

Sistemas Electrónicos Digitales

Tema #4

Diseño mediante Lenguajes de Descripción Hardware

Parte 4.3b

- 5.1 Ventajas de los HDL.
- 5.2 Metodología de Diseño.
- **5.3 VHDL. Sintaxis de VHDL.**
- 5.4 Codificación de circuitos lógicos en VHDL.
- 5.5 Módulos IP.
- 5.6 Sistemas en un Chip (SoC).
- 5.7 Codiseño SW-HW.
- 5.8 SystemC

VHDL: descripción de flujo de datos (I)

- Asignación a señal:

[id:] nombre <= [**guarded**] [**transport** | [**reject** *tiempo*]
inertial] {*forma_de_onda*,}_1 | **unaffected**;

forma_de_onda = *valor* | *expresión* [**after** *tiempo*]

- Ej.:

a <= "11000101";

clk <= **not** clk **after** 50 ns;

reset_n <= '1', '0' **after** 100 ms;

VHDL: descripción de flujo datos (II)

- Asignación condicional a señal:
[id:] nombre **<=** [opciones]
 {forma_onda **when** condicion **else**}₁
 forma_de_onda [**when** condicion];
- Es obligatorio asignar algo a la señal.
- En las ff.oo. pueden usarse también:
 - La misma señal asignada (implica transacción).
 - **unaffected** (no implica transacción).

VHDL: descripción de flujo de datos (III)

- Ej. asignación condicional:

```
s <= '1' when a = b else  
      '0' when a > b else  
      s;
```

```
s <= '1' when a = b else  
      '0' when a > b else  
      unaffected;
```

Aceptada por todos los simuladores, pero no todas las herramientas de síntesis.

VHDL: descripción de flujo datos (IV)

- Asignación **with** / **select** a señal:
 with expresión **select**
 id <= [opciones]
 [{forma_onda **when** caso,}]₁
 forma_de_onda [**when** caso];
- Los casos no pueden solaparse y es obligatorio asignar algo a la señal.
- Como casos pueden usarse:
 - Valor único.
 - Valores múltiples: valor1 | valor2 | valor3
 - Rangos: 1 **to** 3, 5 **downto** 2.
 - **others** (cualquier otro caso).

VHDL: descripción de flujo de datos (V)

- Ej. asignación condicional:

with estado select

```
s <= '0' when "00",  
      '1' when "01" | "10",  
      'Z' when others;
```

with cuenta select

```
s <= '0' when 0 to 7,  
      '1' when 8 to 15;
```

VHDL: descripción de flujo datos (VI)

- Bloque concurrente. Permite agrupar sentencias de ejecución concurrente para hacer diseños modulares:

id_bloque: **block** [(expresión_de_vigilancia)] [**is**]

[cabecera]

[declaraciones]

begin

{sentencia_concurrente}

end block [id_bloque];

- Cabecera: genéricos, entradas/salidas.
- Declaraciones: tipos, ctes., señales, atributos, subprogramas...

VHDL: descripción de flujo de datos (VII)

- Ej. bloque concurrente:

rom: **block**

port(enable: **in** std_logic;

abus: **in** std_logic_vector(11 **downto** 0);

dbus: **out** std_logic_vector(7 **downto** 0));

port map (enable => sys_en, abus => sys_ab, dbus => sys_db);

signal storage **is array**(0 to 2**abus'high - 1) **of** byte;

begin

dbus <= storage(to_integer(abus)) **when** enable = '1' **else**
(**others** => 'Z');

end block;

VHDL: descripción de flujo de datos (VIII)

- La expresión de vigilancia se usa en conjunción con **guarded**.
- Las señales “guardadas” sólo se asignan si la expresión de vigilancia devuelve *true*.
- Las señales declaradas como de clase **bus** o **register** se “desconectan” si la expresión de vigilancia devuelve *false*.

VHDL: descripción de flujo de datos (IX)

- Ej. bloque concurrente con señal guardada:
latch: **block** (enable = '1')
begin
 q <= **guarded** d;
end block latch;
- Ej. bloque concurrente con desconexión de señal:
signal q : bit **register**;
tristate_buffer: **block** (oe = '1')
begin
 q <= **guarded** d;
end;