VHDL

5051158-xxxx

Lecture 4

VHDL Basics – part 3 (arrays, generics)

Test benches

V 1.0, 17.9.2020

Jarno Tuominen

TURKU AMK

TURKU UNIVERSITY OF APPLIED SCIENCES

Contents of this lecture

- VHDL Basics – part 3
  - Arrays and indexing, user types
  - Generics
  - Using generics in hierarchical design
- Creating test benches for simulation

Arrays in VHDL

- 2 ways to combine "bits" to "arrays"
  - Concatenation
  - Aggregate
    - Named association
    - Positional association
- Indexing
  - Accessing a "slice" of an array
  - The direction of the slice (i.e. **to** or **downto**) must match the direction in which the array is declared

```
signal b_bus: std_logic_vector (3 downto 0);
signal x,y,z,w: std_logic;

b_bus <= x & "00" & w;

b_bus <= (2=>x,1=>y,3=>z,0=>d);

b_bus <= (x,y,z,d);
```

```
signal z_bus: std_logic_vector(3 downto 0);
signal a_bus: std_logic_vector (1 to 4);

a_bus <= z_bus; -- ok
z_bus(3 downto 2) <= "00"; --ok
a_bus(2 to 4) <= z_bus(3 downto 1); --ok
z_bus(2 to 3) <= "00"; -- NOT OK
```

Multi-dimensional Arrays and User Types

- An **array** contains multiple elements of same type
  - Note: std_locic_vector is also an array, composed of several std_logic types – defined in std_logic_1164-package
  - When any array object is declared, an existing array type must be used
- Multi-dimensional arrays are especially useful, when using generate loops (we'll see that later)

```
type MY_BUS is array (3 downto 0) of std_logic_vector(15 downto 0);
type RAM is array (0 to 31) of integer range 0 to 255;
signal A_BUS : MY_BUS;
signal RAM_0 : RAM;
```

Entity - Generics

- To improve the reusability and flexibility of your code (especially true for components), you can use **generics**

- **generic** is a parameter- or a "specification", which value is evaluated during compilation/synthesis
  - NOTE: it does not "change" your block run-time
- Generics are listed in entity, before port list. Similarly, component needs to have a generic list as well, as it represents the "interface" of an entity
- On instantiation, values are associated to generics – the default value of generic (in a component) is overridden

```
entity entity_name is
        generic (generic list);
        port    (port list);
end entity_name;

component component_name
        generic (generic_list);
        port (port_list);
end component;

instance_label: component_name
        generic map (generic_association_list)
        port map    (port_association_list);
```

Generics – an example

- Consider a clocked 4:1 bus multiplexer
- What if the width of the data bus changes?
  - In this case the architecture does not require any change (note how we reset the output), but entity need to be changed

```
entity busmux4to1 is
   port (
      clk, n_Reset: in std_logic;
      A,B,C,D : in std_logic_vector(7 downto 0);
      S : in std_logic_vector(1 downto 0);
      Y : out std_logic_vector(7 downto 0)
   );
end busmux4to1;
```

```
architecture rtl of busmux4to1 is
begin
   sync_mux_p: process(clk, n_Reset)
   begin
      if n_Reset = '0' then
         Y <= (others => '0');
      elsif rising_edge(clk) then
         case S is
            when "00" => Y <= A;
            when "01" => Y <= B;
            when "10" => Y <= C;
            when others => Y <= D;
         end case;
      end if; --clk/rst
   end process sync_mux_p;
end architecture rtl;
```

Generics – an example (cont.)

- Added a generic
  - Default value: 8
- Note: No change in architecture (in this case)

```
entity gen_busmux4to1 is
   generic (
      DATA_WIDTH : integer := 8
   );
   port (
      clk, n_Reset: in std_logic;
      A,B,C,D : in std_logic_vector(DATA_WIDTH-1 downto 0);
      S : in std_logic_vector(1 downto 0);
      Y : out std_logic_vector(DATA_WIDTH-1 downto 0)
    );
end gen_busmux4to1;
```

```
architecture rtl of gen_busmux4to1 is
begin
   sync_mux_p: process(clk, n_Reset)
   begin
      if n_Reset = '0' then
         Y <= (others => '0');
      elsif rising_edge(clk) then
         case S is
            when "00" => Y <= A;
            when "01" => Y <= B;
            when "10" => Y <= C;
            when others => Y <= D;
         end case;
      end if; --clk/rst
   end process sync_mux_p;
end architecture rtl;
```

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Generics – an example (cont.)

- How about making it 16-bits wide? Or 32-bits?
- Change the value of generic
  - But, we really don't want to change it every time – better to use it as a component and override the default value when instantiating the component

```
entity gen_busmux4to1 is
   generic (
      DATA_WIDTH : integer := 16
   );
   port (
      clk, n_Reset: in std_logic;
      A,B,C,D : in std_logic_vector(DATA_WIDTH-1 downto 0);
      S : in std_logic_vector(1 downto 0);
      Y : out std_logic_vector(DATA_WIDTH-1 downto 0)
    );
end gen_busmux4to1;
```

```
entity gen_busmux4to1 is
   generic (
      DATA_WIDTH : integer := 32
   );
   port (
      clk, n_Reset: in std_logic;
      A,B,C,D : in std_logic_vector(DATA_WIDTH-1 downto 0);
      S : in std_logic_vector(1 downto 0);
      Y : out std_logic_vector(DATA_WIDTH-1 downto 0)
    );
end gen_busmux4to1;
```

RECAP: VHDL Components and hierarchical design

- A component declaration declares a virtual design entity interface that may be used in component **instantiation** statement
- A component represents an entity/architecture pair. It specifies a subsystem, which can be ***instantiated* in another architecture** leading to a hierarchical specification.
  - Component instantiation is like plugging a hardware component into a socket in a board
- A component must be **declared** before it is **instantiated**
  - The component declaration defines the virtual interface of the instantiated design entity ("the socket")
- Most often, the declaration takes in the main code (in the architecture, before "begin") or in separate packages (more on this later)
- Generics and ports of a component are **copies** of generics and ports of the entity the component represents.

```
component component_name is
    generic (generic_list);
    port (port_list);
end
component component_name;
```

```
architecture rtl of xxx is
 component XOR_4 is
   port(A,B: in std_logic_vector(0 to 3);
              C: out std_logic_vector(0 to 3));
 end component XOR_4;

 signal S1,S2 : std_logic_vector(0 to 3);
 signal S3 : std_logic_vector(0 to 3);

begin
 X1 : XOR_4
 port map(A => S1,B => S2,C => S3);

end architecture rtl;
```

Generics and hierarchy

```
entity mux_top is
    port (
        sysclk: in std_logic;
        S: in std_logic_vector(1 downto 0);
        Q : out std_logic_vector(15 downto 0);
        Q_wide : out std_logic_vector(31 downto 0));
end mux_top;


architecture rtl of mux_top is
    component gen_busmux4to1 is
    generic ( DATA_WIDTH : integer := 8 );
    port (
        clk, n_Reset: in std_logic;
        A,B,C,D : in std_logic_vector(DATA_WIDTH-1 downto 0);
        S : in std_logic_vector(1 downto 0);
        Y : out std_logic_vector(DATA_WIDTH-1 downto 0)
     );
    end component gen_busmux4to1;

    signal E,F,G,H: std_logic_vector(15 downto 0);
    signal X,Y,Z,W: std_logic_vector(31 downto 0);
    signal n_Reset: std_logic;
```

```
begin
  n_Reset <= '1';

  i_busmux16: gen_busmux4to1
    generic map (
        DATA_WIDTH  => 16
    )
    port map (n_Reset => n_Reset,
         clk => sysclk,
         A => E, B => F, C => G, D => H,
         S => S,
         Y => Q);

  i_busmux32: gen_busmux4to1
    generic map (
        DATA_WIDTH  => 32
    )
    port map (n_Reset => n_Reset,
         clk        => sysclk,
         A  => X, B => Y, C  => Z, D => W,
         S => S,
         Y => Q_wide);

end rtl;
```

VHDL test benches

- Test benches are top-level design units **for simulation purposes only**
- Why test bench?
  - Creating a stimulus manually is tedious/difficult
  - Large/complex designs often require automatic testing of outputs (vs inputs).
    - In hierarchical designs, it is useful and easier to test each block separately - comparable to unit testing in SW world
    - Isolation of DUTs makes things easier!
- Test bench contains:
  - Instantiation of Device Under Test (DUT)
  - Stimulus signals for DUT (generators for input waveforms, clock driver, reset etc)
  - (Optionally) generation of reference outputs and comparison to DUT outputs
    - Can provide automatically a pass or fail indication

VHDL functions/commands for TB’s

- Not all VHDL is synthesizable - some commands/functions are intended for simulation only, to be used in test benches

- For example:
  - Delay statements

```
-- Non-Synthesizable Delay Statement:
r_Enable <= '0';
wait for 100 ns;
r_Enable <= '1';
```

  - Assertions

```
assert (A and B = 0) report ”A and B
simultaneously zero” severity warning;
```

VHDL functions/commands for TB’s (2)

- File/Text I/O
  - enables reading of stimulus and writing of results to a file
- In this example stimulus file (ASCII) is created with MATLAB, test bench reads it row by row, on rising edge of a sampling clock and places the data to stimulus vectors (matrix)

```vhdl
library STD;
use STD.textio.all;
...
architecture behavioral of Pate_Top_TB is
  -- file handlers & file op related stuff
  file S_Stimulus: text open read_mode is "S_stimulus.prn";
...
-- read analoque stimulus from files
analoque_stimulus: process
  variable S_row: line;
  variable v_data_read : integer := 0;
begin
  wait until S_SamClk_P'event and S_SamClk_P='1';
  if(n_Reset='0') then
    v_data_read := 0;
  else
  -- read from input file in "row" variable
    if(not endfile(S_Stimulus)) then
      readline(S_Stimulus,S_row);
      for i in 0 to 7 loop
          read(S_row,v_data_read); -- read the integer value
          S_Anal_Data(i) <= std_logic_vector(to_unsigned(v_data_read,14));
      end loop;
    end if;
  end if;
end process;
```

An example test bench

- Get it from Teams and try it out
  - mux_tb_sources.zip
- Notes:
  - The test bench -file (mux_tb.vhd) needs to be added as “simulation source” in Vivado
  - Make sure that mux_tb.vhd is set as top for simulation