LABORATORIO DE ARQUITECTURA DE COMPUTADORES

SESIÓN 0: modelado de puertas lógicas

Objetivo

En esta primera sesión vamos a modelar con VHDL las puertas lógicas básicas de 2 entradas que posteriormente usaremos como base de otros diseños. Además, comprobaremos su correcto funcionamiento familiarizándonos con las herramientas de desarrollo y las diferentes técnicas de *test*.

Aunque la configuración del entorno de desarrollo queda a la libre elección de cada cual, en los guiones de las prácticas utilizaremos las aplicaciones de *software* libre GHDL y GTKWave.

Modelo VHDL de puerta AND

El lenguaje de programación VHDL cumple un variado conjunto de objetivos: descripción y documentación de *hardware*, simulación y *testeo* de circuitos y síntesis en lógica reconfigurable. En el ámbito de este laboratorio lo usaremos como simulador.

VHDL admite tres tipos de descripción: comportamental, estructural y RTL. La primera está más cerca de la programación que de la descripción del *hardware* y puede ser adecuada para síntesis. Las descripciones estructural y RTL están más cerca de la realidad física del circuito. Nosotros utilizaremos la descripción estructural construyendo bloques de complejidad creciente a partir de un desarrollo jerárquico.

No obstante, el modelado de puertas sólo puede realizarse a partir de una descripción comportamental ya que VHDL no permite instanciar bloques de menor entidad.

A continuación proponemos un modelo de puerta AND de dos entradas (`and2.vhd`):

```
--puerta and de 2 entradas

ENTITY and2 IS
  GENERIC (retardo: TIME:= 2 ns);
  PORT (e1, e2: IN BIT; sal: OUT BIT);
END and2;

ARCHITECTURE comportamiento OF and2 IS
BEGIN
  PROCESS (e1, e2)
  BEGIN
    sal <= e1 AND e2 AFTER retardo;
  END PROCESS;
END comportamiento;
```

La entidad declara las entradas y salidas de la puerta así como una constante temporal que indica su **retardo**. No podemos olvidar que los dispositivos electrónicos que están en la base de los diseños de la arquitectura de los computadores son físicos y no lógicos o matemáticos de forma que sufren un cierto retardo de propagación y una disipación de energía. Dado que nosotros evaluaremos los diseños desde un punto de vista temporal, hemos de modelar su retardo de manera que posteriormente podamos cuantificar su rendimiento.

La arquitectura declara un proceso que asigna valor a la señal de salida cada vez que cambia alguna de las señales de entrada. Como se ha dicho más arriba, el modelado es comportamental y se realiza con una única sentencia de asignación.

Las puertas NOT, OR, NAND, NOR y XOR de 2 entradas se modelarán de forma similar. Lo único que vamos a tener en cuenta es que cada una de ellas experimenta un retardo diferente que tiene que ver con la implementación (tipo de transistor, familia lógica, tecnología, etc.). Tomaremos los siguientes retardos:

	NOT	NAND	NOR	AND	OR	XOR
retardo	1 ns	1 ns	1 ns	2 ns	2 ns	3 ns

Test de las puertas

Para comprobar el correcto funcionamiento cada una de las puertas hemos de inyectarle señales al modelo y observar si la salida es correcta. Esto se hace con un modelo VHDL conocido como *test bench* o banco de pruebas.

Un *test bench* es un modelo estructural que conecta una instancia del modelo bajo *test* con un proceso que inyecta señales. Es, por tanto, una entidad sin relación con el exterior (sin declaración de señales) y una arquitectura cuya parte declarativa enumera el circuito bajo estudio y las señales involucradas.

A continuación proponemos un ejemplo de *test bench* sencillo (`tb_and2.vhd`):

```
--este es el test-bench de una puerta AND de 2 entradas
ENTITY tb_and2 IS
END tb_and2;

ARCHITECTURE estructural OF tb_and2 IS
--declaración del componente a estudiar
  COMPONENT and2
    PORT (e1, e2: IN BIT; sal: OUT BIT);
  END COMPONENT;

--especificación de la configuración del componente
  FOR componente: and2 USE ENTITY WORK.and2(comportamiento);

--declaración de señales de interconexión
  SIGNAL a, b: BIT:= '0';
  SIGNAL s: BIT:='0';

--comienza la descripción de los procesos
BEGIN
--primer proceso: instanciación del componente a estudiar
  componente: and2 PORT MAP (a, b, s);

--segundo proceso: estimulo de las señales de entrada
  PROCESS
  BEGIN
    a<= '0';
    b<= '0';
    WAIT FOR 5 ns;
    a<= '1';
    b<= '0';
    WAIT FOR 5 ns;
    a<= '0';
    b<= '1';
    WAIT FOR 5 ns;
    a<= '1';
    b<= '1';
    WAIT FOR 5 ns;
    WAIT;
  END PROCESS;
END estructural;
```

Compilación y simulación

Como hemos dicho más arriba, vamos a utilizar herramientas de *software* libre. Concretamente compilaremos y ejecutaremos los modelos con GHDL.

Abrimos una ventana de comandos y nos situamos en el subdirectorio en el que hemos salvado los modelos. Invocamos GHDL para analizar los modelos comprobando su corrección sintáctica así:

```
> ghdl -a and2.vhd
> ghdl -a tb_and2.vhd
```

Los argumentos pasados son `-a` y el nombre del fichero que contiene el modelo. Una vez que estamos seguros de que no hay errores, invocamos GHDL con el fin de compilar (elaborar) los modelos así:

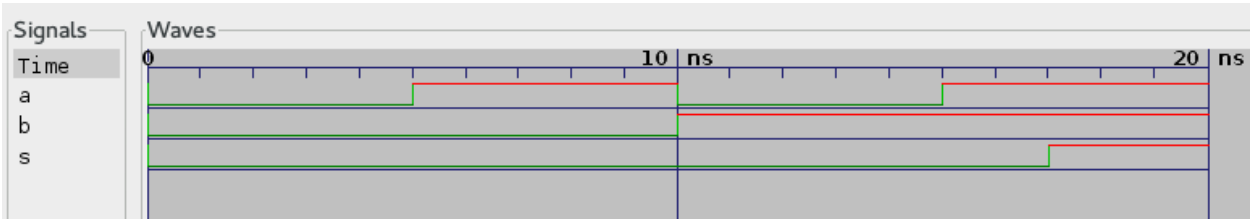
```
> ghdl -e and2
> ghdl -e tb_and2
```

Ahora el argumento es `-e` y el nombre de la entidad del modelo. Si todo ha sido correcto se habrán generado sendos binarios que se puede ejecutar. Evidentemente, hemos de correr el *test bench* que es el encargado de inyectar señales al modelo de la puerta. Con el fin de poder ver luego esas señales vamos a salvarlas en un fichero. El comando será el siguiente:

```
> ghdl -r tb_and2 --vcd=cronograma.vcd
```

Una vez obtenido el fichero con las señales, podemos verlo con GTKWave al que invocaremos así:

```
> gtkwave cronograma.vcd
```



Un *test bench* más sofisticado

A continuación se ofrece un *test bench* que ilustra algunas características del lenguaje VHDL que permiten realizar pruebas potentes de una manera elegante. Veremos cómo inyectar señales desde una tabla de verdad y cómo comprobar que las salidas son correctas de manera automática.

```
--este es el test-bench de una puerta AND de 2 entradas
ENTITY tb_and2 IS
END tb_and2;

ARCHITECTURE estructural OF tb_and2 IS

--declaración del componente a estudiar
  COMPONENT and2
    PORT (e1, e2: IN BIT; sal: OUT BIT);
  END COMPONENT;

--especificación de la configuración del componente
  FOR componente: and2 USE ENTITY WORK.and2(comportamiento);

--declaración de señales de interconexión
  SIGNAL a, b: BIT:= '0';
  SIGNAL s: BIT:='0';
```

```
--comienza la descripción de los procesos
BEGIN
--primer proceso: instanciación del componente a estudiar
  componente: and2 PORT MAP (a, b, s);

--segundo proceso: estimulo de las señales de entrada
  PROCESS
    TYPE tipo_patron IS RECORD
--entradas a la puerta
      a, b: BIT;
--salida
      s: BIT;
    END RECORD;
--el patrón
    TYPE array_patron IS ARRAY (NATURAL RANGE <>) of tipo_patron;
    CONSTANT tabla_verdad: array_patron :=
      (('0', '0', '0'),
       ('0', '1', '0'),
       ('1', '0', '0'),
       ('1', '1', '1'));

  BEGIN
--inyecta las señales desde la tabla de verdad
    FOR i IN tabla_verdad'RANGE LOOP
      a <= tabla_verdad(i).a;
      b <= tabla_verdad(i).b;

--espero al resultado
      WAIT FOR 5 ns;

--compruebo si es correcto
         ASSERT s = tabla_verdad(i).s REPORT "resultado incorrecto" SEVERITY
ERROR;
    END LOOP;

--mensaje final
    ASSERT false REPORT "fin del test" SEVERITY NOTE;
    WAIT;
  END PROCESS;
END estructural;
```

Observamos que en la parte declarativa del proceso encargado de inyectar señales se declara un tipo `RECORD` consistente en una estructura de 3 señales de tipo `BIT` que sólo por claridad se han escrito en líneas diferentes. A continuación se declara un *array* de esas estructuras y se rellena con los valores constantes correspondientes a la tabla de verdad de la puerta `AND`.

Ya en el cuerpo del proceso (detrás del `BEGIN`) encontramos un bucle que lee cada una de las componentes del *array* para inyectar las señales de entrada y comprobar la salida con una orden `ASSERT`.

Una orden `ASSERT` comprueba la condición y si no se cumple emite el mensaje asignado y marca un nivel de aviso (`ERROR`, `NOTA`, etc.).

Prácticas

A) Escriba los modelos VHDL correspondientes a las puertas NOT, NAND, NOR, OR y XOR.

Cada modelo comportamental implementará un tipo de puerta lógica y declarará un tipo genérico de clase temporal para configurar el retardo físico.

B) Escriba el *test bench* de cada puerta y compruebe su correcto funcionamiento.

C) Presente en GTKWave los cronogramas obtenidos de cada puerta.

D) Elabore una memoria con los programas, las salidas de GHDL y los cronogramas.