DISEÑO DIGITAL I

Ejercicios de modelado eficiente y VHDL-2008

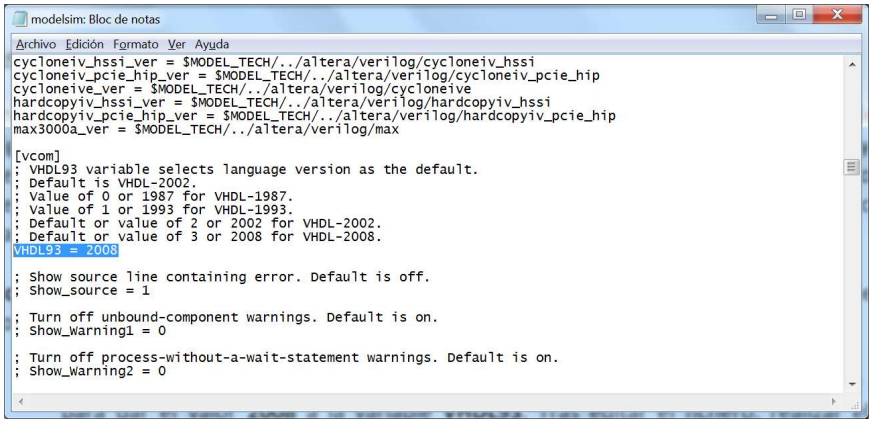
DD1-A2

Introducción

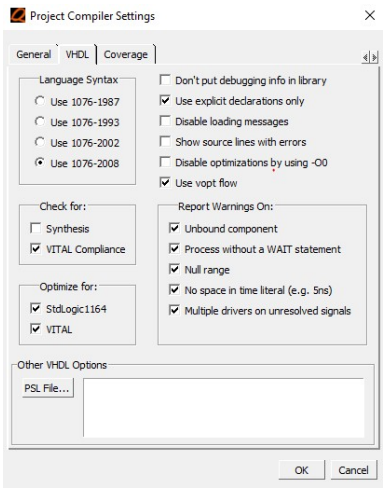
En esta actividad realizará tres ejercicios relacionados con el modelado eficiente de sistemas digitales. Para la realización de los ejercicios tendrá que utilizar el simulador *Questa* y la documentación adjunta (fichero **DD1_A2.zip** que debe descomprimir en su directorio de trabajo). Los ficheros descargados utilizan sintaxis de VHDL-2008, por lo que deberán ser compilados para esta versión del lenguaje.

Nota: Para configurar *Questa* de modo que compile los ficheros con sintaxis VHDL-2008 puede utilizar dos procedimientos:

1. Editar el fichero **modelsim.ini**, que está ubicado en la carpeta de instalación de *Questa*, para dar el valor **2008** a la variable **VHDL93**. Tras editar el fichero, realizar el cambio y arrancar *Questa*, cualquier fichero que se cree o añada a un nuevo proyecto se compilará de acuerdo con el estándar VHDL-2008.



2. Seleccionando el fichero que contiene código VHDL-2008 en la pestaña **Project**, pulsando el botón derecho del ratón para desplegar el menú y seleccionar **Compile** -> **Compile Properties…** En la ventana que aparece, seleccione la pestaña **VHDL** y, en ella, marque la opción **Use 1076-2008**. Con esta alternativa, únicamente los ficheros a los que se haya cambiado la propiedad anterior serán compilados con la sintaxis y reglas de VHDL-2008.



Ejercicio 1

En la documentación adjunta dispone de un modelo VHDL de un circuito **complementador** (un circuito que deja pasar inalterado un dato o invierte todos sus bits en función del valor de la entrada de **control**) que se muestra a continuación:

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

entity complementador is
port(d_in:   in  std_logic_vector(31 downto 0);
     ctrl:   in  std_logic;
     d_out:  out std_logic_vector(31 downto 0));
end entity;

architecture rtl_2002 of complementador is
begin
  process(ctrl, d_in)
  begin
    for i in 0 to 31 loop
      d_out(i) <= d_in(i) xor ctrl;

    end loop;
  end process;
end rtl_2002;

architecture rtl_2008 of complementador is
begin


end rtl_2008;
```

Complete la arquitectura **rtl_2008** para realizar un modelo más eficiente haciendo uso de las operaciones entre arrays y escalares que incluye la versión del lenguaje VHDL de 2008. Realice un *test-bench* para verificar el funcionamiento

Ejercicio 2

El siguiente código corresponde al modelo VHDL de un decodificador binario de 6 a 64 líneas:

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

entity decodificador is
port(d_in:   in  std_logic_vector(5 downto 0);
     enable: in  std_logic;
     d_out:  out std_logic_vector(63 downto 0));
end entity;

architecture rtl_2008 of decodificador is
begin
  process(all)
    variable tmp: std_logic_vector(63 downto 0);

  begin
    tmp := (others => '0');
    tmp(conv_integer(d_in)) := enable;
    d_out <= tmp;

  end process;
end rtl_2008;
```

El modelo se ha construido haciendo uso de una variable. Las variables VHDL tienen un mecanismo de actualización similar al de las variables de los lenguajes de programación, se actualizan tras la ejecución de una sentencia que les asigna valor; mecanismo distinto al de las señales VHDL, que no se actualizan hasta que el proceso en que *se proyecta* la asignación de valor se ha suspendido. El código anterior ilustra cómo las variables pueden emplearse para codificar modelos sintetizables de circuitos:

- Un decodificador binario activa el bit de salida cuyo *número* corresponde a la *decodificación* de la combinación de entrada en binario natural, manteniendo el resto de los bits inactivos.

- En el ejemplo, que corresponde a un decodificador con entradas y salidas activas a nivel alto, la variable **tmp**, que es una variable local al proceso, se declara con el mismo tipo de datos y número de bits que el puerto de salida.

- Cuando se ejecuta el proceso, lo primero que se hace es poner todos los bits de la variable a cero; a continuación, si el decodificador está habilitado (si la entrada **enable** está a '1'), se pone a '1' el bit de la variable cuyo índice corresponde al código de entrada y, finalmente, se transfiere el valor de la variable a la señal que modela el puerto de salida (**d_out**).

El uso de variables en el modelado RTL está justificado cuando proporciona una sustancial reducción en el nivel de complejidad del código.

a) Ejecute una simulación del modelo del decodificador, utilizando el *test-bench* **decodificador_tb.vhd,** y compruebe su correcto funcionamiento. Ambos ficheros están en la documentación adjunta.

Este otro código hace uso de una variable para modelar el funcionamiento de un codificador con prioridad de 8 a 3 líneas:

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

entity codificador is
port(D_in:       in  std_logic_vector(7 downto 0);
     enable:     in  std_logic;
     D_out:      out std_logic_vector(2 downto 0));
end entity;

library ieee;
use ieee.std_logic_unsigned.all;

architecture rtl_2008 of codificador is
begin
  process(all)
    variable tmp: std_logic_vector(2 downto 0);

  begin
    tmp := 3X"0";
    if enable then
      for i in 7 downto 0 loop
        if D_in(i) then
          tmp := tmp + i;
          exit;

        end if;
      end loop;
    end if;
    D_out <= tmp;

  end process;
end rtl;
```

Analice el código para contestar a las siguientes cuestiones:

b) ¿Cumple el código del proceso las reglas de modelado VHDL de circuitos combinacionales? Recuerde que las reglas son dos: el proceso debe ser sensible a todas las entradas del combinacional y debe asignar valor a todos los bits de salida para todas las combinaciones de entrada.

c) ¿Qué valor toman los bits de salida cuando la entrada **enable** vale '0'?

d) Realice un modelo del codificador utilizando la sentencia **case** con indiferencias (**case?**) introducida en VHDL-2008. Realice un *test_bench* para verificar el funcionamiento de ambos modelos.

   Compare esta solución con la anterior para distinguir las ventajas e inconvenientes de cada una y argumentar cuál de las dos le parece más apropiada para modelar codificadores con prioridad.

Ejercicio 3

En general resulta indeseable que en los procesos de modelos sintetizables se pueda proyectar la asignación de valor a una señal más de una vez durante la ejecución del proceso. Cuando esto ocurre, la proyección de valor efectiva es la correspondiente a la última sentencia de asignación ejecutada[1]. Por ejemplo, en un proceso en el que se ejecuten dos sentencias de asignación, como el que se muestra a continuación, el valor cuya asignación se proyecta es el correspondiente a la segunda sentencia y, por tanto, el valor que tomará **S** cuando el proceso se suspenda será "101".

```
process(all)
begin
  S <= "000";
  ...
  ...
  S <= "101";

end process;
```

En el siguiente ejemplo, como la segunda sentencia sólo asigna valor a uno de los bits de **S**, el valor proyectado para ese bit es el que le asigna la segunda sentencia, mientras que el valor proyectado para los otros dos es el especificado en la primera sentencia de asignación. El valor que tomará **S** será, en consecuencia, "010".

```
process(all)
begin
  S <= "000";
  ...
  ...
  S(1) <= '1';

end process;
```

Aunque, como ya se ha recordado más arriba, resulta en general indeseable que se den este tipo de casos en el código de procesos de modelos sintetizables –porque, aún en el caso de que puedan corresponder a modelos correctos, suelen perjudicar la inteligibilidad y eficiencia del código–, en determinados casos puede aprovecharse este *mecanismo* de actualización para evitar el uso de variables y simplificar el código del modelo. Por ejemplo, el proceso que modela el decodificador con ayuda de una variable, en el ejercicio 2, podría haberse realizado de la siguiente manera:

```
process(all)
begin
  d_out <= (others => '0');
  d_out(conv_integer(d_in)) <= enable;

end process;
```

[1] Esto es así porque en los modelos para síntesis no se especifican retardos y, por tanto, el retardo nominal para la asignación es 0 y el mecanismo de retardo que se aplica por defecto es el **inercial**. Cuando se emplea el modo de retardo **inercial**, al ejecutarse una sentencia de asignación se actualiza la lista de valores proyectados para la señal y se borran de ella todos aquellos que tengan un retardo menor o igual al de la sentencia ejecutada.

a) Modifique el código del proceso incluido en el fichero **decodificador.vhd** por la versión del mismo de la página anterior y ejecute una simulación (empleando el *test-bench* **decodificador_tb.vhd**) para comprobar que el funcionamiento observado en la simulación es el mismo.

b) Determine si en el código del proceso que modela el codificador del ejercicio 2, y que para su comodidad se reproduce aquí, es posible, al igual que en el caso del decodificador, sustituir la variable **tmp** por la señal que modela el puerto de salida, **D_out**.

```
process(all)
  variable tmp: std_logic_vector(2 downto 0);

begin
  tmp := 3X"0";
  if enable then
    for i in 7 downto 0 loop
      if D_in(i) then
        tmp := tmp + i;
        exit;

      end if;
    end loop;
  end if;
  D_out <= tmp;

end process;
```

c) Determine si en el siguiente código, que modela el funcionamiento de un detector de paridad, es posible sustituir la variable **tmp** por la señal **par_OK**.

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

entity paridad_32b is
port(D_in:       in  std_logic_vector(31 downto 0);
     bit_par:    in  std_logic;
     odd_even:   in  std_logic;
     par_OK:     out std_logic);
end entity;

architecture rtl_2002 of paridad_32b is
begin
  process(D_in, bit_par, odd_even)
    variable tmp: std_logic;

  begin
    tmp := odd_even xor bit_par;
    for i in 31 downto 0 loop
      tmp := tmp xor D_in(i);

    end loop;
    par_OK <= tmp;

  end process;
end rtl_2002;
```

d) Teniendo en cuenta los resultados de los análisis realizados en este ejercicio, extraiga la condición o condiciones que deben cumplirse para que sea posible sustituir una variable por una señal en un proceso *sintetizable*.