Introducción al VHDL

VHDL orientado a la síntesis de circuitos en Dispositivo Lógicos Programables

Curso - 2023

Introducción

Lenguaje de descripción de dispositivos Hardware.

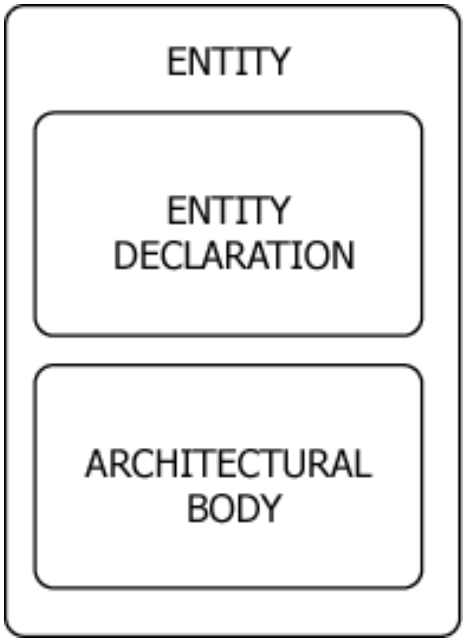
Diferencias con lenguajes de programación:

- describe procesos que ocurren en paralelo.
- Una descripción VHDL, representa el comportamiento o la estructura de un sistema.
- Permite describir un sistema como la interconexión de componentes.

Niveles de representación

- Descripción:
  - **comportamental**: describe comportamiento de las salidas en función de las entradas. (expresión booleana, máquinas de estado,etc.)
  - **estructural**: describe el sistema como un grupo de compuertas o bloques que se interconectan. La descripción estructural se asemeja a un esquemático.

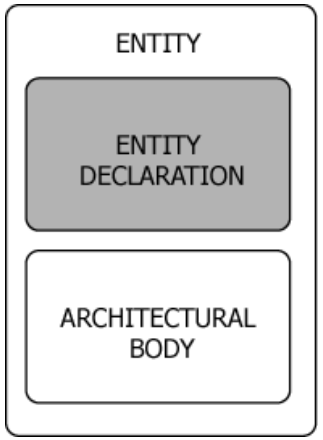
Estructura básica: entidad



- entity declaration: define las entradas y salidas de la entidad.
- architectural body: contiene la descripción de la entidad (lo que hace o contiene), interconexión de componentes y otras entidades, procesos, etc.

Entity declaration

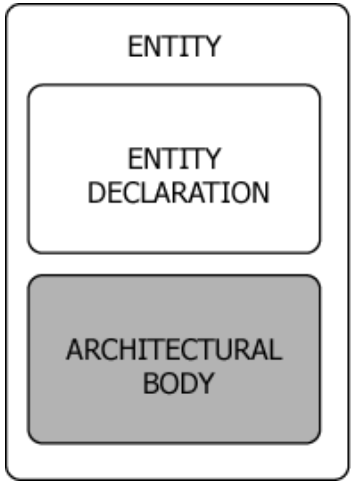
entity NAME_OF_ENTITY is [generic generic_declarations);]
    port (signal_names: mode type;
        signal_names: mode type;
            :
        signal_names: mode type);
end [NAME_OF_ENTITY] ;




```
entity ALARMA is
   port (PUERTA, ENCENDIDO, CINTO: in std_logic;
         SEÑAL: out std_logic);
   end ALARMA;
```

Architecture body

```
architecture architecture_name of NAME_OF_ENTITY is
-- Declarations
   -- component declarations
   -- signal declarations
   -- constant declarations
   -- function declarations
   -- procedure declarations
   -- type declarations
   :
begin
   -- Statements
   :
end architecture_name;
```



Modelo comportamental

- Las asignaciones describen la alarma a nivel de comportamiento del circuito.
- Operadores lógicos permitidos: and, or, nand, nor, xor, xnor and not.

```
architecture comportamental of ALARMA is
   begin
      SEÑAL <= (not PUERTA and ENCENDIDO) or
   (not CINTO and ENCENDIDO);
end comportamental;
```

Ejemplo comportamental

```
entity XNOR2 is
   port (A, B: in std_logic;
         Z: out std_logic);
end XNOR2;
architecture behavioral_xnor of XNOR2 is
   -- signal declaration (of internal signals X, Y)
   signal X, Y: std_logic;
begin
   X <= A and B;
   Y <= (not A) and (not B);
   Z <= X or Y;
end behavioral_xnor;
```

Modelo estructural

- Facilita diseño jerárquico
- Permite reutilizar componentes
- Puede utilizarse para representar resultado de síntesis (lista de nodos)

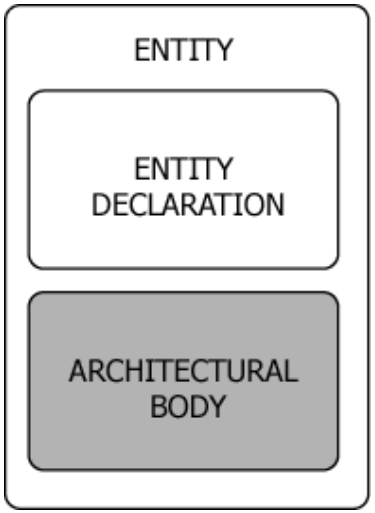
Modelo estructural

- a continuación del encabezado donde se define la entidad (arqu.) se declaran los componentes a utilizar:

```
component NOT1
    port (in1: in std_logic;
          out1: out std_logic);
    end component;
```

Ejemplo: Alarma

```
architecture estructural of ALARMA is
   -- Declaraciones
   component AND_2
      port (in1, in2: in std_logic;
            out1: out std_logic);
   end component;
   component OR_2
      port (in1, in2: in std_logic;
            out1: out std_logic);
   end component;
   component NOT_1
      port (in1: in std_logic;
            out1: out std_logic);
   end component;
```



Instanciación de componentes

Luego del begin se instancian los componentes previamente declarados.

Dos formas:

- instanciación posicional:

label: component-name

port map (signal1,signal2,...signaln);

```
U0: NOT_1 port map (PUERTA, PUERTA_NOT);
```

Instanciación de componentes

- instanciación explícita:

label: component-name port map
(port1=>signal1,
port2=> signal2,…
port3=>signaln);

```
U0: NOT_1 port map
   (in1 => PUERTA,
     out1 => PUERTA_NOT);
```

Ejemplo: Alarma (cont.)

```
-- declaracion de señales para interconectar compuertas
   signal PUERTA_NOT, CINTO_NOT, B1, B2: std_logic;
   begin
   -- Instanciación de componentes
      U0: NOT1 port map (
         in1 => PUERTA,
         out1 => PUERTA_NOT);
      U1: NOT1 port map (CINTO, CINTO_NOT);
      U2: AND2 port map (ENCENDIDO, PUERTA_NOT, B1);
      U3: AND2 port map (ENCENDIDO, CINTO_NOT, B2);
      U4: OR2  port map (B1, B2, SEÑAL);
   end structural;
```

Roteiro

- ✓ Estructura básica
- ✓ Representación comportamental y estructural
- Asignaciones concurrentes
- Procesos: asignaciones secuenciales
- Bibliotecas y paquetes
- Tipos de datos, Signals, Variables

Asignaciones concurrentes

- Asignaciones (<=)
  - establece conexiones entre señales, no necesariamente transferencia a un registro.
- Asignación Incondicional

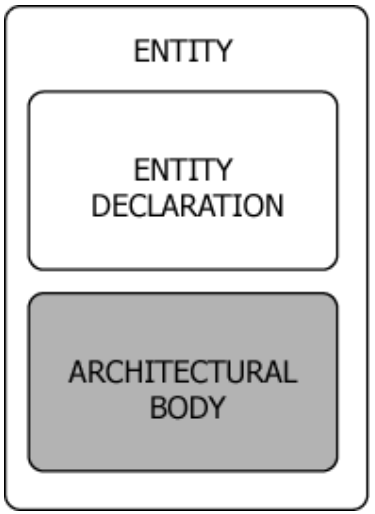
```
A <= B
```

- Asignación Condicional

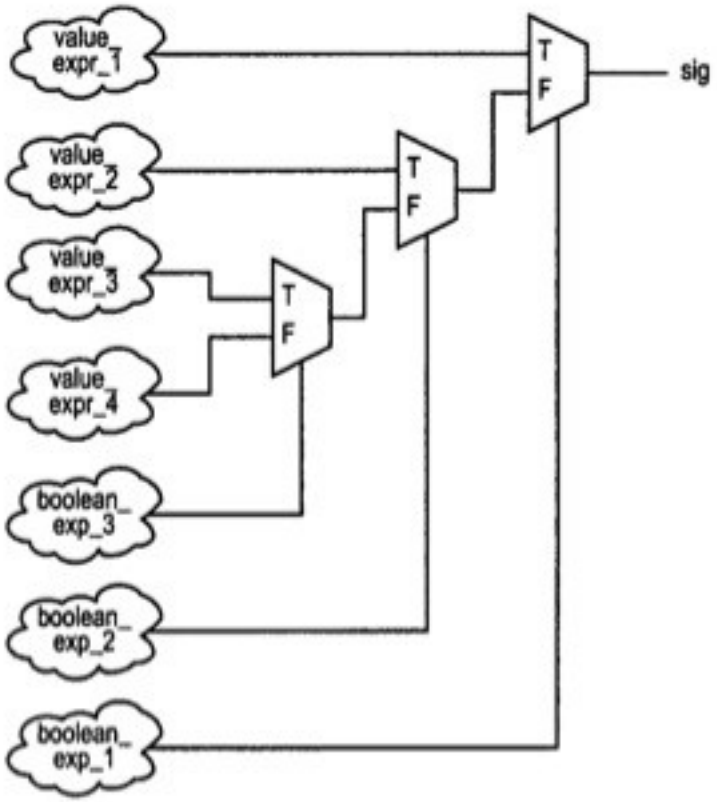
```
Z <=   B when cond1 else
C when cond2 else
A;
```

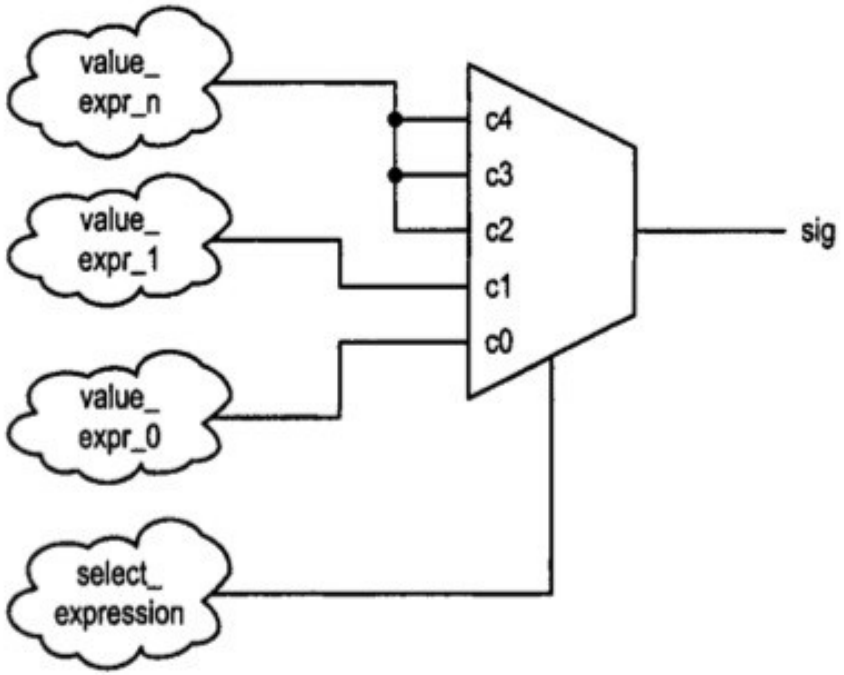
- Select

```
With EXPRESSION
  Z <=   B when choice1 else
  C when choice2 else
  A when others;
```



Condicional vs select





Condicional vs Select

Select:

- todas las combinaciones de entrada tienen un valor.
- se adapta bien para tablas de la verdad

Condicional:

- permite priorizar expresiones de selección
- puede introducir más retardo en la salida

Procesos: Statements secuenciales.

declaración de procesos:

[process_label:] process [(sensitivity_list)] [is]

begin

list of sequential statements such as:

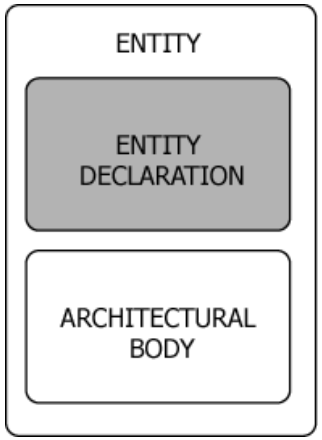
- signal assignments
- variable assignments
- case statement
- exit statement
- if statement

end process [process_label];

Ejemplo de proceso

```
selector: process(A,B,C,cond)
begin
  if (cond= cte1) then
    Z <= B;
  elsif (cond= cte2) then
    Z <= C;
  else
    Z <= A
  end if;
end process;
```

Ejemplo procesos



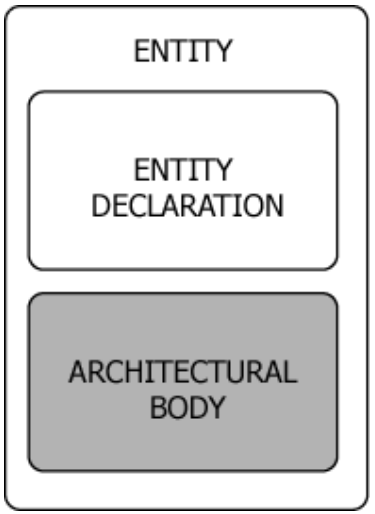

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all;

ENTITY registro IS
GENERIC( WIDTH: integer
        );
PORT(
    rstn : IN    STD_LOGIC;
    clk  : IN    STD_LOGIC;
    ena  : IN    STD_LOGIC;
    D    : IN    STD_LOGIC_VECTOR(WIDTH-1 downto 0);

    Q    : OUT   STD_LOGIC_VECTOR(WIDTH-1 downto 0)
    );
END registro;
```

Ejemplo procesos

```
ARCHITECTURE behav OF registro IS
BEGIN
    D_registro: PROCESS(clk,rstn, ena)  --generacion de los D-FF
        BEGIN
            IF (rstn='0') THEN
                Q <= (OTHERS => '0');
            ELSIF (clk'event AND clk='1') THEN
                IF (ena='1') THEN
                    Q <= D;
                END IF;
            END IF;
    END PROCESS D_registro;
END behav;
```



Procesos: sensitivity list.

declaración de procesos:

[process_label:] process [(**sensitivity_list**)] [is]

  begin

    ...

end process [process_label];

Para **simulación**: el proceso se re-evalúa cada vez que hay un cambio en la "sensitivity list".

Para **síntesis**: la "sensitivity list" es ignorada. El circuito resultante es sensible a cambios en todas las señales que aparecen del lado derecho de una asignación o en la condición de una sentencia condicional.

Concurrentes vs. Secuenciales

- Implementaciones equivalentes:
  - Asignación condicional <=> if ..., elsif ..., else ..., end if
  - Asignación con select <=> case ..., when ..., end case

- Para especificar un registro hace falta un proceso sensible al reloj

Roteiro

- ✓ Estructura básica
- ✓ Representación comportamental y estructural
- ✓ Statements concurrentes
- ✓ Procesos: statements secuenciales
- Bibliotecas y paquetes
- Tipos de datos, Signals, Variables

Bibliotecas y paquetes

- Library: lugar donde el compilador guarda información relacionada con el proyecto actual.
- Package: archivo o módulo que contiene declaración de constantes, componentes, funciones, etc. que deben ser compartidos entre diferentes módulos VHDL.

Ejemplo bibliotecas y paquetes

- El tipo de datos STD_LOGIC está definido en el **paquete** std_logic_1164 de la **biblioteca** ieee.

- Para utilizarlo en un módulo, al comienzo del archivo se debe incluir:
  - library ieee;
  - use ieee.std_logic_1164.all;

Declaración de paquetes

```
package NOMBRE_PACKAGE is
    --- declaración de constantes
    --- declaración de componentes
end NOMBRE_PACKAGE;
```

Ejemplo de paquete

```
package OPERACIONES_LOGICAS is
   component AND2
      port (in1, in2: in std_logic;
            out1: out std_logic);
   end component;
   component OR2
      port (in1, in2: in std_logic;
            out1: out std_logic);
   end component;
   component NOT1
      port (in1: in std_logic;
            out1: out std_logic);
   end component;
end OPERACiONES_LOGICAS;
```

Ejemplo de uso de paquete

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all;
LIBRARY work ;
USE work.OPERACIONES_LOGICAS.all;

entity ALARMA is
   port (
         PUERTA, ENCENDIDO, CINTO: in std_logic;
        SEÑAL: out std_logic);
   end ALARMA;
```

Ejemplo de uso de paquete

```
architecture estructural of ALARMA is
   -- declaracion de señales para interconectar compuertas
   signal PUERTA_NOT, CINTO_NOT, B1, B2: std_logic;
   begin
   -- Instanciación de componentes
      U0: NOT1 port map (
         in1 => PUERTA,
         out1 => PUERTA_NOT);
      U1: NOT1 port map (CINTO, CINTO_NOT);
      U2: AND2 port map (ENCENDIDO, PUERTA_NOT, B1);
      U3: AND2 port map (ENCENDIDO, CINTO_NOT, B2);
      U4: OR2  port map (B1, B2, SEÑAL);
   end structural;
```

Tipos de datos

- Se recomienda al estandar Std_logic (IEEE 1164) para descripciones Hardware:
  - Tiene hasta 9 posibles estados, lo que permite representar la mayoria de los estados encontrados en circuitos digitales.
  - Toman un valor desconocido como inicial. Esto fuerza a que uno deba inicializar el sistema.
  - No es necesario realizar conversiones de tipo.
  - En síntesis solo se utilizan ‘0’ , ‘1’ y ‘Z’ de los 9 posibles estados.

Signal vs Variables

Signal:

- tiene mapeo HW: cable, cable con memoria

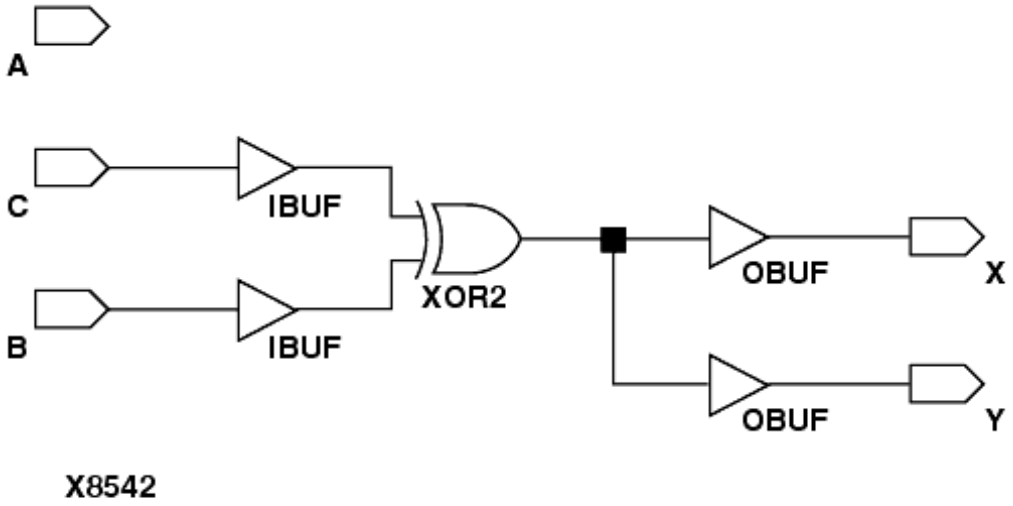
Variable:

- no tiene mapeo HW
- usada en procesos para descripción abstracta de funcionamiento.

Signals vs Variables

```
Library IEEE;
use IEEE.std_logic_1164.all;
entity xor_sig is
     port (A, B, C: in
     STD_LOGIC;
           X, Y: out STD_LOGIC);
end xor_sig;

architecture SIG_ARCH of
     xor_sig is
 signal D: STD_LOGIC;
begin
    SIG:process (A,B,C)
    begin
         D <= A; -- ignored !!
        X <= C xor D;
        D <= B; -- overrides !!
        Y <= C xor D;
    end process;
end SIG_ARCH;
```

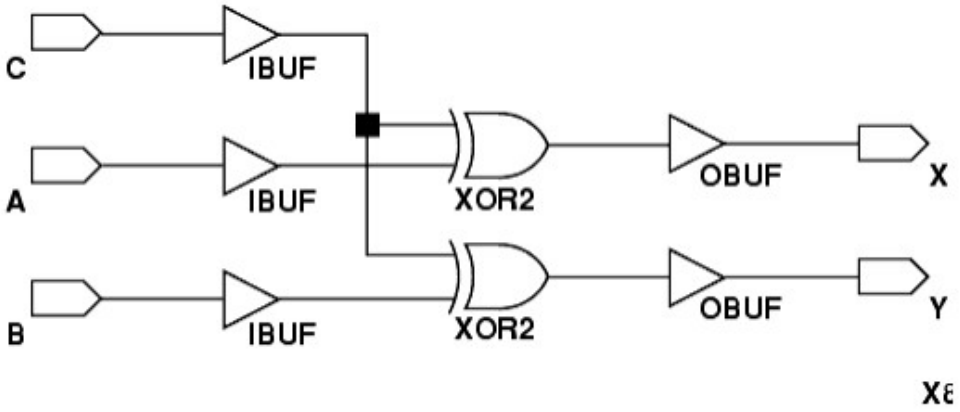


Signals vs Variables

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity xor_var is
 port (A, B, C: in  STD_LOGIC;
       X, Y:    out STD_LOGIC);
end xor_var;

architecture VAR_ARCH of
     xor_var is
begin

    VAR:process (A,B,C)
        variable D: STD_LOGIC;
     begin
       D := A;
        X <= C xor D;
        D := B;
        Y <= C xor D;
    end process;
end VAR_ARCH;
```



Numeric_Std package

B_x_InnA_adc = coefB_InnA * (InnA_adc + coefA_InnA)

```
coefB_InnA <= std_logic_vector(resize(unsigned(coef_InnA(13 downto 0)),
    coefB_InnA'length ));
coefA_InnA <= std_logic_vector(resize(signed(coef_InnA(31 downto 14)),
    coefA_InnA'length ));

InnA_adc_extended <= std_logic_vector(resize(unsigned(InnA_adc),
    InnA_adc_extended'length ));

InnA_adc_lessoffset <= std_logic_vector(signed(InnA_adc_extended) +
    signed(coefA_InnA));

BxInnA: mult2_s16xs16 port map(clk, InnA_adc_lessoffset(15 downto 0), coefB_InnA,
    B_x_InnA_adc);
```

Links y Referencias

- Ejemplo sencillo:
  - https://eva.fing.edu.uy/mod/page/view.php?id=41255
- Referencia rápida:
  - http://www.tcnj.edu/~hernande/r/VHDL_QRC__01.pdf
  - https://nandland.com/common-vhdl-conversions/
- Libros:
  - Chu, P.; "RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability", *Wiley, John & Sons, Incorporated,* **2006** , 694
  - Ashenden, P. J. & Lewis, J.; "The Designer's Guide to VHDL", 3rd. Ed., *Morgan Kaufmann Publishers,* **2008**