UNIVERSITÀ
DEGLI STUDI DI TRIESTE

dipartimento
di ingegneria
e architettura

12 – Generic constants

A.Carini – Progettazione di sistemi elettronici

Parameterizing behavior

```
entity_declaration ⇐
    entity identifier is
        [ generic ( generic_interface_list ) ; ]
        [ port ( port_interface_list ) ; ]
        { entity_declarative_item }
    [ begin
        { concurrent_assertion_statement
            | passive_concurrent_procedure_call_statement
            | passive_process_statement } ]
    end [ entity ] [ identifier ] ;
```

```
generic_interface_list ⇐
    ( identifier { , ... } : subtype_indication [ := expression ] )
    { ; ... }
```

Example

```
entity and2 is
      generic ( Tpd : time );
      port ( a, b : in bit;  y : out bit );
end entity and2;
```

```
architecture simple of and2 is
begin
      and2_function :
            y <= a and b after Tpd;
end architecture simple;
```

Generic map

component_instantiation_statement ⇐
*instantiation*_label :
entity *entity*_name ⟦ (*architecture*_identifier) ⟧
⟦ **generic map** (*generic*_association_list) ⟧
⟦ **port map** (*port*_association_list) ⟧ ;

association_list ⇐ assiciation_element { , ... }

association_element ⇐
⟦ *generic*_name => ⟧ ⟪ expression | **open** ⟫

Example

```
gate1 : entity work.and2(simple)
        generic map ( Tpd => 2 ns )
        port map ( a => sig1,  b => sig2,  y => sig_out );
```

```
gate2 : entity work.and2(simple)
        generic map ( Tpd => 3 ns )
        port map ( a => a1,  b => b1,  y => sig1 );
```

Example

```
entity control_unit is
    generic ( Tpd_clk_out, Tpw_clk : delay_length;
              debug : boolean := false );
    port ( clk : in bit;
           ready : in bit;
           control1, control2 : out bit );
end entity control_unit;
```

Example associations

```
generic map ( 200 ps, 1500 ps, false )
generic map ( Tpd_clk_out => 200 ps, Tpw_clk => 1500 ps )
generic map ( 200 ps, 1500 ps, debug => open )
```

Example flip-flop D

```
entity D_flipflop is
      generic ( Tpd_clk_q, Tsu_d_clk, Th_d_clk : delay_length );
      port ( clk, d : in bit;  q : out bit );
end entity D_flipflop;
-------------------------------------------------------------------------------
architecture basic of D_flipflop is
begin
      behavior : q <= d after Tpd_clk_q when clk = '1' and clk'event;
      check_setup : process is
      begin
            wait until clk = '1';
            assert d'last_event >= Tsu_d_clk
                  report "setup violation";
      end process check_setup;
      check_hold : process is
      begin
            wait until clk'delayed(Th_d_clk) = '1';
            assert d'delayed'last_event >= Th_d_clk
                  report "hold violation";
      end process check_hold;
end architecture basic;
```

Example

```
request_flipflop : entity work.D_flipflop(basic)
    generic map ( Tpd_clk_q => 4 ns,
                  Tsu_d_clk => 3 ns, Th_d_clk => 1 ns )
    port map ( clk => system_clock,
               d => request, q => request_pending );
```

Parameterizing structure

- It’s the second use of *generic contants.*
- For example: for specifying an array port size.
- A practical motivation: a register declared with *array unconstrained ports …*

```
entity reg is
      port ( d : in bit_vector;  q : out bit_vector;  … );
end entity reg;
```

```
signal small_data : bit_vector(0 to 7);
signal large_data : bit_vector(0 to 15);
…
problem_reg : entity work.reg
      port map ( d => small_data,  q => large_data, … );
```

Otherwise with generics

```
entity reg is
        generic ( width : positive );
        port ( d : in bit_vector(0 to width – 1);
                 q : out bit_vector(0 to width – 1);
                 ... );
end entity reg;
```

```
signal in_data, out_data : bit_vector(0 to bus_size – 1);
...
ok_reg : entity work.reg
        generic map ( width => bus_size )
        port map ( d => in_data,  q => out_data, ... );
```

Register architecture

```
entity reg is
      generic ( width : positive );
      port ( d :  in  bit_vector(0 to width – 1);
             q :  out  bit_vector(0 to width – 1);
             clk, reset : in bit );
end entity reg;
-------------------------------------------------------------------
architecture behavioral of reg is
begin
      behavior : process (clk, reset) is
            constant zero : bit_vector(0 to width – 1) := (others => '0');
      begin
            if reset = '1' then
                  q <= zero;
            elsif clk'event and clk = '1' then
                  q <= d;
            end if;
      end process behavior;
end architecture behavioral;
```

Example

```
word_reg : entity work.reg(behavioral)
  generic map ( width => 32 )
  port map ( ... );
```

```
subtype state_vector is bit_vector(1 to 5);
...

state_reg : entity work.reg(behavioral)
  generic map ( width => state_vector'length )
  port map ( ... );
```

See:

- Peter Ashenden, «The designers' guide to VHDL» Morgan Kaufmann,
  - Chapter 12