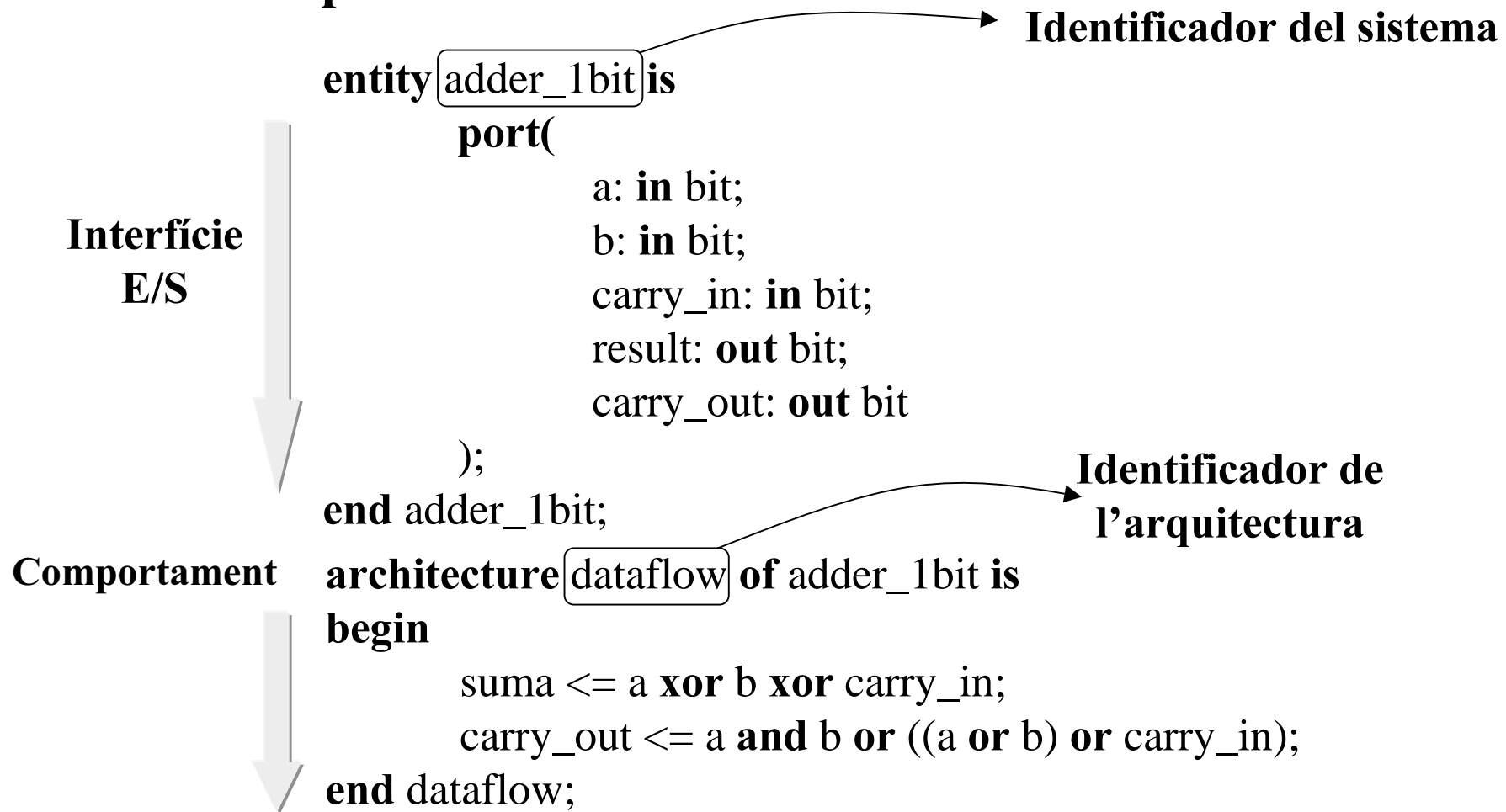


4. Primitives de descripció bàsiques

- Sistema = entity



➤ **Exemple:**



- **Nivells de descripció per a l'arquitectura:**

- **Flux de dades:**
 - Assignacions
 - Blocks
- **Comportament:**
 - Processos
 - Procedures i functions
- **Estructural:**
 - Components
 - Primitiva generate

5. Descripció de flux de dades

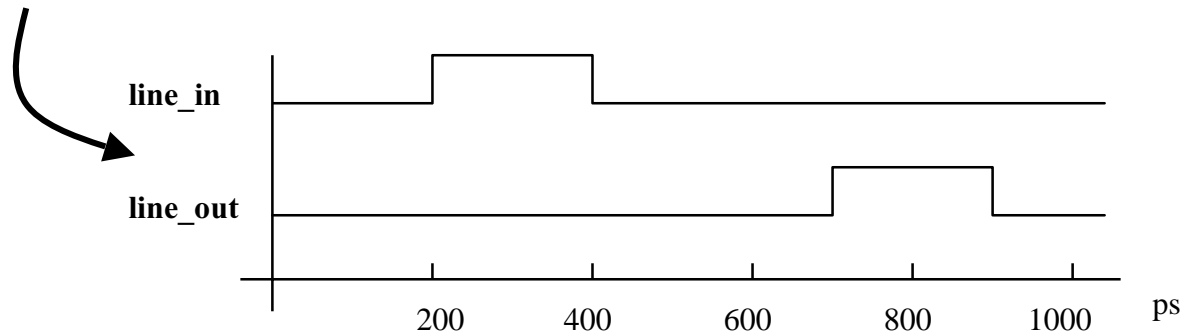
- **Assignació incondicional:**

`A_signal <= valor1 after retard1, valor2 after retard2, ..., valorn after retardn;`

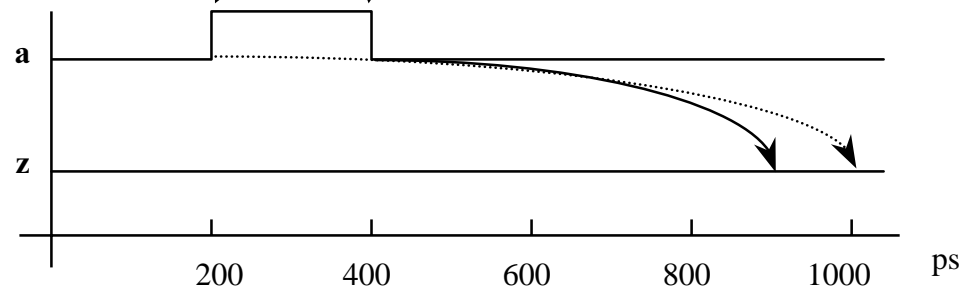
- **Tipus de retards: inertial i transport**

- transport: freqüència infinita (propagació de current)

```
t_line: process (line_in)
begin
    line_out <= transport line_in after 500 ps;
end process t_line;
```



```
asym_delay: process (a)
constant Tpd_01: time:=800 ps;
constant Tpd_10: time:=500 ps;
begin
    if a='1' then
        z<= transport a after Tpd_01;
    else
        z<= transport a after Tpd_10;
    end if;
end process asym_delay;
```



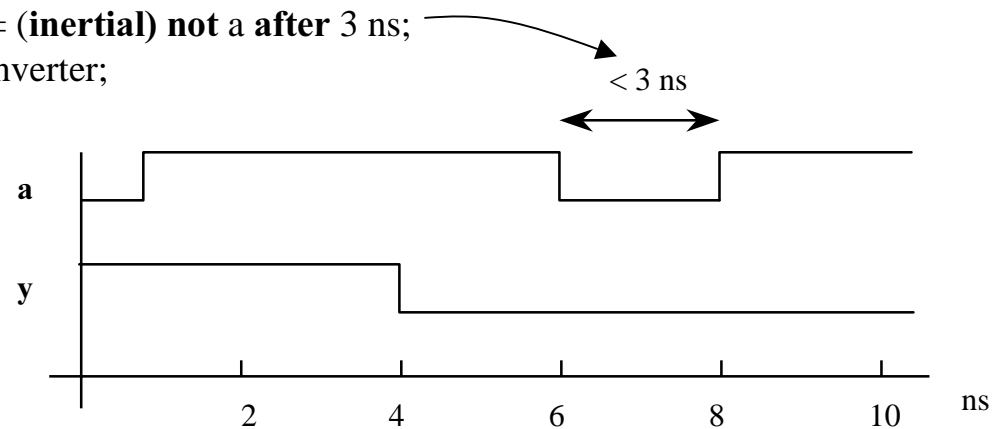
- inertial: tipus de retard per defecte

inverter: **process** (a)

begin

y<= (inertial) not a after 3 ns;

end process inverter;



S<= 1 after 5 ns, 5 after 10 ns, 10 after 15 ns;

|||

S<= 1 after 5 ns;

S<= transport 5 after 10 ns;

S<= transport 10 after 15 ns;

➤ **Assignació condicional:**

```
A_signal <=  valor1 when condicio1  
              else valor2 when condicio2  
              ...  
              else valor_per_defecte;
```

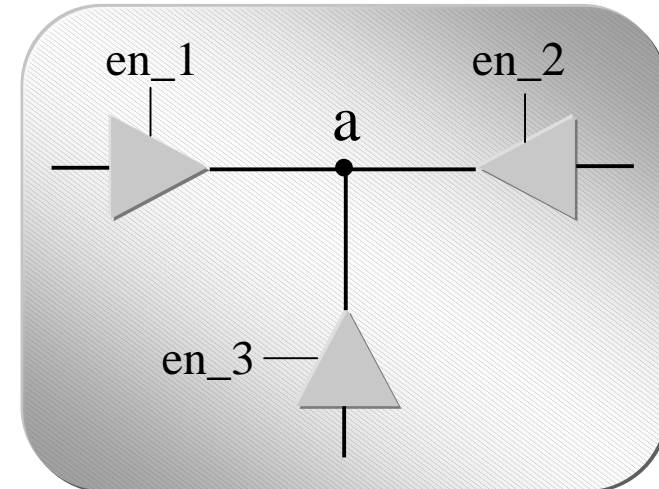
➤ **Assignació selectiva:**

```
with expressio select  
    A_signal <=  valor1 when seleccio1,  
                valor2 when seleccio2,  
                ...  
                valor_per_defecte when others;
```

• Resolució de senyals:

- Definició de *funcions de resolució* que determinen el valor lògic a un node quan aquest està connectat a una sèrie de drivers (actius o no)

```
a <= b or c when (en_1='1')  
      else 'Z';  
a <= d when (en_2='1')  
      else 'Z';  
a <= e xor f when (en_3='1')  
      else 'Z';
```



- **IEEE Standard 1164: Lògica de 9 valors**

'U': No inicialitzat	'W': Desconegut (driver feble)
'X': Desconegut (driver fort)	'L': Nivell baix (driver feble)
'0': Nivell baix (driver fort)	'H': Nivell alt (driver feble)
'1': Nivell alt (driver fort)	'-': Indiferent
'Z': Alta impedància	

- **Utilització dels tipus de dades 1164:**

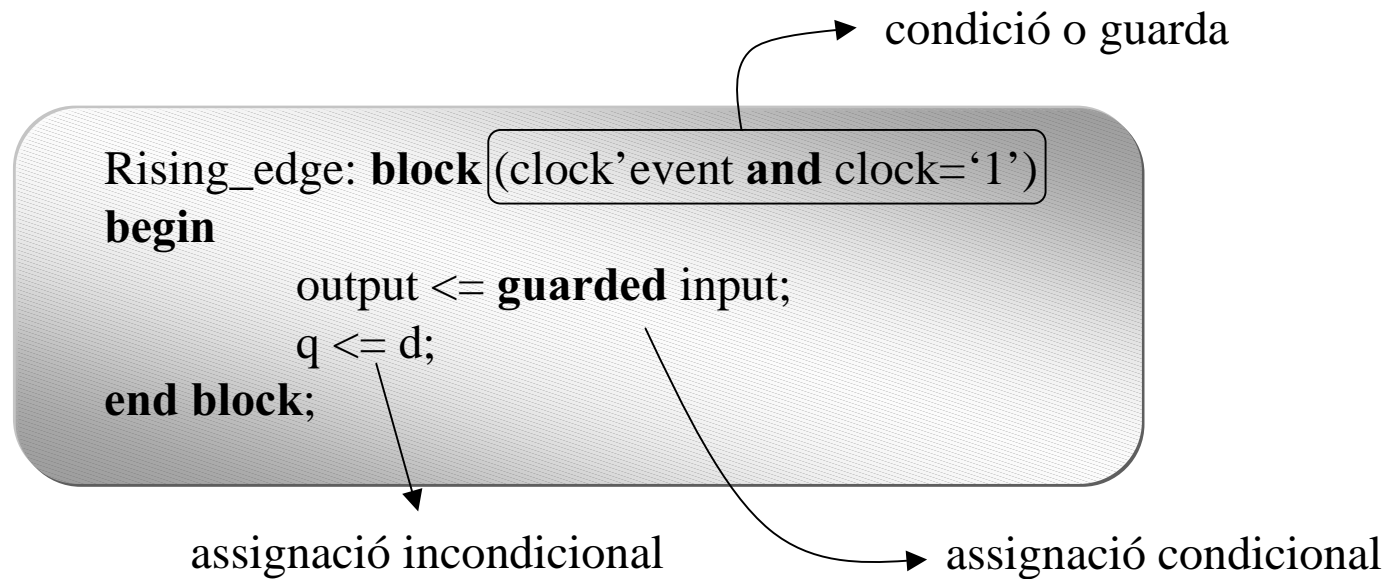
```
library ieee;  
use ieee.std_logic_1164.all;
```

- **Tipus bàsics:**

```
- std_logic ( = bit)  
- std_logic_vector ( = bit_vector)
```

- **Blocks:**

- Subsistema funcional
- Execució **concurrent**
- Interconnexió mitjançant ports
- Definició de guardes



6. Descripció de comportament

- **Processos:**

- Cos d'execució seqüencial
- Activats per canvis de senyals
- Execució **concurrent**

- **Control d'activació:**

- **Llista de sensibilitat:**
 - El procés s'activa si hi ha un canvi en qualsevol dels senyals que figuren a la llista
- **Primitiva wait:**
 - L'execució del procés queda aturada fins que es verifiqui una certa condició

- **Exemple:**

Primitiva wait

```
T_reg: process
begin
  wait until(clk'event and clk='1');
  if (reset='1') then
    q <= '0';
  elsif (enable='1') then
    q <= not q;
  end if;
end process T_reg;
```

Llista de sensibilitat

```
T_reg: process(clk)
begin
  if (clk'event and clk='1') then
    if (reset='1') then
      q <= '0';
    elsif (enable='1') then
      q <= not q;
    end if;
  end if;
end process T_reg;
```

- **Primitives de control seqüencial:**

- **Wait:**

```
wait    [until condicio;]  
         [on senyal1, senyal2, ...;]  
         [for expressio_temporal;]
```

- **If:**

```
if [condicio] then [accions] elsif [accions] ... else [accions] ... end if;
```

- **Case:**

```
case [expressio] is  
    when [seleccio] => [accions];  
    ...  
end case;
```

➤ **Bucle indefinit:**

```
loop [accions] end loop;
```

➤ **Bucle while:**

```
while [condicio] loop  
    [accions];  
end loop;
```

➤ **Bucle for:**

```
for [identificador] in [rang] loop  
    [accions];  
end loop;
```

➤ **Control de bucles: next, exit [when condicio]**

➤ **Exemple:**

**architecture behavior of counter is
begin**

increment: process

variable value: integer:=0;

begin

output <= value;

loop

loop

wait until clk='1' **or** reset='1';

exit when reset='1';

value:= (value+1) **mod** 16;

output <= value;

end loop;

value:=0;

output <= value;

wait until reset='0';

end loop;

end process increment;

end behavior;

- **Procedures i functions:**

Conjunt d'accions utilitzades diverses vegades a una descripció que s'encapsulen a una declaració única

➤ **Procedures:** Poden tornar o no un valor

```
procedure mean is  
  variable partial: real:=0.0;  
  begin  
    for index in matrix'range loop  
      partial:= partial+matrix(index);  
    end loop;  
    mean_value:= partial / real(matrix'length);  
  end procedure mean;
```

➤ **Functions:** Sempre tornen un valor

```
function increment (val:bit_vector) return bit_vector is  
variable result: bit_vector(val'range);  
variable carry: bit;  
begin  
    result(0):= not val(0);  
    carry:= val(0);  
    for i in 1 to val'high loop  
        carry:= carry and val(i-1);  
        result(i):= val(i) xor carry;  
    end loop;  
    return result;  
end increment;
```

7. Descripció estructural

- **Components:**

- Subsistema definit a una llibreria

- Declaració d'un component:

```
component nand3
  generic(Tpd: time:= 1ns);
  port(a,b,c: in bit; f: out bit);
end component;
```

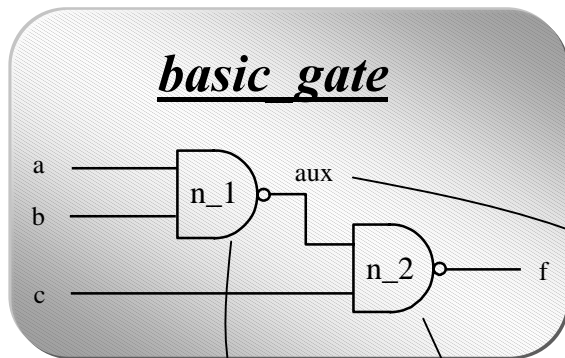
Diagram annotations:

- Arrow from **nand3** to "nom del component"
- Arrow from **generic** to " propietat de l'entitat"

- Instanciació d'un component:

```
enable: nand3
  port map(en1, en2, int_req, interrupt);
```

➤ Example:



```
entity basic_gate is  
port(a,b,c: in std_logic; f: out std_logic);  
end basic_gate;
```

```
architecture structural of basic_gate is  
component nand2  
port(a,b: in std_logic; f: out std_logic);  
end component;
```

```
signal aux: std_logic;  
begin
```

```
    n_1: nand2
```

```
    port map(a,b,f);
```

```
    n_2: nand2
```

```
    port map(a=>aux,b=>c,f=>f);
```

```
end structural;
```

- **Definició d'estructures regulars:**
- **Primitiva generate:**

```
adder: for i in 0 to k-1
    ls_bit: if i=0 generate
        ls_cell: half_adder port map(a(0), b(0), s(0), c_in(1));
    end generate ls_bit;
    middle_bit: if i > 0 and i < k-1 generate
        mid_cell: full_adder port map(a(i), b(i), c_in(i), s(i), c_in(i+1));
    end generate middle_bit;
    ms_bit: if i=k-1 generate
        ms_cell: full_adder port map(a(i), b(i), c_in(i), s(i), carry);
    end generate ms_bit;
end generate adder;
```

8. Organització del disseny

- **Llibreria:** Objecte (directori, fitxer, ...) que conté mòduls ja dissenyats
- **Definició d'una llibreria de referència:**

```
library library_name;
```

- **Utilització d'elements inclosos a una llibreria:**

```
use library_name.defined_element;
```

- **Configuració d'objectes (binding):**

nom de la configuració

```
library basic_flip_flop;  
use basic_flip_flop.ff_d_rising_edge;  
  
configuration reg4_structural of reg4 is  
  for structural  
    for bit0: flipflop  
      use entity ff_d_rising_edge(drive4);  
    end for;  
    for others: flipflop  
      use entity ff_d_rising_edge(drive1);  
    end for;  
  end for;  
end configuration reg4_structural;
```

arquitectura
de reg4

- **Declaracions encapsulades (packages)**

- declaració + definició del cos
- referència: **variable** PC: `data_types.address;`
- utilització: **use** data_types.all;

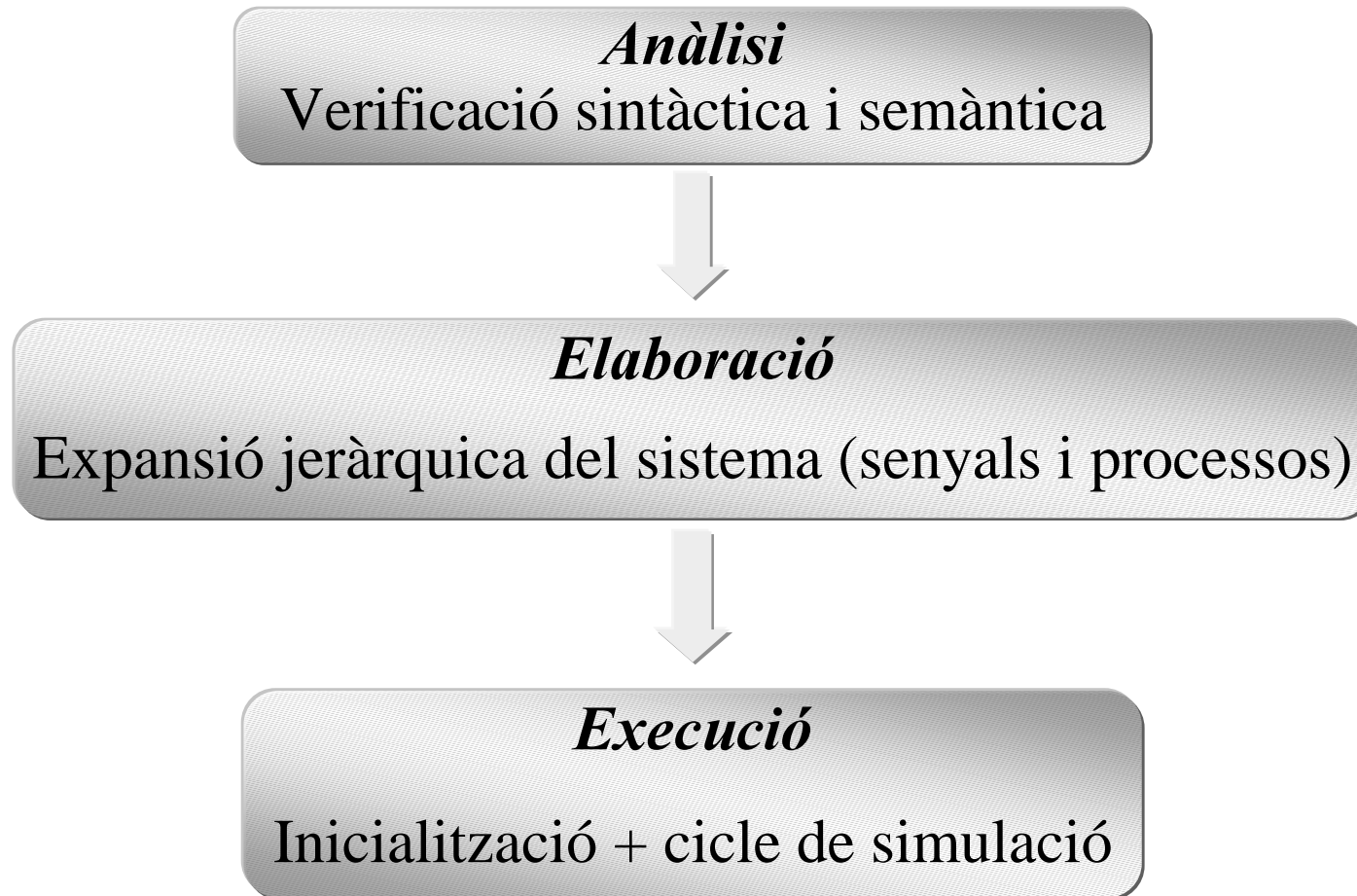
nom del package

tipus definit al package

- **Simulació:**

- **Test Bench:** Model VHDL on es declaren estímuls per a un sistema (i/o on es verifiquen les seves sortides)

- **Organització del procés de simulació:**



- **Cicle de simulació:**

- 1) S'avança el temps fins a l'instant on s'hagin de produir transaccions sobre senyals
- 2) Es realitzen les transaccions sobre els senyals
- 3) S'activen i executen els processos que siguin sensibles als esdeveniments que s'hagin produït

```
process ...  
begin  
    ...  
    s <= '1';  
    ...  
    if s='1' then ...;  
    ...  
end process;
```

