

Diseño Digital I

Modelado Eficiente VHDL 2008



BANCO DE REGISTROS

```
architecture rtl of reg_file_4_8bits is
    signal reg0:      std_logic_vector(7 downto 0);
    signal reg1:      std_logic_vector(7 downto 0);
    signal reg2:      std_logic_vector(7 downto 0);
    signal reg3:      std_logic_vector(7 downto 0);

begin
    process(clk, rst_n)
    begin
        if rst_n = '0' then
            reg0 <= (others => '0');
            reg1 <= (others => '0');
            reg2 <= (others => '0');
            reg3 <= (others => '0');

        elsif clk'event and clk = '1' then
            if WR = '1' then
                case Dir_WR is
                    when "00" =>
                        reg0 <= Dato_in;
                    when "01" =>
                        reg1 <= Dato_in;
                    when "10" =>
                        reg2 <= Dato_in;
                    when "11" =>
                        reg3 <= Dato_in;
                    when others =>
                        null;
                end case;
            end if;
        end if;
    end process;

    Dato_out <= reg0 when Dir_RD = 0 else
                reg1 when Dir_RD = 1 else
                reg2 when Dir_RD = 2 else
                reg3 when Dir_RD = 3 else
                (others => 'X');

end rtl;
```

- Empleando las técnicas básicas de modelado VHDL se puede abordar la realización de modelos sintetizables de todo tipo de sistemas.
- Pero el lenguaje dispone de recursos que permiten aumentar la eficiencia a la hora de codificar modelos o *test-benches*
- Por ejemplo: ¿Cuántas líneas de código VHDL se precisan para completar el modelo sintetizable de un banco de 32 registros de 32 bits con tres puertos de lectura?

Modelado en VHDL de un Banco de Registros

```
entity reg_file_32_32bits_3port_RD is
port(clk:          in      std_logic;
     rst_n:        in      std_logic;
     WR:           in      std_logic;
     Dir_WR:       in      std_logic_vector(4 downto 0);
     Dir_RD_OP1:   in      std_logic_vector(4 downto 0);
     Dir_RD_OP2:   in      std_logic_vector(4 downto 0);
     Dir_RD_Out:   in      std_logic_vector(4 downto 0);
     Dato_in:      in      std_logic_vector(31 downto 0);
     Dato_OP1:     buffer std_logic_vector(31 downto 0);
     Dato_OP2:     buffer std_logic_vector(31 downto 0);
     Dato_out:     buffer std_logic_vector(31 downto 0));

end entity;

architecture rtl of reg_file_32_32bits_3port_RD is
    type reg_file_32bits is array ( NATURAL RANGE <>) of std_logic_vector(31 downto 0);
    signal reg_file_op: reg_file_32bits(31 downto 0);

begin
    process(clk, rst_n)
    begin
        if rst_n = '0' then
            for i in 31 downto 0 loop
                reg_file_op(i) <= (others => '0');
            end loop;

        elsif clk'event and clk = '1' then
            if WR = '1' then
                reg_file_op(conv_integer(Dir_WR)) <= Dato_in;

            end if;
        end if;
    end process;

    Dato_OP1 <= reg_file_op(conv_integer(Dir_RD_OP1));
    Dato_OP2 <= reg_file_op(conv_integer(Dir_RD_OP2));
    Dato_out <= reg_file_op(conv_integer(Dir_RD_out));

end rtl;
```

DECLARACIÓN DE UN TIPO DE DATOS DE USUARIO (ARRAY)

SE CREA UNA SEÑAL DE ESTE NUEVO TIPO DE DATOS PARA MODELAR EL BANCO

BUCLE **FOR** EN MODELO SINTETIZABLE

PARA RESUMIR EL CONJUNTO DE SENTENCIAS DE RESET (UNA POR REGISTRO)

FUNCIONES DE CONVERSIÓN

PARA DECODIFICAR Y EVITAR EL USO DE UNA SENTENCIA **CASE**, O MULTIPLEXAR SIN USAR UNA SENTENCIA **IF**

Arrays

- En VHDL pueden declararse tipos de datos con estructuras **matriciales**, es decir, **ARRAYS**
- A esta clase de tipos de datos pertenecen, por ejemplo, los tipos **std_logic_vector** o **bit_vector**, que se definen, respectivamente, como un **array** de valores de tipo **std_logic** y **bit**:

```
type std_logic_vector is array (natural range <>) of std_logic;
```

```
type bit_vector is array (natural range <>) of bit;
```

- También pueden definirse **arrays** en los que el tipo básico es, a su vez, otro **array**:

```
type reg_file_32bits is array (natural range <>) of std_logic_vector(31 downto 0);
```

Funciones VHDL (I)

- Las **funciones** VHDL devuelven un **valor** que calculan en base a los **parámetros** de entrada.
 - Dentro de ellas no puede asignarse valor a señales o incluirse sentencias **WAIT**.

- Sintaxis de la Declaración de funciones:

function *nombre* (lista de parámetros) **return** *tipo de datos*;

- Sintaxis de la Definición de funciones:

function *nombre* (lista de parámetros) **return** *tipo de datos* **is**

ZONA DE DECLARACIÓN

begin

ALGORITMO DE PROCESAMIENTO SECUENCIAL

end function;

Funciones VHDL (II)

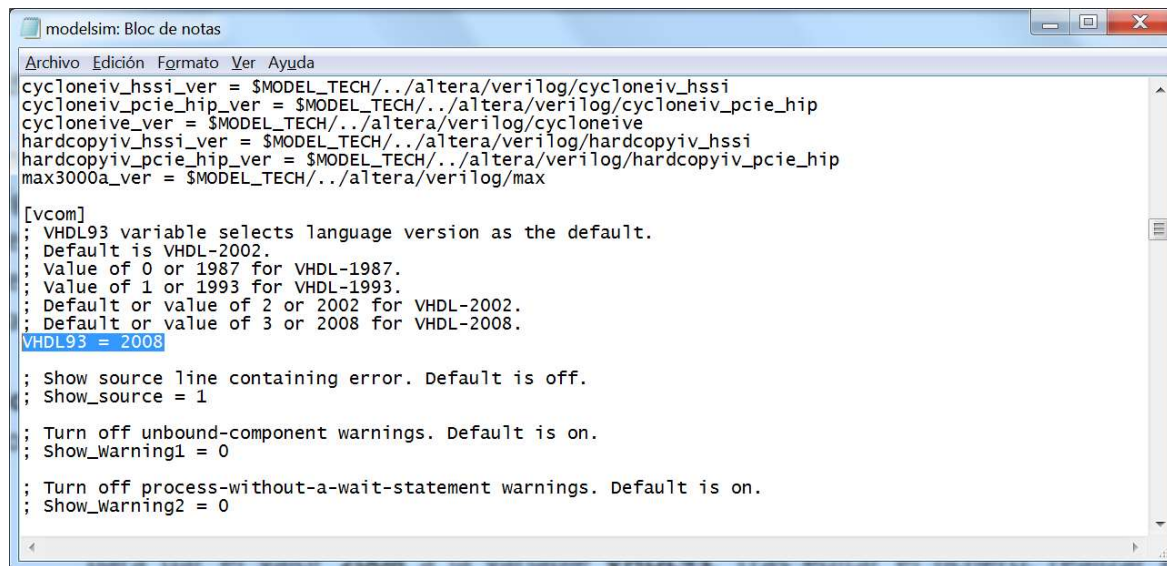
- La función **CONV_INTEGER()** está definida en los paquetes **std_logic_unsigned** y **std_logic_signed**.
 - La función del paquete **std_logic_unsigned** convierte a entero un número expresado en binario natural
 - La función del paquete **std_logic_signed** convierte a entero un número expresado en complemento a dos
- Sobrecarga: puede tenerse visibilidad simultáneamente sobre múltiples funciones u operaciones con el mismo nombre si sus parámetros **formales** son diferentes (en el número o en el tipo de datos)
- En este caso, la función u operación a que se hace referencia en el código se deduce de los parámetros **actuales** que se le pasen en la llamada

VHDL 2008

- En la última revisión del lenguaje VHDL (VHDL-2008) se introducen bastantes cambios para facilitar la codificación e inteligibilidad de los modelos y *test-benches*:
 - Nuevos tipos de **operaciones lógicas**: de **reducción** y para operar un **escalar** y un **array**
 - Interpretación, en ciertos contextos, de **niveles lógicos como valores booleanos**
 - Adición de la cláusula **ALL** para las listas de sensibilidad de procesos y sentencias **CASE** con indiferencias
 - Uso de la sintaxis de las sentencias concurrentes de asignación condicional y selección dentro de procesos, y equiparación de las direccionalidades **OUT** y **BUFFER** y mejora de las constantes
 - Expresiones booleanas y agregados

VHDL 2008

- Puede ser necesario configurar **Questa**: 3 opciones
 - Editar /path_to_questa/**modelsim.ini**
 - Editar /path_to_current_project/**project_name.mpf**
 - En Questa → Project Compiler Settings desde la ventana Project → Use 1076-2008



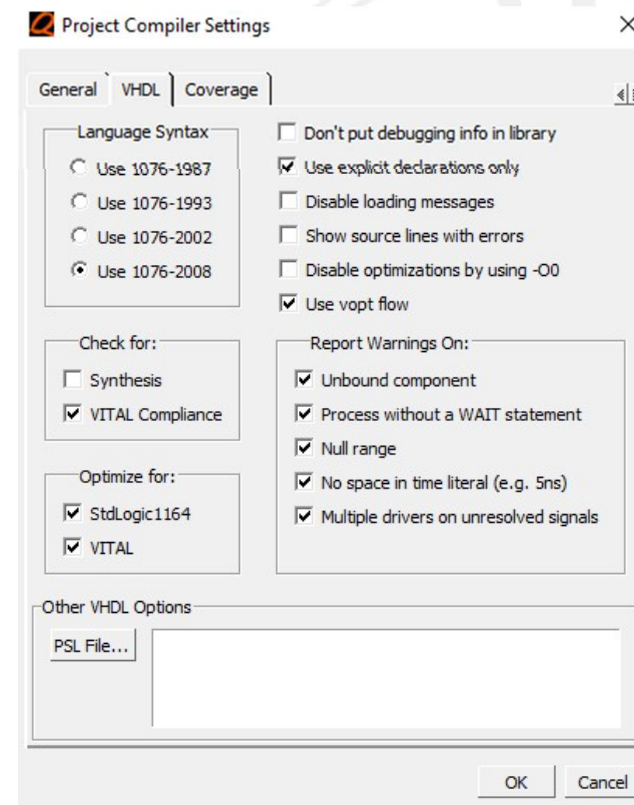
```
modelsim: Bloc de notas
Archivo Edición Formato Ver Ayuda
cycloneiv_hssi_ver = $MODEL_TECH/../../altera/verilog/cycloneiv_hssi
cycloneiv_pcie_hip_ver = $MODEL_TECH/../../altera/verilog/cycloneiv_pcie_hip
cycloneive_ver = $MODEL_TECH/../../altera/verilog/cycloneive
hardcopyiv_hssi_ver = $MODEL_TECH/../../altera/verilog/hardcopyiv_hssi
hardcopyiv_pcie_hip_ver = $MODEL_TECH/../../altera/verilog/hardcopyiv_pcie_hip
max3000a_ver = $MODEL_TECH/../../altera/verilog/max

[vcom]
; VHDL93 variable selects language version as the default.
; Default is VHDL-2002.
; Value of 0 or 1987 for VHDL-1987.
; Value of 1 or 1993 for VHDL-1993.
; Default or value of 2 or 2002 for VHDL-2002.
; Default or value of 3 or 2008 for VHDL-2008.
VHDL93 = 2008

; Show source line containing error. Default is off.
; Show_source = 1

; Turn off unbound-component warnings. Default is on.
; Show_warning1 = 0

; Turn off process-without-a-wait-statement warnings. Default is on.
; Show_warning2 = 0
```



Ejemplo de reducción

Detector de paridad de 4 bits

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

entity paridad_4b is
port(D_in:      in  std_logic_vector(3 downto 0);
     bit_par:   in  std_logic;
     odd_even:  in  std_logic;  -- '0' odd, '1' even
     par_OK:    out std_logic);
end entity;

architecture rtl_2002 of paridad_4b is
begin
    par_OK <= D_in(0) xor D_in(1) xor D_in(2) xor D_in(3) xor bit_par xor odd_even;
end rtl_2002;
```

VHDL-2002

Esquema ineficiente

Cuando el número de bits del dato que se opera es grande, la expresión tiene un gran número de términos

Detector de paridad de 32 bits

```
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;
```

Declaración de variable

Las variables son útiles para construir algoritmos que representan el funcionamiento del sistema que se modela

Algoritmo de procesamiento

Los bucles son útiles para reducir el número de líneas de código

Ejemplos de reducción e interpretación booleana

VHDL-2008

Detector de paridad de 32 bits

```
architecture rtl_2008 of paridad_32b is
begin
    par_OK <= xor (D_in&bit_par&odd_even);
end rtl_2008;
```

Operación de reducción

La operación se aplica asociativamente a todos los bits del vector

Detector de paridad de 32 bits

```
architecture rtl_2008_bis of paridad_32b is
begin
    par_OK <= xor (D_in&bit_par) when not odd_even
               else not(xor (D_in&bit_par));
end rtl_2008_bis;
```

Std_logic a Boolean

Los valores del tipo std_logic se interpretan, en determinados contextos, como valores booleanos ('0' es FALSE y '1' es TRUE).

Ejemplo de operaciones array-escalar

Sumador-Restador de 4 bits

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

entity sumador_restador is
port(ctrl:  in    std_logic;
      A:    in    std_logic_vector(3 downto 0);
      B:    in    std_logic_vector(3 downto 0);
      S:    buffer std_logic_vector(3 downto 0));

end entity;

architecture rtl_2002 of sumador_restador is
begin
  S <= A + B when ctrl = '0' else
      A + (not B) + 1;

end rtl_2002;
```

VHDL-2002

Estructura condicional

Las operaciones solo se pueden hacer array-array o escalar-escalar, dando lugar a esquemas condicionales

Sumador-Restador de 4 bits

```
architecture rtl_2008 of sumador_restador is
begin
  S <= A + (B xor ctrl) + ctrl;

end rtl_2008;
```

VHDL-2008

Operación array-escalar

La operación se aplica entre el escalar y cada uno de los elementos del array. El resultado es un array

Ejemplos de cláusula *all* y sentencia *case* con indiferencias

VHDL-2002

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

entity codificador is
port(D_in:      in  std_logic_vector(3 downto 0);
     enable:    in  std_logic;
     D_out:     out std_logic_vector(1 downto 0));
end entity;

architecture rtl_2002 of codificador is
begin
  process(enable, D_in)
  begin
    if enable = '1' then
      if D_in(3) = '1' then
        D_out <= "11";

      elsif D_in(2) = '1' then
        D_out <= "10";

      elsif D_in(1) = '1' then
        D_out <= "01";

      else
        D_out <= "00";

      end if;

    else
      D_out <= "00";

    end if;

  end process;
end rtl_2002;
```

VHDL-2008

```
architecture rtl_2008 of codificador is
begin
  process(all)
  begin
    if enable then
      case? D_in is
        when "1---" => D_out <= "11";
        when "01--" => D_out <= "10";
        when "001-" => D_out <= "01";
        when others => D_out <= "00";
      end case?;

    else
      D_out <= "00";

    end if;

  end process;
end rtl_2008;
```

CLÁUSULA ALL

Incluye en la lista de sensibilidad todas las entradas que se leen en el proceso

CASE?

Sentencia CASE con indiferencias. Evalúa únicamente los bits que no valen '-'

Direccionalidad **out**, constantes y sentencias concurrentes

Contador BCD

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

entity contador is
port(clk:    in std_logic;
     nRst:   in std_logic;
     enable: in std_logic;
     Q:      out std_logic_vector(3 downto 0));
end entity;

architecture rtl_2008 of contador is
begin
  process(clk, nRst)
  begin
    if not nRst then
      Q <= 4D"0";

    elsif clk'event and (?? clk) then
      Q <= Q + 1 when enable and (Q /= "1001") else
        4D"0" when enable
        unaffected;

    end if;
  end process;
end rtl_2008;
```

VHDL-2008

DIRECCIONALIDAD TIPO OUT

En VHDL-2008 los puertos de tipo OUT y tipo BUFFER tienen las mismas propiedades

FORMATOS DE CONSTANTES

En VHDL-2008 se puede especificar el número de bits correspondientes a una constante octal, decimal o hexadecimal

SINTAXIS DE SENTENCIAS

En VHDL-2008 se puede utilizar la sintaxis de las sentencias concurrentes dentro de un proceso

OPCIONAL

Aporta claridad

?? clk → TRUE si **clk = '1'**

Q /= "1001" → '1' si **Q /= "1001"**

Ejemplos de expresiones booleanas y std_logic

Contador BCD

```
architecture rtl_2008 of contador is
begin
  process(clk, nRst)
  begin
    if not nRst then
      Q <= 4D"0";

    elsif clk'event and (?? clk) then
      Q <= Q + 1 when enable and (Q /= "1001") else
        4D"0" when enable else
        unaffected;

    end if;
  end process;
end rtl_2008;
```

Expresión de tipo boolean

?? clk → TRUE si clk = '1'

(?? enable) and (Q /= 9)

VHDL-2008

Expresión de tipo std_logic

Q /= "1001" → '1' si Q /= "1001"

enable and (Q /= "1001")

```
architecture rtl_2008_bis of contador is
begin
  process(clk, nRst)
  begin
    if not nRst then
      Q <= (others => '0');

    elsif clk'event and (?? clk) then
      Q <= Q + 1 when (?? enable) and (Q /= 9) else
        4D"0" when enable else
        unaffected;

    end if;
  end process;
end rtl_2008_bis;
```

Ejemplo de agregados de arrays y escalares

Sumador

VHDL-2008

Agregado

En VHDL-2008 se pueden construir AGREGADOS entre arrays y escalares

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

entity sumador is
port(C_in:  in    std_logic;
     A:     in    std_logic_vector(3 downto 0);
     B:     in    std_logic_vector(3 downto 0);
     S:     buffer std_logic_vector(3 downto 0);
     C_out: buffer std_logic);

end entity;

architecture rtl_2008 of sumador is
begin
    (C_out, S) <= ('0'&A) + ('0'&B) + C_in;
end rtl_2008;
```



POLITÉCNICA

INDUSTRIALES
ETSII | UPM



ETSIT
UPM



escuela técnica superior de
ingeniería
de
diseño
industrial



Telecomunicación
Campus Sur
UPM

MICROELECTRÓNICA
μe-UPM



INSTITUTO
DE ENERGÍA
SOLAR



CEI UPM

Centro de
Electrónica
Industrial



CENTRO
LÁSER
UPM

UNIVERSIDAD
POLITÉCNICA
DE MADRID

ISOM



iptc



CEMDATIC

citSem

➤ Más información: <https://blogs.upm.es/ue-upm/>

➤ Contacto: comunidad.microelectronica@upm.es



MINISTERIO
DE CIENCIA, INNOVACIÓN
Y UNIVERSIDADES



Financiado por
la Unión Europea
NextGenerationEU



Plan de Recuperación,
Transformación y
Resiliencia



AGENCIA
ESTATAL DE
INVESTIGACIÓN

UNIÓN EUROPEA
Fondos estructurales
Invertimos en su futuro



UNIÓN EUROPEA
Fondo Social Europeo

El Fondo Social Europeo invierte en tu futuro



Comunidad
de Madrid

PERTE Chip
microelectrónica y
semiconductores



Plan de Recuperación,
Transformación y Resiliencia



GOBIERNO
DE ESPAÑA

MINISTERIO
PARA LA TRANSFORMACIÓN DIGITAL
Y DE LA FUNCIÓN PÚBLICA

MICROELECTRÓNICA
μe-UPM