Trabajo SED

Cerradura digital

Objetivos:

- Profundizar en el manejo del lenguaje VHDL.

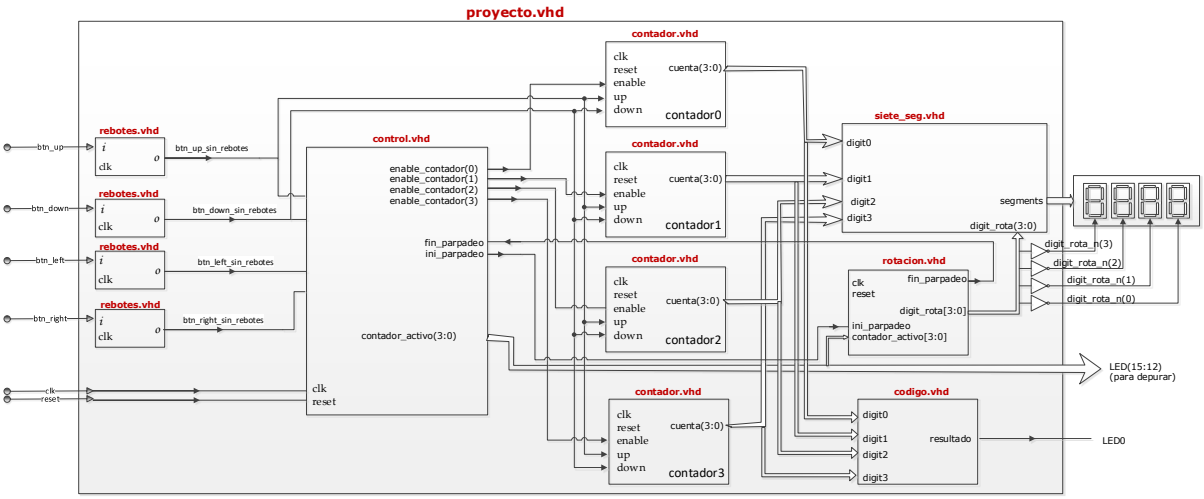
Profesor encargado del proyecto:

- Millán, Rafael rmillan@us.es

1. Descripción del trabajo

El objetivo de este trabajo es el control de una cerradura digital de 4 dígitos utilizando la tarjeta BASYS 3. Se utilizarán los pulsadores izquierda o derecha para seleccionar un dígito del código (su valor se mostrará en uno de los 4 displays) y los pulsadores arriba y abajo para incrementar o decrementar el valor de ese dígito.

En el momento en que se haya introducido la secuencia correcta, se iluminará el LED0 de la placa. El esquema de bloques del diseño es el siguiente:



ENTRADAS:

- **clk,reset**: Solo van a las cajas donde hay biestables: control.vhd, contador.vhd, rotacion.vhd. No se ha dibujo por todo el circuito para facilitar su legibilidad.
- **bnt_left, btn_right**: pulsadores izquierda y derecha. Permiten seleccionar el dígito que desea modificar el usuario. Con cada pulsación se modifica el dígito seleccionado y será visible mediante un parpadeo.
- **bnt_up, bnt_down**: permiten incrementar/decrementar la cuenta del contador seleccionado.

SALIDAS:

- **segments**[6:0]: vector de 7 bits para los segmentos. Está compartido por los 4 displays por lo que hay que realizar una rotación rápida asignando tiempos de iluminación a cada display con un valor detereminado para esta señal de segments.
- **digit_rota**[3:0]: Señal activa a nivel bajo para iluminar uno de los displays en un instante determinado. Por ejemplo, si quiere iluminar el display derecho (unidades), esta salida tomaría el valor "1110".
- **resultado**: Esta conectada al LED0 y se pone a '1' si los 4 dígitos contienen la clave correcta (4321). En caso contrario estaría a '0'.
- **contador_activo**[3:0]: Cuando pulsamos izquierda/derecha, cambiamos el dígito activo que responderá a los botones arriba/abajo para incrementarse o decrementarse. Esta salida no es necesaria. Será útil para depurar el código.

Las entradas y salidas de cada caja tienen los nombres que aparecen en la ENTITY de cada fichero en VHDL.

El fichero trabajo.vhd realiza el cableado de todas las cajas. En su ENTITY estarán las entradas y salidas externas que ya se han comentado. Los cables que unen las cajas internamente se realizan con señales internas de la arquitectura que aparecen en la salida de una caja y en la entrada de otra para que queden conectadas.

2. Ejercicios

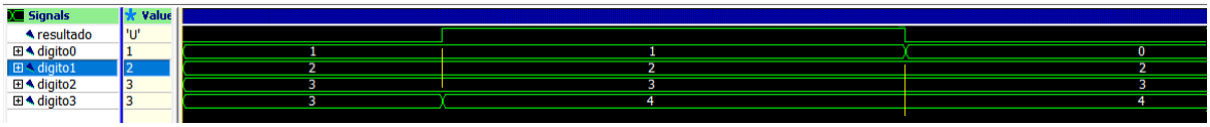
2.1. Comprobación del código (codigo.vhd). (0.5 puntos)

Circuito combinacional simple. Recibe los 4 dígitos de los contadores y los compara con el número "4321". Si el resultado coincide, la salida se pone a '1' y a '0' en caso contrario.

La entidad de esta parte del circuito, según los nombres que aparecen en el esquema anterior, es la siguiente:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity codigo is
    Port (
           digit0 : in STD_LOGIC_VECTOR (3 downto 0);
           digit1 : in STD_LOGIC_VECTOR (3 downto 0);
           digit2 : in STD_LOGIC_VECTOR (3 downto 0);
           digit3 : in STD_LOGIC_VECTOR (3 downto 0);
           resultado : out STD_LOGIC);
end codigo;

architecture codigo_arq of codigo is
```

Se muestra una simulación en la que las entradas toman los valores 3321, 4321 (correcto) y 4320).



Se pide rellenar el código del fichero de diseño y el de estímulos (codigo_tb.vhd) para que realice una simulación en la que las entradas tomen los valores 3789, 4321 (correcto) y 3798.

2.2. Señales para los displays de 7 segmentos (siete_seg.vhd). (0.5 puntos)

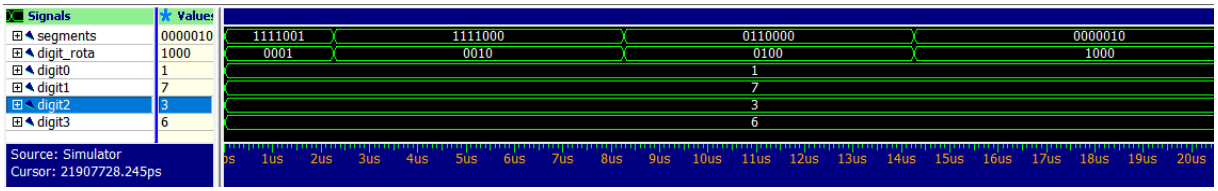
Circuito combinación simple que recibe los dígitos de los 4 contadores. Multiplexa uno de ellos en función de la entrada digit_rota[]. La valor multiplexado se muestra a la salida pero convertido a 7 segmentos. La conversión bcd a 7 segmentos se tiene en los apuntes.

Se muestra una parte del código para la entidad y una señal interna de la arquitectura (bcd) necesaria para realizar la multiplexación.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity siete_seg is
    Port (
           segments   : out STD_LOGIC_VECTOR (6 downto 0);
           digit_rota : in STD_LOGIC_VECTOR (3 downto 0);
           digit0     : in STD_LOGIC_VECTOR (3 downto 0);
           digit1     : in STD_LOGIC_VECTOR (3 downto 0);
           digit2     : in STD_LOGIC_VECTOR (3 downto 0);
           digit3     : in STD_LOGIC_VECTOR (3 downto 0));
end siete_seg;

architecture siete_seg_arq of siete_seg is
    signal bcd : std_logic_vector(3 downto 0);
begin
```

En la siguiente simulación se muestra el caso en el que las entradas valen digit0=1, digit1=7, digit2=3 y digit3=6. La señal digit_rota[] cambia entre los 4 posibles valores que puede tomar y para cada uno de ellos saca a la salida el dígito seleccionado en formato 7-

segmentos. Por ejemplo, si digit_rota=0001 entonces a la salida se vería digit0 que es el 1 (1111001 en formato de 7 segmentos). Cuando vemos un 1 en un display, se iluminan dos segmentos que son los dos ceros que aparecen en el dato anterior.



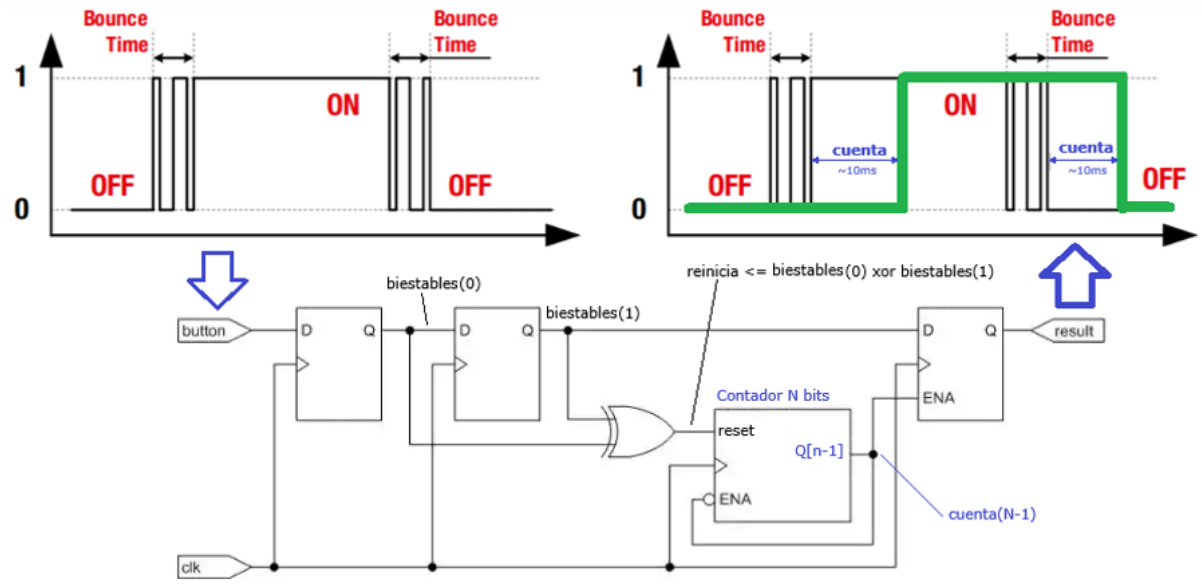
Se pide simular el caso en el que digit0=5, digit1=1, digit2=6, digit3=9 y la entrada digit_rota cambiará al revés. Comenzará por 1000 -> 0100 -> 0010 -> 0001.

2.3. Control de rebotes en los pulsadores (rebotes.vhd). (1 punto)

La interacción de elementos mecánicos (pulsadores e interruptores) con elementos electrónicos genera siempre rebotes. Hay que filtrar la señal para poder operar con ella dentro del circuito.

En la siguiente figura se muestra el circuito que se va a diseñar para asegurarse que se trabaja con la señal en una zona libre de rebotes tras una conmutación de esta. Lo que ve el interior del circuito es la señal de color verde.

Las señales *biestables*(1:0) y *reinicia* son internas de la arquitectura. Si el valor de biestables(0) y biestables(1) son diferentes (xor), el contador se reinicia. El diseño es completamente síncrono (todo se diseña dentro del "if (rising_edge(clk)) then"). En cada ciclo de reloj se actualizan los valores de biestables(0)<=i; y biestables(1)<= biestables(0);

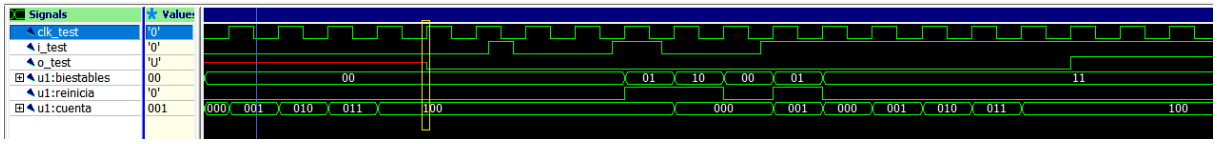


La entidad y las señales internas para gestionar los rebotes se muestran a continuación. El contador utiliza un GENERIC con un valor de N=21 bits. El contador contará desde 0 hasta que se ponga a 1 el bit más significativo (cuenta(N-1)=1), justo cuando ha realizado 2^{N-1} cuentas. Si el reloj del sistema es de 100Mhz, el bit cuenta(N-1) se pondrá a 1 al cabo de ~10ms (clk*2^{20} = 0,01us*1.048.576 ~ 10,46ms).

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY rebotes IS
   GENERIC(
      N21  :  INTEGER := 21
   );
   PORT(
      clk : IN  STD_LOGIC;
      i   : IN  STD_LOGIC;  -- entrada con rebotes
      o   : OUT STD_LOGIC   -- salida  sin rebotes
   );
END rebotes;

ARCHITECTURE rebotes_arq OF rebotes IS
  SIGNAL biestables: STD_LOGIC_VECTOR(1 DOWNTO 0) := "00";               -- muestrea la entrada
  SIGNAL reinicia  : STD_LOGIC := '0';                                   -- sync reset a zero
  SIGNAL cuenta    : STD_LOGIC_VECTOR(N21-1 DOWNTO 0) := (OTHERS => '0'); -- salida del contador
BEGIN

  reinicia <= biestables(0) xor biestables(1);   -- determina si reinicia/cuenta el contador

  PROCESS(clk)
  BEGIN
    IF(clk'EVENT and clk = '1') THEN
```

En la siguiente simulación el valor parametrizable N21 se ha puesto al valor de 3, en lugar de 21. De esta forma se puede simular con pocos ciclos (en la realidad, para 10ms son necesarios del orden de 1 millón con un reloj de 100Mhz). En las curvas se aprecia que se el contador cuenta $2^{N-1}=2^2=4$ ciclos para conseguir que el bit más significativo se ponga a '1' (cuenta(2)= '1').

Al no tener señal de reset, inicialmente la salida tiene un valor indefinido (color rojo). Cuando el contador llega al máximo (cuenta(N-1)=1), la salida captura el valor de biestables(1) de forma sincronizada con un flanco activo de reloj. Es importante visualizar las señales internas de la arquitectura para depurar el programa.



El GENERIC del fichero rebotes.vhd se deja fijo para la implementación que se realizará en la placa con un reloj de 100Mhz. El cambio de dicho parámetro (N21) de 21 al valor de 3 se realiza en el fichero de estímulos cuando se instancia el componente. Ver recuadro rojo.

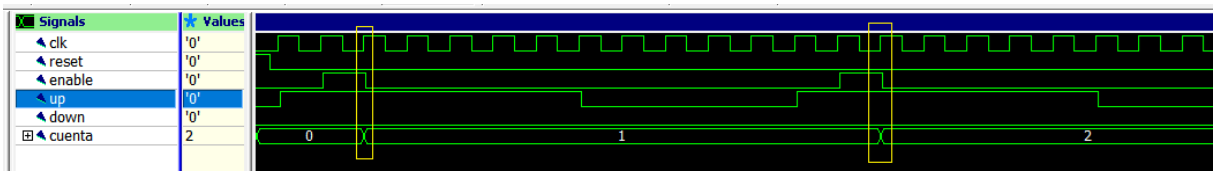
```
-- ********************************************************************************
-- ENTIDAD     (entradas/salidas, el fichero de simulación no tiene)
-- ********************************************************************************
ENTITY test_rebotes IS
END    test_rebotes;

-- ********************************************************************************
-- ARQUITECTURA   (descripción de los estímulos)
-- ********************************************************************************
ARCHITECTURE test_rebotes_arq OF test_rebotes IS
    --Declaración de componentes
    COMPONENT rebotes
          GENERIC(
            N21 : INTEGER := 21  -- Tamaño del contador (ajustable segun la frecuencia del reloj)
          );
          PORT(
            clk : IN  STD_LOGIC;  -- Reloj del sistema
            i   : IN  STD_LOGIC;  -- Entrada del botón con rebotes
            o   : OUT STD_LOGIC   -- Salida debounced
          );
    END COMPONENT;

    SIGNAL CLK_test,I_test,O_test : std_logic;
    constant ciclo: time :=  10 ns;  -- hay que dejar un espacio entre 10 y ns
BEGIN
    -- ////////////////////////////////////////////////////////////////////////////////
    -- Se crea el componente U1 y se conecta a las señales internas de la arquitectura
    -- ////////////////////////////////////////////////////////////////////////////////
    i_rebotes: rebotes
                GENERIC MAP (N => 3)
                PORT    MAP(
                            clk    => CLK_test,
                            i      => I_test,
                            o      => O_test
                          );
```

Se pide simular un flanco de bajada de la señal de entrada con varios rebotes. En las curvas anteriores hay un par de rebotes antes de estabilizarse a nivel alto.

2.4. Contador para los dígitos (contador.vhd). (0.75 puntos)

Se diseñará un contador similar al que se ha visto en prácticas. Se incrementará en 1 si se pulsa el botón 'up' o se decrementará en 1 si se pulsa su botón 'down'. Para que pueda contar debe estar a 1 la entrada 'enable'. Habrá un módulo de control que generará la entrada de enable de forma que, si se pulsa up o down, ese módulo de control generará un pulso de un ciclo de reloj para la señal de enable. De esta forma, el enable solo deja pasar un flanco activo de reloj por lo que el contador se incrementará o decrementará solo en 1, independientemente de la duración de la señal *up* o *down*.

Se muestra una simulación con valores apropiados de *enable* y la señal *up*. Tiene dos pulsos por lo que la cuenta se incrementa en 2.



El contador no es módulo 10. Puede contar desde 0 hasta F. La entidad del circuito correspondiente al esquema del primer apartado del índice sería el siguiente:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity contador is
    Port (
           clk     : in  STD_LOGIC;
           reset   : in  STD_LOGIC;
           enable  : in  STD_LOGIC;
           up      : in  STD_LOGIC;
           down    : in  STD_LOGIC;
           cuenta  : out STD_LOGIC_VECTOR (3 downto 0)
    );
end contador;

architecture contador_arq of contador is
    signal count : unsigned(3 downto 0);
begin
    cuenta <= std_logic_vector(count);

process(clk)
```

Se pide simular 3 pulsos de la entrada 'up', 1 pulso de la entrada 'down' y dos nuevos pulsos de 'up'.

2.5. Iluminación displays mediante rotación (rotacion.vhd).

2.5.1. Iluminación básica (0.75 puntos)

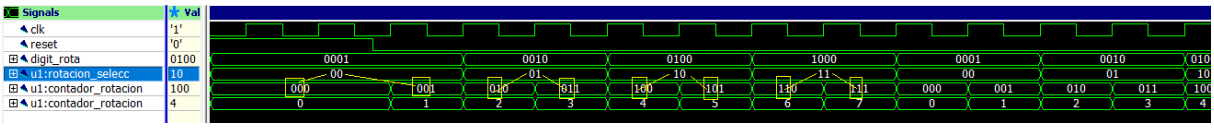
En este caso los displays parecerán que están encendidos todo el tiempo. Si el usuario pulsara 'up' o 'down', para incrementar/decrementar el valor del display seleccionado, no sabría cuál de ellos es hasta que uno de ellos se incrementara o decrementara. Esta rotación básica de los displays está resuelto en la documentación de la primera práctica. Se adaptará este diseño cambiando el nombre de las entradas, salidas y señales internas que se muestran a continuación.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity rotacion is
    GENERIC (
          N20: INTEGER := 20    -- Paramatrizable para cualquier frecuencia de reloj
    );
    Port (
          clk,reset       : in  STD_LOGIC;
          digit_rota      : out STD_LOGIC_VECTOR (3 downto 0)                .
        );
end rotacion;

architecture rotacion_arq of rotacion is
    signal contador_rotacion : unsigned(N20-1 downto 0);  -- unsigned(19 downto 0)
    SIGNAL rotacion_selecc   : UNSIGNED(1 DOWNTO 0);    -- en los apuntes aparece como 'display_activo'
begin
```

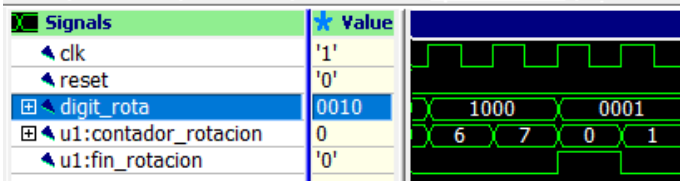
En el fichero de estímulos se creará una instancia del componente con el valor de N20 igual a 3 (contador de 3 bits) para que se pueda ver en la simulación cómo la señal de salida *digit_rota*[] cambia cuando lo hacen los dos bits más significativos de contador.

```
ENTITY test_rotacion IS
END    test_rotacion;

-- ********************************************************************************
-- ARQUITECTURA   (descripción de los estímulos)
-- ********************************************************************************
ARCHITECTURE test_rotacion_arq OF test_rotacion IS
    --Declaración de componentes
    COMPONENT rotacion     -- para la implementación en la placa
        GENERIC (
             N20: INTEGER := 20    -- Paramatrizable para cualquier frecuencia de reloj
        );
        Port (
           clk,reset       : in  STD_LOGIC;
           digit_rota      : out STD_LOGIC_VECTOR (3 downto 0)
        );
    END COMPONENT;

    SIGNAL CLK,RESET: std_logic;
    SIGNAL DIGIT_ROTA : std_logic_vector(3 downto 0);

    constant ciclo: time := 10 ns;  -- hay que dejar un espacio entre 1 y ms
BEGIN
    -- ////////////////////////////////////////////////////////////////////////////////
    -- Se crea el componente U1 y se conecta a las señales internas de la arquitectura
    -- ////////////////////////////////////////////////////////////////////////////////
    U1: rotacion
            GENERIC MAP (N20 => 3)   -- el componente se creará con un contador de 3 bits
            PORT MAP(
                clk         => CLK,
                reset       => RESET,
                digit_rota  => DIGIT_ROTA
            );
```

Cuando *contador_rotacion* desborda y pasa a cero, comienza una nueva secuencia de rotación que reparte tiempos para que se iluminen los displays de uno en un, según el valor que tenga la señal digit_rota[].



Se pide añadir una señal interna llamada *fin_rotacion* que se pondrá a 1 -durante un ciclo de reloj- cuando *contador_rotacion* desborda y pasa a 0. Habrá que indicarle al simulador que muestre esta señal interna.



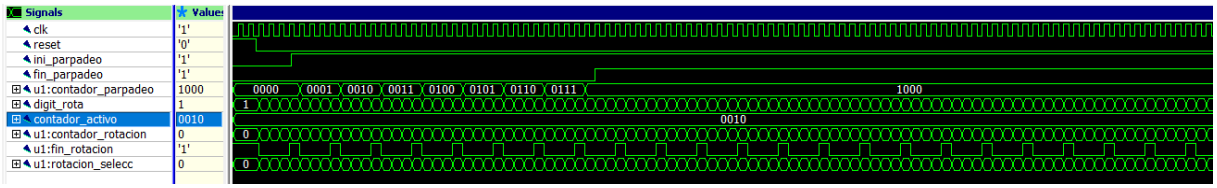
2.5.2. Iluminación temporizada (1.5 puntos)

La señal interna *fin_rotacion* da un pulso de un ciclo de reloj cada ~10,5ms (si *contador_rotacion* tiene 20 bits -para programar en la placa-: 2^{20}*Tclk= 1.048.576 * 10ns= 10,48ms). Se utilizará esta señal como habilitador para un segundo contador de 10 bits. Este contador parará cuando su bit más significativo llegue al valor '1' (bit inútil para la cuenta). Este contador contará con 9 de sus 10 bits (2^9=512). Si se multiplica por 10,5ms, se tienen 5,4 segundos (512*10,5ms). Este será el tiempo que el display seleccionado -para operar con él- pueda parpadear en la placa.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity rotacion is
        GENERIC (
            N20: INTEGER := 20;    -- 20 bits del contador para la rotacion
            N10: INTEGER := 10;    -- 10 bits del contador para el parpadeo
            NP : integer :=  2     -- constante para el cálculo de parpadeos
        );
    Port ( clk,reset       : in  STD_LOGIC;
           contador_activo : in  STD_LOGIC_VECTOR (3 downto 0);
           ini_parpadeo    : in  STD_LOGIC;
           fin_parpadeo    : out STD_LOGIC;
           digit_rota      : out STD_LOGIC_VECTOR (3 downto 0)
         );
end rotacion;

architecture rotacion_arq of rotacion is
    signal contador_rotacion : unsigned(N20-1 downto 0);  -- unsigned(19 downto 0)
    signal contador_parpadeo : unsigned(N10-1 downto 0);  -- unsigned(10 downto 0)
    signal fin_rotacion      : std_logic;
    SIGNAL rotacion_selecc : UNSIGNED(1 DOWNTO 0);
begin
```

Mientras la entrada *ini_parpadeo* se mantenga a '0', la señal *contador_parpadeo* se mantendrá a 0. Si *ini_parpadeo* vale 1, la señal *contador_parpadeo se incrementará* cada 10,5ms hasta que su bit más significativo se ponga a 1. Este contador se incrementa cada vez que coincide un flanco de subida de reloj (clk) y el pulso de la señal *fin_rotacion* (de un ciclo de reloj cada 10,5ms) que actúa de enable para el contador. Cuando el bit más significativo de *contador_parpadeo* se pone a 1, entonces la salida *fin_parpadeo* se pone a 1. El bit más significativo de una señal de tamaño parametrizado se consigue de la siguiente manera: if (*contador_parpadeo(contador_parpadeo'high='1'))*.

En la siguiente figura se muestra una simulación -poco detallada- en el que los parámetros se han modificado (N20=2, N10=4) en el fichero de estímulos. Se aprecia como la señal *contador_parpadeo* para de contar cuando el bit más significativo vale 1 (*contador_parpadeo= 1000).*



Se pide realizar la simulación que se muestra pero solo hasta que la señal *fin_parpadeo* se pone a 1. El valor de contador_activo=2 y la constante NP=2 son irrelevantes, por ahora. Las señales *contador_rotacion, rotacion_selecc y digit_rota* se mostrarán en formato binario.

2.5.3. Iluminación completa (1 punto)

La señal *digit_rota*[] cambia de forma continua (0001->0010->0100->1000) seleccionando uno de los 4 displays para que se ilumine (el resto apagados). En este apartado se hará parpadear al display seleccionado por la entrada *contador_activo.* La duración del parpadeo va desde que *contador_parpadeo* vale 0 hasta que

contador_parpadeo para con su bit más significativo a 1 (ver figura anterior y la tabla siguiente).

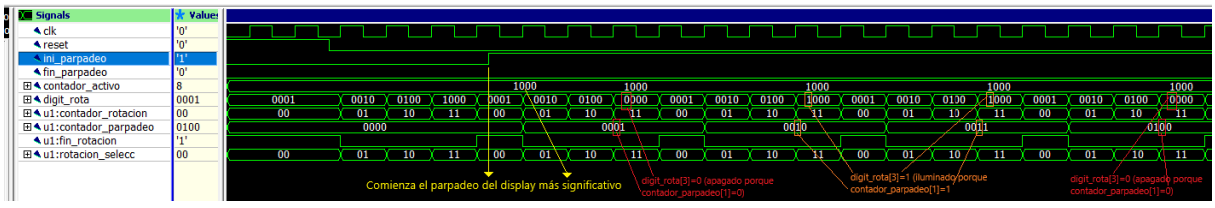
Si *contador_activo*=1000 y *digit_rota*=1000, el display de las unidades de millar se iluminará o apagará según uno de los bits de la señal *contador_parpadeo*. Si utilizamos el segundo bit por la izquierda (NP=**2** -> [3-**2**]=[**1**]), se puede ver los momentos en que se ilumina o apaga ese display para el caso N20=2 y N10=4 bits, respectivamente.

contador_parpadeo[]				
[3]	[2]	[1]	[0]	
0	0	0	0	Apagado
0	0	0	1	
0	0	1	0	Encendido
0	0	1	1	
0	1	0	0	Apagado
0	1	0	1	
0	1	1	0	Encendido
0	1	1	1	
1	0	0	0	Fin parpadeo

En el fichero de diseño, se incluirá un if que apague intermitentemente -durante un tiempo determinado- el display seleccionado. Así se consigue el efecto de parpadeo.

```
rotacion_selecc(1 DOWNTO 0) <=  contador_rotacion(contador_rotacion'high downto contador_rotacion'high-1);

PROCESS (rotacion_selecc)
BEGIN
      case rotacion_selecc is
         when "00" =>
            digit_rota  <= "0001";    -- por defecto el display se iluminará si no se entra en el if

            -- si el displays coincide con el seleccionado (digit_rota=contador_activo, si ini_parpadeo=1 y si el bit
            -- dado por (contador_parpadeo'high-NP) vale '0' -> el display no se ilumina
            IF (xxx and yyy and zzz) THEN
                digit_rota <= "0000";
             end if;
         when "01" =>
            digit_rota  <= "0010";
            IF (xxx and yyy and zzz) THEN
                digit_rota <= "0000";
             end if;
         when "10" =>
            digit_rota  <= "0100";
            IF (xxx and yyy and zzz) THEN
                digit_rota <= "0000";
             end if;
         when others =>
            digit_rota  <= "1000";
            IF (xxx and yyy and zzz) THEN
                digit_rota <= "0000";
             end if;
         end case;
 END PROCESS;
```

El resultado de la simulación con N20=2, N10=3 y NP=2 es el siguiente:



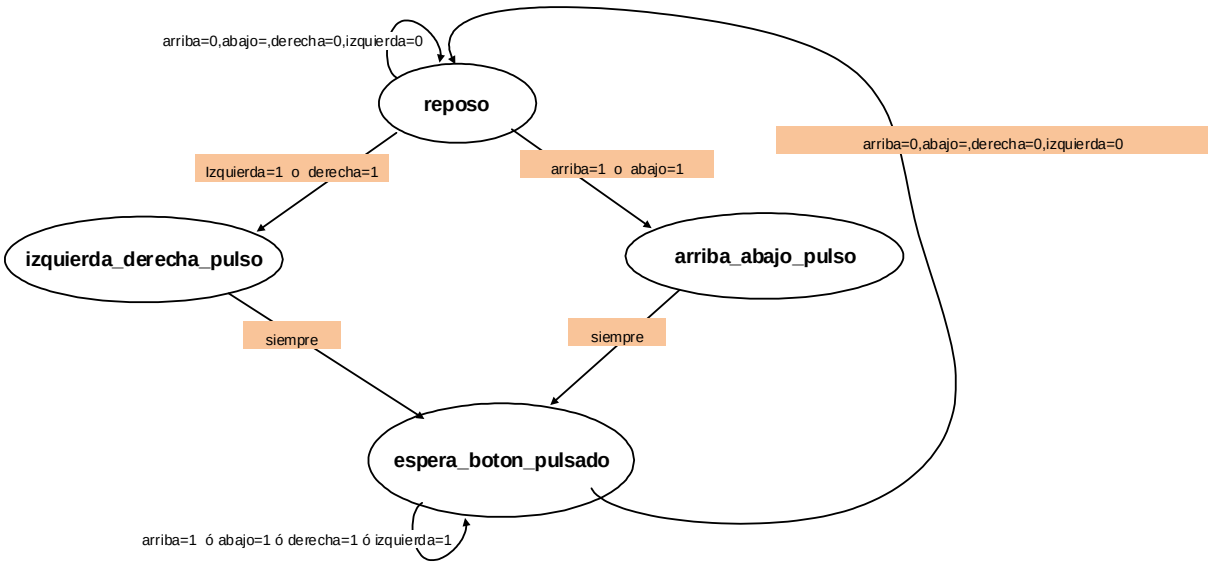
Se pide simular el circuito con N20=2, N10=4 y NP=3.

2.6. Control del sistema (control.vhd).

2.6.1. Control básico del sistema. (1.5 puntos)

El diagrama de estados se muestra en la figura. Mientras no se pulse ningún botón, el sistema permanece en el estado de *reposo*. Si se pulsa un botón irá a un estado transitorio que solo dura un ciclo de reloj (entre dos flancos de subida). Los estados

izquierda_derecha_enable y arriba_abajo_enable no tienen un arco para manterse en dicho estado, como ocurre con *reposo* y *espera_pulsado*. Si en

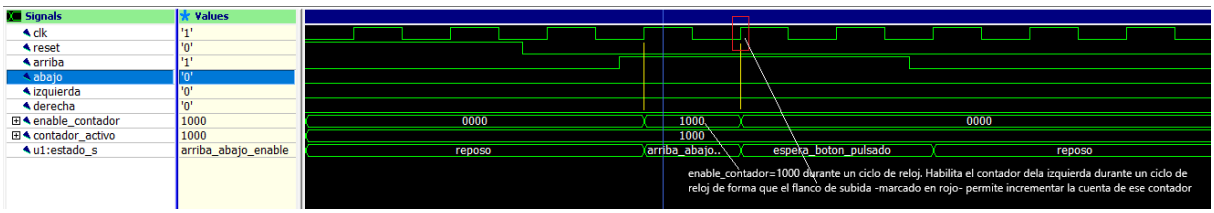


Si se pulsa derecha o izquierda, este módulo desplazará una posición –en el sentido apropiado- un registro interno (*regDesplazaDisplayActivo*) cuyo valor de reset será 1000.

El desplazamiento hay que hacerlo en un proceso que tenga un flanco de subida de reloj, para que los biestables del registro cambien en sincronismo con el reloj. Se preguntará si la señal *estado_s=izquierda_derecha_enable* ya que *estado_s* dura un ciclo de reloj y, por tanto, sirve de enable para dejar pasar un flanco de subida del reloj al registro de desplazamiento. El contenido de este registro (*regDesplazaDisplayActivo)* indica en todo momento cual es el dígito que responderá a los pulsadores 'arriba' o 'abajo', para incrementar o decrementar su valor. El valor de este registro se asignará a la salida *contador_activo* para conocer su estado y depurar el sistema.

Si se pulsa arriba o abajo, este módulo enviará una señal de enable de un ciclo de reloj al contador correspondiente para que se incremente o decremente. Esto hay que hacerlo en un proceso síncrono (dentro del flanco de subida del reloj) y dentro de un if en el que se preguntará si *estado_s = arriba_abajo_enable* para garantizar que solo hay un flanco de subida de reloj para que el incremento/decremento sea de 1. Este módulo pondrá en su salida *enable_contador(3:0)* el valor del registro interno *regDesplazaDisplayActivo pero* Uno de los 4 bits de este registro se activará solo durante 1 ciclo de reloj (pulso). Ese bit llegará a la señal de enable de un contador (ver esquema inicial –completo.

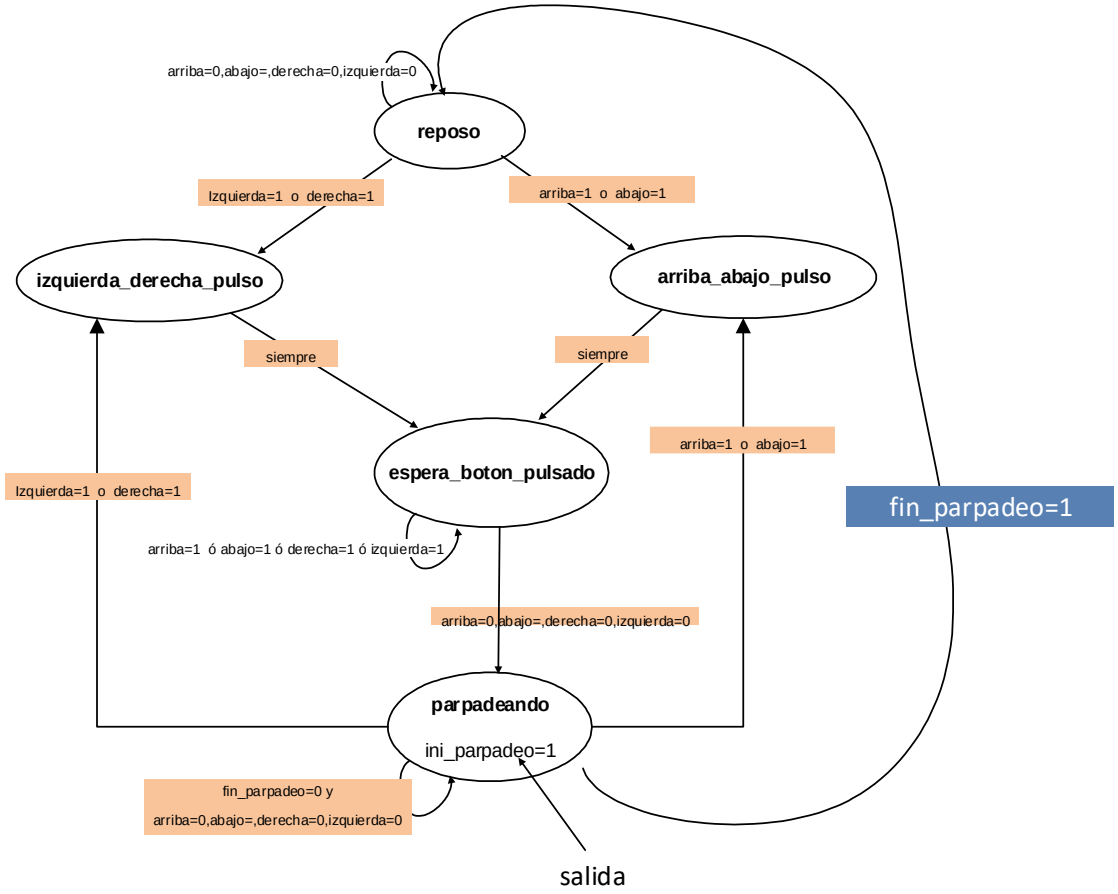
En la siguiente simulación se pulsa el botón arriba. La señal *contador_activo* señala al displays de las unidades de millar (1000) por lo que la señal *enable_contador* se pondrá a 1000 durante un ciclo de reloj para que el contador de este dígito se incremente en 1. El resto del tiempo esta señal (*enable_contador)* permanece a 0. La entrada *fin_parpadeo* y la salida *ini_parpadeo* no se utilizan en esta versión básica.



Se pide simular un pulso en la entrada *izquierda* y, al cabo de unos ciclos, un pulso en la entrada derecha. Esto hará que el registro de desplazamiento rote en un sentido y el contrario de forma que la salida contador_activo también lo hará.

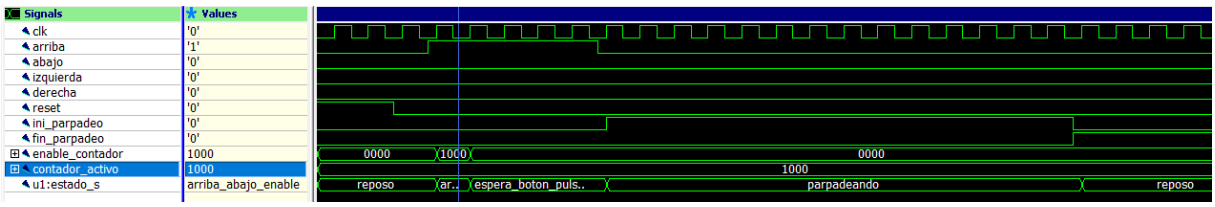
2.6.2. Control con parpadeo. (1.5 puntos)

Al control básico se le añadirá un estado de espera mientra el display activo esté parpadeando durante 5,4 segundos debido a que el usuario ha pulsado un botón. En realidad el parpadeo comienza cuando deja de pulsar el botón. La señal *ini_paradeo* irá al módulo de rotación que devuelve la señal *fin_parpadeo*.



Cuando el autómata activa la señal *ini_parpadeo=1* -en el estado *parpadeando-,* el módulo de rotación comienza a hacer parpadear el *contador_activo*. Cuando ha pasado el tiempo del parpadeo, pone la señal *fin_parpadeo=1* que es una entrada para este bloque y habrá que asignarle valores en el fichero de estímulos.

Se muestra la simulación de la pulsación del botón *arriba*. Activa la señal *enable_contador* durante un ciclo de reloj (estado arriba_abajo_pulso) y, cuando se deja de pulsar, pasa al estado *parpadeando.* Este termina cuando recibe la entrada fin_parpadeo con el valor '1'.



Se pide simular las pulsaciones consecutivas de los botones izquierda y abajo.

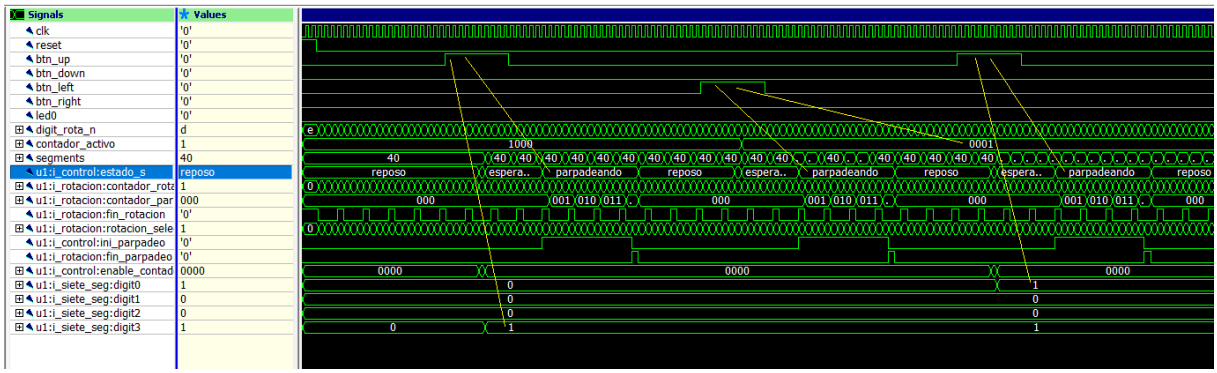
2.7. Diseño completo. (proyecto.vhd) (1 punto)

El fichero proyecto.vhd permite cablear todos los bloques que aparecen en la primera figura de este fichero. Su entidad contiene las entradas y salidas externas. Los cables

internos que unen unas cajas con otras se programan con variables internas de la arquitectura. Esta parte del diseño es mecánica y se deja resuelta. Lo único que falta es copiar todos los ficheros simulados en los apartados anteriores (rebotes.vhd, control.vhd, siete_seg.vhd, …) en la carpeta '2.7.proyecto'. Así podéis compilar todo el proyecto completo y simularlo.

También se deja el fichero de estímulos en el que se han adaptado los GENERICS a valores pequeños para que quepa la simulación en la pantalla: N21=2, N20=2, N10=3 y NP=2. El resultado que me ha salido a mí lo tenéis en la siguiente imagen pero es mucho mejor abrir el fichero de curvas que os dejo y hacer zoom donde os convenga para ver con detalle cómo funciona.

En esta simulación se puede ver que se han pulsado 3 botones: btn_up, btn_left y btn_up. El primer botón hace que el contador dado por la señal *contador_activo* se incremente en 1. El segundo botón rota a la izquierda la señal *contador_activo* de forma que el contador sobre el que actuarán los botones arriba y abajo será el menos significativo. Al pulsar nuevamente sobre el botón btn_up se aprecia que ahora se incrementa el nuevo contador (display) seleccionado.
La pulsación de cualquiera de los 3 botones saca al autómata del estado de reposo y vemos que todos ejecutan el estado 'parpadeando' donde la señal *ini_parpadeo* la pone a 1 el autómata y cuando recibe la señal *fin_parpadeo='1' (del módulo de rotación),* pon la señal *ini_parpadeo a '0'.* El módulo de rotación poniendo, a su vez, la señal *fin_parpadeo* también a 0.



Se pide implementar el diseño completo en Vivado y dejar en la documentación el fichero 'bitstream' generado. El fichero de restricciones lo tenéis en la documentación.

3. Observaciones

- Fecha de finalización: 24 **de abril a las 14h**.
- Se evaluará la presentación del fichero PDF así como el código. Se penalizará si el código no está bien comentado ni indentado.
- El diseño de trabajo es secuencial. La realización de un apartado depende de lo realizado anteriormente. En los apartados se dejan plantillas con los nuevos elementos que se deben añadir pero el diseño deber realizarse sobre los ficheros diseñados con los apartados anteriores.
- Los diez primeros trabajos serán bonificados con un 10% extra de nota. Si la nota supera los 10 puntos, saturará a este valor. Solo se permite subir los trabajos una sola vez a la enseñanza virtual. Hay que estar seguro de lo que se envía.
- El trabajo se subirá a la EV con el título "TRABAJO GRUPO N". Se adjuntará un fichero comprimido ZIP. Se incluirá la misma estructura de la plantilla sustituyendo el fichero MEMORIA_trabajo1.docx por el fichero MEMORIA_trabajo1.pdf obtenido a partir del anterior. El fichero comprimido se llamará trabajo_grupo_N.zip siendo N el número del grupo. En las carpetas de cada apartado debe estar el fichero con las curvas para poder visualizarlas con el simulador Simili.
- Todos los miembros del grupo deben subir el trabajo para que el sistema permita evaluarlos. En la primera página del fichero MEMORIA_trabajo1.pdf se indicará el nombre de todos los miembros del grupo de forma clara. La fecha de entrega se considerará la del último miembro del grupo que suba la práctica.
- Si el trabajo se entrega fuera de plazo se penalizará con un 10% por cada día de retraso respecto a la fecha límite.
- Cualquier explicación, comentario o sugerencia por parte del profesorado será universal a través del foro. No se resolverán dudas en el despacho de los profesores porque podrían suponer una ventaja respecto al resto de grupos. Tanto alumnos como profesores pueden contestar a cualquier cuestión planteada en el foro. El foro será anónimo y un usuario puede borrar sus mensajes si no han tenido respuesta.
- El tamaño apropiado de los grupos será de 3 miembros pero se permitirán grupos 2. El trabajo no es una actividad individual. En el proyecto docente consta como un trabajo en equipo. Si alguien no encuentra compañeros para el trabajo, enviadme un correo y yo os agrupo.
- El trabajo se puede probar en clases de prácticas a partir de la semana del 25 de marzo. El profesor llevará placas que dejará a los grupos que quieran hacer pruebas para el último apartado.
- En el proyecto de vivado no se modifican los valores de los GENERICs (N21=21, N20=20, N10=10 y NP=4).