De momento se resuelven los impares

Capítulo 3

1. ¿ Cuáles de los siguientes identificadores son válidos? ¿Por qué?

entrada_1 ⇒ CORRECTO: comienza con letra, finaliza con número o letra, no contiene dos «_» seguidos ni tampoco ningún carácter no permitido
Entr_sal ⇒ CORRECTO
Entrada__2 ⇒ CORRECTO
Modulo 1 ⇒ INCORRECTO: no se permite utilizar espacios en blanco
Sal_ ⇒ INCORRECTO: no es possible finalizar con el carácter «_»
sal@2 ⇒ INCORRECTO: no es válido el carácter «@»
1EN ⇒ INCORRECTO: no es posible que comenzar con un número
sal2 ⇒ INCORRECTO: no es possible comenzar con el carácter «»
Valor_2_3 ⇒ CORRECTO
dir_4_pos_7 ⇒ CORRECTO

Los que figuran como correctos lo son porque cumplen los requisitos sintácticos de los identificadores simples:

– El primer carácter ha de ser necesariamente una letra (a÷z, A÷Z) del alfabeto inglés, mayúscula o minúscula (el lenguaje no hace distinción entre ambas, se dice que es *case insensitive*).

– El último carácter no puede ser «_».

– No pueden utilizarse dos caracteres «_» seguidos.

– No se permite la utilización de espacios en blanco

3. Indicar qué tipo de literal es cada uno y su equivalencia en lenguaje convencional (donde fuera preciso).

4.5E–7 ⇒ numérico, real (o en coma flotante): $4,5x10^{-7}$
230_000 ⇒ numérico, entero: 230.000
2#1011# ⇒ numérico, entero representado en binario: 1011_2 o 11_{10}
8#67# ⇒ numérico, entero representado en octal: 67_8 o 55_{10}
7#46_52 ⇒ numérico, entero en base 7: 4652_7 o 1703_{10}
16#A0#E1 ⇒ numérico, entero en hexadecimal: $A0_{16}x16^1$ o 2.560_{10}
2#1110_0110#E–5⇒ numérico, real en binario: 11100110.2^{-5} o $7,1875_{10}$
–5.3_16E3 ⇒ numérico, real: -5316
"B" ⇒ *string*: B
" " ⇒ *string*: espacio en blanco
' ' ⇒ *character*: espacio en blanco
"ESTA ES UNA ENTIDAD" ⇒ *string*: ESTA ES UNA ENTIDAD
"ESTA ES UNA ARQUITECTURA" & "Y ESTO ES UN PAQUETE" ⇒ *string*: ESTA ES UNA ENTIDAD Y ESTO ES UN PAQUETE
B"1110_0110" ⇒ *bit string literal* o literal de cadena de *bits*: 11100110
X"B7A" ⇒ *bit string literal* expresado en hexadecimal: 101101111010
O"650" ⇒ *bit string literal* expresado en octal: 011001010000

5. ¿Cuáles serán los valores de A y B una vez ejecutadas las dos sentencias que siguen a las siguientes declaraciones?

```
-- declaraciones:
subtype valor is integer range 0 to 420;
variable A: valor;
variable B: valor:= 25;
constant C: integer := 4;
```

```
-- sentencias:
A:= (B–4) * 4 + (B rem 4);
B:= B mod 4;
```

La ejecución es secuencial por cuanto A y B son variables, luego en primer lugar A recibe el valor 82 y a continuación B pasa a valer 1.

7. Detectar posibles errores en las siguientes asignaciones, justificándolo, a partir de las siguientes declaraciones de objetos:

```
constant A: time:= 7.5 sec;
variable B, C: time;
variable R, S: bit(2 downto 0);
variable T: bit_vector(3 downto 0);

C:= A/3;
C:= 3/A;          ⇒ ERROR: no es posible un tipo físico únicamente en el
                    denominador.
C:= A + 1;        ⇒ ERROR: no existe un operador de suma predefinido
                    que acepte como operandos un tipo físico y un entero.
                    Habría que definir una función que los admitiese. Las
                    funciones se explican el capítulo 6
C:= B/A * C;
T(0 to 2):= R;    ⇒ ERROR: no es posible el cambio en la dirección del
                    rango de la variable T. Dicho cambio solo se podría
                    llevar a cabo declarando un alias. Los alias se
                    estudian en el capítulo 6.
T:= R & 1;        ⇒ ERROR: el operador de concatenación sólo admite
                    operandos de tipo array y/o elementos del mismo tipo
                    para dar lugar a un array como resultado.
S:= S and not R;
S:= R & S;        ⇒ ERROR: el resultado de la operación tiene 6 bits,
                    mientras que la variable que recibe la asignación solo
                    es de 3 bits.
```

9. Detectar, justificándolo, posibles fallos en las siguientes declaraciones de tipos y subtipos de datos:

```
type estatura is range 0 to 3;
subtype altos is estatura range 1.80 to 2;   ⇒ ERROR: «estatura» es de tipo
                                                entero, luego no es admissible
                                                un literal real como límite de
                                                rango del subtipo «altos».
type VECTOR1 is array (altos) of bit;        ⇒ ERROR: el subtipo «altos» está
                                                mal declarado; por tanto no se
                                                puede utilizar como índice de
                                                un array.
```

11. En cuanto a los literales, ¿qué afirmación es incorrecta?

– Los numéricos permiten la notación exponencial.
– Los caracteres individuales se escriben entre comillas simples.
– 2#10110# es un literal de tipo *bit_vector*. ⇒ INCORRECTA: se trata de un literal numérico, por tanto, no puede ser un *bit_vector*.
– Al definir un tipo enumerado pueden crearse nuevos literales.
– El literal X"CF" es una cadena de *bits* en formato hexadecimal.

13. Indicar por qué cada uno de los siguientes *arrays* está correcta o incorrectamente declarado:

type A **is array** (0 **to** 5) **of** *bit_vector*; ⇒ INCORRECTO: es preciso indicar el rango del tipo **array** *bit_vector.*

type B **is array** (*positive* **range** 6 **downto** 1) **of** *bit*; ⇒ CORRECTO

subtype C **is** correcto (3 **downto** 1); ⇒ CORRECTO, siempre y cuando se haya definido previamente el tipo «correcto».

type D **is array** (**range** 0 **to** 8) **of** *bit*; ⇒ INCORRECTO: sobra la palabra reservada **range**.

15. Calcular y justificar el resultado de las operaciones:

(–12) **rem** 7; ⇒ -5, dado que ese el resto de la división y el resultado, en el caso de este operador, conserva el signo del numerador

12 **rem** (–7); ⇒ 5, por razones análogas

(12) **rem** (–7); ⇒ 5, ídem

12 **mod** 7; ⇒ 5, ídem

(–12) **mod** 7; ⇒ 5, ya que es el resultado de (-12)-7*N, con N=-2, dado que este es el menor (en valor absoluto) entero que satisface tanto $|a \textbf{ mod } b| < |b|$ como signo (a **mod** b) = signo b

12 **mod** (–7); ⇒ -2, ya que es el resultado de 12-(-7)*N, con N=-2 también

(12) **mod** (–7); ⇒ idéntico al anterior, los paréntesis aquí son irrelevantes

–12 **mod** 7; ⇒ -5, dado que 5 es el resultado de 12 **mod** 7 y –12 **mod** 7 = -(12 **mod** 7) por las reglas de prioridad de este operador

¿Se puede prescindir de los paréntesis en todos los casos?

Según el estándar, los operadores de multiplicación tienen superior prioridad a los de signo, por tanto, solamente resultan obligatorios los paréntesis en (-12) **rem** 7 y en (-12) **mod** 7.

Capítulo 4

1. Indicar las afirmaciones correctas sobre instanciación de componentes:

– La transmisión de valores (parámetros locales) a los puertos (parámetros formales) de un componente en la sentencia **port map** sólo puede hacerse mediante señales. ⇒ FALSO: puede hacerse tanto mediante señales como constantes, literales y expresiones, con la única restricción de la compatibilidad entre tipos

– Si no se definen valores en **generic map**, los parámetros toman los valores especificados en la declaración de la entidad. ⇒ CORRECTO

– Los parámetros en la instancia a un componente sólo puede indicarse mediante asociación posicional. ⇒ FALSO: puede utilizarse tanto la asociación posicional (implícita) como la nominal (explícita)

– En una instanciación de componente la etiqueta es opcional. ⇒ FALSO: es obligatoria

3. Escribir la sintaxis correcta de **assert-report** para dar lugar a los mensajes y niveles de error siguientes:

– si S = 1111: mensaje «demasiado grande» nivel *FAILURE*

– si S = 0011: mensaje «puede ser excesivo» nivel *WARNING*

– si S < 0011: mensaje «valor correcto» nivel *NOTE*

en una arquitectura donde se efectúa la operación aritmética S = IN1+IN2.

```
architecture X of XX is
    -- supongamos que S, IN1 e IN2 son puertos, que han sido delarados por tanto en la entidad
begin
    S <= IN1+IN2;
    assert (S /= 1111)
    report "demasiado grande"
    severity FAILURE;
    assert (S /= 0011)
    report "puede ser excesivo"
    severity WARNING;
    assert (S >= 1111)
    report "valor correcto"
    severity NOTE;
end;
```

5. Dado el bloque combinacional de la figura y su tabla de verdad:

– Escribir la declaración de entidad de este circuito

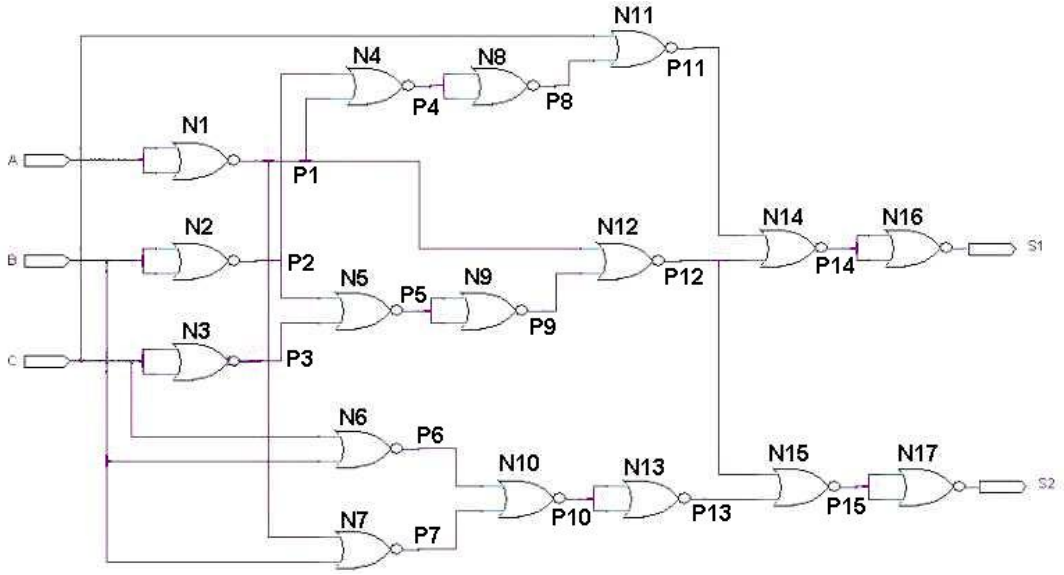
– Escribir una posible arquitectura en el nivel de descripción estructural utilizando exclusivamente componentes «NOR2»:

```
component NOR2
    port (a, b: in bit; s: out bit);
end NOR2;
```

```
entity COMBINACIONAL is
    port ( A, B, C: in bit;
           S1, S2: out bit );
end;
```

A	B	C	S1	S2
0	0	0	0	1
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	0	0

Una vez obtenidas las ecuaciones lógicas simplificadas y convertidas a sumas negadas de dos entradas, el circuito resultante es el siguiente:



Dada la regularidad del circuito, en cuanto a que sólo se emplea un tipo de puerta, es adecuada la utilización de un blucle con la sentencia “generate” (véase apartado 6.10) con lo que se evitan 17 sentencias de instanciación.

```
entity COMBINACIONAL is
    port (A, B, C: in bit; S1, S2: out bit);
end COMBINACIONAL;
architecture ESTUCTURAL_CON_GENERATE of COMBINACIONAL is
    -- nets internas:
    signal P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14,P15,P16,P17,P18,P19:bit;
    type MATRIZ is array(1 to 17) of bit_vector(1 to 3);
    signal N : MATRIZ;     -- terminales de las 17 puertas NOR
begin
    N(1) <= (A, A, P1);            N(2) <= (B, B, P2);
    N(3) <= (C, C, P3);            N(4) <= (P2, P1, P4);
    N(5) <= (P2, P3, P5);          N(6) <= (C, B, P6);
```

```
    N(7) <= (P1, B, P7);          N(8) <= (P4, P4, P8);
    N(9) <= (P5, P5, P9);         N(10)<= (P6, P7, P10);
    N(11)<= (A, P8, P11);         N(12)<= (P1, P9, P12);
    N(13)<= (P10, P10, P13);      N(14)<= (P11, P12, P14);
    N(15)<= (P12, P13, P15);      N(16)<= (P14, P14, P18);
    N(17)<= (P15, P15, P19);
    PUERTAS : for I in 1 to 17 generate
        P_NOR : entity WORK.PNOR port map (N(I)(1), N(I)(2), N(I)(3));
    end generate PUERTAS;
    S1 <= P18;
    S2 <= P19;
end ESTUCTURAL_CON_GENERATE;
```

Se da por existente una entidad PNOR compilada en la biblioteca WORK, tal como:

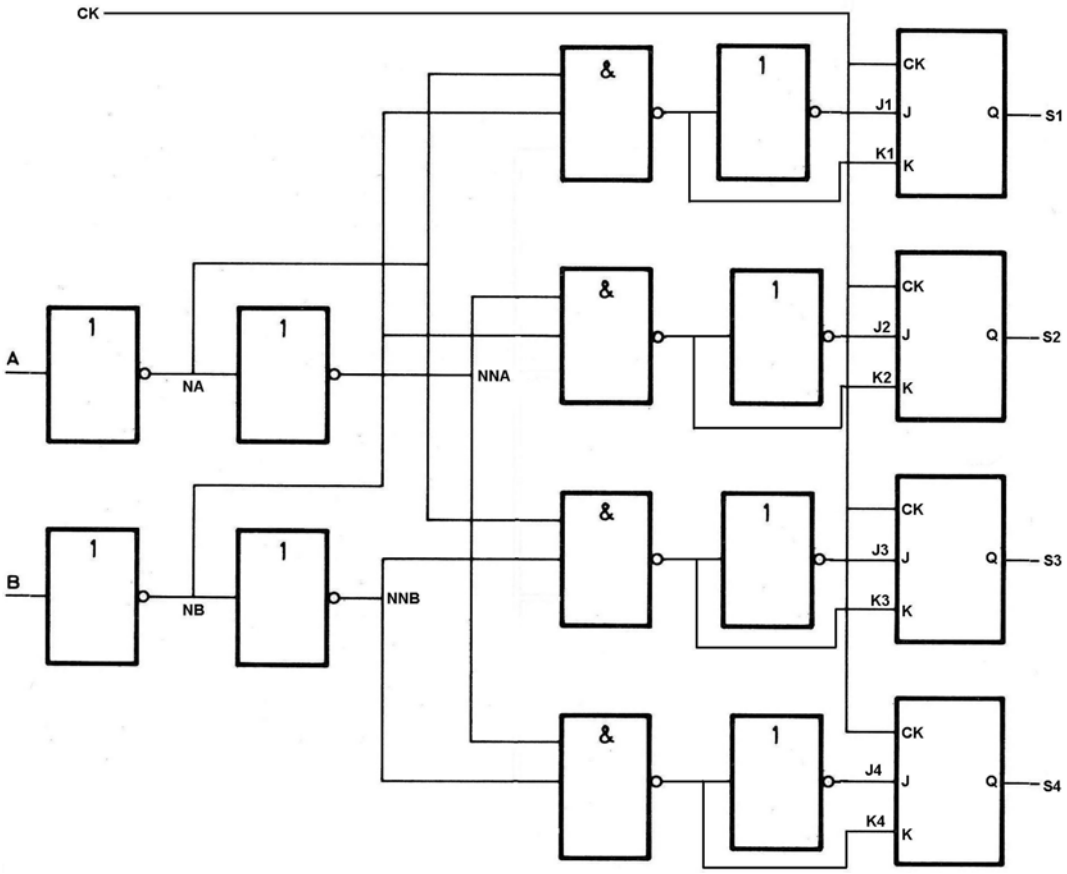
```
entity PNOR is
    port (A, B: in bit; S: out bit);
end PNOR;
architecture RTL of PNOR is
begin
    S <= A nor B;
end RTL;
```

7. Describir las entidades y arquitecturas correspondientes a una puerta inversora y a una puerta NAND de 2 entradas.

```
entity P_NOT is
    port (E: in bit; S: out bit) ;
end P_NOT;
architecture RTL of P_NOT is
begin
    S <= not E;
end RTL;
--------------------
entity P_NAND2 is
    port (E1, E2: in bit;  S: out bit) ;
end P_NAND2;
architecture RTL of P_NAND2 is
begin
    S <= E1 nand E2;
end RTL;
```

9. Modelar un decodificador 1 entre 4 a nivel estructural, con memorización de las señales de salida, instanciando los componentes definidos en los dos ejercicios anteriores.

Es necesaria la utilización de biestables para la memorización. En el ejercicio 8 se modeló un biestable J-K. El esquemático del circuito puede ser el siguiente:




```
entity DECODIFICADOR is
    port (A, B, CK: in bit; S1, S2, S3, S4: out bit);
end DECODIFICADOR;
architecture ESTUCTURAL_CON_GENERATE of DECODIFICADOR is
    -- nets internas:
    signal NA, NB, NNA, NNB, J1, J2, J3, J4, K1, K2, K3, K4, Q1, Q2, Q3, Q4: bit;
    type MATRIZ_1 is array(1 to 12) of bit_vector(1 to 2);
    type MATRIZ_2 is array(1 to 4) of bit_vector(1 to 3);
    type MATRIZ_3 is array(1 to 4) of bit_vector(1 to 4);
    signal N_1 : MATRIZ_1;  -- terminales de las 12 puertas inversoras
    signal N_2 : MATRIZ_2;  -- terminales de las 4 puertas NAND
    signal JK : MATRIZ_3;   -- terminales de los 4 biestables J-K
begin
    N_1(1) <= (A, NA);          N_1(2) <= (B, NB);
    N_1(3) <= (NA, NNA);        N_1(4) <= (NB, NNB);
    N_1(5) <= (K1, J1);         N_1(6) <= (K2, J2);
    N_1(7) <= (K3, J3);         N_1(8) <= (K4, J4);
    N_2(1) <= (NA, NB, K1);     N_2(2) <= (NNA, NB, K2);
    N_2(3) <= (NA, NNB, K3);    N_2(4) <= (NNA, NNB, K4);
    JK(1) <= (CK, J1, K1, Q1);  JK(2)<= (CK, J2, K2, Q2);
    JK(3) <= (CK, J3, K3, Q3);  JK(4)<= (CK, J4, K4, Q4);
    INVERSORES : for I in 1 to 8 generate
        ET_1 : entity WORK.INV port map (N_1(I)(1), N_1(I)(2));
    end generate INVERSORES;
    PUERTAS_NAND : for I in 1 to 8 generate
        ET_2 : entity WORK.P_NAND port map (N_2(I)(1), N_2(I)(2));
    end generate PUERTAS_NAND;
    BIESTABLES : for I in 1 to 8 generate
        ET_3 : entity WORK.JK_RTL port map (JK(I)(1), JK(I)(2), JK(I)(3), JK(I)(4));
```

```
    end generate BIESTABLES;
    S1 <= Q1;
    S2 <= Q2;
    S3 <= Q3;
    S4 <= Q4;
end ESTUCTURAL_CON_GENERATE;
```

11. Partiendo del modelo de una puerta *AND*:

```
entity PUERTA_AND is
    port (A1, A2: in bit; ZN: out bit);
end PUERTA_AND;
architecture SENCILLA of PUERTA_AND is
begin
    ZN <= A1 and A2;
end SENCILLA;
```

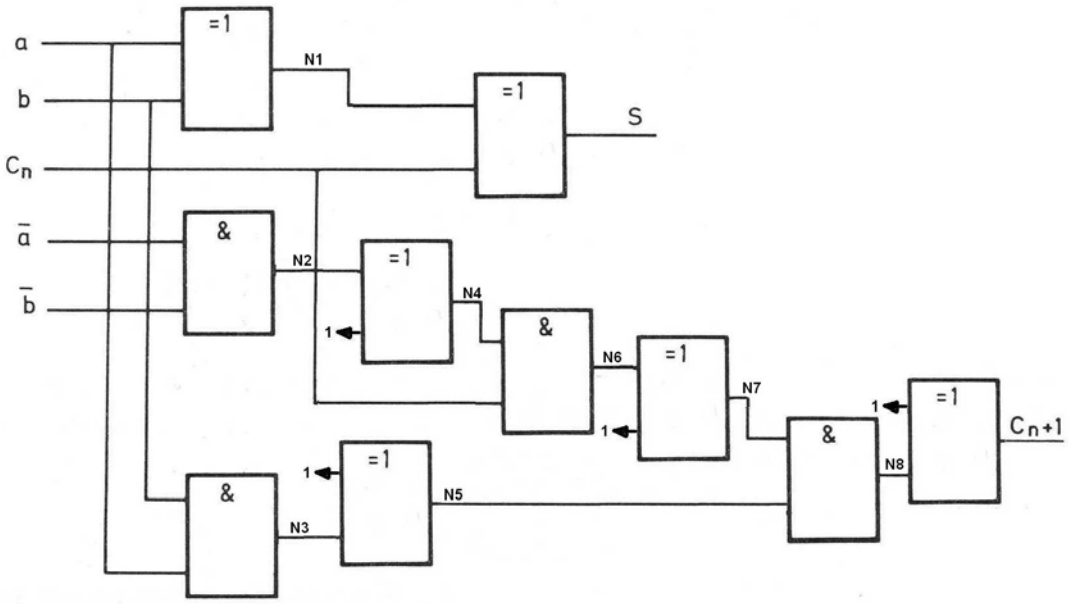
explicar lo que es erróneo y por qué en los siguientes modelos que hacen uso del anterior:

```
entity CIRCUITO_1 is              -- CIRCUITO_1
    port ( X: in bit; Y: out bit; Z: inout bit);
end CIRCUITO_1;
architecture ERRONEA of CIRCUITO_1 is
    component PUERTA_AND
        port (A1, A2: in bit; ZN: out bit);
    end component;
begin
    P1: PUERTA_AND port map (X, Y, Z);
end ERRONEA;
-----------------------------------------------
entity CIRCUITO_2 is              -- CIRCUITO_2
    port ( X: in bit; Y: out bit; Z: inout bit );
end CIRCUITO_2;
architecture ERRONEA of CIRCUITO_2 is
    component PUERTA_AND
        port (A1, A2: in bit; ZN: inout bit);
    end component;
begin
    P1: PUERTA_AND port map (X, Y, Z);
end ERRONEA;
```

- El puerto Y (parámetro local) de CIRCUITO_1 se ha declarado de modo out, por tanto no puede ser asociado al puerto A2 (parámetro formal) de PUERTA_AND, que es de modo in.
- Exactamente lo mismo ocurre en CIRCUITO_2.
- No es errónea sin embargo la asociación del “formal” ZN, de modo out, con el “local” Z, de modo inout, dado que dicha asociación está permitida por la norma (véase apartado 4.5).

13. Escribir el modelo de un sumador total de dos *bits* instanciando puertas *and* y puertas *xor*, ambas de dos entradas.

El esquemático a modelar es el siguiente :




```
entity SUMADOR is
    port (A, B, NA, NB, CN: in bit; S, \CN+1\: out bit);
end SUMADOR;
architecture ESTUCTURAL_CON_GENERATE of SUMADOR is
    -- nets internas:
    signal N1, N2, N3, N4, N5, N6, N7, N8, S_TMP, \CN+1_TMP\: bit;
    type MATRIZ is array(1 to 10) of bit_vector(1 to 3);
    signal NN: MATRIZ;    -- terminales de las 10 puertas
begin
    NN(1)     <= (A, B, N1);          NN(2)     <= (N1, CN, S_TMP);
    NN(3)     <= (N2, '1', N4);       NN(4)     <= ('1', N3, N5);
    NN(5)     <= (N6, '1', N7);       NN(6)     <= ('1', N8, \CN+1_TMP\);
    NN(7)     <= (NA, NB, N2);        NN(8)     <= (A, B, N3);
    NN(9)     <= (N4, CN, N6);        NN(10)    <= (N7, N5, N8);
    P_XOR : for I in 1 to 6 generate
        ET_1 : entity WORK.PXOR port map (NN(I)(1), NN(I)(2), NN(I)(3));
    end generate P_XOR;
    P_AND : for I in 7 to 10 generate
        ET_2 : entity WORK.PAND port map (NN(I)(1), NN(I)(2), NN(I)(3));
    end generate P_AND;
    S <= S_TMP;
    \CN+1\ <= \CN+1_TMP\;
end ESTUCTURAL_CON_GENERATE;
```

Capítulo 5

1. Utilizar la sentencia **wait** para que un proceso:

– Sea sincronizado por niveles.

– Sea sincronizado por flancos.

– Asigne un retardo de 10 ns.

– Se suspenda durante 40 ns si el operando de entrada es mayor que un valor predeterminado C.

– Sea activado por flancos descendentes con un retardo de 6 ns.

– Por niveles:

```
process
begin
    …
    wait until C='1';
    ...
end process;
```

– Por flancos:

```
process
begin
    …
    wait on C;
    ...
end process;
o bien :
process (C)
begin
    …
end process;
```

– Asigne un retardo de 10 ns:

```
process (…)
begin
    …
    C <= <valor> after 10ns;
    ...
end process;
o bien :
process
begin
    …
    wait for 10 ns;
    C <= <valor>;
    ...
end process;

process
begin
    …
    wait for 10 ns;
    C <= ….
    ...
```

```
end process;
```

– Se suspenda durante 40 ns si el operando de entrada es mayor que un valor predeterminado C.

```
process
begin
    …
    if OPERANDO > C then
        wait for 10 ns;
    end if;
    ...
end process;
```

– Sea activado por flancos descendentes con un retardo de 6 ns.

```
process
begin
    …
    wait until C='0' and C'event;
    wait for 6 ns;
    ...
end process;
```

3. Determinar las diferencias implícitas, si las hay, en el uso de las siguientes descripciones:

```
--------------------
A<= B and C;
--------------------
process (B, C)
begin
    A<= B and C;
end process;
--------------------
process
begin
    wait on B, C;
    A<= B and C;
end process;
--------------------
process
begin
    A<= B and C;
    wait on B, C;
end process;
```

No existe diferencia alguna entre ellas. La asignación a la señal A tiene lugar siempre que ocurra un evento en B o C, y luego de un ciclo delta, en cualquiera de los cuatro casos.

5. ¿Son equivalentes las sentencias concurrentes de asignación a señal when-else y with-select?

No. La primera es una sentencia de asignación condicional y la última selectiva. Ello no implica que no puedan utilizarse ambas para modelar el comportamiento del mismo circuito, como ocurre con los ejemplos de los apartados 5.1.3 y 5.1.4.

7. ¿Qué ocurre cuando la condición de selección en una sentencia **case-when** es de tipo discreto pero con un número ilimitado de elementos?

En este caso se hace obligatorio indicar

when others => <sentencias secuenciales>

porque la sentencia obliga a especificar todos los posibles valores de la expresión que se evalúa tras la palabra reservada case.

9. Modelar un registro de 4 *bits*, que carga datos o los desplaza a la derecha bajo el control de una señal P_S, mediante un bloque.

```
entity REG_DESP is
    port (E: bit_vector(3 downto 0);
          CLK, P_S: bit;
          S: bit_vector(3 downto 0);
end REG_DESP;
architecture COMP of REG_DESP is
    signal AUX: bit;
begin
    R_G: block (CLK='1' and CLK'event)
    begin
        AUX <= guarded E when P_S='0' else
                AUX srl 1;
    end block R_G;
    S <= AUX;
end COMP;
¿por qué no funciona srl?
¿   "     "       "     guarded?
```

11. Modelar el sumador anterior con sentencias secuenciales. Añadir una entrada asíncrona de puesta a cero mediante una sentencia **wait**.

```
entity SUMADOR_2_BITS is
    port ( A, B: bit_vector(1 downto 0);
           P_CERO: bit;
           SUMA: out bit_vector(2 downto 0) );
end SUMADOR_2_BITS;
architecture SECUENCIAL of SUMADOR_2_BITS is
begin
    process
        variable AC_P: bit;
    begin
        case P_CERO is
            when '0' => SUMA <= "000";
            when '1' =>
                SUMA(0) <= A(0) xor B(0);
                AC_P := (A(0) and B(0));
                SUMA(1) <= A(1) xor B(1) xor AC_P;
                SUMA(2) <= (A(1) and B(1)) or (A(1) and AC_P) or
                           (B(1) and AC_P);
        end case;
        wait on P_CERO;
    end process;
end SECUENCIAL;
```

13. Supongamos que la siguiente sentencia forma parte de un modelo:

```
...
wait on a, b;
...
```

determinar las afirmaciones correctas respecto a la misma:

1. a y b pueden ser señales y variables, pero no constantes.
2. Esta sentencia puede usarse en procesos y bloques.
3. Puede sustituir a la lista de sensibilidad de un proceso.
4. Detiene la ejecución secuencial hasta que haya un evento en las señales a y b (en ambas simultáneamente).
5. Resulta equivalente a **wait until** (a'*event* **or** b'*event*);

1. INCORRECTO => Nunca pueden ser variables, sólo pueden ser señales
2. INCORRECTO => Es una sentencia secuencial. Por tanto puede usarse en procesos pero nunca en bloques
3. CORRECTO
4. INCORRECTO => No tiene porqué haber un evento simutáneo en ambas señales. Es sufiente con que tenga lugar en una sola de ellas
5. CORRECTO

15. Modelar un decodificador 1 entre 4 con memorización de las salidas, a nivel estructural, instanciando puertas básicas y biestables. Escribir los modelos de estos últimos en el propio fichero.

```
-- Puerta inversora
entity P_NOT is
    port (E: in bit; S: out bit);
end P_NOT;
architecture RTL of P_NOT is
begin
    S <= not E;
end RTL;
-- Puerta AND
entity P_AND is
    port (E1, E2: in bit; S: out bit);
end P_AND;
architecture RTL of P_AND is
begin
    S <= E1 and E2;
end RTL;
-- Biestable tipo D
entity B_D is
    port (D, CK: in bit; Q: out bit);
end B_D;
architecture RTL of B_D is
begin
    Q <= D when CK'event and CK='1' else
            unaffected;
end RTL;
-- Decodificador
entity DECOD_1_4 is
    port ( CK: in bit;
            ENT: bit_vector(1 downto 0);
            SALIDA: out bit_vector(3 downto 0) );
end DECOD_1_4;
architecture ESTRUCTURAL of DECOD_1_4 is
    signal N_ENT: bit_vector(1 downto 0);
    signal S: bit_vector(3 downto 0);
begin
```

```
    P_N_1: entity WORK.P_NOT port map(ENT(0), N_ENT(0));
    P_N_2: entity WORK.P_NOT port map(ENT(1), N_ENT(1));
    P_A_1: entity WORK.P_AND port map(N_ENT(0), N_ENT(1), S(0));
    P_A_2: entity WORK.P_AND port map(ENT(0), N_ENT(1), S(1));
    P_A_3: entity WORK.P_AND port map(N_ENT(0), ENT(1), S(2));
    P_A_4: entity WORK.P_AND port map(ENT(0), ENT(1), S(3));
    B_D_1: entity WORK.B_D port map(S(0), CK, SALIDA(0));
    B_D_2: entity WORK.B_D port map(S(1), CK, SALIDA(1));
    B_D_3: entity WORK.B_D port map(S(2), CK, SALIDA(2));
    B_D_4: entity WORK.B_D port map(S(3), CK, SALIDA(3));
end ESTRUCTURAL;
```

NOTA: En las sentencias de instanciación del modelo anterior se hizo uso de opción de instanciación directa, introducida en la revisión del lenguaje de 1993. La instanciación directa elimina la necesidad de declarar los componentes que van a ser instanciados y en su lugar ha de especificarse, antes del identificador del componente, la librería que contiene la pareja entidad-arquitectura a la que dicho componente hace referencia, así como la palabra reservada entity. Supone por tanto una reducción en el la cantidad de líneas del modelo.

17. Seleccionar las afirmaciones correctas:

– En la zona de declaraciones de un proceso pueden definirse localmente funciones y procedimientos.

⇒ CORRECTO

– Los componentes sólo pueden instanciarse en el cuerpo de una arquitectura.

⇒ CORRECTO. Pueden ser instanciados también en un bloque pero éste ha de formar parte inexorablemente de una arquitectura

– Los bloques ejecutan su código interno de modo secuencial y en paralelo con las demás sentencias incluidas en el cuerpo de una arquitectura.

⇒ INCORRECTO. Las sentencias internas de un bloque son necesariamente concurrentes

– Los bloques sólo pueden existir en el cuerpo de una arquitectura.

⇒ CORRECTO

19. Partiendo de las siguientes sentencias:

```
type hexadecimal is (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F);
signal puerto_serie: hexadecimal;
...
case puerto_serie is
    when 0 => salida<= entrada;
    when 1 to 6  => salida (1 to 6)<= (others => '0');
    when 3 | A | B =>
        if OE = '1' then
            Registro:= entrada;
            Salida<= (others => '0');
        end if;
    when others => null;
end case;
```

determinar las respuestas correctas si «puerto_serie» vale 3:

– Se realizan las asignaciones correspondientes a la tercera opción.

⇒ INCORRECTO

– Se produce un error porque las opciones tras when no son todas excluyentes entre sí.

⇒ CORRECTO. La sentencia case-when exige que se contemplen todos los valores del objeto o expresión tras case, pero tan sólo una vez. No debe de haber ambigüedades

– No se puede evaluar una señal en una sentencia case-when.

⇒ INCORRECTO
– El identificador «puerto_serie» no es válido.

⇒ INCORRECTO

21. Identificar las afirmaciones correctas sobre los objetos que pueden conformar la lista de eventos de un proceso:

– Variables y señales.

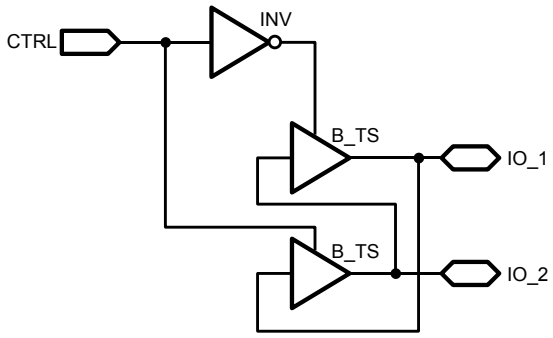
⇒ INCORRECTO
– Constantes.

⇒ INCORRECTO
– Un puerto de la entidad.

⇒ CORRECTO
– Una señal declarada en la arquitectura.

⇒ CORRECTO

23. Modelar adecuadamente el circuito de la figura a nivel RTL haciendo uso de la sentencia concurrente **when-else**. ¿Es posible declarar los puertos «IO_1» e «IO_2» del modo **buffer**?




```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity EJERCICIO_23 is
    port (CTRL: in std_logic; IO_1, IO_2: buffer std_logic);
end EJERCICIO_23;
architecture RTL of EJERCICIO_23 is
begin
    IO_1 <=  IO_2 when CTRL='0' else
             'Z';
    IO_2 <=  IO_1 when CTRL='1' else
             'Z';
end RTL;
```

Es posible en este caso la declaración de los puertos «IO_1» e «IO_2» de modo buffer, dado que sólo reciben una fuente de señal (la propia salida de las puertas de 3 estados).

Capítulo 6

1. Explicar las diferencias entre función y procedimiento en cuanto a su funcionalidad.

Ambos son recursos del lenguaje que ayudan a mejorar la descripción, estructuración y legibilidad de los modelos y, por tanto, a la posible reutilización del código. Son muy similares a los que se definen en los lenguajes de programación de alto nivel (HLL) de propósito general.
El objetivo principal de las funciones es realizar un cálculo puntual y devolver el resultado sin que avance el tiempo. Por tanto, se puede pensar en ellas como en una generalización de las expresiones o una forma de definir nuevos operadores.
Los procedimientos suelen estar más ligados a la naturaleza de lenguaje de descripción *hardware* de VHDL y pueden constituir pequeñas descripciones encargadas de realizar prácticamente cualquier tarea, aunque presenta una mayor utilidad en los bancos de prueba para el acceso a ficheros externos también y en el modelado de la "parte *software*" de un circuito / sistema, como pueden ser la instrucciones de un procesador.

3. Un procedimiento es una herramienta útil para la ejecución de un código que debe repetirse varias veces sobre valores diferentes. ¿Es posible declarar un procedimiento sin lista de parámetros de entrada? ¿Qué ocurre?

Sí es posible declarar un procedimiento sin parámetro alguno, de entrada o de salida o de entrada/salida. La norma no obliga a ello. Sería un procedimiento que no intercambia datos a través de sus parámetros pero sí puede tener acceso a los objetos que pertenezcan a su ámbito de alcance y visibilidad. Si se trata de señales, entonces el procedimiento debe de formar parte de un proceso para realizar asignaciones a las mismas, debido a los efectos colaterales, tal como se explica en el apartado 7.3 del texto.

5. Explicar porqué no es posible utilizar una sentencia **wait** dentro de una función. ¿Y de un procedimiento?

No es posible utilizar sentencias wait en funciones dado que éstas por definición devuelven los resultados sin retardo alguno (ni siquiera retardo delta), lo cual es incompatible con la propia naturaleza de wait.
En el caso de los procedimientos no existe la restricción anterior, por lo que pueden utilizar una o más sentencias wait, al igual que los procesos.

7. Definir una segunda función estadística que devuelva el valor máximo de un conjunto de valores enteros. Posteriormente definir un **alias** para ambas funciones, incluyendo la firma.

```
-- supongamos que la 1ª función es la siguiente:
function MAX (E1, E2, E3: integer) return natural is
begin
    return 0;
end MAX;
-- ahora sobrecargamos la función anterior
function MAX (E1, E2, E3: integer) return integer is
begin
    if E1>=E2 and E1>=E3 then return E1;
    elsif E2>=E1 and E2>=E3 then return E2;
    else return E3;
    end if;
end MAX;
-- declaramos sendos alias con firma para diferenciar ambas funciones
alias MAX_1 is MAX [integer, integer, integer return integer];
alias MAX_2 is MAX [integer, integer, integer return natural];
```

9. Si se ha declarado la variable S como compartida, ¿cuál será el resultado de ejecutar el siguiente código?:

```
process (A)
begin
    S:= X and Y;
end process;
process (A)
```



```
begin
    S:= X or Y;
end process;
```

Es un ejemplo paradigmático del no determinismo a que puede dar lugar la utilización de variables compartidas. En este caso la sentencia de asignación a S que prevalece puede ser tanto la primera como la segunda, en función de que proceso ejecute el simulador en último lugar.

11. Indicar los valores que toman las distintas variables declaradas:

```
type entero is range 2 to 20;
type conj1_bits is array (0 to 6) of bit;
type conj2_bits is array (-10 to 10) of bit;
type estado is (Inicial, Lectura, Escritura, Alta_impedancia);
--
variable V1: entero:= entero'val(2);
variable V2: integer:= conj1_bits'right;
variable V3: integer:= conj2_bits'high;
V3:= conj2_bits'length;
variable V4: integer:= estado'pos(Lectura);
variable V5: estado;
V5:= estado'succ(Inicial);
V5:= estado'leftof(Alta_impedancia);
```

V1 → 2
V2 → 6
V3 → 10
V3 → 21
V4 → 1
V5 → Inicial
V5 → Lectura
V5 → Escritura

13. Suponiendo las siguientes declaraciones de tipos y objetos:

```
type tabla is array (1 to 3, -2 to 5) of bit;
type vocales is ('a', 'e', 'i', 'o', 'u');
variable matriz: tabla;
signal reloj: bit;
```

¿qué afirmaciones son correctas?:

– matriz'*left* devuelve –2. ⇒ INCORRECTO, devuelve 1

– matriz'*high* devuelve 3. ⇒ CORRECTO

vocales'*pos*('u') devuelve 1. ⇒ INCORRECTO, devuelve 4

– reloj'*delayed*(5 ns) devuelve una señal idéntica a «reloj» retrasada 5 ns. ⇒ CORRECTO

– boolean'*right* devuelve *false*. ⇒ INCORRECTO, devuelve TRUE

15. Identificar la secuencia de valores asignados (forma de onda) y actualizados en la señal Y de tipo *bit*, dentro del siguiente proceso:

```
process
begin
    Y<= transport '1' after 10 ns;
    wait for 5 ns;
    Y<= transport '0' after 7 ns;
    wait for 8 ns;
```

```
    Y<= transport '1' after 10 ns;
    wait for 2 ns;
    Y<= transport '0' after 3 ns;
end process;
```

– ('0', 0 ns; '1', 10 ns; '0', 12 ns; '1', 23 ns)

– error de compilación por asignación múltiple a Y

– ('0', 0 ns; '1', 10 ns; '0', 12 ns)

– ('0', 0 ns; '1', 10 ns; '0', 12 ns; '1', 23 ns; '0', 28 ns)

La tercera respuesta es la correcta, dado que la secuencia indefinida es:
('0', 0 ns; '1', 10 ns; '0', 12 ns; '1', 25 ns, '0', 27 ns; …)

Capítulo 7

1. ¿Cuál es la primera entidad que se analiza cuando se compila un diseño?

En cuanto a entidades, deben de ser compiladas en orden jerárquico ascendente, de tal modo que cuando se haga referencia como componente a una pareja entidad-arquitectura (entidad mínima de diseño), ésta debe de estar ya compilada.

Por tanto, si todo un diseño jerárquico se especifica en un solo fichero fuente y se especifican al final las entidades de menor jerarquía, el compilador devuelve mensajes de error porque no encuentra las entidades a que hacen referencia las declaraciones de los componentes y la compilación no se lleva a cabo.

3. Nombrar todos los elementos que pueden ser declarados en un paquete

Tipos, subtipos, constantes, señales, ficheros, variables compartidas, **alias**,
subprogramas, componentes y atributos, especificación de atributos y desconexiones y sentencias **use**.
También es posible declarar grupos y plantillas de grupos, conceptos introducidos en la revisión de 1993, que no se explican en el texto.

5. ¿Existe alguna jerarquía en las diferentes bibliotecas de un sistema?

No entre ellas. Sí puede existir jerarquía en la contrucción de cada biblioteca dentro del sistema de archivos, pero ello no viene definido por la norma y por tanto depende de cada herramienta en particular.

7. Identificar las respuestas correctas:

- Todo diseño VHDL utiliza al menos una biblioteca. ⇒ CORRECTO. Al menos la biblioteca WORK

- La biblioteca IEEE es visible para todo diseño y no es preciso declararla. ⇒ INCORRECTO

- La palabra reservada **library** hace visibles los elementos de una biblioteca en el diseño actual. ⇒ CORRECTO

- El paquete *textio* define los recursos para el uso de ficheros de texto. ⇒ CORRECTO

9. Diseñar un paquete que declare cuatro componentes: «sumador_total», «puerta_O», «puerta_Y», «semisumador» y «puerta_XOR» y declare y defina dos funciones: una de conversión de los enteros 0 al 7 al tipo *bit_vector* y otra de conversión del resultado de la anterior al tipo *string*.

```
package EJERCICIO_9 is
    component SUMADOR_TOTAL
        generic (tamanho_operandos: integer:= 8);
        port (A, B: in bit_vector (tamanho_operandos-1 downto 0);
                SUMA: out bit_vector (tamanho_operandos downto 0));
```

```
    end component SUMADOR_TOTAL;
    component PUERTA_O
        port (E1, E2: in bit; S: out bit);
    end component PUERTA_O;
    component PUERTA_Y
        port (E1, E2: in bit; S: out bit);
    end component PUERTA_Y;
    component PUERTA_XOR
        port (E1, E2: in bit; S: out bit);
    end component PUERTA_XOR;
    component SEMISUMADOR
        port (A, B: in bit; S: out bit);
    end component SEMISUMADOR;
    function ENT_BV(A: integer; N_BITS: natural) return bit_vector;
    function BV_STRING (A: bit_vector) return string;
end package EJERCICIO_9;
--------------------------------------
package body EJERCICIO_9 is
    function ENT_BV(A: integer; N_BITS: natural) return bit_vector is
        variable RESULTADO: bit_vector(N_BITS-1 downto 0);
        variable SIGNO: bit := '0';
        variable VALOR_INTERMEDIO: integer := A;
    begin
        if (A < 0) then
            SIGNO := '1';
            VALOR_INTERMEDIO := -(A+1);
        end if;
        for I in 0 to N_BITS-1 loop
            if (VALOR_INTERMEDIO mod 2) = 0 then
                RESULTADO(I) := SIGNO;
            else
                RESULTADO(I) := not SIGNO;
            end if;
            VALOR_INTERMEDIO := VALOR_INTERMEDIO/2;
        end loop;
        if   ((VALOR_INTERMEDIO/=0) or
            (SIGNO/=RESULTADO(RESULTADO'left))) then
            report "RESULTADO TRUNCADO";
        end if;
        return RESULTADO;
    end function ENT_BV;
    function BV_STRING (A: bit_vector) return string is
        variable RESULTADO: string (A'length downto 1);
    begin
        for I in 0 to A'length-1 loop
            if A(I) = '0' then
                RESULTADO(I+1) := '0';
            else
                RESULTADO(I+1) := '1';
            end if;
        end loop;
        return RESULTADO;
    end function BV_STRING;
end package body EJERCICIO_9;
```

Capítulo 8

1. Modelar los siguientes circuitos a nivel RTL (en el siguiente capítulo se encuentran los respectivos modelos a nivel estructural, en unos casos, y algorítmico en otros, salvo el multiplicador), así como sus bancos de prueba para simulación:

- Biestable RS activo por nivel.

- Biestable JK activado por flancos.

- Memoria RAM de dimensión genérica.

- Memoria LIFO de dimensión genérica.

- Memoria FIFO de dimensión genérica.

- Contador síncrono de 8 bits.

- Contador BCD de 3 dígitos.

- Multiplicador binario de 4 bits.

```
-----------------------------------------------------
-- BIESTABLE RS ACTIVO POR NIVEL
-----------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity B_RS is
    port ( R, S, CLK: in std_ulogic;
           Q, NQ : out std_ulogic );
end B_RS;
architecture RTL of B_RS is
    signal TEMP: std_logic;
begin
    TEMP <= '0' when R='1' and S='0' else
            '1' when R='0' and S='1' else
            'X' when R='1' and S='1' else
            unaffected;
    Q <= TEMP when CLK='1' else
         unaffected;
    NQ <= not TEMP when CLK='1' else
          unaffected;
end RTL;
---------------------------------
-- BANCO DE PRUEBAS
---------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity RS_TEST is end;
architecture TEST of RS_TEST is
    signal R, S, CLK,   Q, NQ: std_ulogic;
begin
    CUT: entity WORK.B_RS port map (R, S, CLK,  Q, NQ);
    process
    begin
        CLK <= '0', '1' after 10 ns;
        wait for 20 ns;
    end process;
        R <= '1', '0' after 12 ns, '1' after 15 ns, '0' after 18 ns;
```

```
        S <= '0', '1' after 12 ns, '1' after 15 ns, '0' after 18 ns;
end TEST;
-------------------------------------------------------------
-- BIESTABLE JK ACTIVADO POR FLANCOS
-------------------------------------------------------------
entity B_JK is
    port (J,K, CLK: in bit;
            Q,NQ: inout bit);
end B_JK;
architecture RTL of B_JK is
begin
    process(CLK)
    begin
        if (CLK='1' and CLK'event) then
            if (J='1' and K='1') then Q<= not Q; NQ<= not NQ;
            elsif (J='1' and K='0') then Q<= '1'; NQ<= '0';
            elsif (J='0' and K='1') then Q<= '0'; NQ<= '1';
            end if;
        end if;
    end process;
end RTL;
---------------------------------
-- BANCO DE PRUEBAS
---------------------------------
entity JK_TEST is end;
architecture TEST of JK_TEST is
    signal J, K, CLK,    Q, NQ: bit;
begin
    CUT: entity WORK.B_JK port map (J, K, CLK, Q, NQ);
    process
    begin
        CLK <= '0', '1' after 2 ns;
        wait for 4 ns;
    end process;
    J <= '1' after 5 ns, '0' after 9 ns, '1' after 15 ns;
    K <= '0' after 5 ns, '1' after 9 ns, '1' after 15 ns;
end TEST;
-------------------------------------------------------------
-- MEMORIA RAM DE DIMENSIÓN GENÉRICA
-------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
entity RAM is
    generic(BUS_DIR, BUS_DATOS: natural:= 8);
    port ( DIR: in std_logic_vector(BUS_DIR-1 downto 0);
            DATOS: inout std_logic_vector(BUS_DATOS-1 downto 0);
            RD, WR: in std_logic ) ;
end RAM;
architecture RTL of RAM is
    type MEMO is array(0 to 2**BUS_DIR-1) of
        std_logic_vector(BUS_DATOS-1 downto 0);
    signal CONTENIDO: MEMO;
    constant T_ACCESO: time := 15 ns;
begin
    DATOS <= CONTENIDO(to_integer(unsigned(DIR))) after T_ACCESO
                    when RD='1' and WR='0' else
                (others => 'Z') after T_ACCESO
```

```
                    when RD='0' and WR='0' else
                unaffected;
    CONTENIDO(to_integer(unsigned(DIR)))<=DATOS after T_ACCESO
                    when RD='0' and WR='1' else
                unaffected;
end RTL;
--------------------------------
-- BANCO DE PRUEBAS
--------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity TEST_RAM is end TEST_RAM;
architecture TEST of TEST_RAM is
    signal DIR: std_logic_vector(3 downto 0):= X"0";
    signal DATOS: std_logic_vector(3 downto 0):= "ZZZZ";
    signal RD, WR: std_logic:= '0';
begin
    CUT: entity WORK.RAM generic map (4, 4)
            port map (DIR, DATOS, RD, WR);
    process
    begin
        wait for 40 ns;
        WR <= '1';
        DIR <= X"A";
        DATOS <= X"F";
        wait for 40 ns;
        DATOS <= "ZZZZ";
        WR <= '0';
        RD <= '1';
        wait for 40 ns;
        RD <= '0';
        wait;
    end process;
end TEST;
-------------------------------------------------------------
- MEMORIA LIFO DE DIMENSIÓN GENÉRICA
-------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity LIFO is
    generic(ANCHO_PALABRA, N_POS: natural:= 8);
    port(ENTRADA: in std_logic_vector(ANCHO_PALABRA-1 downto 0);
            CLK, RD, WR: in bit;
            SALIDA: out std_logic_vector(ANCHO_PALABRA-1 downto 0) );
end LIFO;
architecture FUNCIONAL of LIFO is
    type LIFO_t is array(0 to N_POS-1) of std_logic_vector(ANCHO_PALABRA-1 downto 0);
    signal LIFO_s: LIFO_t;
begin
    ESCRIBE_LEE : process (CLK)
        variable LIFO_v : LIFO_t;
        variable INDICE : integer := 0;
        variable LIFO_VACIA : boolean;
    begin
        if (CLK='1' and CLK'event) then
            if (RD='1' and LIFO_VACIA) then SALIDA <= LIFO_v(0);       -- Lectura
                for i in 0 to INDICE-1 loop
                    LIFO_v(i) := LIFO_v(i+1);
                end loop;
                LIFO_v(INDICE) := (others=>'U');
```

```
                INDICE := INDICE-1;
                if INDICE=0 then LIFO_VACIA := true;     -- LIFO vacía
                end if;
            else SALIDA <= (others => 'Z');      -- SALIDA en alta impedancia
                if (WR='1') then
                    while (INDICE /= 0) loop
                        LIFO_v(INDICE) := LIFO_v(INDICE-1);
                        INDICE := INDICE-1;
                    end loop;
                    if (INDICE<15) then         -- si la LIFO está llena se sobrescriben
                        INDICE := INDICE+1; -- los datos más antiguos
                    end if;
                    LIFO_v(0) := ENTRADA;
                    LIFO_VACIA := false;          -- existen datos en la LIFO
                end if;
            end if;
        end if;
        LIFO_s <= LIFO_v;
    end process ESCRIBE_LEE;
end FUNCIONAL;
-----------------------
-- Banco de pruebas
-----------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity TEST_LIFO is end TEST_LIFO;
architecture TEST of TEST_LIFO is
    component LIFO
        generic(ANCHO_PALABRA, N_POS: natural:= 8);
        port(ENTRADA: in std_logic_vector(ANCHO_PALABRA-1 downto 0);
                CLK, RD, WR: in bit;
                SALIDA: out std_logic_vector(ANCHO_PALABRA-1 downto 0) );
        end component;
    signal CLK, RD, WR: bit;
    signal ENTRADA, SALIDA: std_logic_vector(7 downto 0);
begin
    DUT: LIFO generic map (8, 16) port map (ENTRADA, CLK, RD, WR, SALIDA);
    RELOJ: process
    begin
        wait for 15 ns;
        CLK <= not (CLK);
    end process RELOJ;
    VALORES: process
    begin
        -- ciclo de escritura 1
        WR <= '1';
        ENTRADA <= "00000001";
        wait for 15 ns;
        WR <= '0';
        wait for 15 ns;
        -- ciclo de escritura 2
        WR <= '1';
        ENTRADA <= "00000010";
        wait for 15 ns;
        WR <= '0';
        wait for 15 ns;
        -- ciclo de escritura 3
        WR <= '1';
```

```
        ENTRADA <= "00000100";
        wait for 15 ns;
        WR <= '0';
        wait for 15 ns;
        -- ciclo de escritura 4
        WR <= '1';
        ENTRADA <= "00001000";
        wait for 15 ns;
        WR <= '0';
        wait for 15 ns;
        -- ciclo de lectura 1
        RD <= '1';                              -- inhibición de salida
        wait for 15 ns;
        wait for 15 ns;
        -- ciclo de lectura 2
        RD <= '0';                              -- inhibición de salida
        wait for 15 ns;
        wait for 15 ns;
        -- ciclo de lectura 3
        RD <= '1';                              -- inhibición de salida
        wait for 15 ns;
        RD <= '0';
        wait for 15 ns;
        -- ciclo de lectura 4
        RD <= '1';                              -- inhibición de salida
        wait for 15 ns;
        RD <= '0';
        wait for 15 ns;
        wait;
    end process VALORES;
end TEST;
-------------------------------------------------------------
- MEMORIA FIFO DE DIMENSIÓN GENÉRICA
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity FIFO is
    generic(ANCHO_PALABRA, N_POS: natural:= 8);
    port(ENTRADA: in std_logic_vector(ANCHO_PALABRA-1 downto 0);
            CLK, OE,WE: in bit;
            WP: inout bit;
            SALIDA: out std_logic_vector(ANCHO_PALABRA-1 downto 0);
end FIFO;
architecture base of FIFO is
    type FIFO_t is array(0 to N_POS-1) of std_logic_vector (ANCHO_PALABRA-1 downto 0);
begin
    escribe_lee: process(clk)
        variable celula: FIFO_t;
        variable indice: integer:=0;
    begin
        if (clk='1' and clk'event) then
            if (OE='0' and indice>0) then
                salida<=celula(0);
                for i in 0 to indice-1 loop
                    celula(i):=celula(i+1);
                end loop;
                celula(indice):=(others=>'U');
                indice:=indice-1;
            else
                salida<=(others=>'Z');
```

```
                if (WE='0' and WP='0') then
                    celula(indice):= entrada;
                    indice:= indice+1;
                    if indice>15 then    -- pila llena,
                        WP<= '1';             --impide la escritura
                    end if;
                end if;
            end if;
        end if;
    end process;
end base;
--------------------------------
-- BANCO DE PRUEBAS
--------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity testFIFO is
end testFIFO;
architecture comportamiento of testFIFO is
    component FIFO
        generic(ANCHO_PALABRA, N_POS: natural:= 8);
        port(ENTRADA: in std_logic_vector(ANCHO_PALABRA-1 downto 0);
            CLK, OE,WE: in bit;
            WP: inout bit;
            SALIDA: out std_logic_vector(ANCHO_PALABRA-1 downto 0);
    end component;
constant semiciclo: time:= 15 ns;
signal clk,oe,we,wp: bit;
signal entrada, salida: std_ulogic_vector(7 downto 0);
begin
    DUT: FIFO generic map (8,16) port map(entrada,clk,oe,we,wp,salida);
    reloj: process
    begin
        wait for semiciclo;
        clk<='1';
        wait for semiciclo;
        clk<='0';
    end process;
    valores: process
    begin
    ------------------------------- initialization de variables
        wait for 10 ns ;
        oe<='1';
        we<='1';
        wait for 25 ns ;
        ----------------------- ciclo de escritura1
        we<='0';
        entrada<="11100111";
        wait for semiciclo;
        we<='1';
        wait for semiciclo;
        -------------------------- ciclo de escritura2
        we<='0';
        entrada<="10000001";
        wait for semiciclo;
        we<='1';
        wait for semiciclo;
        -------------------------- ciclo de escritura3
```

```
        we<='0';
        entrada<="00001111";
        wait for semiciclo;
        we<='1';
        wait for semiciclo;
        --------------------------- ciclo de lectura1
        oe<='0';
        wait for semiciclo;
        oe<='1';            -- inhibición de salida
        wait for semiciclo;
        --------------------------- ciclo de lectura2
        oe<='0';
        wait for semiciclo;
        oe<='1';            -- inhibición de salida
        wait for semiciclo;
        --------------------------- ciclo de lectura3
        oe<='0';
        wait for semiciclo;
        oe<='1';            -- inhibición de salida
        wait for semiciclo;
        --------------------------- ciclo de lectura4
        oe<='0';
        wait for semiciclo;
        oe<='1';            -- inhibición de salida
        wait for semiciclo;
        wait;
    end process;
end comportamiento;
-------------------------------------------------------
-- CONTADOR SÍNCRONO DE 8 BITS
-------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_BIT.all;
entity CONTADOR is
    port(   ENTRADA :in unsigned(7 downto 0);
            AD,LOAD,RESET,CLK : in bit;
            SALIDA : out unsigned(7 downto 0)     );
end CONTADOR;
architecture RTL of CONTADOR is
    signal Q: unsigned(7 downto 0);
begin
    Q <= X"00" when RESET='1' else
        X"00" when (CLK='1' and CLK'event) and LOAD='0' and AD='0' and Q=X"FF" else
        X"FF" when (CLK='1' and CLK'event) and LOAD='0' and AD='1' and Q=X"00" else
        Q+X"01" when (CLK='1' and CLK'event) and LOAD='0' and AD='0' else
        Q-X"01" when (CLK='1' and CLK'event) and LOAD='0' and AD='1' else
        ENTRADA when (CLK='1' and CLK'event) and LOAD='1' else
        unaffected;
    SALIDA <= Q;
end RTL;
---------------------------
-- Banco de pruebas
---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_BIT.all;
entity TEST_CONTADOR is
end TEST_CONTADOR;
architecture PRUEBA of TEST_CONTADOR is
```

```
    component CONTADOR
        port(   ENTRADA :in unsigned(7 downto 0);
                AD,LOAD,RESET,CLK : in bit;
                SALIDA : out unsigned(7 downto 0)     );
    end component;
    signal T_AD,T_LOAD,T_RESET,T_CLK: bit;
    signal T_ENTRADA,T_SALIDA : unsigned(7 downto 0);
begin
    DUT: CONTADOR port map
                        (T_ENTRADA,T_AD,T_LOAD,T_RESET,T_CLK,T_SALIDA);
    RELOJ: process
    begin
        wait for 5 ns;
        T_CLK<='1';
        wait for 5 ns;
        T_CLK<='0';
    end process;
    VALORES: process
    begin
        wait for 100 ns;
        ---------------- entrada en parallel
        T_ENTRADA<=X"CD";
        T_LOAD<='1';
        wait for 30 ns;
        T_LOAD<='0';
        ----------------- peseta a cero síncrona
        wait for 100 ns;
        T_RESET<='1';
        wait for 25 ns;
        T_RESET<='0';
        ------------------ connate
        wait for 3000 ns;
        T_AD<='1';
        wait for 3000 ns;
        T_AD<='0';
        ------------------
        wait;
    end process;
end PRUEBA;
-----------------------------------------------
-- CONTADOR BCD DE 3 DÍGITOS
-----------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_BIT.all;
entity BCD_3D is
    port (  ENTRADA : in unsigned (11 downto 0);
            LD, RST, AD, CK : in bit;
            SP : out bit;
            Q : out unsigned (11 downto 0)  );
end BCD_3D;
architecture RTL of BCD_3D is
    signal ESTADO : unsigned (11 downto 0):= X"000";
    alias E1 : unsigned (3 downto 0) is ESTADO (3 downto 0);
    alias E2 : unsigned (3 downto 0) is ESTADO (7 downto 4);
    alias E3 : unsigned (3 downto 0) is ESTADO (11 downto 8);
    signal SP1, SP2 : bit;
begin
```

```
    E1 <= X"0" when RST='1' else
            ENTRADA(3 downto 0) when LD='1' and (CK='1' and CK'event) else
            E1+X"1" when (CK='1' and CK'event) and AD='0' and E1/=X"9" else
            E1-X"1" when (CK='1' and CK'event) and AD='1' and E1/=X"0" else
            X"0" when (CK='1' and CK'event) and AD='0' and E1=X"9" else
            X"9" when (CK='1' and CK'event) and AD='1' and E1=X"0" else
            unaffected;
    SP1   <= '1' when (AD='0' and E1=X"9") or (AD='1' and E1=X"0") else
               '0';
    E2 <= X"0" when RST='1' else
            ENTRADA(7 downto 4) when LD='1' and (CK='1' and CK'event) else
            E2+X"1" when (CK='1' and CK'event) and SP1='1' and AD='0' and E2/=X"9" else
            E2-X"1" when (CK='1' and CK'event) and SP1='1' and AD='1' and E2/=X"0" else
            X"0" when (CK='1' and CK'event) and SP1='1' and AD='0' and E2=X"9" else
            X"9" when (CK='1' and CK'event) and SP1='1' and AD='1' and E2=X"0" else
            unaffected;
    SP2   <= '1' when SP1='1' and ((AD='0' and E2=X"9") or (AD='1' and E2=X"0")) else
               '0';
    E3 <= X"0" when RST='1' else
            ENTRADA(11 downto 8) when LD='1' and (CK='1' and CK'event) else
            E3+X"1" when (CK='1' and CK'event) and SP2='1' and AD='0' and E3/=X"9" else
            E3-X"1" when (CK='1' and CK'event) and SP2='1' and AD='1' and E3/=X"0" else
            X"0" when (CK='1' and CK'event) and SP2='1' and AD='0' and E3=X"9" else
            X"9" when (CK='1' and CK'event) and SP2='1' and AD='1' and E3=X"0" else
            unaffected;
    SP     <= '1' when SP2='1' and ((AD='0' and E3=X"9") or (AD='1' and E3=X"0")) else
               '0';
    Q  <= E3&E2&E1;
end RTL;
----------------------------
-- Banco de pruebas
----------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_BIT.all;
entity TEST_BCD_3D is end TEST_BCD_3D;
architecture PRUEBA of TEST_BCD_3D is
    component BCD_3D
        port ( ENTRADA : in unsigned (11 downto 0);
                LD, RST, AD, CK : in bit;
                SP : out bit;
                Q : out unsigned(11 downto 0) );
    end component;
    signal T_ENTRADA : unsigned (11 downto 0);
    signal T_LD, T_RST, T_AD, T_CK, T_SP : bit;
    signal T_Q : unsigned (11 downto 0);
begin
    DUT: BCD_3D port map (T_ENTRADA, T_LD, T_RST, T_AD, T_CK, T_SP, T_Q);
    RELOJ : process
    begin
        wait for 15 ns; T_CK <= '1';
        wait for 15 ns; T_CK <= '0';
    end process RELOJ;
    VALORES : process
    begin
        wait for 10 ns;
        T_ENTRADA <= X"099"; T_LD <= '1';      -- entrada en parallel
        wait for 30 ns; T_LD <= '0';
        wait for 30 ns; T_AD <= '1';           -- cambio sent do de connate
        wait for 60 ns; T_RST <= '1';          -- peseta a cero síncrona
```

```
        wait for 25 ns; T_RST <= '0';
        wait for 25 ns; T_AD <= '0';                    -- cambio sent do de connate
        T_ENTRADA <= X"989"; T_LD <= '1';       -- entrada en parallel
        wait for 45 ns; T_LD <= '0';
        wait for 25 ns; T_AD <= '1';                    -- cambio sent do de connate
        wait for 50 ns; wait;
    end process VALORES;
end PRUEBA;
---------------------------------------
-- MULTIPLICADOR BINARIO
---------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_BIT.all;
entity MULTIPLICADOR_4 is
    port (  ENTRADA_1, ENTRADA_2 : in unsigned (3 downto 0);
            SALIDA : out unsigned (7 downto 0)    );
end MULTIPLICADOR_4;
architecture RTL of MULTIPLICADOR_4 is
begin
    SALIDA <= ENTRADA_1 * ENTRADA_2;
end RTL;
---------------------------
-- Banco de pruebas
---------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_BIT.all;
entity TEST_MULTIPLICADOR_4 is end TEST_MULTIPLICADOR_4;
architecture PRUEBA of TEST_MULTIPLICADOR_4 is
    component MULTIPLICADOR_4
        port (  ENTRADA_1, ENTRADA_2 : in unsigned (3 downto 0);
                SALIDA : out unsigned (7 downto 0)    );
    end component;
    signal T_ENTRADA_1, T_ENTRADA_2 : unsigned (3 downto 0);
    signal SALIDA : unsigned (7 downto 0);
begin
    DUT: MULTIPLICADOR_4 port map (T_ENTRADA_1, T_ENTRADA_2, SALIDA);
    VALORES : process
    begin
        for I in 0 to 9 loop            -- toad la tabla del 7
            T_ENTRADA_1 <= X"7";
            T_ENTRADA_2 <= TO_UNSIGNED(I,4);
            wait for 10 ns;
        end loop;
    end process VALORES;
end PRUEBA;
```

3. Identificar las afirmaciones correctas acerca del código formado por las siguientes sentencias:

```
signal a : integer := 0;
....
while a < 15 loop
    a <= a+1;
    entrada(a) := B(a) and C(a);
end loop;
```

– El bucle se ejecuta indefinidamente => CORRECTO. La señal "a" nunca puede actualizarse porque el bucle

carece de una sentencia **wait**

– El valor final de a es 14 => INCORRECTO. Su valor es 0 indefinidamente

– No se puede utilizar la señal a como índice de un vector => INCORRECTO

– La condición de salida del bucle (a>14) no se cumple nunca. => CORRECTO. La razón es la misma que para la primera respuesta

5. ¿Cuál de las dos opciones consigue intercambiar los valores iniciales de los objetos «codigo1» y «codigo2»? ¿Por qué?

a)

```
process
    variable codigo1 : integer := 57;
    variable codigo2 : integer := 92;
begin
    codigo1 := codigo2;
    codigo2 := codigo1;
    wait;
end process;
```

b)

```
Signal codigo1: integer := 57;
signal codigo2 : integer := 92;
...
process (codigo1, codigo2)
begin
    codigo1 <= codigo2;
    codigo2 <= codigo1;
end process;
```

En a) los valores finales de ambos objetos coinciden puesto que la actualización de variables es instantánea y en secuencia, con lo cual fácilmente se deduce que ambas acaban adquiriendo el valor 92 al ejecutarse el proceso. Por tanto NO SE CONSIGUE EL INTERCAMBIO.

En b) sin embargo los objetos son señales, con lo que se actualizan luego de un ciclo delta cuando el proceso se detiene. En primer lugar se "anota" que "codigo1" tomará el valor 92 cuando el proceso se detenga pero hasta entonces sigue valiendo 57. Lo mismo con "codigo2": tomará el valor 57 pero mientras el proceso está activo sigue valiendo 92. Así pues, cuando el proceso se para, AMBAS INTERCAMBIAN SUS VALORES luego de un ciclo delta.

Capítulo 10

1. Razonar si las siguientes descripciones son sintetizables y si se trata de diseños combinacionales o secuenciales.

a)

```
entity EJEMPLO_13 is
port ( D, CLK: in bit; Q: out bit);
end EJEMPLO_13;
architecture VHDL of EJEMPLO_13 is
begin
process(CLK)
begin
```

```
if not CLK' stable and CLK='1' then Q<=D;
else Q<='0';
end if;
end process;
end VHDL;
```

No es sintetizable. Podría parecer la descripción de un biestable activado por flancos ascendentes de reloj (if not CLK' stable and CLK='1') pero el 'else' que sigue a la condición 'if' no tiene significado hardware puesto que los biestables no pueden realizar un 'else'.

b)

```
entity EJEMPLO_14 is
port ( D, CLK, ENABLE: in bit; Q: out bit);
end EJEMPLO_14;
architecture VHDL of EJEMPLO_14 is
begin
process
begin
wait until CLK'event and CLK='1';
if ENABLE='1' then Q<=D after 30 ns;
end if;
end process;
end VHDL;
```

Sí es sintetizable. Se trata de un diseño secuencial, un biestable activado por flanco de subida con entrada de inhibición. Sin embargo, la parte de la descripción donde se establece un comportamiento temporal 'after 30 ns' no puede ser sintetizada y normalmente será ignorada por el sintetizador con un mensaje de advertencia de que no ha podido sintetizarlo.

c)

```
entity EJEMPLO_15 is
port (a, b: in integer range 0 to 3; c: out integer range 0 to 4);
end EJEMPLO_15;
architecture VHDL of EJEMPLO_15 is
begin
process(a, b)
begin
c<=a+b;
end process;
end VHDL;
```

Sí es sintetizable. Es un circuito combinacional cuya salida depende de todas las entradas simultáneamente.