VHDL for synthesis: review

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

The VHDL you (should) already know

- Structure of a VHDL file
- Library section
- Entity section
- Architecture section
  - Before begin
  - After begin

Sections of a VHDL file

`Library`

`Entity`

`Architecture`

~~`Configuration`~~ (not commonly used)

Library section

Library

Inclusion of libraries and packages with:

Data types, functions, components, …

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

Library section

Example:

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

Syntax:

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

Library section

Packages to use:

```
ieee.std_logic_1164.all;

ieee.numeric_std.all;
```

Syntax:

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

Entity section

Entity

‘Black box’ description: inputs, outputs and parameters (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;
```

Entity section

Syntax:

Direction must be in, out or 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;
```

Architecture section

Two distinct parts:

- Before `begin`
- After `begin`

Before begin

- Data type definitions
- Signal declarations
- Component declarations

```
type t_state is (stop, slow, mid, fast);
signal state, n_state: t_state;
signal count, n_count: 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;
```

Before begin

Component declarations:

```
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;
```

Subsection Generic and Port are exactly equal to those of the entity which we are declaring as a component

After begin

- Concurrent statements
- Process
- Instances of components

Concurrent statements

Concurrent statements:

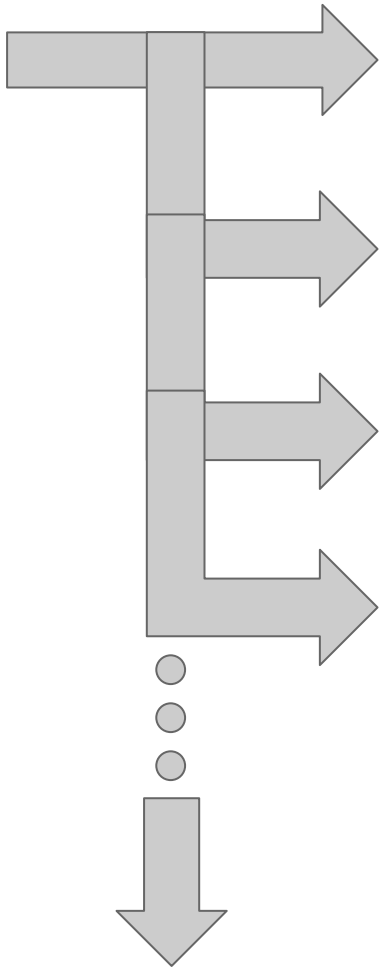
- Assignments: `b <= a;`
- Logic operations: `c <= a and (not b);`
- Arithmetic operations: `f <= d + e;`
- `when … else`
- `with … select`

Concurrent statements

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;
```



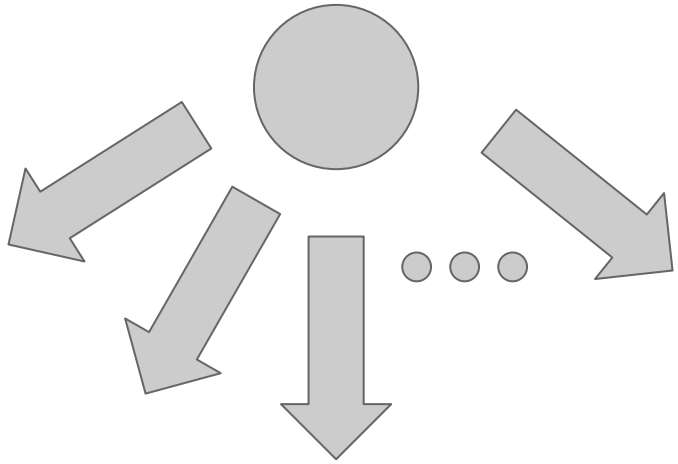
- Cannot be used **inside a process**

Concurrent statements

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;
```



- Cannot be used **inside a process**

Process

Process:

- Assignments: `b <= a;`
- Logic operations: `c <= a and (not b);`
- Arithmetic operations: `f <= d + e;`
- **if**… **elsif** … **else**
- **case** … **when**

Architecture: after **begin**

Process

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

Processs

```
proc_name: process (sensitivity_list)
  begin
    <Statements>
  end process;
```

Synchronous process

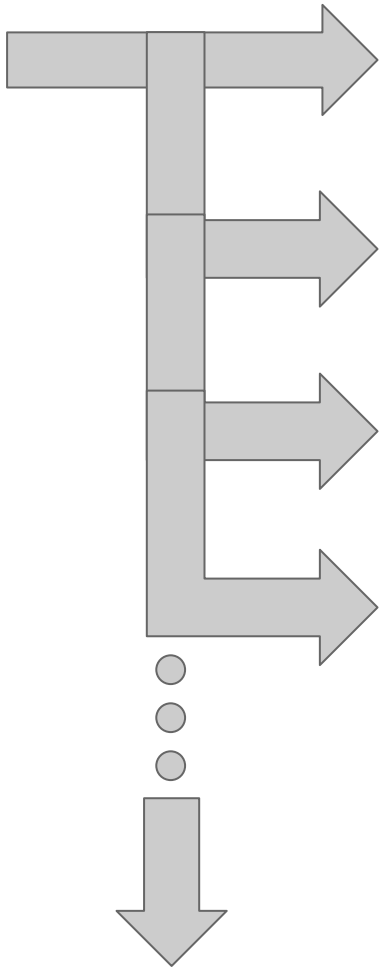
```
sync: process (rst, clk)
  begin
    if (rst='1') then
      cont <= (others=>'0');
    elsif (rising_edge(clk)) then
      cont <= n_cont;
    end if;
  end process;
```

Synchronous processes are always described in the same way

Process

If… elsif… else:

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

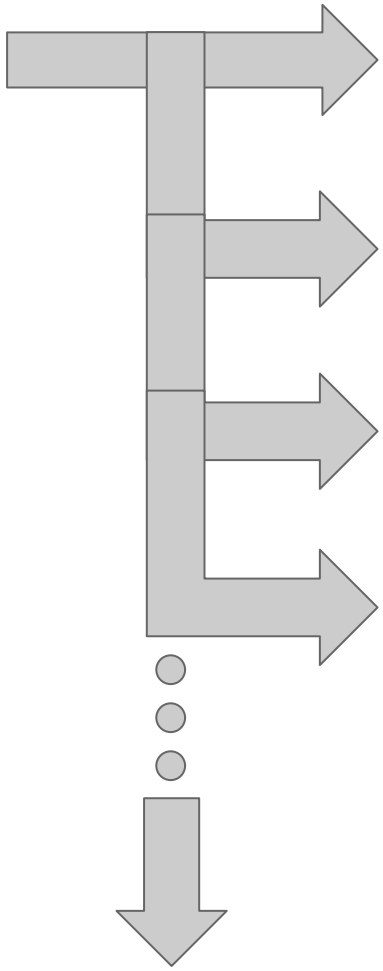


- Cannot be used **outside of a process**

Process

If… elsif… else:

```
if (cond1) then
  <statements>
elsif (cond2) then
  <statements>
<... more elsif …>
else
  <statements>
end if;
```

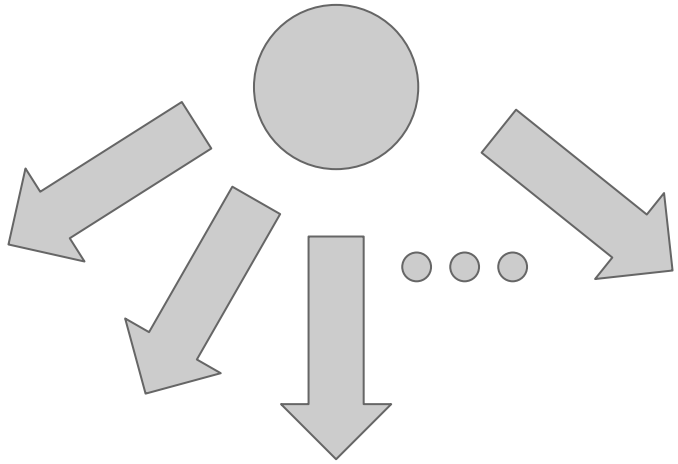


- Cannot be used **outside of a process**

Process

Case… when:

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



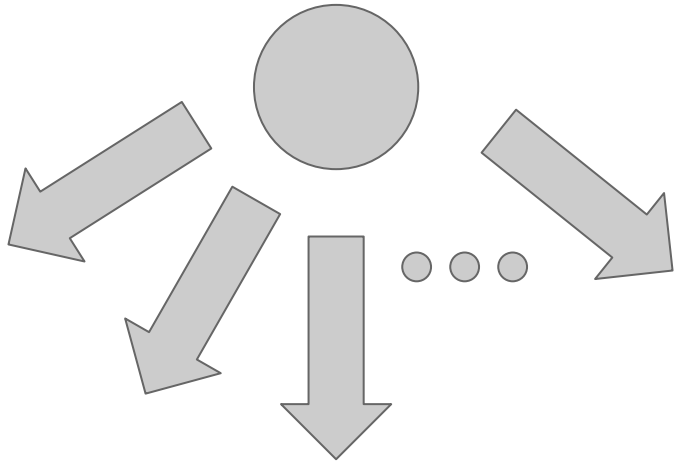
Widely used to describe state machines

- Cannot be used **outside of a process**

Process

Case… when:

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



Widely used to describe state machines

- Cannot be used **outside of a process**

Instances of components

```
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
    );
```

Instances of components

```
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,

Exercise

Let's design and simulate an N-bit counter (instanced with N=8)

Inputs:

```
clk, rst, enable : std_logic;
```

Salidas;

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