Lección 3.

Ejemplos de descripciones por comportamiento de dispositivos de propósitos generales genéricos

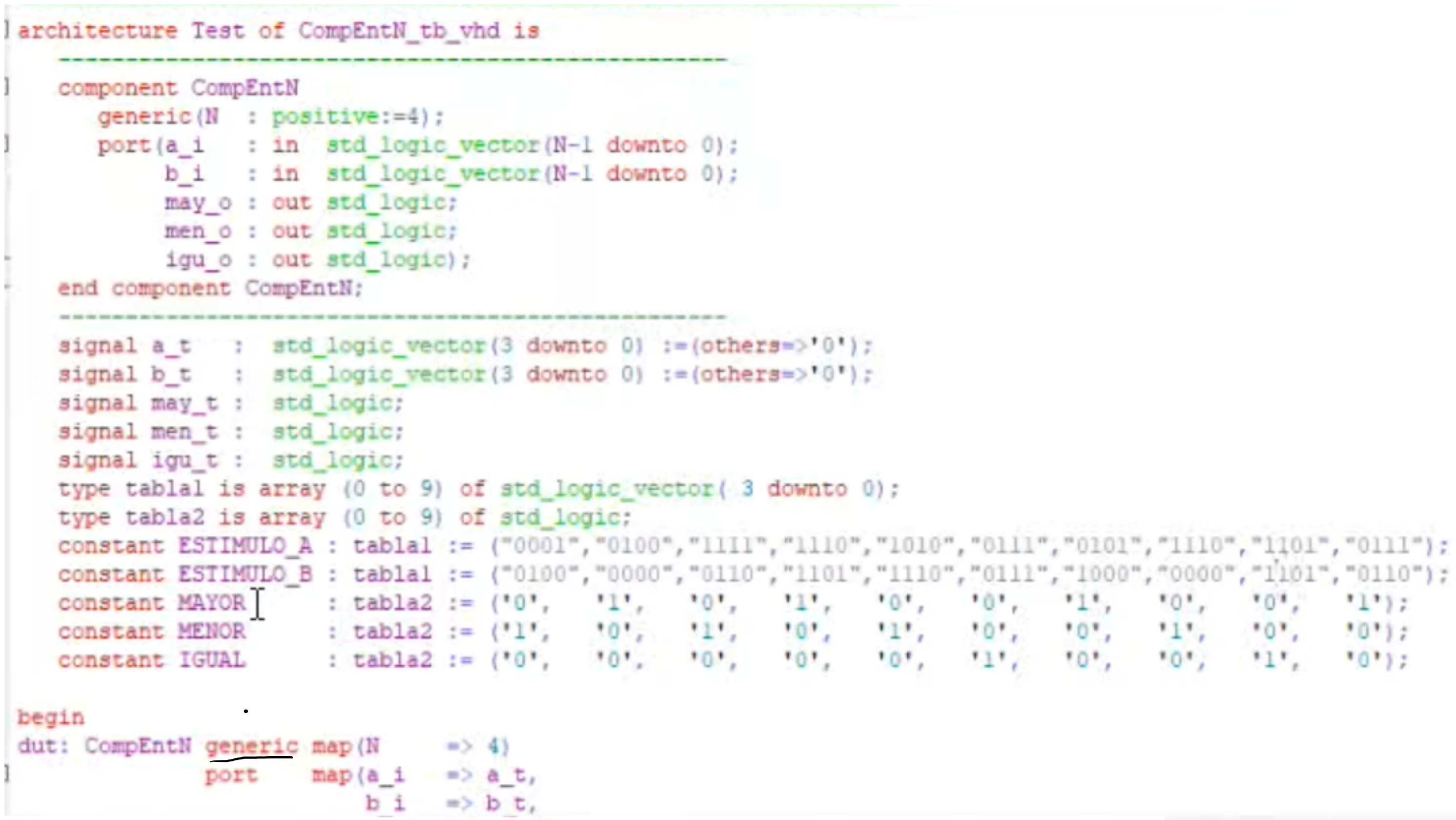
Uso de "generic"

Para un numero N de entradas

Comparador genérico $\boxed{V13}$

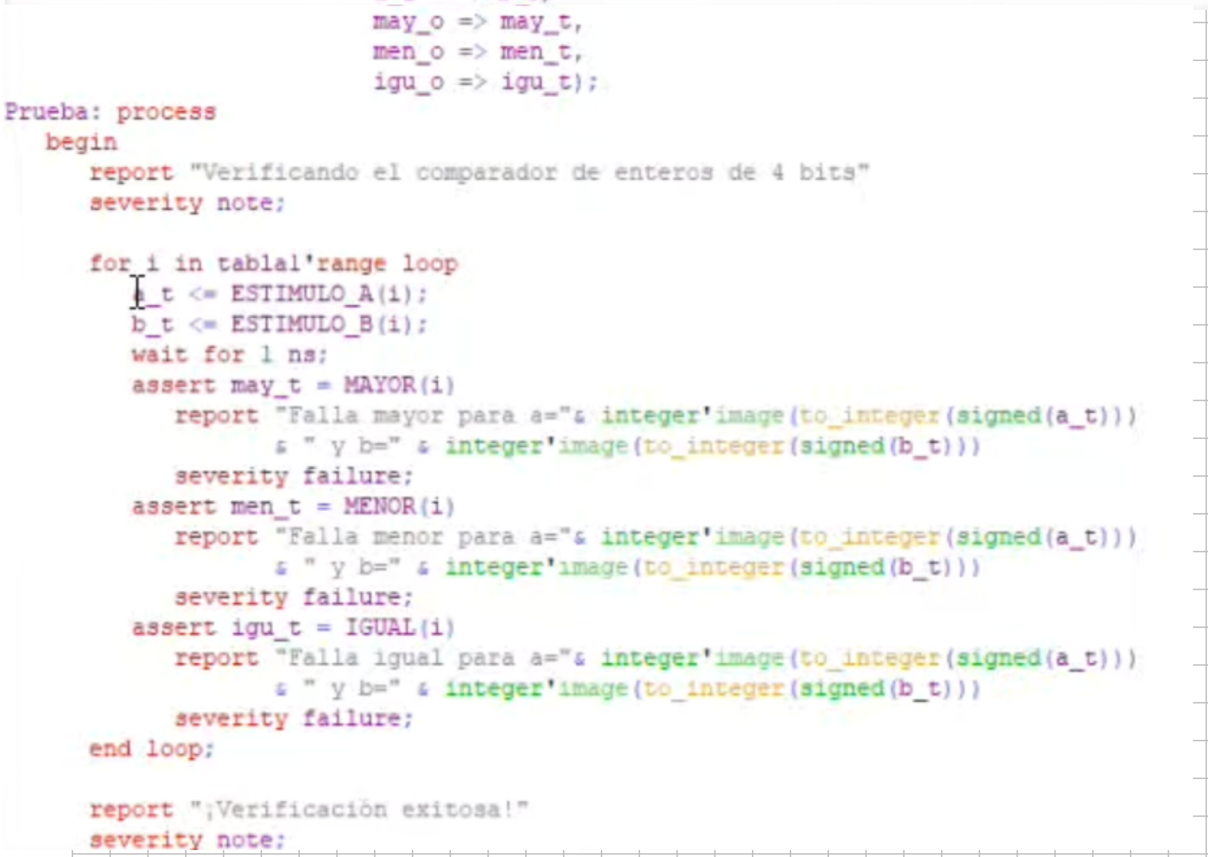
Vamos a necesitar el paquete ieee.numeric_std

```
entity CompEntN is
   generic (N : positive := 4)
   port( a_i  : in  std_logic_vector (N-1 downto 0);
         b_i  :             ''                     ;
         may_o : out std_logic;
         men_o : out std_logic;
         igu_o : out std_logic
       );
end entity CompEntN;
architecture Arq of CompEntN is
 begin
    may_o <= '1' when signed (a_i) > signed (b_i) else
              '0' ;
         :
end architecture Arq;
```



```vhdl
architecture Test of CompEntN_tb_vhd is
   --------------------------------------------------
   component CompEntN
      generic(N   : positive:=4);
      port(a_i   : in  std_logic_vector(N-1 downto 0);
           b_i   : in  std_logic_vector(N-1 downto 0);
           may_o : out std_logic;
           men_o : out std_logic;
           igu_o : out std_logic);
   end component CompEntN;
   --------------------------------------------------
   signal a_t   :  std_logic_vector(3 downto 0) :=(others=>'0');
   signal b_t   :  std_logic_vector(3 downto 0) :=(others=>'0');
   signal may_t :  std_logic;
   signal men_t :  std_logic;
   signal igu_t :  std_logic;
   type tabla1 is array (0 to 9) of std_logic_vector( 3 downto 0);
   type tabla2 is array (0 to 9) of std_logic;
   constant ESTIMULO_A : tabla1 := ("0001","0100","1111","1110","1010","0111","0101","1110","1101","0111");
   constant ESTIMULO_B : tabla1 := ("0100","0000","0110","1101","1110","0111","1000","0000","1101","0110");
   constant MAYOR      : tabla2 := ('0',   '1',   '0',   '1',   '0',   '0',   '1',   '0',   '0',   '1');
   constant MENOR      : tabla2 := ('1',   '0',   '1',   '0',   '1',   '0',   '0',   '1',   '0',   '0');
   constant IGUAL      : tabla2 := ('0',   '0',   '0',   '0',   '0',   '1',   '0',   '0',   '1',   '0');

begin
dut: CompEntN generic map(N     => 4)
              port    map(a_i   => a_t,
                          b_i   => b_t,
```

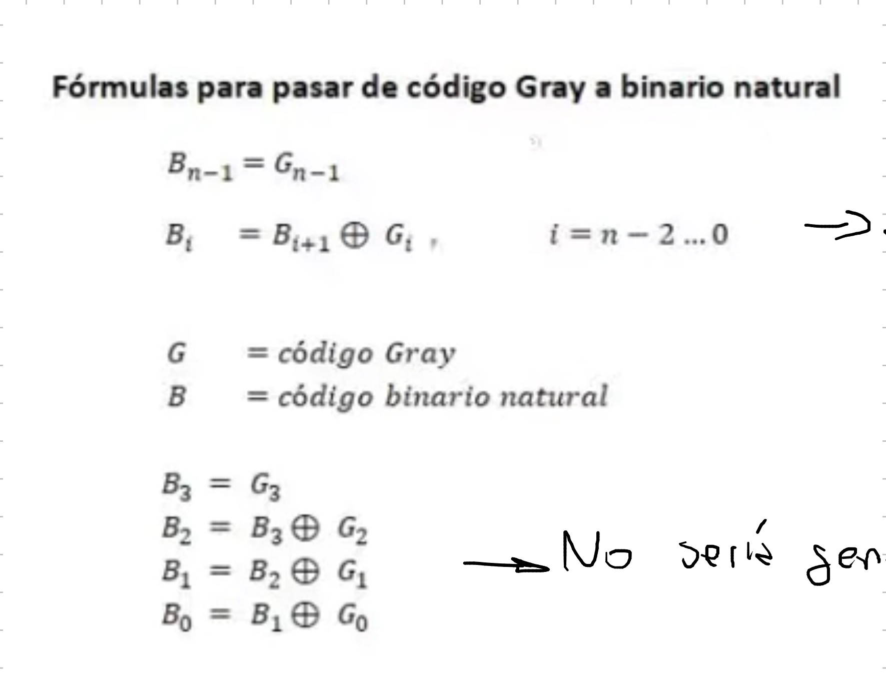


```vhdl
                          may_o => may_t,
                          men_o => men_t,
                          igu_o => igu_t);
Prueba: process
   begin
      report "Verificando el comparador de enteros de 4 bits"
      severity note;

      for i in tabla1'range loop
         a_t <= ESTIMULO_A(i);
         b_t <= ESTIMULO_B(i);
         wait for 1 ns;
         assert may_t = MAYOR(i)
            report "Falla mayor para a="& integer'image(to_integer(signed(a_t)))
                   & " y b=" & integer'image(to_integer(signed(b_t)))
            severity failure;
         assert men_t = MENOR(i)
            report "Falla menor para a="& integer'image(to_integer(signed(a_t)))
                   & " y b=" & integer'image(to_integer(signed(b_t)))
            severity failure;
         assert igu_t = IGUAL(i)
            report "Falla igual para a="& integer'image(to_integer(signed(a_t)))
                   & " y b=" & integer'image(to_integer(signed(b_t)))
            severity failure;
      end loop;

      report "¡Verificación exitosa!"
      severity note;
```

Conversor Código Gray → binario natual $\boxed{V14}$



Fórmulas para pasar de código Gray a binario natural

$B_{n-1} = G_{n-1}$

$B_i \quad = B_{i+1} \oplus G_i \,, \qquad i = n-2 \ldots 0$ → algoritmo

$G \quad = código\ Gray$
$B \quad = código\ binario\ natural$

$B_3 = G_3$
$B_2 = B_3 \oplus G_2$
$B_1 = B_2 \oplus G_1$ → No sería genérico
$B_0 = B_1 \oplus G_0$

```
entity Gray2BinN is:
   generic(N : integer :=4); -- puede ser 'positive'
   port(g_i  : in  std_logic_vector(N-1 downto 0);
        b_o : out std_logic_vector(N-1 downto 0).
       );
end entity Gray2BinN
```

Esto es lo que queremos hacer.

```
architecture Comportamiento of Gray2BinN is:
 begin
       b_o (N-1) <= g_i(N-1);
       b_o (N-2 downto 0) <= b_o(N-1 downto 1) xor
                              g_i(N-2 downto 0)
 end architecture Comportamiento;
```

la operación xor se hace elemento a elemento

No se puede realizar porque una salida no se puede leer!!!

Necesitamos una señal auxiliar

```
architecture Comportamiento of Gray2BinN is
 Signal baux : std_logic_vector (N-1 downto 0);
begin
      baux (N-1) <= g_i (N-1);
      baux (N-2 downto 0) <= baux (N-1 downto 1) xor
                              g_i(N-2 downto 0)
      b_o <= baux;

end architecture Comportamiento;
```

Test bench

```
      port   (g_i  : in  std_logic_vector(N-1 downto 0);
              b_o  : out std_logic_vector(N-1 downto 0));
   end component Gray2BinN;
   ------------------------------------------------------

   signal g_t  : std_logic_vector (3 downto 0) := (others => '0');
   signal b_t  : std_logic_vector (3 downto 0);
   type tabla is array(0 to 2**4-1) of std_logic_vector(3 downto 0);
   constant TABLA_GRAY : tabla := ("0000", "0001", "0011", "0010",
                                   "0110", "0111", "0101", "0100",
                                   "1100", "1101", "1111", "1110",
                                   "1010", "1011", "1001", "1000");
begin
dut: Gray2BinN generic map(N   => 4)
               port    map(g_i => g_t,
                           b_o => b_t);
Prueba:
   process begin
      report "Probando el conversor de Gray a binario de 4 bits"
      severity note;

      for i in tabla'range loop
         g_t <= TABLA_GRAY(i);
         wait for 1 ns;
         assert b_t = std_logic_vector(to_unsigned(i, 4))
            report "Falla para "& integer'image(to_integer(unsigned(g_t)))
            severity failure;
      end loop;

      report "!Prueba exitosa!"
      severity note;
      wait;
```

Usa una tabla

casteo