Repaso de VHDL para síntesis

Hipólito Guzmán Miranda
Profesor Titular
Universidad de Sevilla
hguzman@us.es

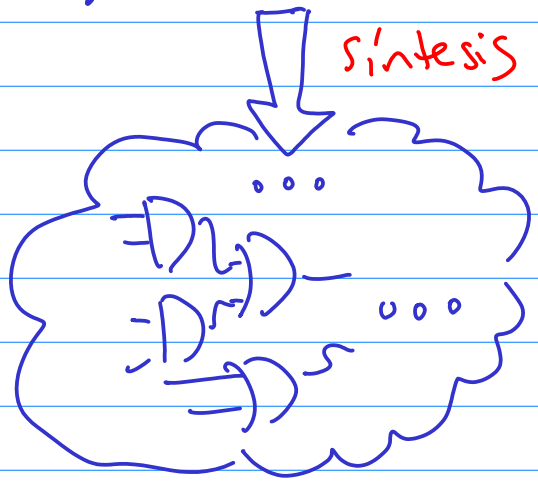
VHDL es un lenguaje de Descripción HW (HDL)

VHDL
- VHSIC
- HDL

Se usa para describir circuitos electrónicos (digitales) a alto nivel

```
y <= A + B;   VHDL
```

síntesis



Nivel de puerta

Análogo

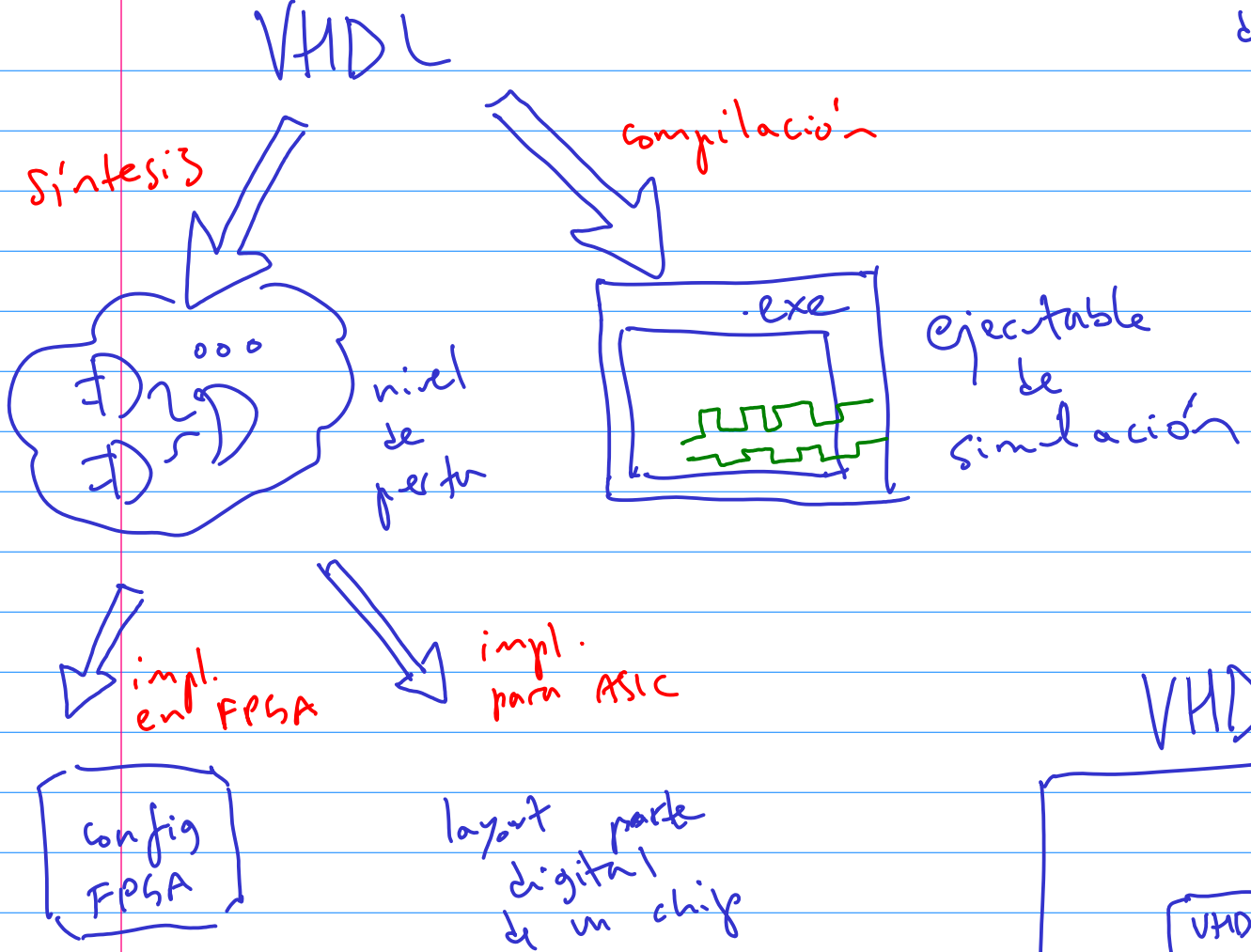
```
y = A + B;   lenguaje C
```

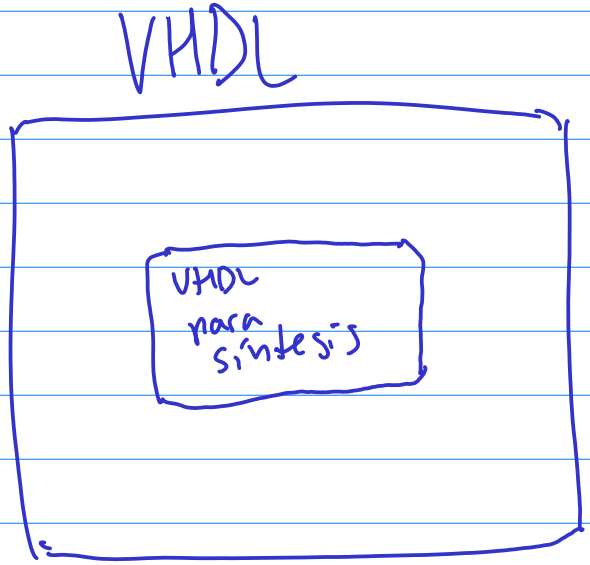
compilación

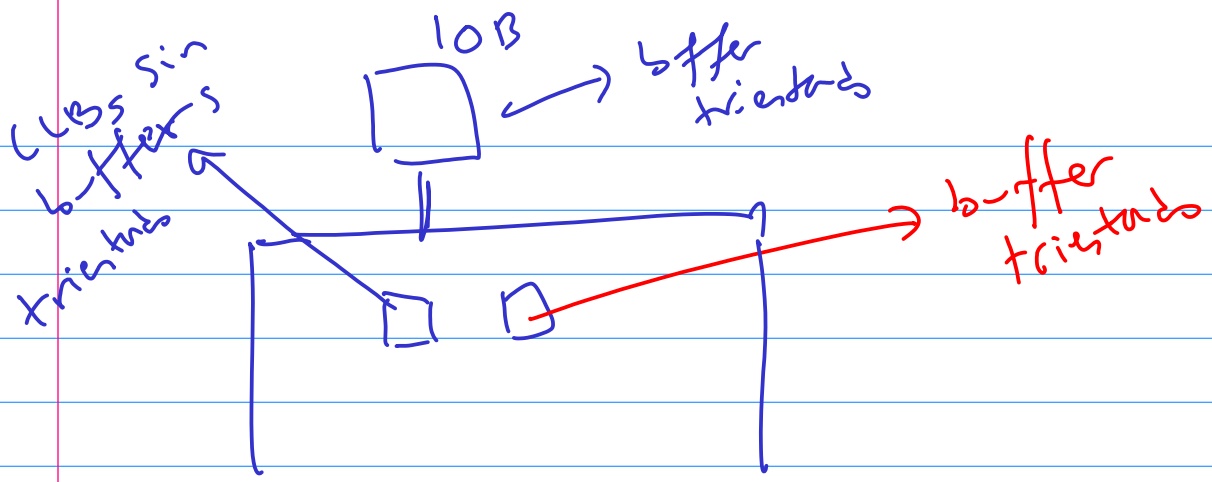
```
LD
Sum ...
JMP
```

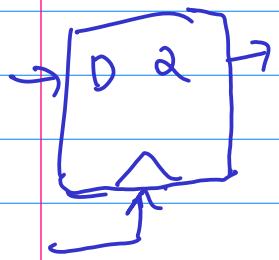
ensamblador

VHDL "desciende" del lenguaje Ada



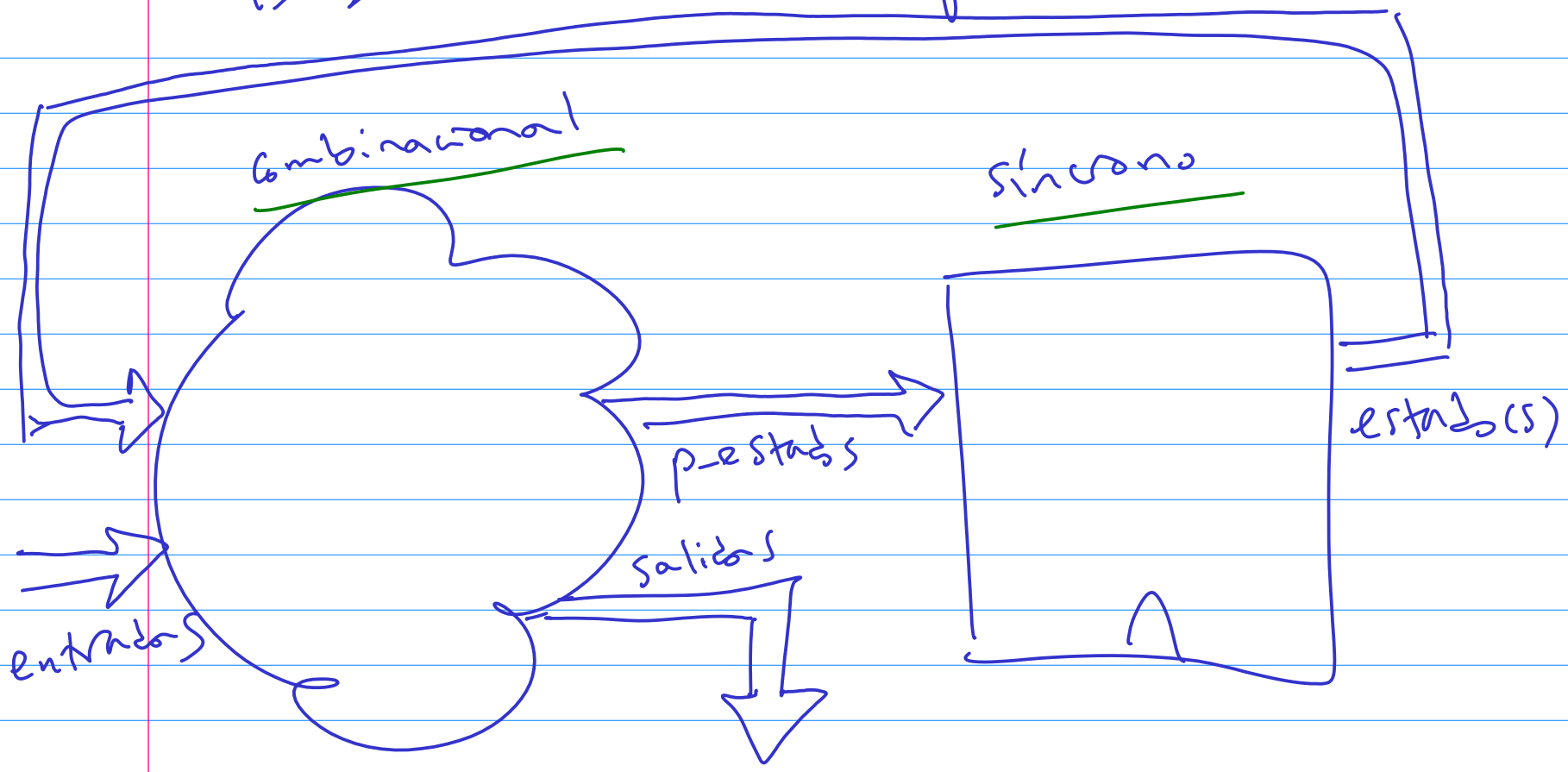






No todo lo que puede escribirse con VHDL legal (sintaxis correcta) describe un circuito implementable.

Diseño con 2 procesos:



toma las decisiones

implementa elementos de memoria (biestables)

El VHDL que conocéis

- Estructura de un fichero VHDL → describe 1 entity
- Sección Library
- Sección Entity
- Sección Architecture
  - Antes del begin
  - Después del begin
- [Sección Configuration] uso más minoritario

Secciones de un fichero VHDL

fichero.vhd

Library

Entity

Architecture

~~**Configuration**~~ (no se suele usar)

Sección Library

Análogo a los #include en C

Library

Inclusión de librerías y paquetes con:

Tipos de datos, Funciones, Componentes, ...

define tipo de dato std_logic y std_logic_vector

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

define tipo unsigned y signed y op. aritméticas

Sección Library

Ejemplo:

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

Sintaxis:

```
library lib_name;
use lib_name.package_name.all;
```

Sección Library

Paquetes a utilizar:

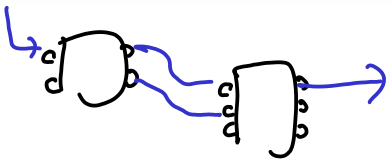
ieee.std_logic_1164.**all**;

ieee.numeric_std.**all**;

Sintaxis:

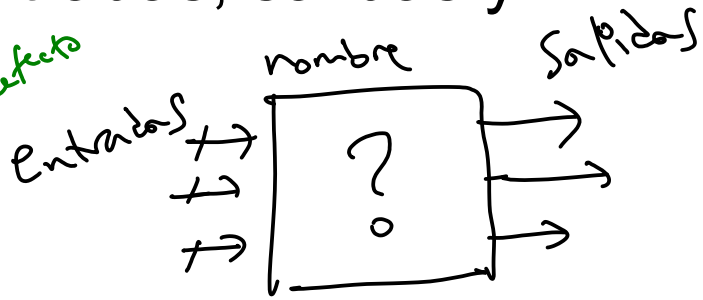
```
library lib_name;
use lib_name.package_name.all;
```

Sección Entity



Entity

Descripción de 'caja negra': entradas, salidas y parámetros (generics)

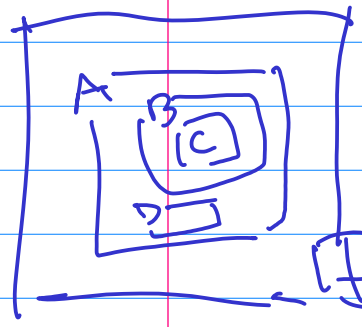



```
entity counter is
   Generic (N : integer := 8);
   Port ( rst    : in  STD_LOGIC;
          clk    : in  STD_LOGIC;
          enable : in  STD_LOGIC;
          count  : out STD_LOGIC_VECTOR (N-1 downto 0)
        );
end counter;
```

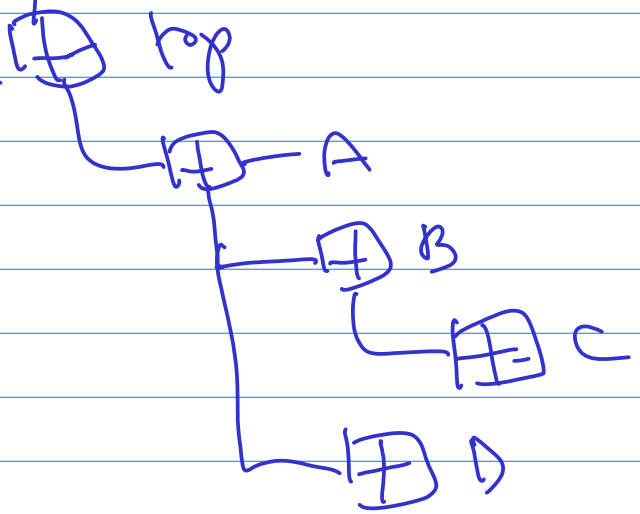
Unidad mínima de funcionalidad

en VHDL:

top

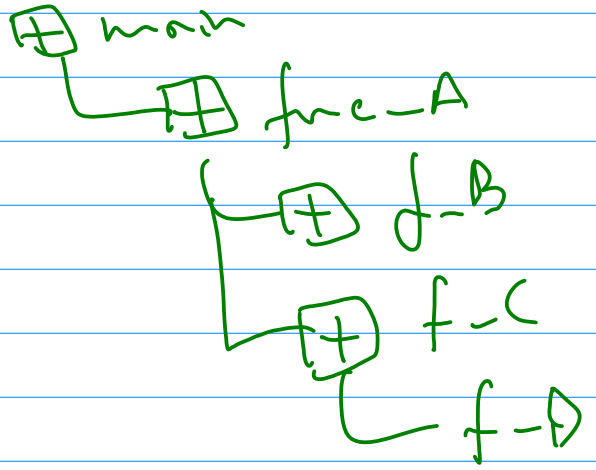


en entity



en C:

function



Sección Entity

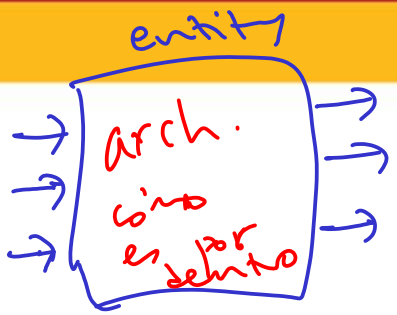
Sintaxis:

Direction debe ser in, out o bidir

```
entity entity_name is
  Generic (gen_name : data_type := default_value;
           <another generic>;
           <last port doesn't have separating ;>
          );
  Port ( port_name  : direction data_type;
         <another port>;
         <last port doesn't have separating ;>
        );
end entity_name;
```

Generics deben tomar valores ESTÁTICOS

- conocidos en tiempo de síntesis / compilación
- no cambian durante el funcionamiento del circuito



Sección Architecture

Dos partes diferenciadas:

- Antes del `begin`

Declaración de elementos que vamos a utilizar después

1)—
2)—
3)—

- Después del `begin`

"Todo lo que parezca un circuito funcionando"

1)—
2)—
3)—

Antes del begin

1) Se usa principalmente para FSMs

- Definición de tipos de dato
- Declaración de señales
- Declaración de componentes

```
1) type t_estado is (parada, lento, medio, rapido);
2) signal estado, p_estado: t_estado;
2) signal cuenta, p_cuenta: std_logic_vector(7 downto 0);

1) type enum_data_type is (first, second, third, fourth);
2) signal signal_name: data_type;
2) signal signal1, signal2: data_type;
```

Antes del begin

Declaración de componentes:

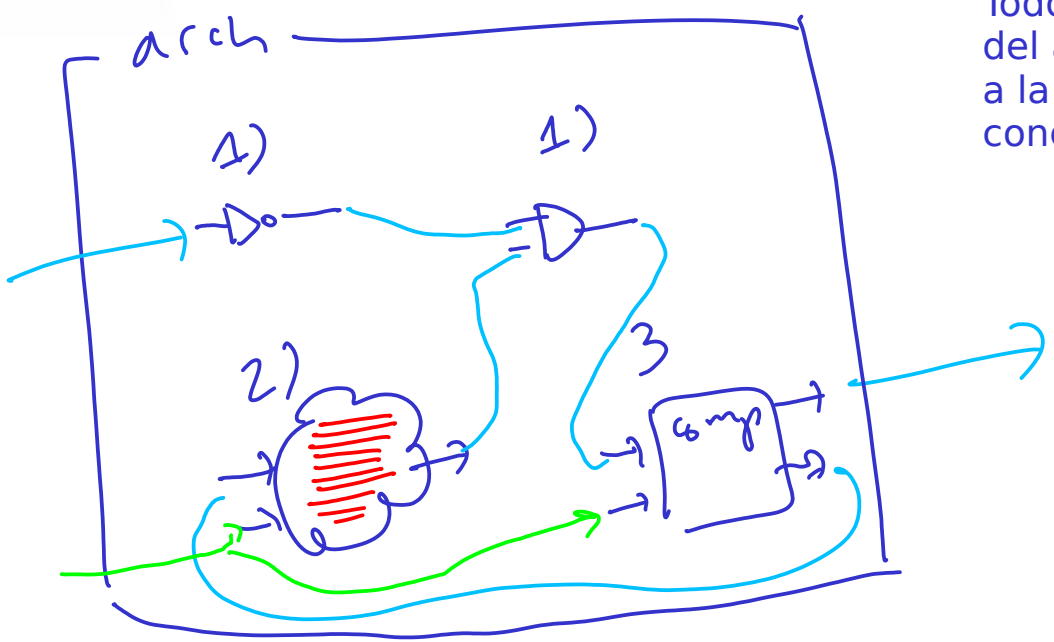
entidad (definida en otro fichero) que instancio

```
component counter is
  Generic (N : integer := 8;
           M : integer := 10);
  Port ( rst    : in  STD_LOGIC;
         clk    : in  STD_LOGIC;
         enable : in  STD_LOGIC;
         count  : out  STD_LOGIC_VECTOR
                       (N-1 downto 0));
end component;
```

Subsección Generic y Port son exactamente iguales a las de la entidad que estamos declarando como componente

Después del `begin`

1) • Sentencias concurrentes
2) • Process
3) • Instancias de componentes

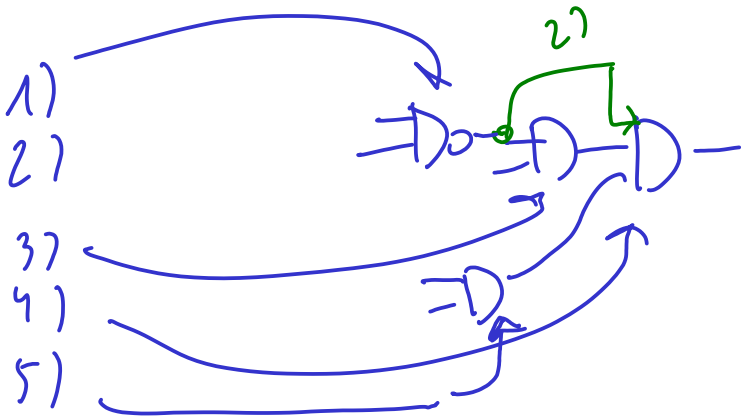


Todo lo que está tras el begin del architecture está funcionando a la vez (en parelelo, de manera concurrente)

Sentencias concurrentes

Sentencias concurrentes:

- Asignaciones: `b <= a;`
- Operaciones lógicas: `c <= a and (not b);`
- When… else
- With… select



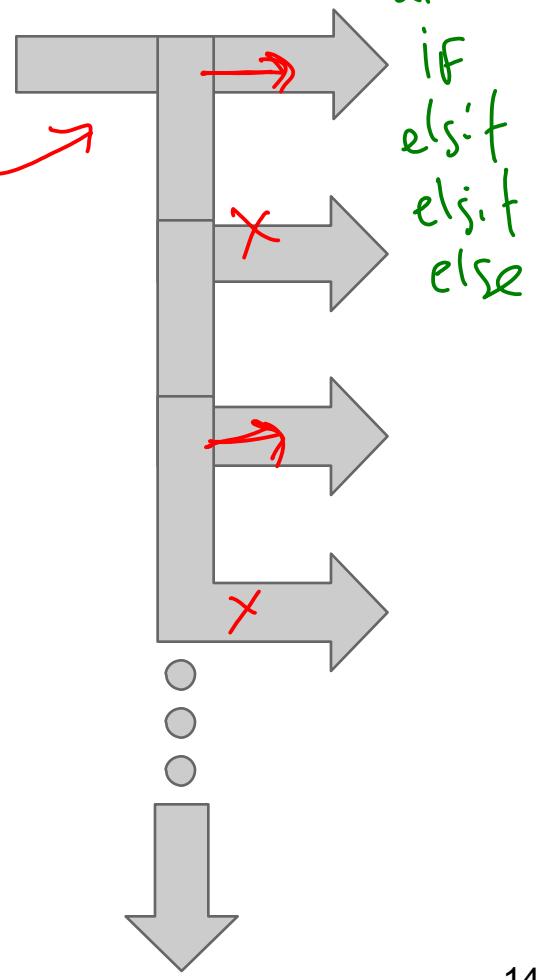
Se usa para implementar operaciones sencillas, usando pocas sentencias concurrentes en un architecture

Para implementar funcionalidades complejas es preferible usar process

Sentencias concurrentes

When… else:

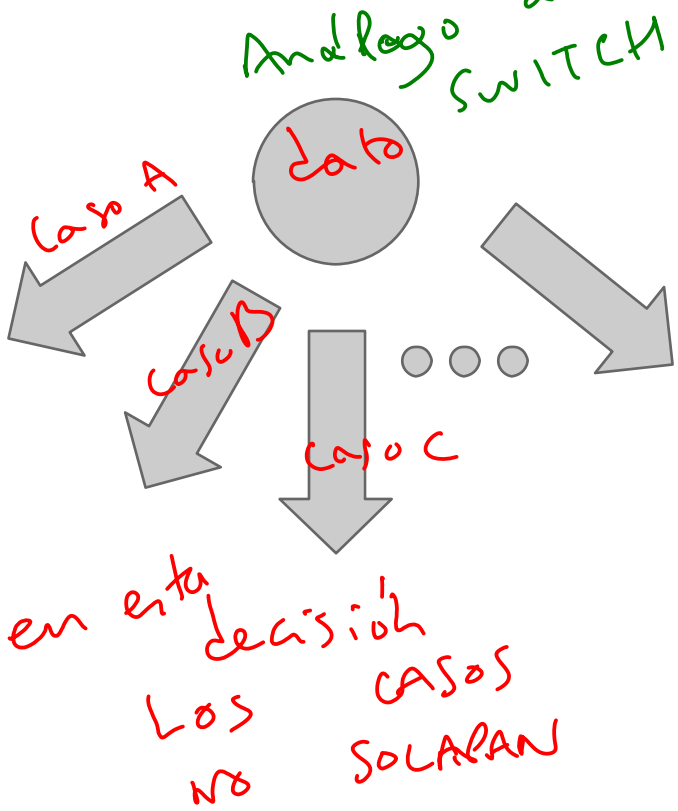
```
d <= (not a) when e="01" else
          b when e="10" else
          c;


sig1 <= expr1 when cond1 else
        expr2 when cond2 else
        <...>
        exprN;
```



Sentencias concurrentes

With… select:

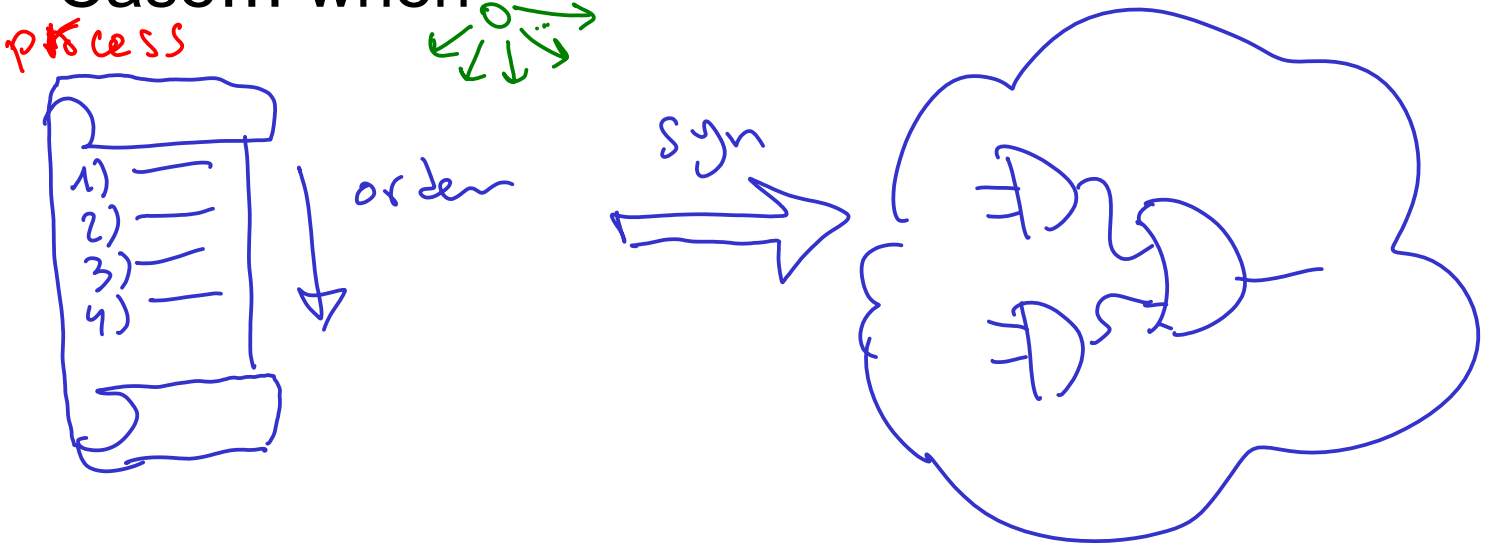
```
with e select
   d <= not a when "01",
             b when "10",
             c when others;

with sig1 select
   sig2 <= expr1 when value1,
           expr2 when value2,
           <...>
           expr3 when others;
```



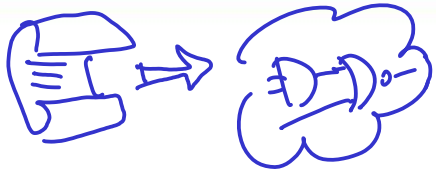
Después del `begin`

Procesos: (process)

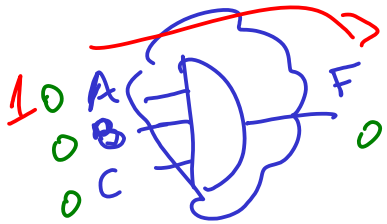
- Asignaciones: `b <= a;`
- Operaciones lógicas: `c <= a and (not b);`
- If… elsif… else
- Case… when

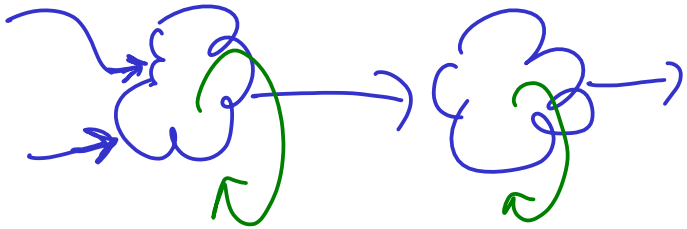


Procesos



```
comb: process (cont, enable)
  begin
    if (enable = '1') then
      p_cont <= cont + 1;
    else
      p_cont <= cont;
    end if;
  end process;
```





Procesos

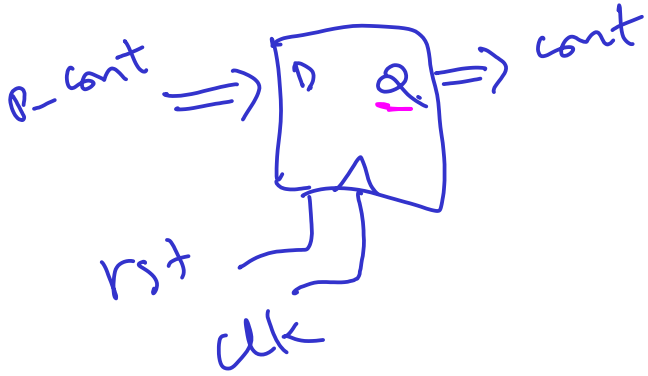
opcional

```
proc_name: process (lista_sensibilidad)
   begin
      <Sentencias>
   end process;
```

Señales o puertos que se leen
(todo lo que esté dentro de la condición
de un IF, CASE, y todo lo que esté
a la derecha de una asignación <=)

Procesos síncronos

```
sinc: process (rst, clk)
  begin
    if (rst='1') then
      cont <= (others=>'0');
    elsif (rising_edge(clk)) then
      cont <= p_cont;
    end if;
  end process;
```



No es sensible a p_cont porque sólo puede cambiar si hay un flanco de clk

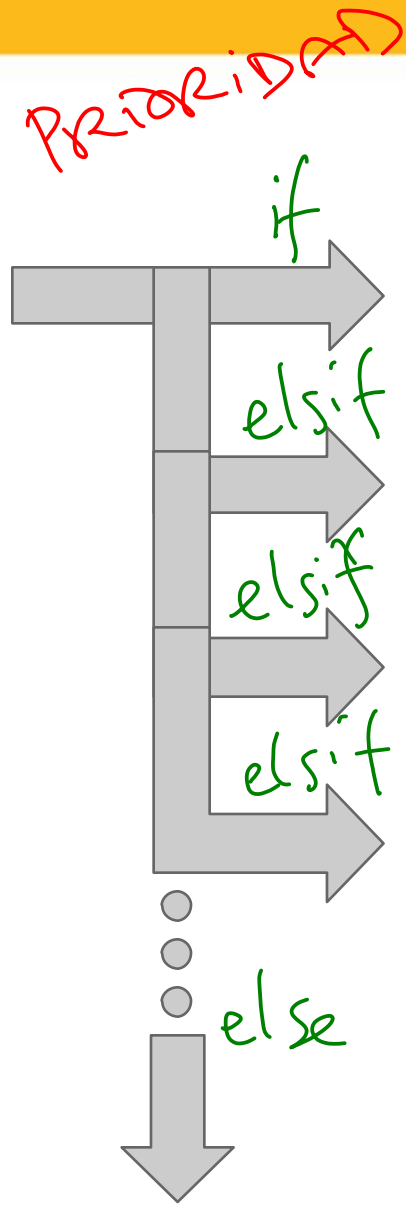
Los procesos síncronos siempre se describen de la misma manera

Procesos

If… elsif… else:

```
if (rst_sync = '1') then
   p_cont <= (others=>'0');
elsif (enable = '1') then
   p_cont <= cont + 1;
else
   p_cont <= cont;
end if;
```

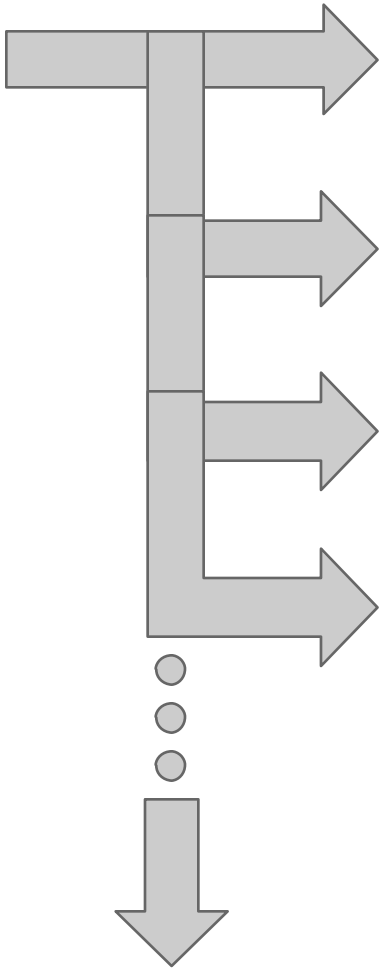
elsif y else son opcionales



Procesos

If… elsif… else:

```
if (cond1) then
  <sentencias>
elsif (cond2) then
  <sentencias>
<... más elsif ...>
else
  <sentencias>
end if;
```

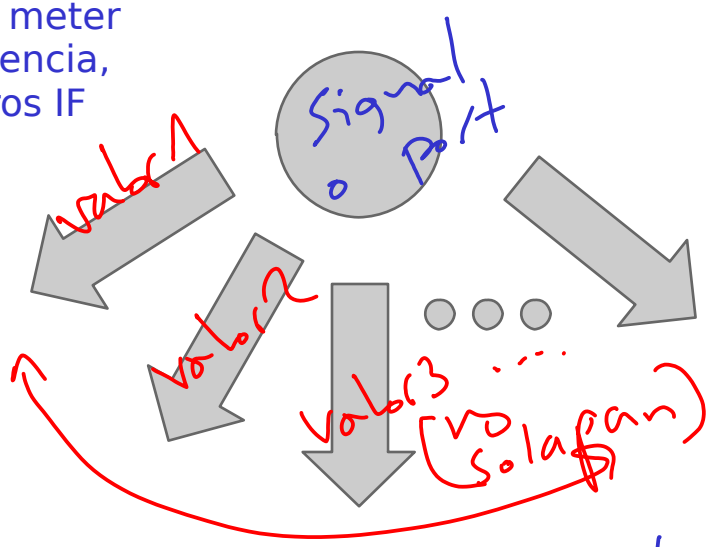


Procesos

Case… when:

```
case state is
     when idle =>
       <sentencias>
     when count =>
       <sentencias>
     when header =>
       <sentencias>
     when others =>
       <sentencias>
end case;
```

aquí podemos meter cualquier sentencia, incluyendo otros IF y case



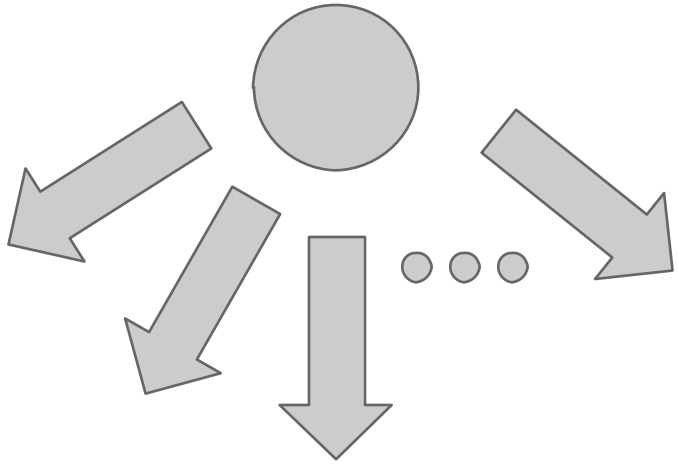
Muy utilizado para describir máquinas de estados (FSM)

Procesos

Case… when:

signal o port (object)

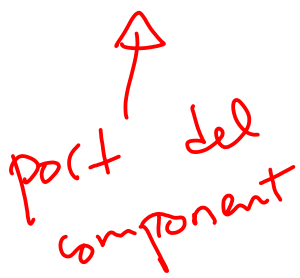
```
case sig1 is
    when value1 =>
      <sentencias>
    when value2 =>
      <sentencias>
    when value3 =>
      <sentencias>
    when others =>
      <sentencias>
end case;
```

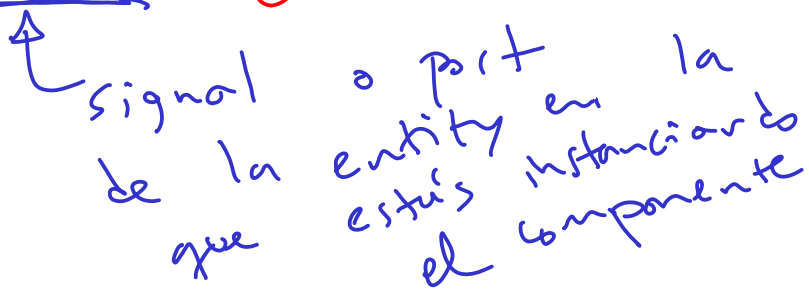


Muy utilizado para describir máquinas de estados

Instancias de componentes

```
cont_inst: counter
generic map ( N => 8, M => 10 )
port map (
        rst_high => sig_rst_high,
        enable => sig_enable,
        clk => clk,
        data_out => sig_data_out
    );
```





Instancias de componentes

```
inst_name: component_name
generic map ( gen1 => val1, gen2 => val2 )
port map (
      port1 => sig_top1,
      port2 => sig_top2,
      port3 => sig_top3,
      <...>
      portN => sig_topN
    );
```

Component_port => top_signal_or_port,

VHDL

Library IEEE → std_logic_1164 (siempre)
→ numeric_std

Entity
Generic
Port → in, out, [bidir]

Architecture
1) tipos de datos
2) signals
3) components

begin
1) sent. concurrentes
2) PROCESS → combinacionales
→ síncronos
3) inst. componentes

Ejercicio

Diseñemos y simulemos un contador de N bits (instanciado con N=8) con Xilinx ISE

Entradas:

Clk, rst, enable : std_logic;

Salidas;

Cuenta : std_logic_vector (7 downto 0);