Module 5 - Structural Programming

1. Structural Programming in VHDL

1.1 Structural Style

Structural Style Programming is an approach in VHDL that allows designers to create digital circuits by using basic components connected to each other to form a more complex system. In this approach, a circuit is represented as a collection of entities linked in a specific way to achieve the desired function.

1.2 Port Mapping

Port mapping is the process of associating (mapping) the ports of a component (entity) in VHDL with the signals present in the architecture. This allows us to connect a defined entity to the actual circuit in the design.

Some important points:

- **Entity Definition**: An entity must be defined before port mapping.
- **Port-Map List**: A list that maps ports to signals.
- **Port Mapping Order**: The order must match the entity's port definition.
- **Signal Declaration**: Signals must be declared beforehand.

Port Mapping Example

```
-- Entity definition
entity AND2 is
  port (
    A, B: in std_logic;
    Y: out std_logic
  );
end entity;

-- Port mapping in architecture (this is actually the entity's behavior)
architecture RTL of AND2 is
```

```
begin
  Y <= A and B;
end architecture;

-- Using the entity with port mapping in a higher-level design
D1: AND2 port map (
  A => input_signal_A,
  B => input_signal_B,
  Y => output_signal
);
```

2. Generic Map

2.1 Generic Map Explanation

A generic map is the process of associating a generic value in an entity with a value in the architecture. Generics are parameters used to configure a component.

Some important points:

- **Generic**: A parameter to change the characteristics of an entity.
- **Generic Map**: Sets the value of the generic during instantiation.
- **Default Value**: Used if a value is not explicitly set.

2.2 Generic Map Example

```
entity Counter is
  generic (
    WIDTH: positive := 8;
    ENABLED: boolean := true
  );
  port (
    clk: in std_logic;
    reset: in std_logic;
    count: out std_logic_vector(WIDTH-1 downto 0)
  );
end entity;

architecture RTL of MyDesign is
  signal my_counter_output: std_logic_vector(7 downto 0);
begin
  my_counter_inst: Counter
    generic map (
      WIDTH => 8,
      ENABLED => true
    )
    port map (
      clk => system_clock,
```

```
      reset => reset_signal,
      count => my_counter_output
    );
end architecture;
```

3. VHDL Modularity

3.1 VHDL Modularity Explanation

Example: **4-bit Ripple Carry Adder** using 4 Full Adders. A Ripple Carry Adder adds binary numbers with a chained carry.

3.1.1 Stage 1: Full Adder

```
entity Full_Adder is
  port (
    A, B, Cin: in std_logic;
    Sum, Cout: out std_logic
  );
end entity Full_Adder;

architecture RTL of Full_Adder is
begin
  Sum <= (A xor B) xor Cin;
  Cout <= (A and B) or ((A xor B) and Cin);
end architecture;
```

3.1.2 Stage 2: 4-bit Ripple Carry Adder

```
entity Four_Bit_RCA is
  port (
    A, B: in std_logic_vector(3 downto 0);
    Sum: out std_logic_vector(3 downto 0);
    Cout: out std_logic
  );
end entity Four_Bit_RCA;

architecture RTL of Four_Bit_RCA is
  signal Carry: std_logic_vector(3 downto 0);
begin
```

```
  FA0: Full_Adder port map (A(0), B(0), '0', Sum(0), Carry(0));
  FA1: Full_Adder port map (A(1), B(1), Carry(0), Sum(1), Carry(1));
  FA2: Full_Adder port map (A(2), B(2), Carry(1), Sum(2), Carry(2));
  FA3: Full_Adder port map (A(3), B(3), Carry(2), Sum(3), Cout);
end architecture;
```

3.1.3 Stage 3: Testbench

```
entity RCA_tb is
end entity RCA_tb;

architecture RTL of RCA_tb is
  signal A, B, Sum: std_logic_vector(3 downto 0);
  signal Cout: std_logic;
  signal Clock: std_logic := '0';
  constant Clock_Period: time := 10 ns;

  component Four_Bit_RCA
    port (
      A, B: in std_logic_vector(3 downto 0);
      Sum: out std_logic_vector(3 downto 0);
      Cout: out std_logic
    );
  end component;
begin
  Clock_Process: process
  begin
    while now < 1000 ns loop
      Clock <= not Clock;
      wait for Clock_Period / 2;
    end loop;
    wait;
  end process;

  A <= "1101";
  B <= "0011";

  RCA: Four_Bit_RCA port map (A, B, Sum, Cout);
```

```
  process
  begin
    wait until rising_edge(Clock);
    if Cout = '1' then
      report "The addition result is overflowing";
    end if;
    report "Addition result: " & to_string(Sum);
    wait;
  end process;
end architecture RTL; -- Corrected from Main_Architecture to RTL
```

4. Array and Type

4.1 Array and Type in VHDL

4.1.1 Array

An array is a collection of elements of the same data type. It can be one-dimensional or multi-dimensional.

4.1.2 Type

A type is a new data definition. It can be a built-in type (`std_logic`, `integer`) or a user-derived type.

4.2 Type and Array Example

```
type RegisterArray is array (0 to 7) of std_logic_vector(7 downto 0);
signal registers: RegisterArray := (others => (others => '0'));
```