LABORATORIO DE ARQUITECTURA DE COMPUTADORES

SESIÓN 2: operadores de *n* bits

Objetivo

En la sesión anterior hemos escrito los modelos de operadores lógicos de 4 bits pero, ¿qué sucedería si los quisiéramos de más bits? ¿Habría que repetir tediosamente una línea de código por instancia? ¿Y si los necesitamos de *n* bits siendo *n* una variable? Para solucionar este problema VHDL nos ofrece la sentencia GENERATE que vamos a aprovechar para modelar operadores lógicos de tamaño *n* parametrizable.

En un segundo paso, crearemos algunos operadores que nos serán muy útiles en el futuro y que están gobernados por una señal de control (enable). En realidad son pequeñas modificaciones de los anteriores.

A la vez, vamos a aprender a pasar un valor entero a su equivalente binario sobre un bus para programar *test benches* cada vez más útiles.

Modelo VHDL de un operador AND de *n* bits con GENERATE

Asumimos que ya disponemos de nuestra puerta AND de 2 entradas escrita en VHDL (and2.vhd). Un posible modelo estructural de un operador AND de *n* bits sería el siguiente (op_and_n.vhd):

```
--operador AND de n bits

ENTITY op_and_n IS
  GENERIC (n: INTEGER:=8);
  PORT (x, y: IN BIT_VECTOR(n-1 DOWNTO 0); r: OUT BIT_VECTOR(n-1 DOWNTO 0));
END op_and_n;

ARCHITECTURE estructural OF op_and_n IS
  COMPONENT and2
    PORT (e1, e2: IN BIT; sal: OUT BIT);
  END COMPONENT;
BEGIN
  bateria: FOR i IN 0 TO n-1 GENERATE
    puertas: and2 PORT MAP (x(i), y(i), r(i));
  END GENERATE;
END estructural;
```

Nótese que declaramos un entero n inicializado a 8 con la sentencia GENERIC. Es un parámetro que nos sirve para establecer el tamaño de los buses y del operador. Este parámetro puede adoptar cualquier valor cuando instanciemos este modelo posteriormente. El hecho de que lo inicialicemos no significa que tenga que valer eso siempre. Se le asigna un valor para prevenir el caso de que no se lo demos posteriormente.

La sentencia GENERATE funciona en combinación con un bucle automatizando la descripción de un modelo que tiene una estructura regular. Sólo tenemos que preocuparnos de conectar correctamente los puertos, indexados en el índice del bucle siempre que sea posible.

GENERATE también puede usarse en combinación con sentencias condicionales, por ejemplo para distinguir el primer o el último elemento de una estructura *hardware*.

Test del operador

Un *test bench* del operador AND de *n* bits podría ser el siguiente (tb_op_and_n.vhd):

```
--este es el test-bench de un operador AND de TAMANO_PALABRA bits

ENTITY tb_op_and_n IS
END tb_op_and_n;

ARCHITECTURE estructural OF tb_op_and_n IS
  COMPONENT op_and_n
     GENERIC (n: INTEGER:=8);
     PORT (x, y: IN BIT_VECTOR(n-1 DOWNTO 0); r: OUT BIT_VECTOR(n-1 DOWNTO 0));
  END COMPONENT;

  FOR operador: op_and_n USE ENTITY WORK.op_and_n(estructural);

  --se puede declarar en un paquete que invocaremos con USE
  CONSTANT TAMANO_PALABRA: integer := 2;

  SIGNAL A, B, C: BIT_VECTOR(TAMANO_PALABRA-1 DOWNTO 0);

BEGIN
  operador: op_and_n GENERIC MAP (TAMANO_PALABRA) PORT MAP(A, B, C);

  PROCESS
  VARIABLE valor, limite: INTEGER;
  BEGIN
    limite:= 2**TAMANO_PALABRA-1;

    FOR i IN 0 TO limite LOOP
      valor:=i;

      FOR k IN A'REVERSE_RANGE LOOP
        A(k)<=BIT'VAL(valor REM 2);
        valor:=valor/2;
      END LOOP;

      FOR j IN 0 TO limite LOOP
        valor:=j;

        FOR k IN B'REVERSE_RANGE LOOP
          B(k)<=BIT'VAL(valor REM 2);
          valor:=valor/2;
        END LOOP;
        WAIT FOR 2 ns;

      END LOOP;
    END LOOP;

    WAIT FOR 2 ns;
    WAIT;
  END PROCESS;
END estructural;
```

Vemos como a pesar de que el tamaño n inicializado en el operador original es 8, ahora forzamos a que adopte otro valor mediante la sentencia GENERIC MAP y la constante TAMANO_PALABRA. En este ejemplo declaramos la constante en la propia arquitectura del modelo de *test bench* pero ya veremos que es posible declararla en un módulo aparte que se invoca con el comando USE y que funciona a modo de fichero *include*.

Otro elemento nuevo que introducimos en esta sesión es la rutina que pasa de valor entero a señal de tipo bus. Observamos que es un bucle en el que aplicamos el algoritmo de codificación de enteros realizando sucesivas divisiones por la base. Lo copiamos seguidamente para estudiarlo en detalle:

```
FOR k IN A'REVERSE_RANGE LOOP
  A(k)<=BIT'VAL(valor REM 2);
  valor:=valor/2;
END LOOP;
```

Lo que queremos hacer es asignar el bit correspondiente a cada línea del bus A de acuerdo a la codificación binaria del entero valor. Para ello barremos todo el rango del bus desde el peso 0 al $n - 1$ asignando el resto de la división entera entre la base 2.

De esta forma podemos programar *test* que utilicen rangos de datos fácilmente.

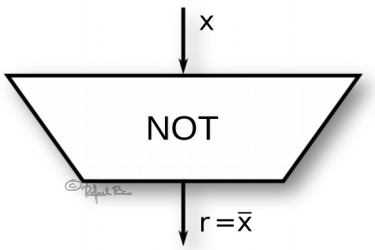
Prácticas

A) Escriba los modelos VHDL de los operadores lógicos NOT, OR, NAND, NOR y XOR de *n* bits.

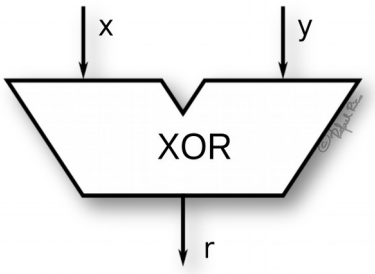
B) Escriba los *test benches* adecuados para los operadores anteriores.

C) Escriba el modelo de un inversor condicional y su *test bench.*

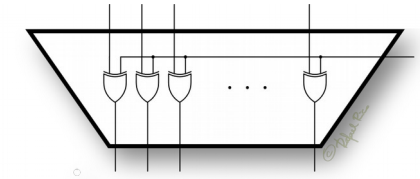
Un inversor de *n* bits realiza su operación lógica siempre, incondicionalmente.



Por otra parte, sabemos que la XOR de un bit d_i con '1' conmuta el valor de d_i mientras que la XOR de d_i con '0' deja d_i tal cual. Aprovechando esta lógica, podemos diseñar un inversor condicional. La XOR de un dato X con un operando y formado por todo '0' copia X a la salida tal cual mientras que la XOR de un dato X con un operando y formado por todo '1' conmuta los bits de X entregando como resultado el inverso de X.



Pues bien, dado que el operando y controla el funcionamiento del inversor condicional y que todos sus bits son siempre iguales a '0' o inguales a '1', podemos realizar el siguiente diseño en el que el operando y toma valor de una señal de control ENABLE.



D) Elabore una memoria con los modelos realizados y los resultados de las pruebas con los *test bench*.