<table>
<tr><td colspan="3" rowspan="2"> 
UNIVERSIDAD POLITÉCNICA DE MADRID

ETSIS de Telecomunicación</td><td colspan="6">APELLIDOS: SOLUCION</td></tr>
<tr><td colspan="4">NOMBRE:</td><td colspan="2">DNI:</td></tr>
<tr><td colspan="4">ASIGNATURA: DISEÑO DIGITAL I</td><td colspan="2">Bloque: 2, ORDINARIO (escrito)</td></tr>
<tr><td colspan="3">Fecha</td><td>Curso</td><td colspan="4">TITULACIÓN: ☐ Electrónica de Comunic. ☐ Sistemas de Telecom. ☐ Sonido e Imagen ☐ Telemática</td><td colspan="2">Nota Final</td></tr>
<tr><td>11</td><td>01</td><td>2021</td><td>TERCERO</td><td colspan="4">Calificaciones parciales</td><td colspan="2"></td></tr>
</table>

ADVERTENCIAS PARA LA REALIZACIÓN DE LA PRIMERA PARTE DEL EXAMEN

- Rellene **AHORA** los datos personales que deben figurar en esta hoja.
- Mientras dure el examen deberá exponer su D.N.I. encima de la mesa.
- **NO SE ADMITIRÁN** exámenes escritos a lapicero ni con tinta roja o verde.
- **COMPRUEBE** que su ejemplar del examen consta de **2** ejercicios en **9** páginas numeradas.
- En este examen **NO PUEDEN UTILIZARSE CALCULADORAS, LIBROS, APUNTES NI DISPOSITIVOS DE TELECOMUNICACIÓN**. Retírelos ahora de la mesa.
- La duración de esta parte del examen es de **85 minutos**.
- Rellene la siguiente tabla que registra el puesto de laboratorio donde está realizando el examen

Aula:	**Puesto:**

Esta hoja se ha dejado en blanco intencionadamente

Ejercicio 1	Monoestable multipulso		
		3,5 puntos	30 minutos

El siguiente código modela el funcionamiento de un monoestable capaz de generar un tren de tres pulsos en su salida “pulsos” cada vez que se activa su entrada “trigger”. La duración de los pulsos, y el tiempo transcurrido entre cada uno de ellos viene dada por el valor de las constantes p1_on, p1_off, p2_on, p2_off y p3_on.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity monoestable_multi is
port(nRst:    in     std_logic;
     clk:     in     std_logic;
     trigger: in     std_logic;
     pulsos:  buffer std_logic);

end entity;

architecture rtl of monoestable_multi is
  signal cnt_mono: std_logic_vector(6 downto 0);

  constant p1_on:   natural := 10;
  constant p1_off:  natural := 10;

  constant p2_on:   natural := 20;
  constant p2_off:  natural := 20;

  constant p3_on:   natural := 30;


begin
  process(clk, nRst)
  begin
    if nRst = '0' then
      cnt_mono <= (others => '0');

    elsif clk'event and clk = '1' then
      if trigger = '1' and cnt_mono = 0 then
        cnt_mono <= (0 => '1', others => '0');

      elsif cnt_mono /= 0 and
            cnt_mono /= (p1_on + p1_off + p2_on + p2_off + p3_on) then
        cnt_mono <= cnt_mono + 1;

      else
        cnt_mono <= (others => '0');

      end if;
    end if;
  end process;

  pulsos <= '0' when cnt_mono = 0 else
            '1' when cnt_mono <=  p1_on else
            '0' when cnt_mono <= (p1_on + p1_off) else
            '1' when cnt_mono <= (p1_on + p1_off + p2_on) else
            '0' when cnt_mono <= (p1_on + p1_off + p2_on + p2_off) else
            '1' when cnt_mono <= (p1_on + p1_off + p2_on + p2_off + p3_on);

end rtl;
```

1.- La sentencia concurrente pretende modelar un circuito combinacional. Justifique si el modelo del circuito es o no correcto. **(0.5 puntos)**

Pretende ser un modelo de un circuito combinacional que activa la señal de salida en los tramos en "on" y la desactiva en los tramos en "off". Sin embargo, no es correcto como modelo combinacional porque no se asigna valor a la salida para todas las posibles combinaciones de entrada. De hecho, en el proceso de síntesis Quartus infiere un latch a partir de este modelo.

La corrección figura en el apartado siguiente.

Si se cambia la sentencia concurrente por la siguiente: **(0.5 puntos)**

```
pulsos <= '0' when cnt_mono = 0 else
          '1' when cnt_mono <=  p1_on else
          '0' when cnt_mono <= (p1_on + p1_off) else
          '1' when cnt_mono <= (p1_on + p1_off + p2_on) else
          '0' when cnt_mono <= (p1_on + p1_off + p2_on + p2_off) else
          '1' when cnt_mono <= (p1_on + p1_off + p2_on + p2_off + p3_on) else
          '1';
```

¿El circuito modelado por esta nueva sentencia es el mismo que el modelado por la sentencia original? Justifique su respuesta.

En sí misma no es equivalente. Este sí es un modelo correcto para el circuito combinacional que activa la señal de salida en los tramos en "on" y la desactiva en los tramos en "off". El "else" final evita el problema de la sentencia original.

¿Funcionaría el circuito *monoestable_multi* correctamente? Justifique su respuesta para cada una de las dos versiones de la sentencia concurrente.

No obstante, el circuito funciona correctamente con ambas versiones de la sentencia concurrente, ya que el contador jamás alcanza un valor superior al contemplado en el último "when", por lo que, en realidad, la parte del "else" final nunca es utilizado.

2.- Si modificamos la sentencia *elsif* del proceso que modela el contador por el código siguiente:

```
elsif cnt_mono /= 0 and
      cnt_mono < (p1_on + p1_off + p2_on + p2_off + p3_on) then
  cnt_mono <= cnt_mono + 1;
```

¿cambiaría el funcionamiento del circuito? Justifique su respuesta **(0.5 puntos)**

No. Al ser cnt_mono un contador ascendente, la primera condición (cuenta /=0 y /=90) y la segunda (cuenta /=0 y cuenta < 90) son equivalentes.

3.- ¿Es el monoestable modelado redisparable? Justifique su respuesta. **(0.5 puntos)**

No es redisparable, ya que el trigger está habilitado con el valor cero en el contador cnt_mono. Por lo tanto, se necesita que los pulsos de salida hayan finalizado.

4.- Modifique el modelo para que se pueda redisparar el circuito únicamente tras la generación del primero de los pulsos. Incluya únicamente las sentencias con cambios **(0.5 puntos)**

Hay que cambiar la línea:

```
if trigger = '1' and cnt_mono = 0 then
```

por la siguiente:

```
if trigger = '1' and (cnt_mono = 0 or cnt_mono > p1_on) then
```

5.- Describa los cambios que habría que realizar en el modelo para que al disparar el circuito se genere un tren de 4 pulsos a nivel alto con una duración de 20 ciclos de reloj cada uno separados por un nivel bajo con una duración de 20 ciclos **(1 punto)**

Esta es una posible solución, la más directa, pero habría otros posibles enfoques.

- *Crear dos nuevas constantes: p3_off y p4_on*
- *Todas las constantes igualadas a 20*
- *En el "elsif" habría que poner*
  cnt_mono < (p1_on + p1_off + p2_on + p2_off + p3_on + p3_off + p4_on)

- *En la sentencia concurrente habría que incluir dos nuevas líneas:*
  '0' when cnt_mono <= (p1_on + p1_off + p2_on + p2_off + p3_on + p3_off) else
  '1' when cnt_mono <= (p1_on + p1_off + p2_on + p2_off + p3_on + p3_off + p4_on)

- *El contador cnt_mono debería tener 8 bits ya que debe ser capaz de contar hasta 140.*

Ejercicio 2	Análisis de un modelo		
		5,5 puntos	55 minutos

En este ejercicio deberá analizar el siguiente modelo y contestar a las preguntas que se indican a continuación.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity monoestable_tren is
port(nRst:    in     std_logic;
     clk:     in     std_logic;
     trigger: in     std_logic;
     N:       in     std_logic_vector(6 downto 0);
     ancho:   in     std_logic_vector(6 downto 0);
     pulsos:  buffer std_logic);

end entity;



architecture rtl_1 of monoestable_tren is
  signal cnt_N:       std_logic_vector(6 downto 0);
  signal cnt_ancho:   std_logic_vector(7 downto 0);

begin
  process(clk, nRst)
  begin
    if nRst = '0' then
      cnt_N     <= (0 => '1', others => '0');
      cnt_ancho <= (others => '0');

    elsif clk'event and clk = '1' then
      if trigger = '1' and cnt_ancho = 0 then
        cnt_ancho <= (0 => '1', others => '0');
        cnt_N <= (0 => '1', others => '0');

      elsif cnt_ancho < ancho&'0' and cnt_ancho /= 0 then
        cnt_ancho <= cnt_ancho + 1;

      elsif cnt_N < N then
        cnt_ancho <= (0 => '1', others => '0');
        cnt_N <= cnt_N + 1;

      else
        cnt_ancho <= (others => '0');

      end if;
    end if;
  end process;

  pulsos <= '1' when cnt_ancho > ancho else
            '0';

end rtl_1;
```

1.- Explique el funcionamiento del circuito cuando *N* y *ancho* son distintos de cero **(1,5 puntos)**

Genera un tren de "N" pulsos con una duración de "ancho" ciclos de reloj y distanciados "ancho" ciclos de reloj entre ellos. Cuando se recibe un pulso en la señal "trigger".

No es redisparable, hay que esperar hasta el final del tren de pulsos para efectuar un nuevo disparo.

2.- Deduzca y explique el funcionamiento del circuito cuando *N* es 0 y *ancho* es distinto de 0 **(0.5 puntos)**

Genera un pulso de duración "ancho" ciclos de reloj, retardado "ancho" ciclos de reloj desde la activación del "trigger". Es decir, un pulso con las características descritas en el apartado anterior.

3.- Deduzca y explique el funcionamiento del circuito cuando *ancho* es 0 y *N* es distinto de cero **(0.5 puntos)**

Genera un pulso de duración "N" ciclos de reloj tras la activación del "trigger". Es decir, un pulso como el de un monoestable convencional con las características descritas en el apartado anterior.

4.- ¿En alguno de los dos anteriores casos el circuito opera como un monoestable? Si es que no, indique las diferencias **(0.5 puntos)**

En propiedad, solamente en el caso del apartado 3 podemos hablar de un monoestable. En el caso del apartado 2 el pulso se genera con retardo.

5.- Calcule el número de ciclos mínimos entre dos disparos en función de N y ancho, cuando N y ancho son distintos de 0 **(0.5 puntos)**

*El monoestable no puede ser rearmado hasta 2*N*ancho + 1 ciclos de reloj después del disparo anterior.*

Para las siguientes dos preguntas suponga que el cuerpo de arquitectura que modela el funcionamiento del monoestable es el siguiente.

```
architecture rtl_2 of monoestable_tren is
  signal cnt_N:        std_logic_vector(6 downto 0);
  signal cnt_ancho:    std_logic_vector(   downto 0);
  signal ancho_i:      std_logic_vector(   downto 0);

begin
  process(clk, nRst)
  begin
    if nRst = '0' then
      cnt_N     <= (0 => '1', others => '0');
      cnt_ancho <= (others => '0');
      ancho_i   <= (others => '0');

    elsif clk'event and clk = '1' then
      if trigger = '1' and cnt_ancho = 0 then
        cnt_ancho <= (0 => '1', others => '0');
        ancho_i <= "0···0"&ancho; -- Se concatenan '0' hasta completar vector

        cnt_N <= (0 => '1', others => '0');

      elsif cnt_ancho < ancho_i&'0' and cnt_ancho /= 0 then
        cnt_ancho <= cnt_ancho + 1;

      elsif cnt_N < N then
        cnt_N <= cnt_N + 1;
        cnt_ancho <= (0 => '1', others => '0');
        ancho_i   <= ancho_i + ancho;

      else
        cnt_ancho <= (others => '0');

      end if;
    end if;
  end process;

  pulsos <= '1' when cnt_ancho > ancho_i else
            '0';
end rtl_2;
```

7.- Describa el funcionamiento del circuito cuando *N* y *ancho* son distintos de 0
(1 punto)

*La salida pulsos genera, tras cada disparo, un tren de N pulsos, siendo el pulso N-simo de anchura 2*N*ancho y ciclo de trabajo 50%*

8.- Deduzca el número de bits que son necesarios para las señales de *cnt_ancho* y *ancho_i*. Justifique su respuesta **(1 punto)**

*ancho_i debe ser capaz de almacenar un valor máximo de desde 0 hasta 127*127 = 16129, por lo que debe disponer de 14 bits. cnt_ancho debe almacenar un valor el doble de grande, por lo que debe tener 15 bits*