Sistemas Electrónicos Digitales

Tema #**4**

Diseño mediante Lenguajes de Descripción Hardware

Parte 4.3c

Índice

- 4.1 Ventajas de los HDL.
- 4.2 Metodología de Diseño.
- 4.3 VHDL. Sintaxis de VHDL.
- 4.4 Codificación de circuitos lógicos en VHDL.
- 4.5 Módulos IP.
- 4.6 Codiseño SW-HW.
- 4.7 SystemC

VHDL: descripción funcional (I)

- Descripción de tipo algorítmica: secuencia de instrucciones.
- Se usan construcciones **process**.
- Estas construcciones son las únicas de ejecución secuencial en VHDL.
- Una misma arquitectura puede contener varias construcciones **process** que se ejecutarían de forma concurrente.

VHDL: asignación a variable

- Las variables sólo pueden declararse y usarse dentro de **process**.
- El operador para asignar un valor a una variable es diferente al que se usa para señales:

  nombre_variable **:=** valor;
- Ej.:

  variable count: natural;

  count **:=** 1;

VHDL: descripción funcional (II)

- Sentencia **process**:

  id_proc:
  [**postponed**] **process** [(lista_sensibilidad)] [**is**]
  [declaraciones]
  begin
  {sentencia_ejecución_secuencial}
  end [**postponed**] **process** [id_proc];
- Declaraciones: tipos, ctes., variables, señales, atributos, subprogramas…

VHDL: descripción funcional (III)

- El orden de ejecución de los procesos en una arquitectura que contiene varios de ellos no está definido.
- Se puede declarar un proceso **postponed** para garantizar que se ejecuta después de todos los otros procesos.
- Si se declaran varios procesos **postponed** el orden de ejecución entre ellos no está definido.
- Si un diseño depende del orden de ejecución probablemente no sea un buen diseño.

VHDL: descripción funcional (IV)

- Una sentencia **process** se ejecuta en un bucle infinito implícito.
- Todas las sentencias, salvo **wait**, se ejecutan instantáneamente (no corre el tiempo de simulación).
- Sólo se ejecuta si hay un cambio en alguna de las señales de su lista de sensibilidad.
- Si no tiene lista de sensibilidad es obligatorio usar sentencias **wait** en su interior.
- En caso contrario el proceso se ejecutaría de forma continua bloqueando el simulador.

VHDL: descripción funcional (V)

- En cada iteración de un proceso:
  - Las asignaciones a variables tienen efecto inmediato.
  - Las asignaciones a una señal sólo tienen efecto en la siguiente iteración del proceso.
  - Si se hacen varias asignaciones a una señal en la misma iteración, sólo tiene efecto la última.
  - Si no se hace ninguna asignación, las variables y señales conservan su valor.
- Esto garantiza la causalidad y coherencia de la descripción, pero ojo con efectos imprevistos.

VHDL: descripción funcional (VI)

```
architecture A of E is
    signal A, B, C, X, Y: integer;
begin
    P1: process(A, B, C)
    begin
        C <= A;
        X <= C + 2;
        C <= B;
        Y <= C + 2;
    end process;
end;
```

- Valores iniciales:
  - A = 3, B = 2, C = 0, X = ?, Y = ?
- Valores finales:
  - A = 3, B = 2, C = 2, X = 2, Y = 2

VHDL: descripción funcional (VII)

```
architecture A of E is
    signal A, B, X, Y: integer;
begin
    P1: process(A, B)
        variable C: integer;
    begin
        C := A;
        X <= C + 2;
        C := B;
        Y <= C + 2;
    end process;
end;
```

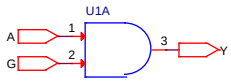
- Valores iniciales:
  - A = 3, B = 2, C = 0, X = ?, Y = ?
- Valores finales:
  - A = 3, B = 2, C = 2, X = 5, Y = 4

VHDL: descripción funcional (VIII)

- Tener en cuenta el comportamiento diferente de variables y señales.
- Si el diseño es combinacional y su lógica es compleja, es buena idea dar un valor a las salidas al principio del proceso para evitar la inferencia de memoria (latches).
- Incluir siempre todas las entradas asíncronas en la lista de sensibilidad del proceso.

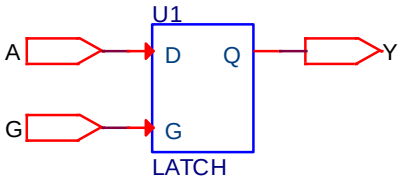
VHDL: descripción funcional (IX)

```
entity E1 is
    port (A, G: in std_logic;
          Y: out std_logic);
end;
architecture A1 of E1 is
begin
    U1A: process(A, G)
    begin
        Y <= '0';
        if G = '1' then
            Y <= A;
        end if;
    end process;
end;
```



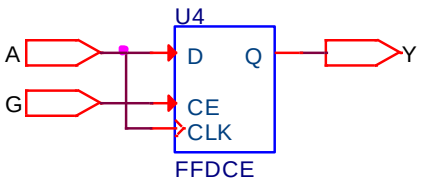
VHDL: descripción funcional (X)

```
entity E1 is
    port (A, G: in std_logic;
          Y: out std_logic);
end;
architecture A1 of E1 is
begin
    U1: process(A, G)
    begin
        if G = '1' then
            Y <= A;
        end if;
    end process;
end;
```



VHDL: descripción funcional (XI)

```
entity E1 is
    port (A, G: in std_logic;
          Y: out std_logic);
end;
architecture A1 of E1 is
begin
    U1: process(A)
    begin
        if G = '1' then
            Y <= A;
        end if;
    end process;
end;
```



Todos los simuladores entienden lo mismo, pero algunas herramientas de síntesis ignoran nuestra lista y suponen que todas las entradas están en ella.

VHDL: descripción funcional (XII)

- Sentencia **wait**: permite esperar hasta que se produce un suceso o transcurre cierto tiempo:

  [id:] **wait** [**on** {signal,}$_1$] [**until** condición] [**for** tiempo];

- Sin argumentos detiene la ejecución para siempre.
- Espera hasta que:
  - hay algún cambio en las señales de la cláusula **on**, o
  - se cumple la condición de la cláusula **until**, o
  - pasa el tiempo de la cláusula **for**.
- Todas las señales usadas en la cláusula **until** se añaden a la lista de sensibilidad de una cláusula **on** implícita.

VHDL: descripción funcional (XIII)

- Ej. sentencia **wait**:

  wait on pulse;
  wait until count > 7;
  wait for 1 us;
  wait on interrupt **for** 25 us;
  wait on clk, sensor **until** count = 3 **for** 100 ns;

VHDL: descripción funcional (XIV)

- Sentencia **if then else**:

  [id:] **if** condición **then**
  {sentencia_ejecucion_secuencial}$_1$
  [**elsif** condición **then**
  {sentencia_ejecucion_secuencial}$_1$]
  [**else**
  {sentencia_ejecucion_secuencial}$_1$]
  end if [id];

VHDL: descripción funcional (XV)

- Sentencia **case**:

  [id:] **case** expresión **is**
  {**when** caso =>
  {sentencia_ejecucion_secuencial}$_1$]}$_1$
  [**when others** =>
  {sentencia_ejecucion_secuencial}$_1$]
  end case [id];

VHDL: descripción funcional (XVI)

- Los casos no pueden solaparse y es obligatorio cubrir todas las posibilidades.
- Como casos pueden usarse:
  - Valor único.
  - Valores múltiples: valor1 | valor2 | valor3
  - Rangos: 1 **to** 3, 5 **downto** 2.
  - **others** (cualquier otro caso).
- Se suele utilizar siempre **others** al final.

VHDL: descripción funcional (XVII)

- Bucles:

  [id:] tipo_de_bucle **loop**

  $\{sentencia_ejecucion_secuencial\}_1$

  end loop [id];

  tipo_de_bucle:
  - Bucle tipo *for*: **for** nombre **in** rango
  - Bucle tipo *while*: **while** condición
  - Si no se indica, bucle infinito.
- Pueden ser sintetizables, pero no siempre.

VHDL: descripción funcional (XVIII)

- Ej. bucles:

```
for count in 5 downto 0 loop
    table(count) <= count * 2;
end loop;

count := 5;
while count >= 0 loop
    table(count) <= count * 2;
    count := count – 1;
end loop;
```

VHDL: descripción funcional (XIX)

- Es posible finalizar la iteración actual:
  [id:] **next** [id_bucle] [**when** condición];
- o finalizar el bucle completo:
  [id:] **exit** [id_bucle] [**when** condición];
- Puede añadirse una condición para la interrupción e indicarse el bucle afectado.
- Si no se indica bucle se entiende que es el más interno.

VHDL: descripción funcional (XX)

- Ej. interrupción iteración y bucle:

```
exterior: while a < 10 loop
    interior: for i in 1 to 10 loop
        …
        next exterior when i = a;
        …
    end loop interior;
    exit when hecho;
end loop exterior;
```

Finaliza el bucle *interior* y comienza la siguiente iteración del bucle *exterior*.

Finaliza el bucle *exterior*.

VHDL: descripción funcional (XXI)

- En VHDL el código secuencial puede estructurarse en subprogramas.
- Estos subprogramas se pueden usar también en construcciones concurrentes.
- En tal caso se comportan como **process** con una lista de sensibilidad igual a sus parametros de entrada y entrada/salida.
- Si no tienen parámetros de estos tipos deberían contener **wait**.

VHDL: descripción funcional (XXII)

- Tipos y características de los subprogramas en VHDL:

	function	procedure
Devuelve resultado	Siempre	Nunca
Parámetros	Sólo de entrada	Entrada/salida
Efectos colaterales	No	Sí
Llamada	Debe formar parte de una expresión	Puede ir sola en una sentencia
Uso de **wait**	No permitido	Permitido

VHDL: descripción funcional (XXIII)

- Declaración de un procedimiento:

  procedure nombre [({parametros;}$_1$)] **is**
  [declaraciones]
  begin
  {sentencias_ejecución_secuencial}$_1$
  end [procedure] [nombre];

- Parámetros: por defecto **variable in**

  [**constant**|**variable**|**signal**] {nombre,}$_1$: [**in**|**inout**|**out**] tipo |
  file {nombre,}$_1$: tipo

- Declaraciones: tipos, variables, subprogramas.

VHDL: descripción funcional (XXIV)

- Ej. procedimiento

```
procedure limits(constant data: integer_vector;
  min, max: out integer) is
    variable i: integer;
begin
    min := data(data'left);
    max := data(data'right);
    for i in data'range loop
        if min > data(i) then min := data(i); end if;
        if max < data(i) then max := data(i); end if;
    end loop;
end limits;
```

VHDL: descripción funcional (XXV)

- Declaración de una función: por defecto **pure**

  [**pure**|**impure**]
  function nombre [({parametros;}$_1$)] **return** tipo **is**
    [declaraciones]
  begin
    {sentencias_ejecución_secuencial}$_1$
  end;

- Parámetros: por defecto **constant in**

  [**constant**|**variable**|**signal**] {nombre,}$_1$: [**in**] tipo
  file {nombre,}$_1$: tipo

- Declaraciones: tipos, variables, subprogramas.

VHDL: descripción funcional (XXVI)

- Ej. función

```
function max(constant data: integer_vector) return integer is
    variable i: integer;
    variable result: integer;
begin
    result := integer'low;
    for i in data'range loop
        if result < data(i) then result := data(i); end if;
    end loop;
    return result;
end max;
```

VHDL: descripción funcional (XXVII)

- Ej. Función pura / impura
- Función pura: el resultado sólo depende de los parámetros que se pasan a la función:

```
pure function sin(x: real) return real;
```

- Función impura: el resultado no sólo depende de los parámetros que se pasan a la función:

```
impure function now return delay_length;
```

Valor que mantiene el simulador con el tiempo de simulación transcurrido.

VHDL: descripción funcional (XXVIII)

- Llamada a procedimiento:

  procedure read_block(length, offset: positive);
  - Por posición: read_block(64, 0);
  - Por nombre: read_block(offset => 0, length => 64);
  - No son necesarios los paréntesis si no hay parámetros.
- Parámetros por defecto:

  procedure read_block(length: positive := 256);

  read_block(256) ≡ read_block;
- Declaración adelantada (*forward*): necesaria si se llama a los subprogramas antes de definirlos:

  procedure nombre[({parametros,}$_1$)];

  [**pure** | **impure**] **function** nombre[({parametros,}$_1$)] **return** tipo;

VHDL: descripción funcional (XXIX)

- Sobrecarga de funciones y operadores:
  - Se puede sobrecargar cualquier función.
  - Pero la sobrecarga más útil es la de operadores.
  - El compilador elige la sobrecarga a utilizar por el tipo y número de los parámetros.

```
function "+"(a, b: complex) return complex is
  variable temp: complex;
begin
  temp.RE := a.RE + b.RE;
  temp.IM := a.IM + b.IM;
  return temp;
end "+";
```

Sobrecarga del operador "+" para sumar complejos

Uso del operador "+" original para sumar números reales