CPE 462
VHDL: Simulation and Synthesis

Topic #04 - d) Generic

Summary of operators

Operators.

Operator type	Operators	Data types
Assignment	<=, :=, =>	Any
Logical	NOT, AND, NAND, OR, NOR, XOR, XNOR	BIT, BIT_VECTOR, STD_LOGIC, STD_LOGIC_VECTOR, STD_ULOGIC, STD_ULOGIC_VECTOR
Arithmetic	+, −, *, /, ** (mod, rem, abs)◆	INTEGER, SIGNED, UNSIGNED
Comparison	=, /=, <, >, <=, >=	All above
Shift	sll, srl, sla, sra, rol, ror	BIT_VECTOR
Concatenation	&, (, , ,)	Same as for logical operators, plus SIGNED and UNSIGNED

◆ = un-synthesizable

Summary of attributes

Attributes.

Application	Attributes	Return value
For regular DATA	d'LOW	Lower array index
	d'HIGH	Upper array index
	d'LEFT	Leftmost array index
	d'RIGHT	Rightmost array index
	d'LENGTH	Vector size
	d'RANGE	Vector range
	d'REVERSE_RANGE	Reverse vector range
For enumerated DATA	d'VAL(pos)◆	Value in the position specified
	d'POS(value)◆	Position of the value specified
	d'LEFTOF(value)◆	Value in the position to the left of the value specified
	d'VAL(row, column)◆	Value in the position specified
For a SIGNAL	s'EVENT	True when an event occurs on s
	s'STABLE	True if no event has occurred on s
	s'ACTIVE◆	True if s is high

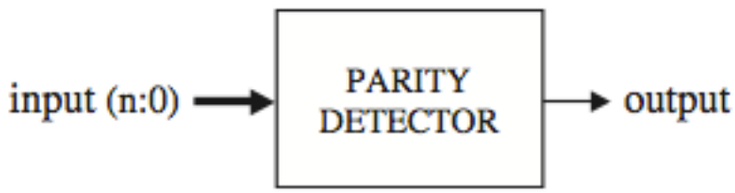
◆ = un-synthesizable

Generic

GENERIC... What is it?

- GENERIC is a way of specifying a generic parameter
- This is, a static parameter that can be easily modified and adapted to different applications

- The purpose is to confer the code more flexibility and reusability
- For example, you create an adder circuit that can work with either 2 inputs of 4 bits or 2 inputs of 1million bits
- Other example... a parity detection circuit that works with an n-input data-stream.



Syntax

- A GENERIC statement, when employed, must be declared in the ENTITY.

- The specified parameter will then be truly global (that is, visible to the whole design, including the ENTITY itself).

- Its syntax is shown below.

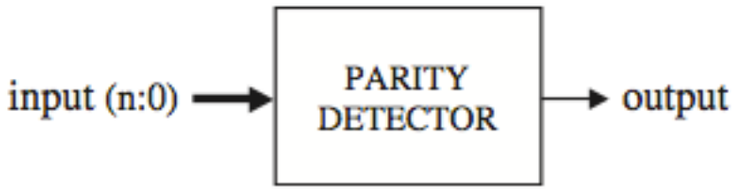
```
GENERIC (parameter_name : parameter_type := parameter_value);
```

Examples

- Next we will discuss several complete design examples, with the purpose of further illustrating the use of operators, attributes and GENERIC.

- Keep in mind, that up to now we have just discussed the basic foundations, so a lot of the coding techniques will still be unfamiliar.

Generic Parity Detector

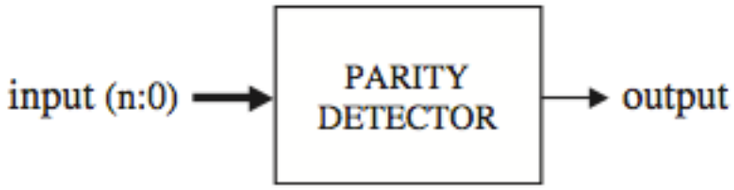


- The circuit must provide output = '0' when the number of '1's in the input vector is even, or output = '1' otherwise.

- Notice in the VHDL code that the ENTITY contains a GENERIC statement (line 3), which defines **n** as 7. This code would work for any other vector size being only necessary to change the value of **n** in that line.

```
entity parity_det is
   generic (n: integer := 7);
   port (input : in bit_vector (n downto 0);
   output : out bit);
end entity;

architecture myarch of parity_det is
begin
   process(input)
   variable temp : bit;
   begin
        temp:='0';
        for i in input'range loop
            temp := temp XOR input(i);
        end loop;
        output<=temp;
   end process;
end architecture;
```

Generic Parity Detector



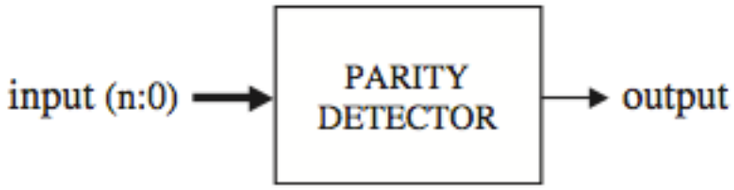
n is the generic parameter

Why generic? We want to modify our circuit so that it is modular enough to work with other input sizes.

```
entity parity_det is
   generic (n: integer := 7);
   port (input : in bit_vector (n downto 0);
   output : out bit);
end entity;
```

```
architecture myarch of parity_det is
begin
   process(input)
   variable temp : bit;
   begin
        temp:='0';
        for i in input'range loop
             temp := temp XOR input(i);
        end loop;
        output<=temp;
   end process;
end architecture;
```

Generic Parity Detector



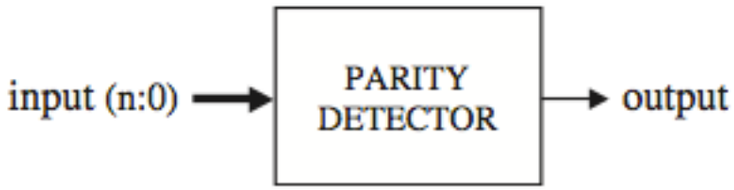
The port will be a bit_vector but with n inputs, where n was specified on the line:

```
generic (n: integer := 7);
```

```
entity parity_det is
   generic (n: integer := 7);
   port (input : in bit_vector (n downto 0);
   output : out bit);
end entity;
```

```
architecture myarch of parity_det is
begin
   process(input)
   variable temp : bit;
   begin
        temp:='0';
        for i in input'range loop
             temp := temp XOR input(i);
        end loop;
        output<=temp;
   end process;
end architecture;
```

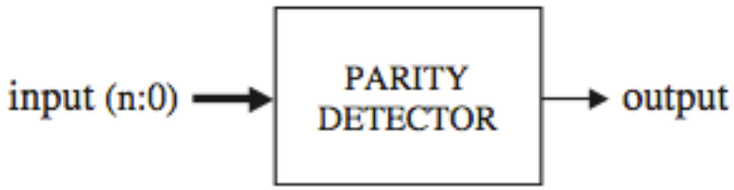
Generic Parity Detector




```
entity parity_det is
   generic (n: integer := 7);
   port (input : in bit_vector (n downto 0);
   output : out bit);
end entity;

architecture myarch of parity_det is
begin
   process(input)
   variable temp : bit;
   begin
        temp:='0';
        for i in input'range loop
            temp := temp XOR input(i);
        end loop;
        output<=temp;
   end process;
end architecture;
```

We haven't learned all the bolded instructions, but you can figure out what they do.

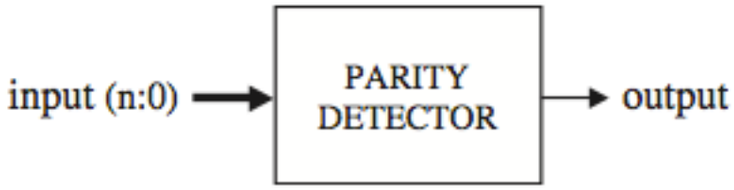
Generic Parity Detector



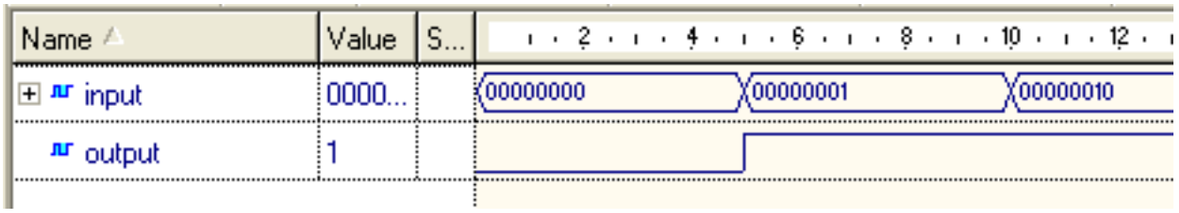

```
entity parity_det is
   generic (n: integer := 7);
   port (input : in bit_vector (n downto 0);
   output : out bit);
end entity;

architecture myarch of parity_det is
begin
   process(input)
   variable temp : bit;
   begin
        temp:='0';
        for i in input'range loop
             temp := temp XOR input(i);
        end loop;
        output<=temp;
   end process;
end architecture;
```

This is the data attribute. It will return the range of data on the input signal.

Generic Parity Detector




```
entity parity_det is
   generic (n: integer := 7);
   port (input : in bit_vector (n downto 0);
   output : out bit);
end entity;
```




```
architecture myarch of parity_det is
begin
   process(input)
   variable temp : bit;
   begin
        temp:='0';
        for i in input'range loop
           temp := temp XOR input(i);
        end loop;
        output<=temp;
   end process;
end architecture;
```

It will keep looping until all input(i) bits are XORed... That is how we compute parity.

Test-benches with Generic

You need to perform some minor changes to the test-bench template so you can include generic modules.

```
entity testparity is
end;

architecture bench of testparity is
 constant n : positive := 7;
  component parity_det
   generic (n: integer := n);
   port (input : in bit_vector (n downto 0);
   output : out bit);
  end component;

  signal input : bit_vector (n downto 0);
  signal output : bit;

begin
 input   <= "00000000" , "00000001" after 5 ns
 , "00000010" after 10 ns , "00000011" after 15 ns;

 m: parity_det generic map(n) port map (input,output);

end bench;
```

Test-benches with Generic

You must add a constant with the generic name and value.

```
entity testparity is
end;

architecture bench of testparity is
  constant n : positive := 7;
  component parity_det
   generic (n: integer := n);
   port (input : in bit_vector (n downto 0);
   output : out bit);
  end component;

  signal input : bit_vector (n downto 0);
  signal output : bit;

begin
 input   <= "00000000" , "00000001" after 5 ns
 , "00000010" after 10 ns , "00000011" after 15 ns;

 m: parity_det  generic map(n) port map (input,output);

end bench;
```

Test-benches with Generic

Inside the component, make sure you include the generic line.

```
entity testparity is
end;

architecture bench of testparity is
  constant n : positive := 7;
  component parity_det
  generic (n: integer := n);
  port (input : in bit_vector (n downto 0);
  output : out bit);
  end component;

  signal input : bit_vector (n downto 0);
  signal output : bit;

begin
 input   <= "00000000" , "00000001" after 5 ns
 , "00000010" after 10 ns , "00000011" after 15 ns;

 m: parity_det  generic map(n) port map (input,output);

end bench;
```

Test-benches with Generic

Add the generic map into the last line of the test-bench

```
entity testparity is
end;

architecture bench of testparity is
  constant n : positive := 7;
  component parity_det
   generic (n: integer := n);
   port (input : in bit_vector (n downto 0);
   output : out bit);
  end component;

  signal input : bit_vector (n downto 0);
  signal output : bit;

begin
 input   <= "00000000" , "00000001" after 5 ns
 , "00000010" after 10 ns , "00000011" after 15 ns;

 m: parity_det  generic map(n) port map (input,output);

end bench;
```

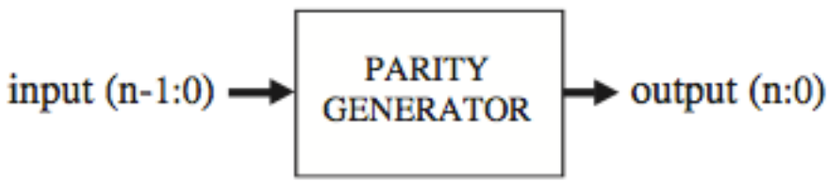
Main program

```
entity parity_det is
   generic (n: integer := 7);
   port (input : in bit_vector (n downto 0);
   output : out bit);
end entity;

architecture myarch of parity_det is
begin
   process(input)
   variable temp : bit;
   begin
        temp:='0';
        for i in input'range loop
            temp := temp XOR input(i);
        end loop;
        output<=temp;
   end process;
end architecture;
```

Test-bench

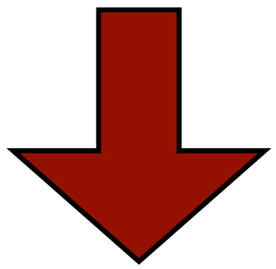
```
entity testparity is
end;

architecture bench of testparity is
  constant n : positive := 7;
  component parity_det
   generic (n: integer := n);
   port (input : in bit_vector (n downto 0);
   output : out bit);
  end component;

   signal input : bit_vector (n downto 0);
   signal output : bit;

begin
 input   <= "00000000"
 , "00000001" after 5 ns
 , "00000010" after 10 ns
 , "00000011" after 15 ns;

 m: parity_det  generic map(n)
    port map (input,output);

end bench;
```

Another Example: Generic Parity Generator



- The circuit must add one bit to the input vector (on its left)
- Such bit must be a '0' if the number of '1's in the input vector is even, or a '1' if it is odd, such that the resulting vector will always contain an even number of '1's (even parity)

Main program



```
entity parity_gen is
   generic (n: integer := 7);
   port (input : in bit_vector (n-1 downto 0);
   output : out bit_vector(n downto 0));
end    entity;

architecture myarch of parity_gen is
begin
   process(input)
   variable temp1 : bit;
   variable temp2 : bit_vector (output'range);
   begin
        temp:='0';
        for i in input'range loop
            temp1 := temp XOR input(i);
            temp2(i) := input(i);
        end loop;
        temp2(output'high):=temp1;
        output<=temp;
   end process;
end architecture;
```

- In HW#7 you will write the test-bench that will test this circuit.
- You will also have to describe out what each line is doing.