

Repaso de VHDL para síntesis

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

El VHDL que conocéis

- Estructura de un fichero VHDL
- Sección Library
- Sección Entity
- Sección Architecture
 - Antes del begin
 - Después del begin

Secciones de un fichero VHDL

Library

Entity

Architecture

~~Configuration~~ (no se suele usar)

Sección Library

Library

Inclusión de librerías y paquetes con:

Tipos de datos, Funciones, Componentes, ...

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

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;
```

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;
```

Sección Architecture

Dos partes diferenciadas:

- Antes del **begin**
- Después del **begin**

Antes del begin

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

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

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

Antes del begin

Declaración de componentes:

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));
```

Subsección

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

end component;

Después del begin

- Sentencias concurrentes
- Process
- Instancias de componentes

Sentencias concurrentes

Sentencias concurrentes:

- Asignaciones: $b \leq a;$
- Operaciones lógicas: $c \leq a \text{ and } (\text{not } b);$
- Operaciones aritméticas: $f \leq d + e;$
- **when ... else**
- **with ... select**

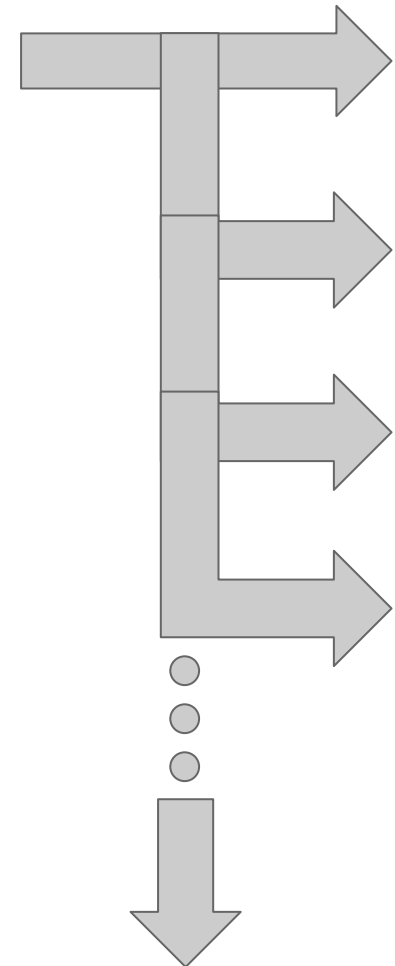
Sentencias concurrentes

When... else:

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

```
obj1 <= expr1 when cond1 else
      expr2 when cond2 else
      <...>
      exprN;
```

- No pueden usarse dentro de un process



Sentencias concurrentes

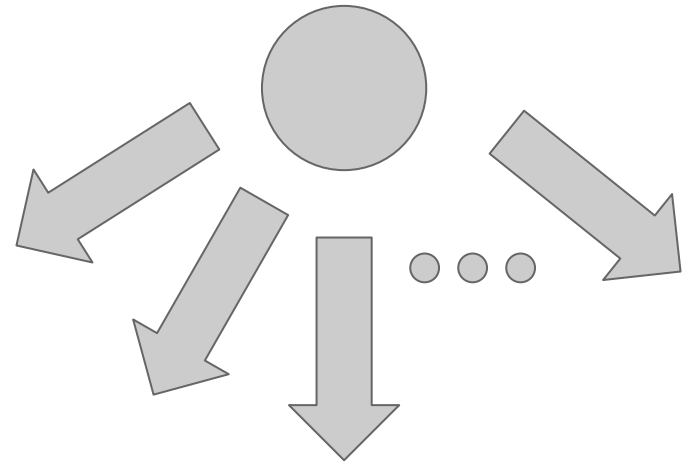
With... select:

with e select

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

with obj1 select

```
obj2 <= expr1 when value1,
      expr2 when value2,
      <...>
      expr3 when others;
```



- No pueden usarse dentro de un process

Procesos

Procesos:

- Asignaciones: $b \leq a;$
- Operaciones lógicas: $c \leq a \text{ and } (\text{not } b);$
- Operaciones aritméticas: $f \leq d + e;$
- **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

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

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;
```

Los procesos síncronos siempre se describen de la misma manera

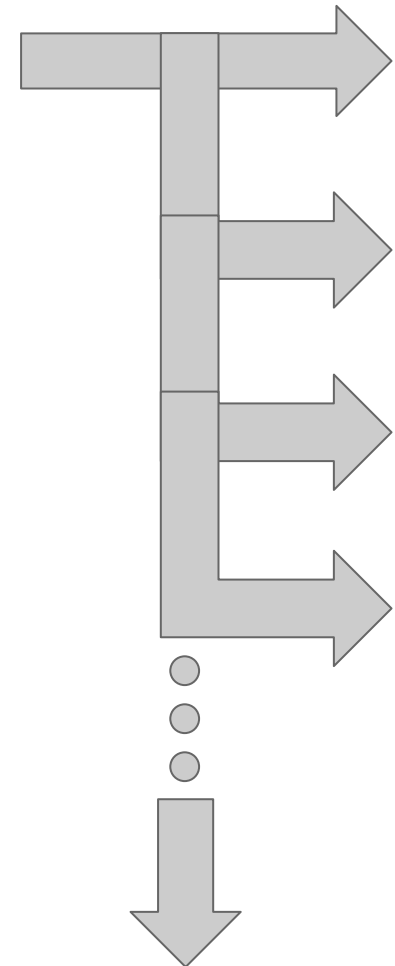
Procesos

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;
    
```

- No puede usarse fuera de un process

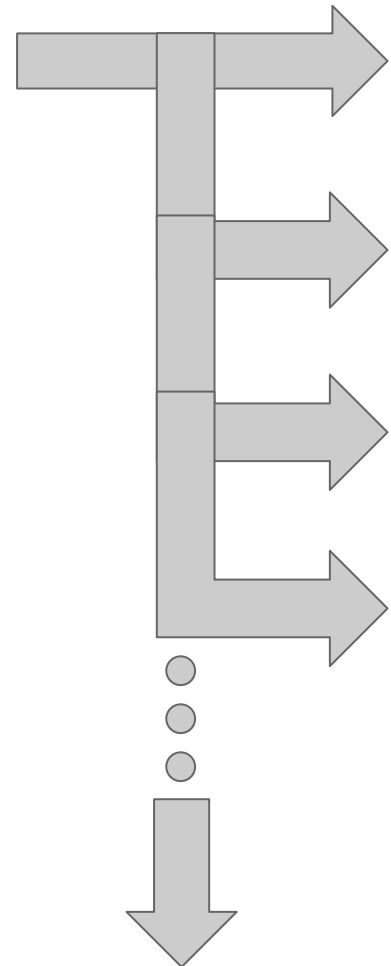


Procesos

if... elsif... else:

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

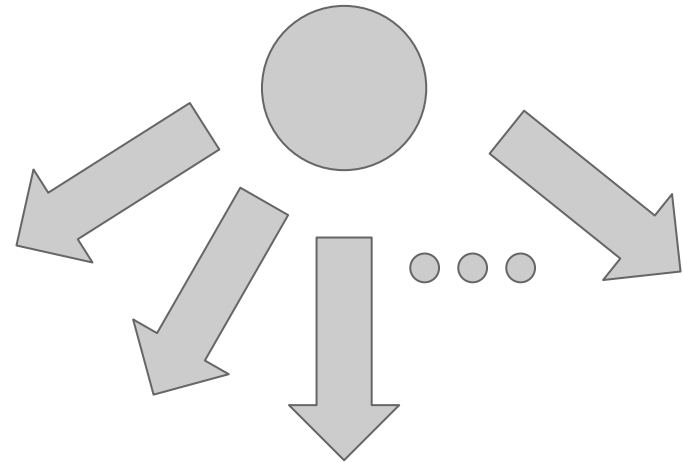
- No puede usarse fuera de un process



Procesos

Case... when:

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



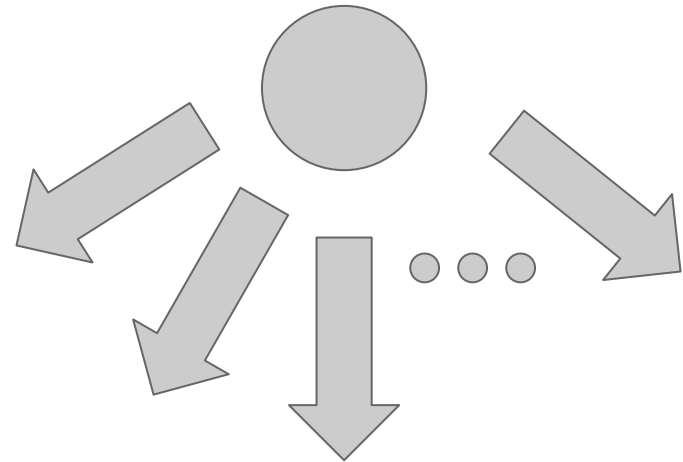
Muy utilizado
para describir
máquinas de
estados

- No puede usarse fuera de un process

Procesos

Case... when:

```
case obj1 is  
  when value1 =>  
    <sentencias>  
  when value2 =>  
    <sentencias>  
  when value3 =>  
    <sentencias>  
  when others =>  
    <sentencias>  
end case;
```



Muy utilizado
para describir
máquinas de
estados

- No puede usarse fuera de un process

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,

Ejercicio

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

Entradas:

`clk, rst, enable : std_logic;`

Salidas:

`Q : unsigned (N-1 downto 0);`