13 – Components and configurations

A.Carini – Progettazione di sistemi elettronici

Components

- We have already seen how to build a hieratical structure composed by interconnected systems by directly instantiating *entity – architecture* pairs.
- An alternative approach considers writing some *component declarations* (that shall not be confused with the entity declarations) in the declarative part of an *architecture body* or a *package*.
- We will then instantiate the components in the architecture bodies of our hierarchy.

Component declaration

```
component_declaration ⇐
    component identifier [ is ]
        [ generic ( generic_interface_list ) ; ]
        [ port ( port_interface_list ) ; ]
    end component [ identifier ] ;
```

- Example:

```
component flipflop is
    generic ( Tprop, Tsetup, Thold : delay_length );
    port ( clk : in bit;  clr : in bit;  d : in bit;
           q : out bit );
end component flipflop;
```

Component instantiation

component_instantiation_statement ⇐
*instantiation*_label :
[**component**] *component*_name
[**generic map** (*generic*_association_list)]
[**port map** (*port*_association_list)] ;

Example

```
entity reg4 is
      port ( clk, clr : in bit;  d : in bit_vector(0 to 3);
               q : out bit_vector(0 to 3) );
end entity reg4;
-------------------------------------------------------------------
architecture struct of reg4 is
      component flipflop is
            generic ( Tprop, Tsetup, Thold : delay_length );
            port ( clk : in bit;  clr : in bit;  d : in bit;
                     q : out bit );
      end component flipflop;
begin
      bit0 : component flipflop
            generic map ( Tprop => 2 ns, Tsetup => 2 ns, Thold => 1 ns )
            port map ( clk => clk, clr => clr, d => d(0), q => q(0) );
      bit1 : component flipflop
            generic map ( Tprop => 2 ns, Tsetup => 2 ns, Thold => 1 ns )
            port map ( clk => clk, clr => clr, d => d(1), q => q(1) );
      bit2 : component flipflop
            generic map ( Tprop => 2 ns, Tsetup => 2 ns, Thold => 1 ns )
            port map ( clk => clk, clr => clr, d => d(2), q => q(2) );
      bit3 : component flipflop
            generic map ( Tprop => 2 ns, Tsetup => 2 ns, Thold => 1 ns )
            port map ( clk => clk, clr => clr, d => d(3), q => q(3) );
end architecture struct;
```

Packaging components

```
library ieee;  use ieee.std_logic_1164.all;
package serial_interface_defs is
    subtype reg_address_vector is std_logic_vector(1 downto 0);
    constant status_reg_address : reg_address_vector := B"00";
    constant control_reg_address : reg_address_vector := B"01";
    constant rx_data_register : reg_address_vector := B"10";
    constant tx_data_register : reg_address_vector := B"11";
    subtype data_vector is std_logic_vector(7 downto 0);
    ...      -- other useful declarations
    component serial_interface is
        port ( clock_phi1, clock_phi2 : in std_logic;
               serial_select : in std_logic;
               reg_address : in reg_address_vector;
               data : inout data_vector;
               interrupt_request : out std_logic;
               rx_serial_data : in std_logic;
               tx_serial_data : out std_logic );
    end component serial_interface;
end package serial_interface_defs;
```

Corresponding entity

```
library ieee;  use ieee.std_logic_1164.all;
use work.serial_interface_defs.all;
entity serial_interface is
      port ( clock_phi1, clock_phi2 : in std_logic;
             serial_select : in std_logic;
             reg_address : in reg_address_vector;
             data : inout data_vector;
             interrupt_request : out std_logic;
             rx_serial_data : in std_logic;
             tx_serial_data : out std_logic );
end entity serial_interface;
```

Example of component instance

```
library ieee;  use ieee.std_logic_1164.all;
architecture structure of microcontroller is
    use work.serial_interface_defs.serial_interface;
    ...      -- declarations of other components, signals, etc
begin
    serial_a : component serial_interface
        port map ( clock_phi1 => buffered_phi1,
                   clock_phi2 => buffered_phi2,
                   serial_select => serial_a_select,
                   reg_address => internal_addr(1 downto 0),
                   data => internal_data_bus,
                   interrupt_request => serial_a_int_req,
                   rx_serial_data => rx_data_a,
                   tx_serial_data => tx_data_a );
    ...      -- other component instances
end architecture structure;
```

Basic configuration declaration

```
configuration_declaration ⇐
    configuration identifier of entity_name is
        for architecture_name
            { for component_specification
                binding_indication ;
              end for ; }
        end for ;
    end [ configuration ] [ identifier ] ;
component_specification ⇐
    ( instantiation_label { , ... } | others | all ) : component_name
binding_indication ⇐ use entity entity_name [ ( architecture_identifier ) ]
```

Example of binding indication

```
for bit0, bit1 : flipflop
        use entity work.edge_triggered_Dff(basic);
end for;
```

Complete example

```
library star_lib;
use star_lib.edge_triggered_Dff;
configuration reg4_gate_level of reg4 is
    for struct    -- architecture of reg4
        for bit0 : flipflop
            use entity edge_triggered_Dff(hi_fanout);
        end for;
        for others : flipflop
            use entity edge_triggered_Dff(basic);
        end for;
    end for;    -- end of architecture struct
end configuration reg4_gate_level;
```

Configuring multiple levels of hierarchy

binding_indication ⇐ **use configuration** *configuration*_name

Example:

```
for flag_reg : reg4
      use configuration work.reg4_gate_level;
end for;
```

Example

```
use work.counter_types.digit;  →  subtype digit is bit_vector(3 downto 0);
entity counter is
    port ( clk, clr : in bit;
           q0, q1 : out digit );
end entity counter;
-----------------------------------------------------------------
architecture registered of counter is
    component digit_register is
        port ( clk, clr : in bit;
               d : in digit;
               q : out digit );
    end component digit_register;
    signal current_val0, current_val1, next_val0, next_val1 : digit;
begin
    val0_reg : component digit_register
        port map ( clk => clk, clr => clr, d => next_val0,
                   q => current_val0 );
    val1_reg : component digit_register
        port map ( clk => clk, clr => clr, d => next_val1,
                   q => current_val1 );
    -- other component instances
    ...
end architecture registered;
```

Example

```
configuration counter_down_to_gate_level of counter is
    for registered
        for all : digit_register
            use configuration work.reg4_gate_level;
        end for;
        ...      -- bindings for other component instances
    end for;   -- end of architecture registered
end configuration counter_down_to_gate_level;
```

Hieratical configuration declaration

```
configuration_declaration ⇐
    configuration identifier of entity_name is
        block_configuration
    end [ configuration ] [ identifier ] ;
block_configuration ⇐
    for architecture_name
        {   for component_specification
                binding_indication ;
                [ block_configuration ]
            end for ; }
    end for ;
```

Example of a single configuration

```
library star_lib;
use star_lib.edge_triggered_Dff;
configuration full of counter is
    for registered    -- architecture of counter
        for all : digit_register
            use entity work.reg4(struct);
            for struct    -- architecture of reg4
                for bit0 : flipflop
                    use entity edge_triggered_Dff(hi_fanout);
                end for;
                for others : flipflop
                    use entity edge_triggered_Dff(basic);
                end for;
            end for;    -- end of architecture struct
        end for;
        ...      -- bindings for other component instances
    end for;    -- end of architecture registered
end configuration full;
```

Direct instantiation of configured elements

component_instantiation_statement ⇐
    *instantiation*_label :
        configuration *configuration*_name
        [**generic map** (*generic*_association_list)]
        [**port map** (*port*_association_list)] ;

Example

```
architecture top_level of alarm_clock is
    use work.counter_types.digit;
    signal reset_to_midnight, seconds_clk : bit;
    signal seconds_units, seconds_tens : digit;
    ...
begin
    seconds : configuration work.counter_down_to_gate_level
        port map ( clk => seconds_clk, clr => reset_to_midnight,
                   q0 => seconds_units, q1 => seconds_tens );
    ...
end architecture top_level;
```

Generics and port maps in configurations

```
binding_indication ⇐
    use ⟪  entity entity_name ⟦ ( architecture_identifier ) ⟧
         ⟦ configuration configuration_name ⟫
    ⟦ generic map ( generic_association_list ) ⟧
    ⟦ port map ( port_association_list ) ⟧
```

Example of generics

```
library ieee;  use ieee.std_logic_1164.all;
entity reg is
      generic ( t_setup, t_hold, t_pd : delay_length;
                width : positive );
      port ( clock : in std_logic;
             data_in : in std_logic_vector(0 to width – 1);
             data_out : out std_logic_vector(0 to width – 1) );
end entity reg;
```

Example of generics

```
architecture structural of controller is
    component reg is
        generic ( width : positive );
        port ( clock : in std_logic;
               data_in : in std_logic_vector(0 to width – 1);
               data_out : out std_logic_vector(0 to width – 1) );
    end component reg;
    ...
begin
    state_reg : component reg
        generic map ( width => state_type'length )
        port map ( clock => clock_phase1,
                   data_in => next_state,
                   data_out => current_state );
    ...
end architecture structural;
```

Example of generics

```
configuration controller_with_timing of controller is
    for structural
        for state_reg : reg
            use entity work.reg(gate_level)
            generic map ( t_setup => 200 ps, t_hold => 150 ps, t_pd => 150 ps,
                          width => width );
        end for;
        ...
    end for;
end configuration controller_with_timing;
```

Example of different port maps

```
architecture structure of computer_system is
    component decoder_2_to_4 is
        generic ( prop_delay : delay_length );
        port ( in0, in1 : in bit;
               out0, out1, out2, out3 : out bit );
    end component decoder_2_to_4;
    ...
begin
    interface_decoder : component decoder_2_to_4
        generic map ( prop_delay => 4 ns )
        port map ( in0 => addr(4), in1 => addr(5),
                   out0 => interface_a_select, out1 => interface_b_select,
                   out2 => interface_c_select, out3 => interface_d_select );
    ...
end architecture structure;
```

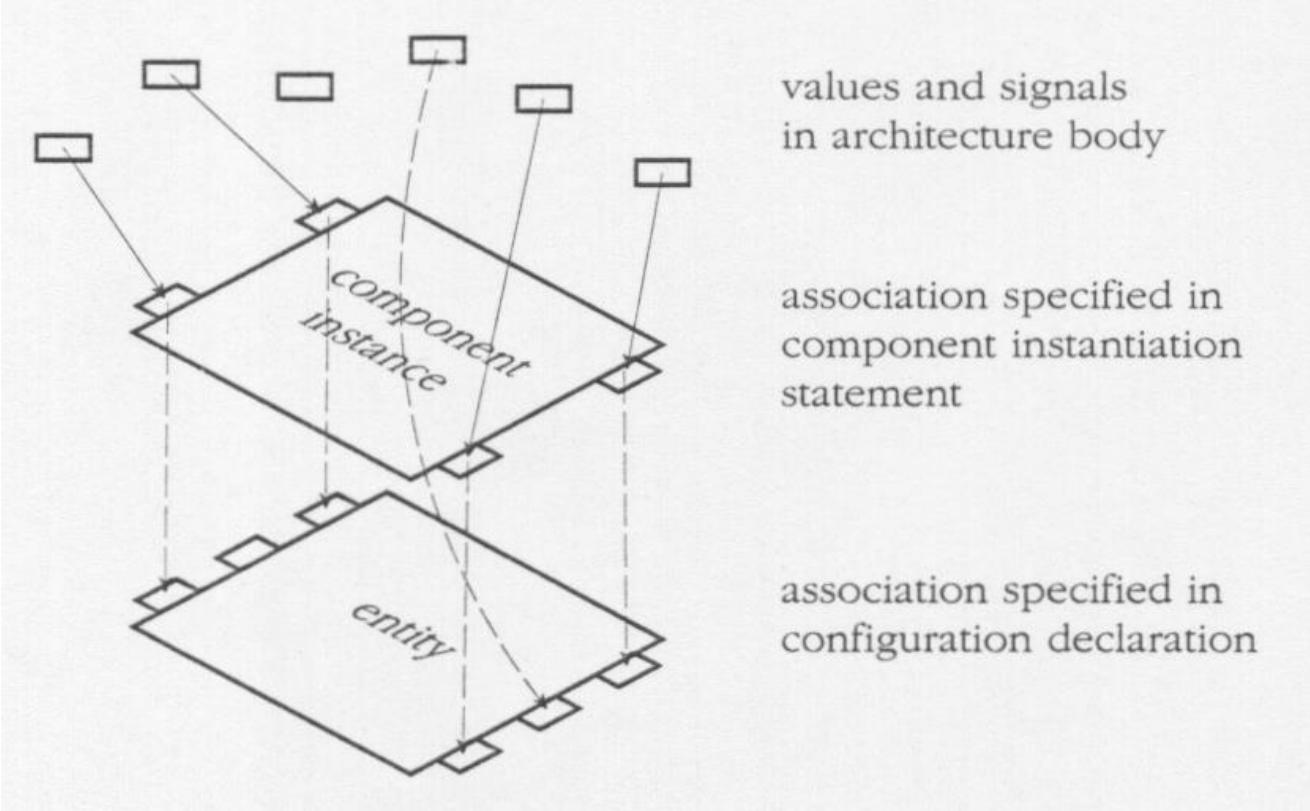
Example of different port maps

```
entity decoder_3_to_8 is
      generic ( Tpd_01, Tpd_10 : delay_length );
      port ( s0, s1, s2 : in bit;
             enable : in bit;
             y0, y1, y2, y3, y4, y5, y6, y7 : out bit );
end entity decoder_3_to_8;
```

Example of different port maps

```
configuration computer_structure of computer_system is
    for structure
        for interface_decoder : decoder_2_to_4
            use entity work.decoder_3_to_8(basic)
            generic map ( Tpd_01 => prop_delay, Tpd_10 => prop_delay )
            port map ( s0 => in0, s1 => in1, s2 => '0',
                       enable => '1',
                       y0 => out0, y1 => out1, y2 => out2, y3 => out3,
                       y4 => open, y5 => open, y6 => open, y7 => open );
        end for;
        ...
    end for;
end configuration computer_structure;
```

The full picture



Default binding

- We call *local generics or ports* those of the component.
- We call *formal generics or ports* those of the bond entity.
- Each formal generic or port is connected to the local generic or port with the same name.
- Formal generics or ports that do not have a local with the same name are left open.
- If a local generic or port does not have a formal generic or port with the same name, the design is wrong.

Error example

```
component nand3 is
      port ( a, b, c : in bit := '1';  y : out bit );
end component nand3;
```

```
gate1 : component nand3
     port map ( a => s1, b => s2, c => open, y => s3 );
```

```
entity nand2 is
      port ( a, b : in bit := '1';  y : out bit );
end entity nand2;
```

```
for gate1 : nand3
      use entity work.nand2(basic);
end for;
```

ERROR !!!

Deferred component binding

binding_indication ⇐ **use open**

- We could also simulate the design:
  - The inputs of the unconnected modules are not used.
  - The outputs of the unconnected modules are not driven.

Example of unbound

```
architecture structural of single_board_computer is
    ...      -- type and signal declarations
    component processor is
        port ( clk : in bit;  a_d : inout word; ... );
    end component processor;
    component memory is
        port ( addr : in bit_vector(25 downto 0); ... );
    end component memory;
    component serial_interface is
        port ( clk : in bit;  address : in bit_vector(3 downto 0); ... );
    end component serial_interface;
begin
    cpu : component processor
        port map ( clk => sys_clk, a_d => cpu_a_d, ... );
    main_memory : component memory
        port map ( addr => latched_addr(25 downto 0), ... );
    serial_interface_a : component serial_interface
        port map ( clk => sys_clk, address => latched_addr(3 downto 0), ... );
    ...
end architecture structural;
```

Example of unbound

```
library chips;
configuration intermediate of single_board_computer is
    for structural
        for cpu : processor
            use entity chips.XYZ3000_cpu(full_function)
            port map ( clock => clk, addr_data => a_d, ... );
        end for;
        for main_memory : memory
            use entity work.memory_array(behavioral);
        end for;
        for all : serial_interface
            use open;
        end for;
        ...
    end for;
end configuration intermediate;
```

See:

- Peter Ashenden, «The designers' guide to VHDL» Morgan Kaufmann,
  - Chapter 13