1. METODOLOGÍA Y HERRAMIENTAS DE DISEÑO DE COMPUTADORES

[Transparencia relacionada con el problema]

1.1. [*Unidades de diseño*, pág. 7].
Añade a la siguiente declaración de entidad una cláusula que defina las constantes genéricas `Tpw_clk_h` y `Tpw_clk_l` del tipo ***`delay_length`*** con un valor por defecto de 3 ns.

```
entity flipflop is
     port (clk, d: in bit; q, q_n: out bit);
end entity flipflop;
```

1.2. [*Unidades de diseño*, pág. 16].
Escribe una cláusula de contexto que haga accesibles las bibliotecas `company_lib` y `proyect_lib` y que haga directamente visibles las entidades `in_pad` y `out_pad` de `company_lib` y todas las entidades de `proyect_lib`.

1.3. [*Unidades de diseño*, pág. 16].
Escribe una cláusula de contexto que haga accesibles la biblioteca `DSP_lib` y que haga directamente visibles la entidad `systolic_FFT` y todos los ítems declarados en un paquete `DSP_types` (la entidad y el paquete pertenecen a la biblioteca `DSP_lib`).

1.4. [*Unidades de diseño*, pág. 7; *Elementos básicos del lenguaje*, pág. 27].
En la siguiente interfaz de entidad se utiliza una constante genérica para especificar los tamaños de los vectores `standard_logic` a que pertenecen los puertos de entrada y los de salida. Completa la interfaz con los tipos de los puertos.

```
entity adder is
     generic (data_length: positive);
     port (a,b: in ...; sum: out ...);
end entity adder;
```

1.5. [*Unidades de diseño*, pág. 12; *Elementos básicos del lenguaje*, págs. 17 y ss.].
Escribe una declaración de paquete que contenga declaraciones de tipo para enteros no negativos representables con ocho bits; números fraccionarios entre –1.0 y +1.0; corrientes eléctricas con unidades de nA, μA, mA y A; y colores de un semáforo.

1.6. [*Elementos básicos del lenguaje*, pág. 3].
Escribe los siguientes literales decimales como literales hexadecimales.

```
1    34   256.0     0.5
```

1.7. [*Elementos básicos del lenguaje*, pág. 3].
¿Cuál es la diferencia entre los literales `16#23DF#` y `X"23DF"`?

1.8. [*Elementos básicos del lenguaje*, pág. 22].
Escribe declaraciones de constante para el número de bits en una palabra de 32 bits y para el número π (3.14159).

1.9. [*Elementos básicos del lenguaje*, pág. 27].
Escribe una declaración para un subtipo de `std_ulogic_vector` que represente un byte. Declara una constante de ese subtipo donde cada elemento tenga el valor 'Z'.

1.10. [*Elementos básicos del lenguaje*, pág. 27].
Se puede extender el signo de un vector `v1` de 8 bits, que representa un entero binario en complemento a dos, para formar un vector `v2` de 32 bits, del siguiente modo: se copia `v1` en las 8 posiciones más a la izquierda de `v2` y, en `v2`, se ejecuta un desplazamiento aritmético a la derecha para llevar los 8 bits de la izquierda a las 8 posiciones de la derecha. Escribe las sentencias de asignación de variable necesarias para hacer esto, precedidas de las declaraciones de `v1` y `v2`.

1.11. [*VHDL Quick Start*, págs. 37 a 39; *Sentencias*, págs. 7 y 8].
En el momento actual, la señal `s` tiene el valor '0'. ¿Cuál es el valor de las variables booleanas `v1` y `v2` después de ejecutar las siguientes sentencias dentro de un proceso?

```
s <= '1';
v1 := s='1';
wait on s;
v2 := s='1';
```

1.12. [*Elementos básicos del lenguaje*, pág. 39; *Sentencias*, for e if].
Escribe un bucle para contar el número de elementos que valen '1' en una variable, `v`, del tipo vector de bits.

1.13. [*Elementos básicos del lenguaje*, pág. 34; *Sentencias*, págs. 27 y 28].
Escribe un bucle `while` para calcular la función exponencial de `x` sumando los términos de la siguiente serie que sean mayores que la suma de los anteriores dividida por 10^5.

$$e^x = 1 + (x/1) + (x^2/2!) + (x^3/3!) + (x^4/4!) + \ldots$$

1.14. [*Elementos básicos del lenguaje*, pág. 34; *Sentencias*, págs. 27 y 28].
Escribe un bucle `for` para calcular la función exponencial de `x` sumando los ocho primeros términos de la serie.

1.15. [*Elementos básicos del lenguaje*, pág. 42; *Sentencias*, pág. 50].
Escribe una sentencia `assert` concurrente para verificar que el tiempo entre cambios de una señal de reloj, `clk`, es, al menos, `T_pw_clk`.

1.16. [*Sentencias*, vectores y sentencia for].
Escribe una declaración de tipo `array` para un vector de 30 enteros y una declaración de variable para una variable de ese tipo. Escribe un bucle `for` para calcular el valor medio de los componentes del vector.

1.17. [*Sentencias*, págs. 7 y 8].
Escribe una sentencia `wait` que suspende un proceso hasta que una señal `s` cambia de '1' a '0' y una señal de habilitación, `en`, es '1'.

1.18. [*Sentencias*, págs. 7 y 8].
Escribe una sentencia `wait` que suspende un proceso hasta que una señal `ready` cambia a '1' o transcurren 5 ms.

1.19. [*Sentencias*, págs. 28 y ss.].
Escribe una sentencia `loop` que consulta un bit de entrada `d` cuando una entrada de reloj `clk` cambia a '1'. Si `d` vale '0' el bucle continúa, pero si vale '1', se sale del bucle.

1.20. [*Sentencias*, págs. 34 y 35].
Escribe una sentencia `assert` que expresa el requisito de que las dos salidas de un flip-flop, `q` y `q_n`, de tipo `std_ulogic`, son complementarias.

1.21. [*Sentencias*, pág. 46].
Escribe el proceso equivalente a la sentencia de asignación de señal condicional

```
mux_logic:
    z <= a and not b after 5 ns when enable='1' and sel='0'
         else x or y after 6 ns when enable='1' and sel='1'
         else '0' after 4 ns;
```

1.22. [*Sentencias*, pág. 51].
Se tiene declarado el siguiente procedimiento

```
procedure shuffle_bytes
    (signal d_in: in std_ulogic_vector(0 to 15);
    (signal d_out: out std_ulogic_vector(0 to 15);
    (signal shuffle_control: in std_ulogic;
    prop_delay: delay_length) is ...
```

Escribe el proceso equivalente a la siguiente llamada concurrente a procedimiento

```
swapper: shuffle_bytes (ext_data, int_data, swap_control,
Tpd_swap);
```

1.23.

Se tiene la siguiente interfaz de un semisumador

```
entity HalfAdder is
     port(I1, I2: in bit; Cout, S: out bit);
end HalfAdder;
```

y una arquitectura para el mismo. Un banco de pruebas para esa entidad sería

```
entity TestHalfAdder is
end TestHalfAdder;

architecture Funcional of TestHalfAdder is

component HalfAdder
port(I1, I2: in bit; Cout, S: out bit);
end component;

signal A, B, COut, Sum: bit;

begin
  HA1: HalfAdder port map(A, B, COut, Sum);

  A <= not(A) after 5 ns;
  B <= not(B) after 10 ns;

end Funcional;
```

Escribe un banco de pruebas para un sumador completo cuya declaración de entidad sea

```
entity FullAdder is
     port(A, B, Cin: in bit; Cout, Sum: out bit);
end FullAdder;
```