PRÁCTICA:

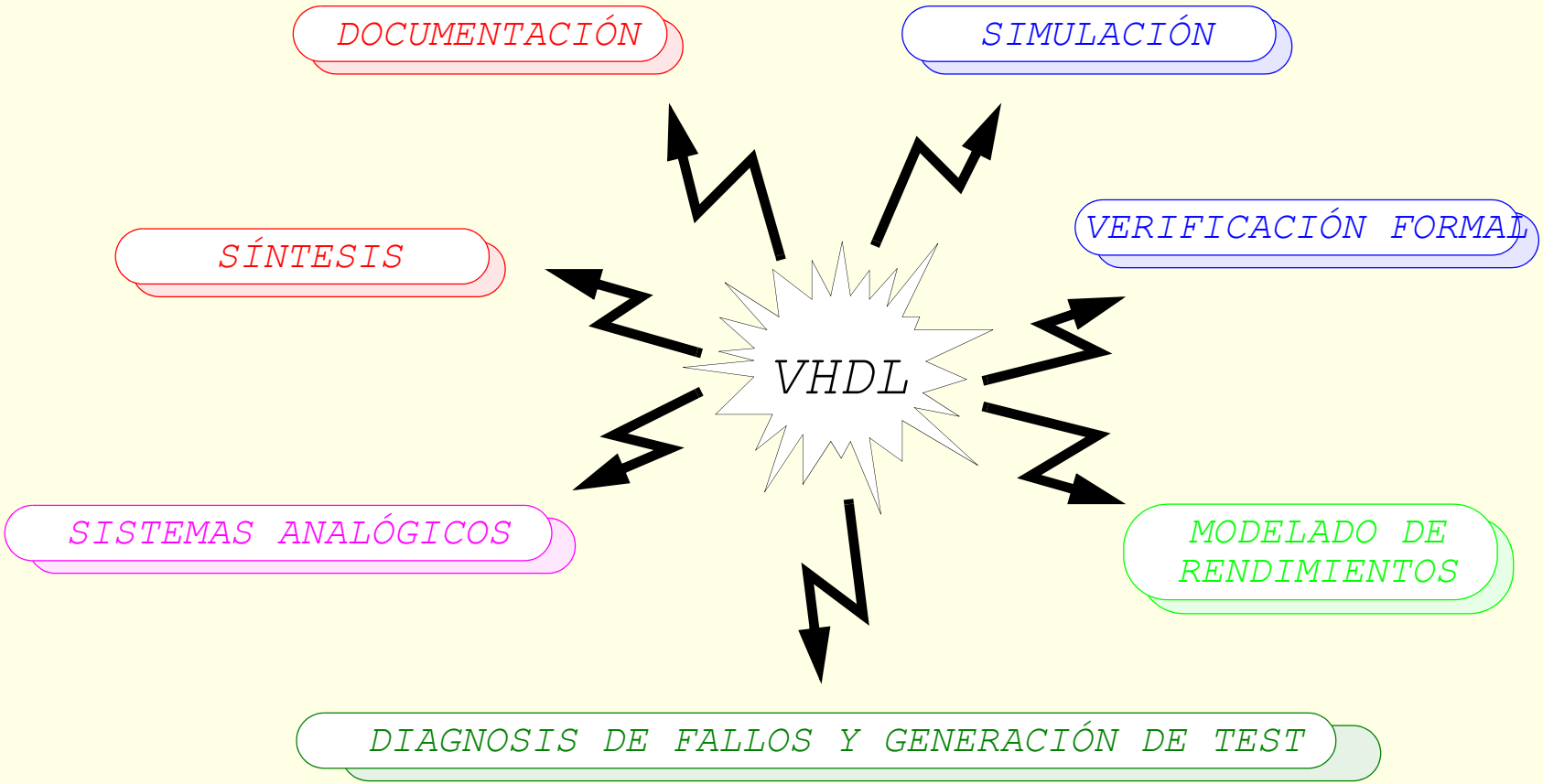
LENGUAJE VHDL

Los lenguajes permiten manejar mejor grandes tamaños

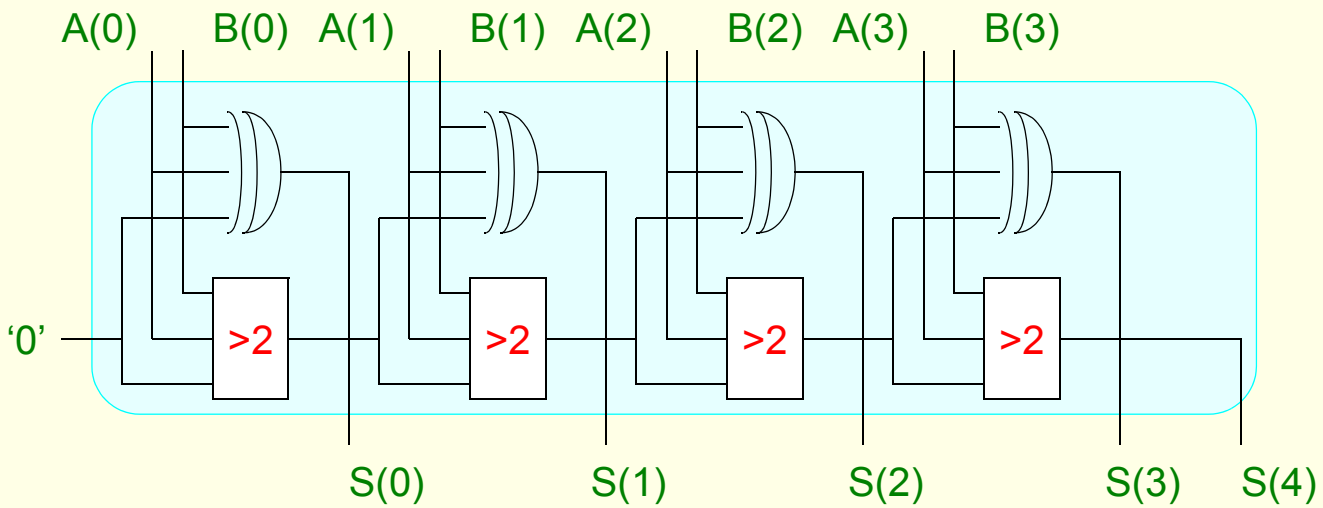
Los lenguajes son más flexibles que las tablas

Los lenguajes son légibles por las máquinas más fácilmente que los gráficos

Los lenguajes son más comprensibles que los métodos matemáticos



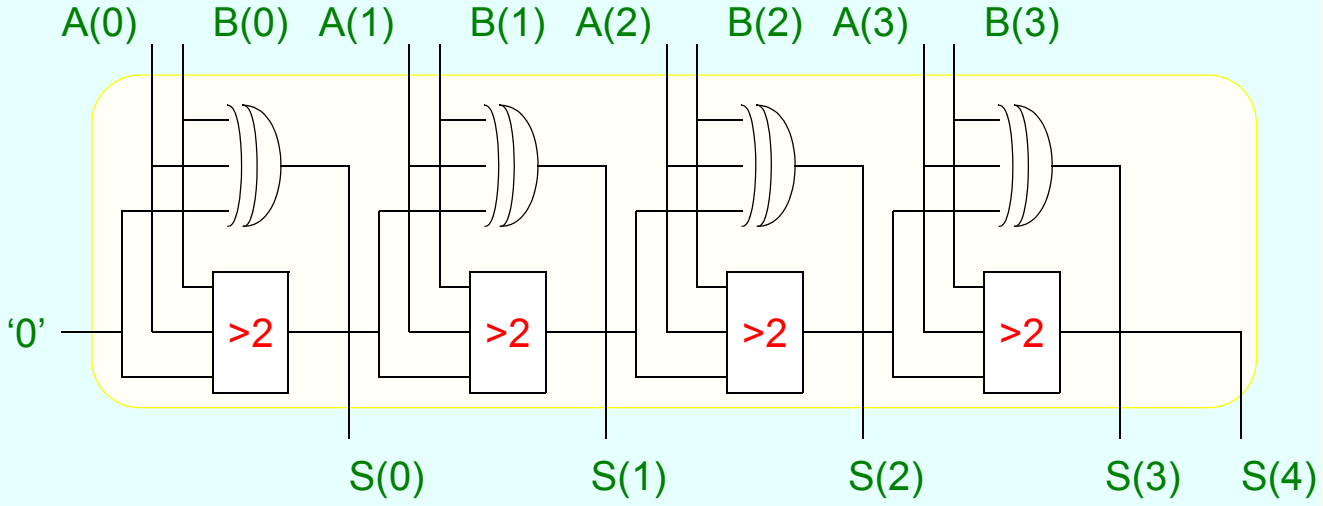
ENTIDAD



Conexión con el exterior

Propiedades genéricas

ARQUITECTURA



Descripción estructural

Descripción de comportamiento

Descripción de flujo de datos

Estructura

```
entity sumador is
port(       A, B, Cin       : in std_logic_vector(3 downto 0);
            suma            : out std_logic_vector(4 downto 0));
end sumador;
```

```
architectura estructura of sumador  is
    signal s1    : std_logic_vector(3 downto 0);
    component xor
    port(       A, B, C     : in std_logic;
                Y           : out std_logic);
    end component;
    component mayor
    port(       A, B, C     : in std_logic;
                Y           : out std_logic);
    end component;
    for all:xor use entity work.xor;
    for all:mayor use entity work.mayor;
begin
    B1:xor port map (A(0), B(0), '0', suma(0));
    M1:mayor port map (A(0), B(0), '0', S1(0));
    B2:xor port map (A(1), B(1), S1(0), suma(1));
    M2:mayor port map (A(1), B(1), S1(0), S1(1));
    B3:xor port map (A(2), B(2), S1(1), suma(2));
    M3:mayor port map (A(2), B(2), S1(1), S1(2));
    B4:xor port map (A(3), B(3), S1(2), suma(3));
    M4:mayor port map (A(3), B(3), S1(2), suma(4));
end;
```

```
architectura flujo of sumador  is
    signal s1    : std_logic_vector(2
        downto 0);
begin
    suma(0) <= A(0) xor B(0);
    S1(0) <= A(0) and B(0);
    suma(1) <= A(1) xor B(1) xor S1(0);
    S1(1) <= (A(1) and B(1)) or
        (A(1) and S1(0)) or (B(1) and S1(0));
    suma(2) <= A(2) xor B(2) xor S1(1);
    S1(2) <= (A(2) and B(2)) or
        (A(2) and S1(1)) or (B(2) and S1(1));
    suma(3) <= A(3) xor B(3) xor S1(2);
    suma(4) <= (A(3) and B(3)) or
        (A(3) and S1(2)) or (B(3) and S1(2));
end;
```

Lenguaje VHDL

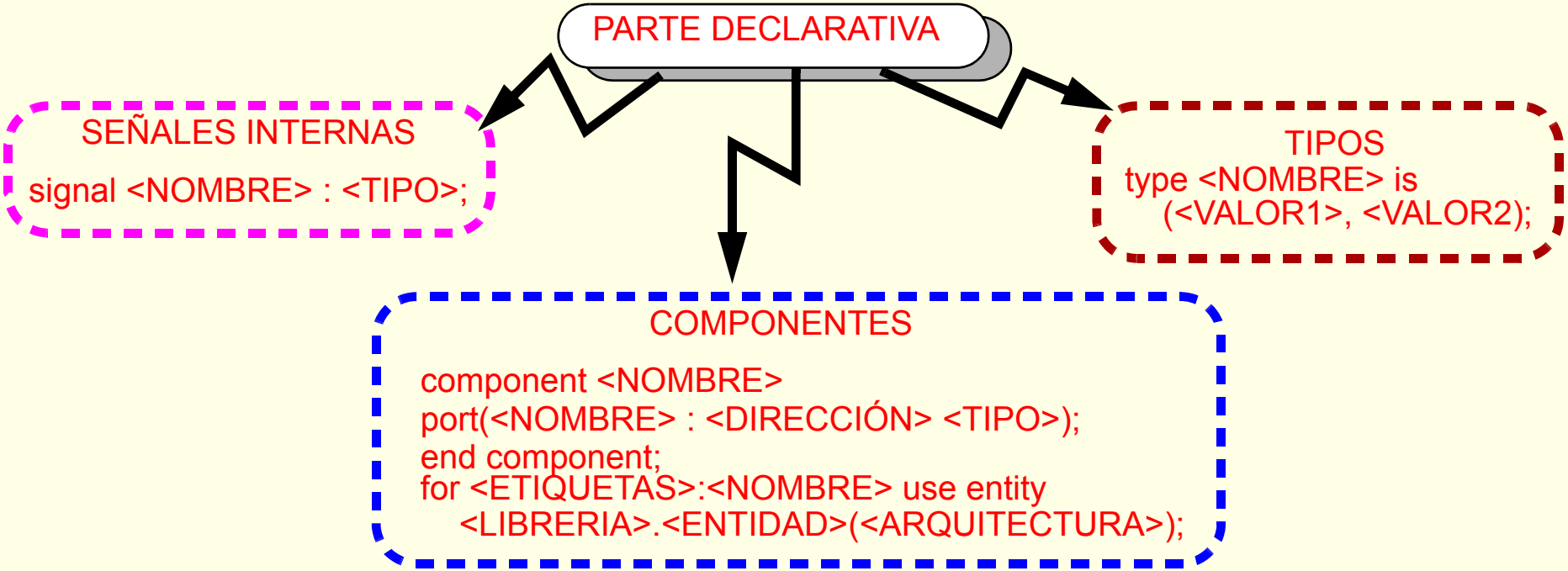
ESQUEMA GENÉRICO DE LA ENTIDAD

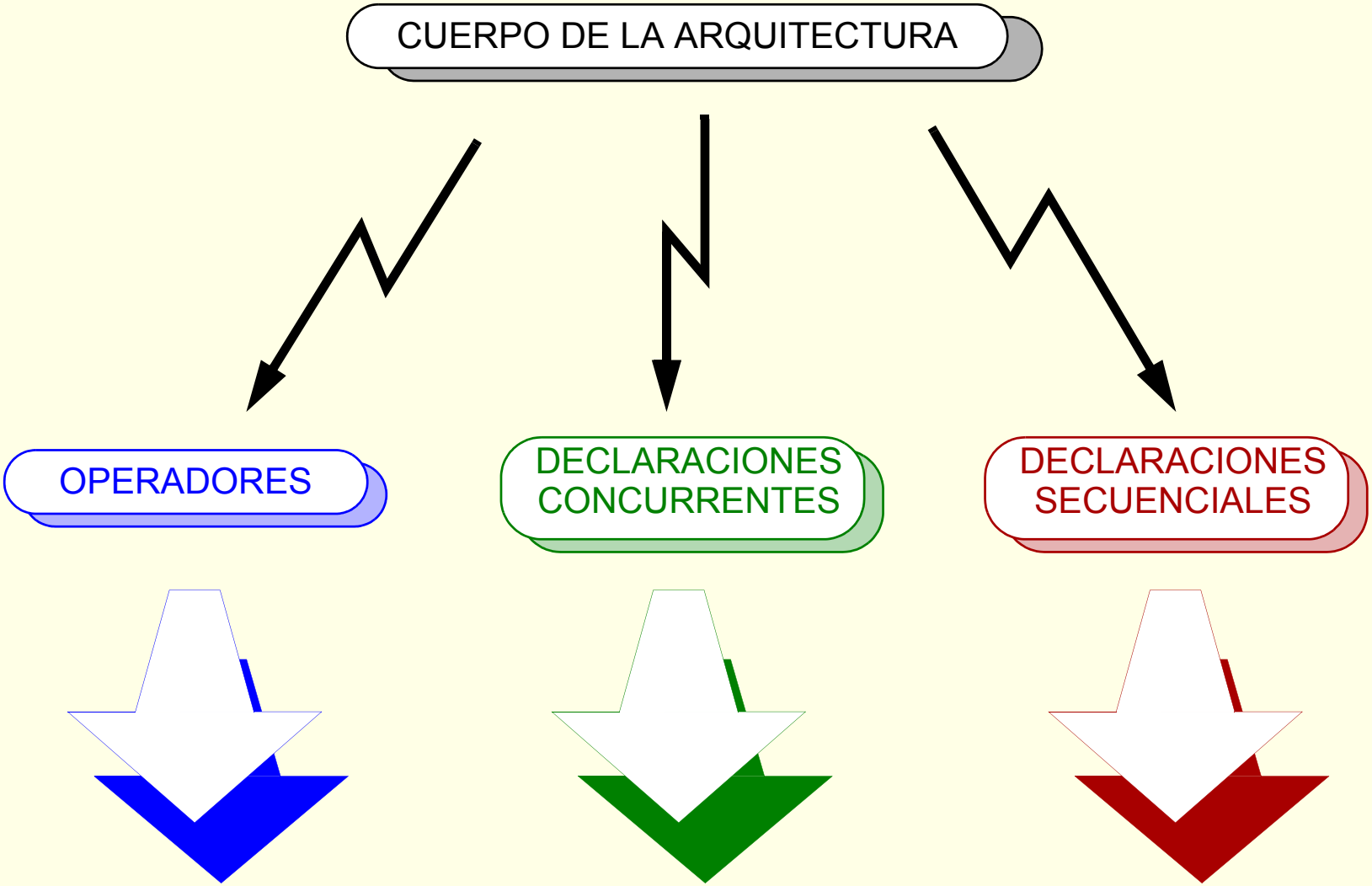
```
entity <NOMBRE> is
generic(<PROPIEDAD>                    : <TIPO> := <VALOR>;
        <PROPIEDAD>                    : <TIPO> := <VALOR>);
port(   <NODO><, NODO>                     : <SENTIDO> <TIPO>;
        <NODO><, NODO>                     : <SENTIDO> <TIPO>;
        <NODO><, NODO>                     : <SENTIDO> <TIPO>);
end <NOMBRE>;
```

- SENTIDOS DE SEÑAL
  - **IN** --> SEÑAL DE ENTRADA. NO SE LE PUEDE ASIGNAR NINGÚN VALOR EN EL INTERIOR DEL MODELO
  - **OUT** --> SEÑAL DE SALIDA. DEBE SER ASIGNADA. NO SE PUEDE TOMAR COMO DATO DE UN ASIGNAMIENTO
  - **INOUT** --> SEÑAL DE ENTRADA/SALIDA. SE LE PUEDE ASIGNAR UN VALOR, Y SER DATO DE UN ASIGNAMIENTO.
- TIPOS PREDEFINIDOS
  - **BIT** (VALORES: 0, 1)
  - **BIT_VECTOR** (VECTOR DE N BITS)
  - STD_LOGIC (VALORES: 0, 1, X, Z, H, L)
  - STD_LOGIC_VECTOR (VECTOR DE N STD_LOGIC)

ESQUEMA GENÉRICO DE LA ARQUITECTURA

```
architecture <NOMBRE> of <NOMBRE ENTIDAD> is
    <PARTE DECLARATIVA DEL CUERPO DE LA ARQUITECTURA>
begin
    <CUERPO DE LA ARQUITECTURA>
end;
```




CUERPO DE LA ARQUITECTURA
OPERADORES
DECLARACIONES CONCURRENTES
DECLARACIONES SECUENCIALES
DESCRICPCIÓN DE FLUJO DE DATOS
DESCRICPCIÓN DE ESTRUCTURA
DESCRICPCIÓN DE ALGORITMO

DEFINICIÓN DE NUEVOS TIPOS

Tipos enumerados

type <NOMBRE> is (<VALOR><, <VALOR>, <VALOR>);

type bit is (‘0’, ‘1’);

type triestado is (‘0’, ‘1’, ‘Z’);

type estados_sistema is (desocupado, test, evaluando);

type <NOMBRE> is range <VALOR INICIAL> to <VALOR FINAL>;

type enteros_pequeños is range 0 to 9;

type reales_pequeños is range 0.0 to 1.0;

type tiempo is range -(2**31-1) to 2**31-1

Tipos matriciales

type <NOMBRE> is array (<VALOR INICIAL> to <VALOR FINAL>) of <TIPO>;

type <NOMBRE> is array (<TIPO DE RANGO> range <>) of <TIPO>;

type bit_vector is array (integer range <>) of bit;

type matriz is array (integer range <>, integer range <>) of reales_pequeños;

type mi_bit is array (triestado range ‘0’ to ‘1’) of bit;

Tipos registros

```
type <NOMBRE> is record
        <NOMBRE1><,<NOMBRE11>>: <TIPO>;
        <NOMBRE2><,<NOMBRE21>: <TIPO>;
end record;
```

```
type coordenadas is record
            X, Y: enteros_pequeños;
end record;
type par_lógica_tiempo is record
            valor_lógico: bit_vector(9 downto 0);
            valor_temporal: tiempo;
end record;

signal S1, S2: par_lógica_tiempo;
signal C1, C2: coordenadas;

C1.X <= 8;
C1.Y <= 9;
C2 <= C1;
S1.valor_lógico(6) <= '0';
S2.valor_lógico <= "0101010101"; ----------> S2.valor_lógico(9) = 0
```

atributos -> referencia a características de tipos

atributos que devuelven elementos

- ‘left -> devuelve el elmento de la izquierda de la lista
- ‘right -> devuelve el elemento de la derecha de la lista
- ‘high -> devuelve el mayor elemento de la lista
- ‘low -> devuelve el menor elemento de la lista
- ‘succ(<ELEmento>) -> devuelve el siguiente elemento al indicado
- ‘val(<índice>) -> devuelve el elemento correspondiente al índice
- ‘pred(<elemento>) -> devuelve el elemento anterior al indicado

atributos que devuelven un valor

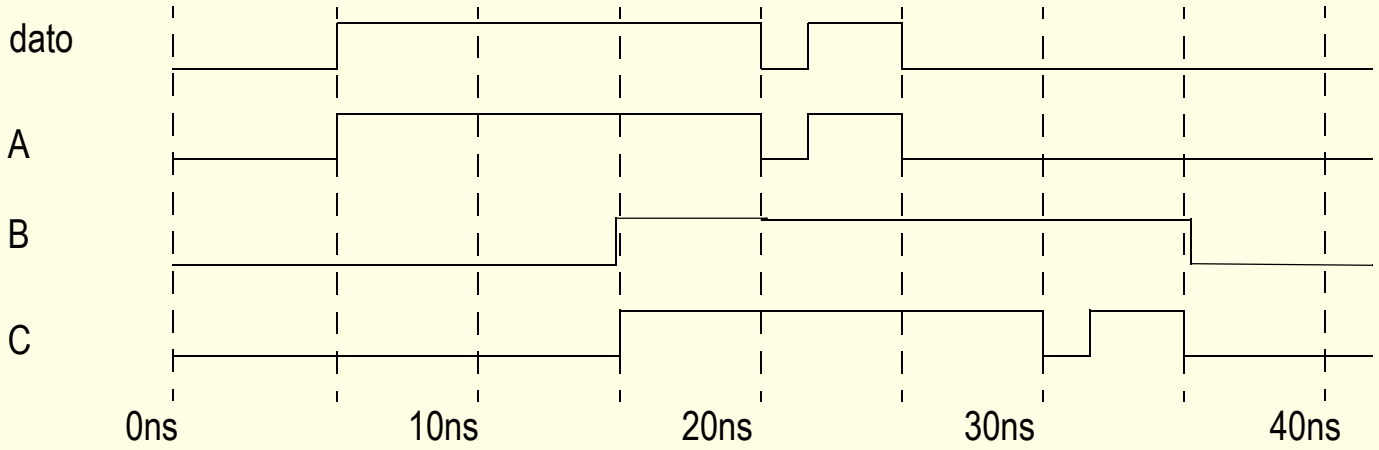
- ‘pos(<elemento>) -> devuelve el índice correspondiente al elemento
- ‘stable -> cierto cuando la no hay cambio en la señal
- ‘event -> cierto cuando hay cambio en la señal
- ‘range -> devuelve el rango del tipo
- ‘length -> devuelve la longitud del tipo

atributos que devuelven una señal

- ‘delayed(<N>) -> devuelve la misma señal retrasada n segundos

Asignamiento de señales: forma básica de modelar un comportamiento en vhdl

```
dato <= '0', '1' after 5ns, '0' after 20ns, '1' after 22ns, '0' after 25ns;
a <= dato;
-- A ES UNA COPIA IDÉNTICA DE DATO
b <= dato after 10ns;
-- B ES UNA COPIA DE DATO RETRASADA INERCIALMENTE 10ns
c <= transport dato after 10ns
-- C ES UNA COPIA DE DATO RETRASADA IDEALMENTE 10ns
```



Operaciones

lógicas

```
and  --> S1 <= A and B and C;
OR   --> S1 <= A or B or C;
xor  --> S1 <= A xor B xor C;
NAND --> S1 <= (A nand B) nand C;
NOR  --> S1 <= (A nor b) nor C;
not  --> S1 <= not A;
```

aritméticas

```
+    --> suma
-    --> resta
*    --> multiplicación
/    --> división
**   --> potenciación
abs  --> valor absoluto
```

relacionales

```
=    --> igual
/=   --> distinto
<    --> menor
<=   --> menor o igual
>    --> mayor
>=   --> mayor o igual
```

Declaraciones concurrentes

Selección

```
with <EXPRESIÓN> select
    <SEÑAL> <= <EXPR1> when <VALOR>,
    <EXPR2> when <VALOR>;
```

Componente

```
<ETIQUETA>: <COMPONENTE> port map (<SEÑALES>);
```

Asignamiento condicional

```
<SEÑAL> <= <EXPR1> when <CONDICIÓN> else
        <EXPR2>;
```

Bucle

```
<ETIQUETA>: for <ÍNDICE> in <RANGO> generate
begin
        <INSTRUCCIONES CONCURRENTES>
end generate;
```

Bloque

```
<ETIQUETA>: block (<CONDICIÓN>)
    <PARTE DECLARATIVA
begin
    <INSTRUCCIONES>
    <SEÑAL> <= guarded <EXPR>;
end block <ETIQUETA>;
```

Procesos

```
<ETIQUETA>: process (<LISTA DE SENSIBILIDAD>)
    <PARTE DECLARATIVA
begin
    <INSTRUCCIONES SECUENCIALES>
end process <ETIQUETA>;
```

Avisos

```
assert <CONDICIÓN> report <MENSAJE>
     severity <TIPO DE AVISO>;
-- LOS TIPOS DE AVISO PUEDEN SER NOTE, WARNING,
--  ERROR y FAILURE.
```

Declaraciones secuenciales

Espera

```
wait on <SEÑAL>;
wait for <TIEMPO>;
wait until <CONDICIÓN>;
```

Condicional

```
<ETIQUETA>: if <CONDICIÓN> then
        <INSTRUCCIONES>
elseif <CONDICIÓN2> then
        <INSTRUCCIONES>
else
        <INSTRUCCIONES>
end if <ETIQUETA>;
```

Selección

```
<ETIQUETA> : case <EXPRESIÓN> is
when <VALOR> =>
        <INSTRUCCIONES>
when others =>
        <INSTRUCCIONES>
end case <ETIQUETA>;
```

Bucles

```
<ETIQUETA>: for <ÍNDICE> in <RANGO> loop
        <INSTRUCCIONES SECUENCIALES>
end loop;
```

```
<ETIQUETA>: while <CONDICIÓN> loop
        <INSTRUCCIONES SECUENCIALES>
end loop;
```

```
<ETIQUETA>: loop
        <INSTRUCCIONES SECUENCIALES>
        next <ETIQUETA> when <CONDICIÓN>;
        exit <ETIQUETA> when <CONDICIÓN>;
end loop <ETIQUETA>;
```

Librerías

Declaración de Paquetes

```
package <NOMBRE> is
        <PARTE DECLARATIVA>
end <NOMBRE>;
```

Cuerpo de Paquetes

```
package body <NOMBRE> is
      <DEFINICIONES>
end <NOMBRE>;
```

```
package funciones_triestado is
        type triestado is (‘0’, ‘1’, ‘Z’);
        function “andt” (X, Y : triestado) return triestado;
        function “ort” (X, Y: triestado) return triestado;
        function “nott” (X : triestado) return triestado;
end funciones_triestado;

package body funciones_triestado is
function “andt” (X, Y : triestado) return triestado is
      signal resultado: triestado;
      begin
      process (X)
      begin
            if X = ‘0’ then resultado <= ‘0’;
            elseif X = ‘1’ then resultado <= Y;
            elseif Y = ‘0’ then resultado <= ‘0’;
            else resultado <= Y;
            end if;
      end process;
end “andt”;
```

Referenciando Librerías

library <NOMBRE1>, <NOMBRE2>, <NOMBRE3>;

library aritmética;

Referencia a paquetes

use <LIBRERIA>.<PAQUETE>.<ELEMENTO>;

use work.sumador;
use work.funciones_triestado.all;
use work.funciones_triestado.andt;
use std.standard.bitl;
use std.textio.all;
use aritmética.funciones_triestado.triestado;

for <ETIQUETA>:<ENTIDAD> use entity
<LIBRERIA>.<ENTIDAD>(<ARQUITECTURA>);

for all:sumador use entity work.sumador;

Ejemplo

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;
entity sumador4 is
    Port ( a : in std_logic_vector(3 downto 0);
           b : in std_logic_vector(3 downto 0);
           cin : in std_logic;
           suma : out std_logic_vector(3 downto 0);
           cout : out std_logic);
end sumador4;

architecture Behavioral of sumador4 is
signal carry :std_logic_vector(4 downto 0);
begin
        carry(0) <= cin;
        F1:for i in a'range generate
        begin
                suma(i) <= a(i) xor b(i) xor carry(i);
                carry(i+1) <= (carry(i) and (a(i) or b(i))) or (a(i) and b(i));
        end generate;
        cout <= carry(4);
end;
```