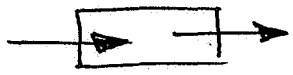
ESTRUCTURAS DE LOS COMPUTADORES

javier.garrido@uam.es
alberto.sanchezgonzalez@uam
↑ prácticas → C-231

ENTIDAD - ARQUITECTURA

La entidad (función) se usa para hacer una descripción "caja negra" del diseño, solo se detallan los puertos de entrada y salida.



equivalente en C:

```
float CalcularMedia (floata, float
```

Una entidad tiene varias arquitecturas, lo que queremos que haga la entidad.

equivalente en C a una arquitec

```
    int c;
    c=(a+b)/2;
    return c;
}
```

PORTS - La conexión con el exterior

PORT
- → NOMBRE: Identificador
- → MODO:
  - → In: entrada (leer)
  - → Out: salida (escribir)
  - → Inout
- → TIPO DE DATO
  - → std_logic (std_logic_1164) → valores: "U", "X", "0", "1", "Z", "W", "L", "H", "-".
    - "U" ← desconocido / unknown
    - "X" ← corto circuito
    - "Z" ← alta impedancia
  - → std_logic_vector
  - → Integer
  - → Boolean

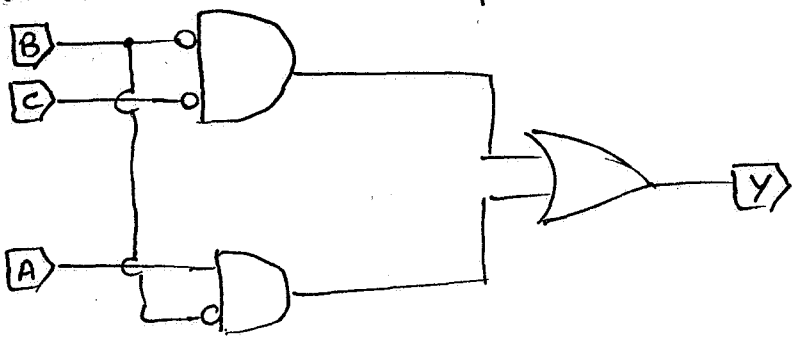
Ejemplo código VHDL

```
library IEEE;                          -- similar a declaración de .h
use IEEE.std_logic_1164.all;           -- para usar std_logic

entity inversor is
    port(a: in std_logic;
         y: out std_logic);
end inversor;

architecture comportamental of inversor is
begin
    y <= not a;                        -- asignación con flecha
end;
```

Síntesis de código VHDL

```
architecture comportamental of ejemplo is
begin
    y <= (not a and not b and not c) or (a and not b and not c) or
end comportamental;
```

SÍNTESIS: Se ha simplificado y (se puede comprobar por tablas de Karnaugh).



Importante

- VHDL no distingue mayúsculas y minúsculas.
- Los nombres no pueden empezar por números
- Se ignoran los espacios, tabuladores, retornos de carro. Comentarios para la ayuda de la corrección.

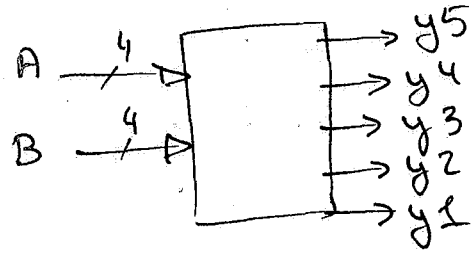
Tipo std_logic

IEEE.std_logic_1164

→ no inicializado, valor por defecto.
→ desconocido fuerte, indica cortocircuito.
→ Salida de una puerta con nivel lógico bajo
→ Salida de una puerta con nivel lógico alto.
→ alta impedancia
→ desconocido débil, terminación de bus
→ 0 débil, resistencia de pull-down.
→ 1 débil, resistencia de pull-up.
→ No importa, utilizado para la síntesis de circuitos.

OPERADORES BITWISE — LOGICA COMBINACIONAL

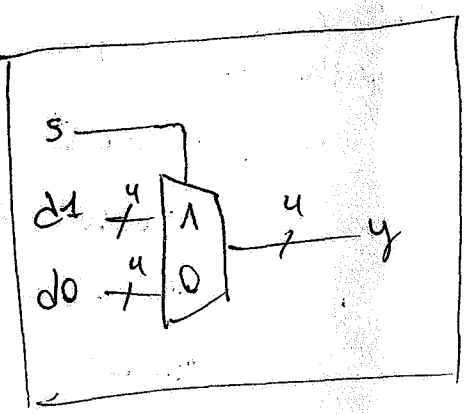
```
entity puertas is
port (a,b: in std_logic_vector(3 downto 0);
      y1, y2, y3, y4, y5 : out std_logic_vector (3 downto 0);
end puertas;
```




```
architecture comport of puertas is        -- pueden actuar sobre bits
                                          -- o sobe puertas
begin
   y1 <= a and b;        -- AND
   y2 <= a or b;         -- OR
   y3 <= a xor b;        -- XOR
   y4 <= a nand b;       -- NAND
   y5 <=a nor b;         -- NOR
end comport;
```

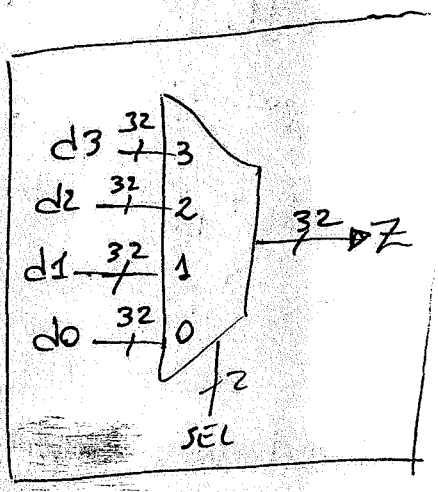
WHEN — ELSE

```
entity mux2a1_4bits is
port (d0, d1: in std_logic_vector (3 downto 0);
      s: in std_logic;
      y: out std_logic_vector (3 downto 0);
end mux2a1_4bits;

architecture comport of mux2a1_4bits is
begin
   y <= d0 when s='0' else d1;
end comport;
```



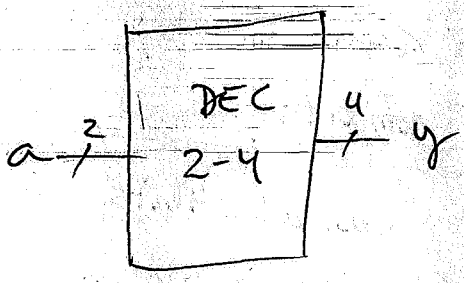
WHEN - ELSE

```
entity mux4a1_32bits is
port (d0, d1, d2, d3: in std_logic_vector (31 downto 0);
      SEL : in std_logic_vector (1 downto 0);
      Z : out std_logic_vector (31 downto 0);
end mux4a1_32bits;

architecture comport of mux4a1_32bits is
begin
   Z <= d3 when SEL = "11" else
        d2 when SEL = "10" else
        d1 when SEL = "01" else
        d0;
end comport;
```



WITH - SELECT

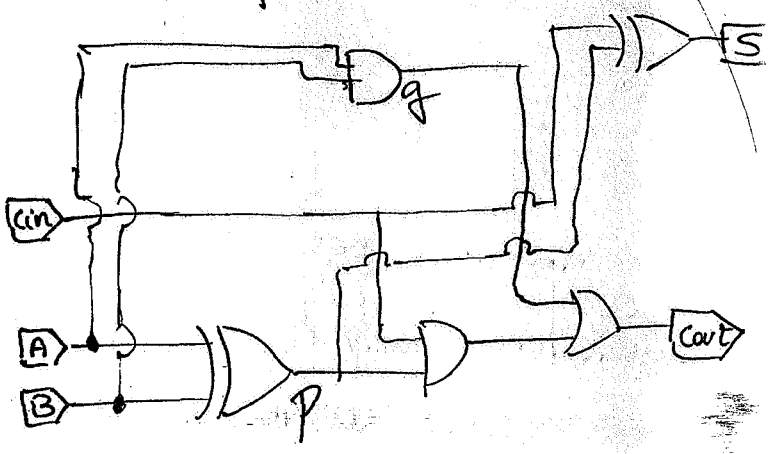
```
entity deco2a4 is
port (a: in std_logic_vector (1 downto 0);
      y: out std_logic_vector (3 downto 0);
end deco2a4;

architecture comport of deco2a4 is
begin
  with a select
    y <= "0001" when "00",
         "0010" when "01",
         "0100" when "10",
         "1000" when others;      --último caso, siempre
                                  -- conviene incluirlo
end comport;
```



```
architecture comport of fulladder is

    signal p, g : std_logic;

begin
    p <= a xor b;
    g <= a and b;
    S <= p xor cin;
    cout <= g or (p and cin);

end comport;
```

Todo el hardware existe a la vez!

-- La declaración de
-- señales internas se
-- pone entre architecture y
-- begin

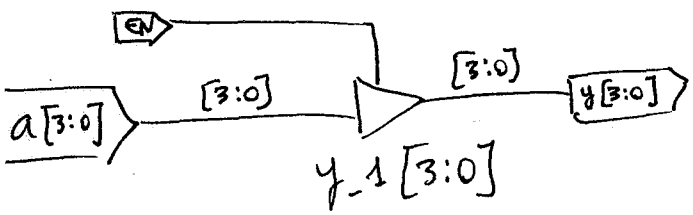


Z: ALTA IMPEDANCIA

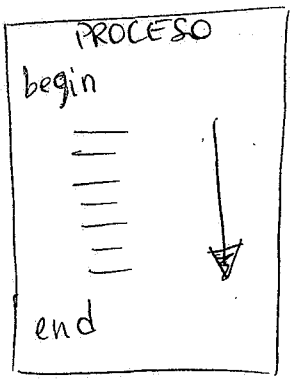
VHDL:

```
signal y, a : std_logic_vector(3 downto 0);
signal en : std_logic;
...
y <= (others => 'z') when en = '0' else a;
```

SÍNTESIS:



PROCESO



- Proceso: código secuencial en su interior → orden importante
- Más intuitivo pero hay más errores.
- Dentro de procesos: if, case, for y whil

ESTRUCTURA DEL "process"

```
architecture nombreArq of nombreEnt is
    ~ PARTE DECLARATIVA, SEÑALES INTERNAS DE LA ARQUITECTURA ~

Begin
  -- los procesos aparecen entre el begin y end de architecture
[ETIQ] process (lista sensibilidad) -- la etiqueta es opcional
    ~ PARTE DECLARATIVA, VARIABLES PERO NO SEÑALES, DE USO INTERNO ~

  begin
    -código
  end process [ETIQ];
end nombreArq;
```

~ Cada vez que cambia alguna señal de la lista sensibilidad, se ejecuta secuencialmente el código ~

IF

```
if condition 1 then
   -- sec instr 1
elseif condition 2 then
   -- sec instr 2
else condition 3 then
   --instr por defecto
end if;
```

CUIDADO: si es lógica combinacional SIEMPRE tiene que haber "else".

Ejemplo:

```
CTRL: process (nivel)
begin
   if nivel > 60 then
      a <= "11";
   elseif nivel >40 then
      a<= "10";
   elseif nivel > 20 then
      a<= "01";
   else
      a <= "00";
   end if;
end process CTRL;
```

CASE

```
case expression is
   when caso_1 =>
      -- sec instr 1
   when caso_2 =>
      -- sec instr 2
   when others =>
      -- instr por defecto
end case;
```

Similar al with-select

Ejemplo:

```
MUX: process (sel, a,b,c,d)
begin
   case sel is
      when "11" => y<=d;
      when "10" => y<=c;
      when "01" => y<=b;
      when others => y<=a;
   end case;
end process MUX;
```

FOR

```
[ETIQ] for indice in rango loop
    -- sec instr;
end loop [ETIQ];
```

Ejemplo:

```
AND 8: process(a,b)
begin
    for i in 0 to 7 loop
        y(i) <= a(i) and b(i);
    end loop;
end process AND 8;
```

```
y <= a and b;   -- hubiera sido más fácil porque and es un operador bitwise
```

Ejemplo

```
AND8: process (a, b)
begin
    for i in 7 downto 0 loop
        y(i) <= a(i) and b(i);
    end loop;
end process AND8;
```

WHILE

```
[ETIQ] while condition loop
    -- secc instr;
end loop [ETIQ]
```

Ejemplo

```
process (...)
begin
    while a='1' loop
        ...
        ...
        ...
        ...
        ...
    end loop;
end process;
```

SENTENCIAS CONDICIONALES Y BUCLES: RESUMEN

- Fuera de proceso
  - ✓ When - else - .... - ELSE - ;
  - ✓ With - select - ..... WHEN OTHERS;

- Dentro de proceso
  - ✓ if ** then - ; elsif - ; else - ; else if
  - ✓ Case ** is when - , ... when others - ; end case;
  - ✓ For * * in * * downto/to * * loop - ; end loop;
  - ✓ While * * loop - ; end loop;

LOGICA ~~CIRCUITOS~~ SECUENCIAL

En VHDL los flip_flops (o señales registradas) se describen empre igual: Con un process en el que aparecen el reloj el reset asíncrono (si lo hay) en la lista de ensibilidad.

FLIP-FLOP TIPO D

```
entity flop is
port (clk : in std_logic;
      D : in std_logic_vector (3 downto 0);
      Q : out std_logic_vector (3 downto 0);
end flop;
architecture sintetizable of flop is
begin
 REG:process (clk)
    begin
      if [clk = '1' and clk'event then] ≡ rising_edge (clk) then
         Q <= D;
      end if;
    end process;
end sintetizable;
```

FLIP-FLOP TIPO D con RESET ASÍNCRONO

```
architecture sintetizable of flop is
begin
    process (Reset, Clk)
        begin
            if Reset = '1' then
                Q <= (others => '0');
            elseif Clk = '1' and Clk'event then
                Q <= D;
            end if;
        end process;
    end sintetizable;
```

FLIP-FLOP TIPO D con RESET SÍNCRONO

```
architecture sintetizable of flop is
begin
    process (Clk) <-- El reset síncrono no va en la lista de sensibilidad
        if Clk='1' and Clk'event then
            if Reset='1' then
                Q <= (others => '0');
            else
                Q <= D;
            end if;
        else if;
    end process;
end sintetizable;
```

FLIP-FLOP TIPO D con ENABLE

```
architecture sintetizable of flop is
begin
    process (Reset, Clk)
    begin
        if Reset = '1' then
            Q <= (others => '0');
        elsif Clk = '1' and Clk'event then
            if En = '1' then
                Q <= D;
            end if;
        end if;
    end process;
end architecture;
```

LATCH (CERROJO)

```
architecture sintetizable of latch is
begin
    process (Clk, D)
    begin
        if Clk = '1' then
            Q <= D;
        end if;
    end process;
end sintetizable;
```

LISTAS DE SENSIBILIDAD (RESUMEN)

- PROCESOS COMBINACIONALES
  - Señales que se leen:
    - ▷ Elementos que se comparan (if, case)
    - ▷ Elementos que se leen en asignaciones
- PROCESOS SÍNCRONOS
  - Reset síncrono
    - ▷ CLK
  - Reset asíncrono
    - ▷ CLK, Reset

TESTBENCH

PARTE 1 - INSTANCIACIÓN

```
entity testbench is    -- no hay entradas ni salidas (ports)
end;

architecture test of testbench1 is
    component MiAnd    -- declaración del uut
    port(a,b : in std_logic;
         y: out std_logic);
    end component;
        -- señales para conectar todos los puertos del uut
    signal a,b, y: std_logic;    --pueden ser nombres distintos
begin
   uut: MiAnd port map(
        a => a,
        b => b,
        y => y);
```

PARTE 2 - GENERACIÓN DE ESTÍMULOS

```
process    -- sin lista de sensibilidad => ¡wait!
begin
    a <= '0'; b <= '0';
    wait for 10 ns;
    a <= '0'; b <= '1';
    wait for 10 ns;
    a <= '1'; b <= '0';
    wait for 10 ns;
    a <= '1'; b <= '1';
    wait for 10 ns;
    wait;    -- "cuelga" este proceso, si no vuelve a empezar
end process;
end;    -- end del architecture
```

El simulador debe analizar el valor de la señal 'y' en cada caso y comprobar el correcto funcionamiento

assert condicion report "Texto" severity nivel;

Verifica que condición se cumple, sino saca "Texto" por el log del simulador y genera una excepción del nivel que se haya especificado.

Dependiendo del nivel (note, warning, error, failure), el simulador parará o no (configurable por usuario).