Utilizar el generate en VHDL

Creador: David Rubio G.

Entrada: https://soceame.wordpress.com/2024/11/26/utilizar-el-generate-en-vhdl/

Blog: https://soceame.wordpress.com/

GitHub: https://github.com/DRubioG

Fecha última modificación: 23/02/2025

Uno de los elementos más potentes que hay en VHDL son los *generate*, que básicamente son directivas para el sintetizador para añada unos módulos o señales a la síntesis.

Básicamente VHDL tiene dos tipos de estructuras *generate*, una condicional y una iterativa.

- La **estructura condicional** se basa en la estructura de un if. Para ello se sigue un patrón como el siguiente.

```
nombre_estructura : if <condición> generate
   <constantes a generar>
   <señales a generar>
  [begin]

-- Estructura que se ejecutará por el sintetizador si se cumple la condición.

end generate;
```

Las estructuras que se puede utilizar son señales o la instanciación de un módulo (que tiene que haber sido definido en la arquitectura).

NOTA: es importante tener en cuenta que el *generate* también puede crear constantes/señales condicionales, de tal forma que se pueden utilizar constantes/señales variables dependiendo de la condición.
Aunque es verdad que el sintetizador elimina las señales que se definen pero no se utilizan, pero antes te da un warning.

Para las condiciones se pueden utilizar genéricos de tipo string.

Pongo un ejemplo:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_misc.all;

entity test is
    generic (
        condition : string := "or"
    );
    port(
        a : in std_logic;
        b : out std_logic
    );
end entity;

architecture arch_test of test is

component and_lc is
    port (
        a : in std_logic_vector(1 downto 0);
        y : out std_logic
    );
end component;
signal h : std_logic_vector(2 downto 0);
begin
```

```
gen_OR: if condition = "or" generate

    begin
        h <= a & a & a;
        b <= and_reduce(h);
    end generate;


gen_AND: if condition = "and" generate
        signal h : std_logic_vector(1 downto 0);
    begin
        h <= a & a;
        inst_and_lc : and_lc
            port map (
                a => h,
                y => b
            );
    end generate;

end architecture;
```

En el ejemplo anterior se puede ver perfectamente el funcionamiento con diferentes estructuras.

NOTA: lo he comentado antes, los *component* se tienen que definir en la estructura aunque luego no se vayan a ejecutar.

- La **estructura iterativa** se basa en un *for*. Este *for* tiene la ventaja de que crea tantas estructuras en paralelo como se le defina.

```
nombre_estructura : for <var> in <condición rango> generate
   <constantes a generar>
   <señales a generar>
  [begin]

-- Estructura que se ejecutará por el sintetizador si se cumple la condición.

end generate;
```

La variable <*var*> se puede considerar de tipo *integer.*

Las estructuras que se puede utilizar son señales o la instanciación de un módulo (que tiene que haber sido definido en la arquitectura.

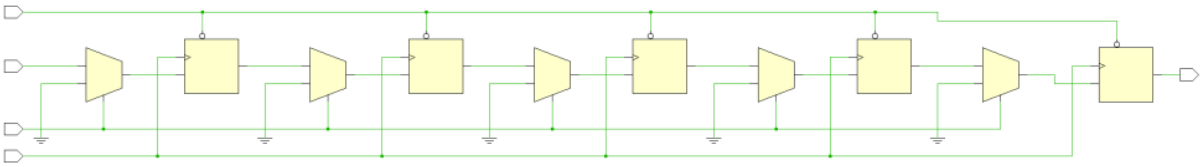
Las estructuras que puede contener son muy parecidas a las que se utilizan en el if *generate*.

El ejemplo que se propone es el de crear un registro de desplazamiento a izquierdas, basado en biestables D, este sistema también se podría utilizar para sincronizar señales entre diferentes dominios de tiempo.

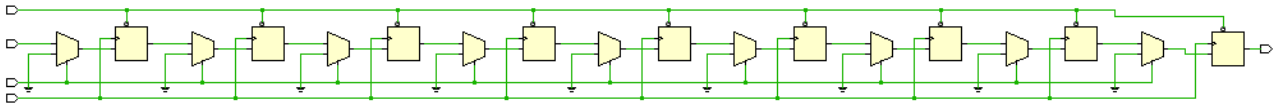
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity D_FF is
    generic(
        N : integer := 4
```

```
    );
    Port (
        clk : in std_logic;
        rst_n : in std_logic;
        en : in std_logic;
        data_in : in std_logic;
        data_out : out std_logic
    );
end D_FF;

architecture Behavioral of D_FF is
signal d_aux : std_logic_vector(N-1 downto 0);
begin

    process(clk, rst_n, en)
    begin
        if rst_n = '0' then
            d_aux(0) <= '0';
        elsif rising_edge(clk) then
            if en = '1' then
                d_aux(0) <= data_in;
            elsif en = '0' then
                d_aux(0) <= '0';
            end if;
        end if;
    end process;


gen_DFF : for i in 0 to N-2 generate
       process(clk, rst_n, en)
    begin
        if rst_n = '0' then
            d_aux(i+1) <= '0';
        elsif rising_edge(clk) then
            if en = '1' then
                d_aux(i+1) <= d_aux(i);
            elsif en = '0' then
                d_aux(i+1) <= '0';
            end if;
        end if;
    end process;

end generate;

    process(clk, rst_n, en)
    begin
        if rst_n = '0' then
            data_out <= '0';
        elsif rising_edge(clk) then
            if en = '1' then
                data_out <= d_aux(N-1);
            elsif en = '0' then
                data_out <= '0';
            end if;
        end if;
    end process;


end Behavioral;
```

La estructura a sintetizar sería la siguiente, donde se ven todas las etapas del registro de desplazamiento.



Si cambiamos el número de etapas a 8, se puede ver como se escala el registro aumentando el número de biestables.



NOTA: con el bucle *for* es muy fácil perderse a la hora de definir estructuras y sus señales, la recomendación es analizar los extremos de la estructura antes de comenzar, porque son los extremos los que tienen contacto con el resto del firmware.

Nota final

Si vas a utilizar un *generate* **aíslalo en un solo módulo**, no juntes un *generate* con el resto del desarrollo porque se puede volver muy complejo de comprender el desarrollo.