A6 Apéndice al capítulo 23

Ejemplos de diseño con V H D L

Este apéndice ofrece una serie de ejemplos de diseño de circuitos de control descritos en VHDL. Los seis primeros se refieren a pequeños sistemas, basados en la utilización de contadores, a saber:

- temporización de la luz de una escalera,
- control de anchura de pulsos de entrada,
- supervisión de velocidad de un motor,
- comparación de anchura de pulsos sucesivos,
- generación de **n** pulsos, siendo **n** la medida del pulso de entrada,
- y pulsos de anchura progresivamente creciente/decreciente.

En estos ejemplos, como referencia para el texto VHDL y, también, como ejercicio de diseño modular, se incluye y se comenta el correspondiente diagrama de bloques.

Otros tres diseños se refieren a máquinas algorítmicas descritas en el capítulo 24:

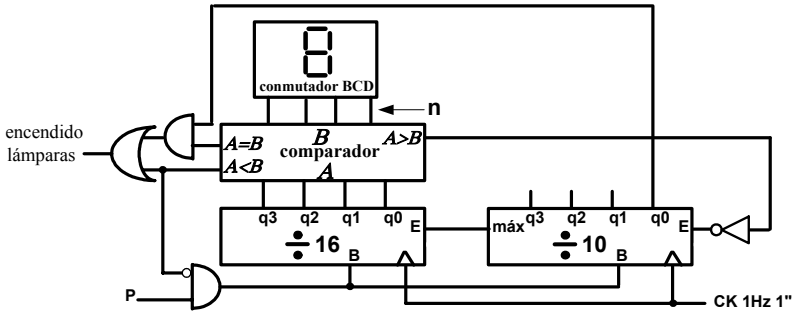
- producto de dos números de 64 dígitos binarios (apartado 24.3.),
- raíz cuadrada de un número de 64 dígitos (ejemplo 24.4.2.),

y, por último, se desarrollan dos diseños más amplios, relativos a un cronómetro controlado mediante dos pulsadores (ejemplo 24.4.5. del capítulo 24) y al control de una encimera de cocina vitrocerámica con cuatro «fuegos».

[Los diez diseños de este apéndice han sido comprobados, mediante compilación y simulación del texto VHDL con la herramienta MAX+plus II de ALTERA.]

1. Temporización de la luz de una escalera

Un conmutador BCD controla las lámparas de iluminación de la escalera de una comunidad de vecinos; de forma que, si el número fijado en el conmutador BCD es **n**, las lamparas deben mantenerse encendidas durante **n** . 10 segundos cada vez que se pulsa un interruptor **P**; durante los 10 segundos siguientes, las lamparas deben apagarse y encenderse 5 veces sucesivas, con intervalos de 1 segundo. Cuando las lámparas están encendidas, si se pulsa **P** no hace nada, pero al hacerlo durante el intervalo de intermitencia, o posteriormente al mismo, se reinicia el ciclo de encendido.



Tomás Pollán Santamaría. Tecnología Electrónica. E.U.I.T.I.Z. Universidad de Zaragoza. tpollan@posta.unizar.es

A partir de un reloj de 1 Hz (1 segundo de período), un contador década determina ciclos de 10" y un segundo contador realiza el contaje de dichos ciclos y la comparación con el número **n** fijado en el conmutador BCD. Mientras el segundo contador se encuentra por debajo de **n** (**n** estados, de 0 a n-1) las lámparas se mantienen encendidas; cuando ambos números son iguales, se produce la intermitencia (10") y, al superar el contador el valor **n** se detiene el contaje para evitar que «se dé la vuelta» (vuelva a 0) e inicie un nuevo ciclo de iluminación.

Para la intermitencia se aprovecha la primera salida (unidades) del primer contador, cuyo semiperíodo (tiempo en **1** y tiempo en **0**) es 1 segundo. El pulsador borra ambos contadores para comenzar una nueva temporización, pero no lo hace cuando las lámparas se encuentran en encendido fijo.

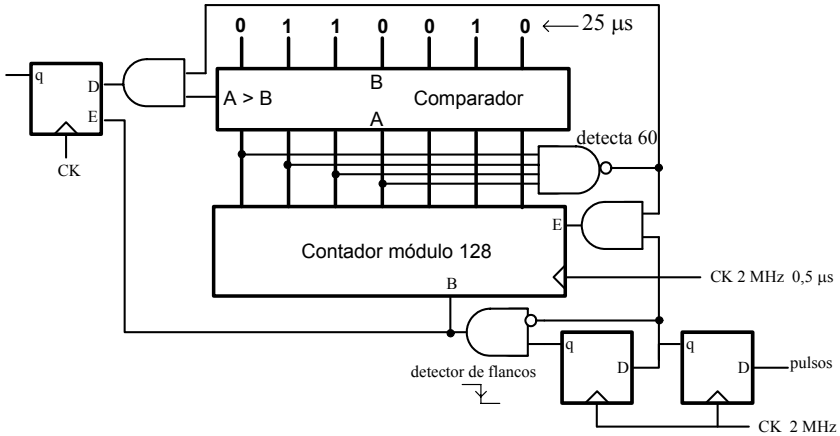
```
library ieee;   use ieee.std_logic_1164.all;   use ieee.std_logic_unsigned.all;

entity ESCALERA is
    port (  CK, RS, P      : in    std_logic;
            conmutador     : in std_logic_vector(3 downto 0);
            luces          : out  std_logic);
end ESCALERA;

architecture TEMPORIZADOR of ESCALERA is
signal cont_1, cont_2              : std_logic_vector(3 downto 0);
-- el contador cont_1 divide por 10: pasa del reloj de 1" a 10"
-- cont_2 cuenta intervalos de 10" (cuando cont_1 está en 9)

begin

-- si cont_2 < conmutador BCD, luz fija; cuando son iguales, luz intermitente
-- la intermitencia se hace con cont_1(0): semiperíodo 1"
luces <= '1' when (cont_2 < conmutador) or ((cont_2 = conmutador) and (cont_1(0) = '1'))
          else '0';

TEMPORIZACION: process
begin                wait until CK = '1';

-- inicialización RS y activación de la temporizción P
if (( RS = '1' ) or ((P = '1') and (cont_2 >= conmutador))) then     cont_2 <= "0000";
                                                                     cont_1 <= "0000";

-- contaje módulo 10 para cont_1 y hasta cont_2 > conmutador BCD para ambos
elsif (cont_2 <= conmutador) then
      if (cont_1 = "1001") then          cont_2 <= cont_2 + 1;     cont_1 <= "0000";
            else                         cont_1  <= cont_1  +1;    end if;
end if;              end process;
end TEMPORIZADOR;
```

2. Control de anchura de pulsos de entrada

Un circuito posee una única línea de entrada que recibe pulsos de diferente duración y presenta una sola línea de salida que debe situarse a **1** si la anchura del último pulso recibido es mayor que 25 µs y menor de 60 µs, con una precisión de 0,5 µs: la salida se actualiza cuando finaliza cada pulso (flanco de bajada).

Se requiere un reloj de 2 MHz para que la precisión sea de 0,5 µs y con dicho reloj:

25 µs = **11001** µs = **110010** x 0,5 µs 60 µs = **111100** µs = **1111000** x 0,5 µs



Un contador «mide» la anchura del pulso de entrada con el reloj de 2 MHz (0,5 µs) y la compara con 25 µs (comparador) y con 60 µs (puerta *Nand*); cuando llega a 60 el contador se detiene para que no «se dé la vuelta» (vuelva a 0) e inicie un nuevo contaje. La comparación se ejecuta (sobre el biestable de salida) al finalizar el pulso de entrada (detección del flanco de bajada) y, a la vez, se borra el contador para la medida del siguiente pulso.

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

entity ANCHURA is   port ( CK, RS, P   : in   std_logic; Y   : out std_logic);
end ANCHURA;

architecture MEDIR of ANCHURA is
signal cont          :std_logic_vector(6 downto 0);
-- cont mide la anchura del pulso; se detiene si llega a 60 µs
-- se borra al detectarse el flanco de bajada

signal detector      :std_logic_vector(2 downto 1);
signal flanco        :std_logic;
-- «detector» son dos biestables para detectar el flanco de bajada:
-- cuando detector(2) = 1 y detector(1) = 0, entonces flanco = 1
```



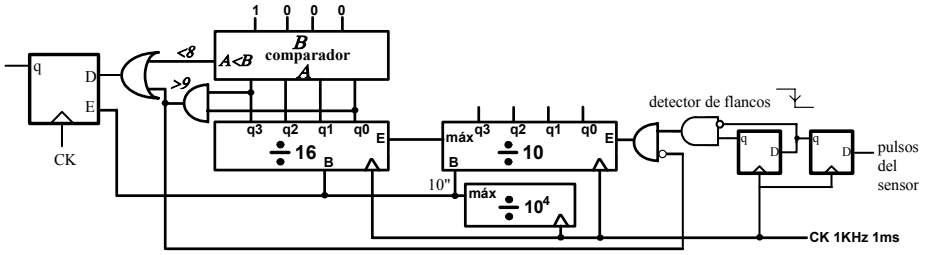
```
signal yy            :std_logic;
-- yy es el biestable de salida que se activa cuando flanco = 1 y 25 < contador < 60

begin

Y <= yy;     flanco <= detector(2) and not detector(1);

RELOJ: process
begin        wait until CK = '1';
detector(1) <= P;                  detector(2) <= detector(1);

-- medida de la anchura de pulso
if (( RS = '1' ) or (flanco = '1')) then                          cont <= "0000000";
elsif (cont < "1111000") and (detector(1) = '1')  then            cont <= cont + 1;
end if;

-- ejecución de las comparaciones para dar lugar a la salida
if flanco = '1' then
      if (cont >= "0110010") and (cont < "1111000") then          yy <= '1';
                  else                                            yy <= '0';    end if;
end if;
end process;
end MEDIR;
```

3. Supervisión de velocidad de un motor

Un motor debe girar con un mínimo de 480 r.p.m. hasta un máximo de 540 (que no debe ser alcanzado); el circuito que vigile la velocidad del motor recibirá pulsos de un sensor que detecta las vueltas del motor y debe proporcionar una señal de alarma cuando se encuentre fuera de dicho intervalo. Se utilizará una frecuencia de reloj de 1 KHz.

Corresponde a un frecuencímetro con una sola salida (de alarma), referida a la comparación de la medida de frecuencia con las dos frecuencias extremas; tomando un tiempo de medida de 10 segundos:

480 r.p.m. = 80 rev. en 10" = **1000** 540 r.p.m. = 90 rev. en 10" = **1001**



El contaje de pulsos se hace en BCD y se compara con 80 y con 90; cuando llega a 90 se detiene para evitar que el contador «se dé la vuelta». El intervalo de medida (10") se obtiene dividiendo por 10^4 la frecuencia de reloj (1 KHz); al final de dicho intervalo se ejecutan las comparaciones (biestable de salida) y se borra el contador para iniciar una nueva medida.

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

entity MOTOR is    port (  CK, RS, S    : in    std_logic;      Y    : out  std_logic);
end MOTOR;

architecture FRECUENCIA of MOTOR is

-- división de frecuencia por 10; se utiliza 4 veces (por ello se define como función)
function mod10 (e :std_logic; qq :std_logic_vector(3 downto 0)) return std_logic_vector is
  variable cont:std_logic_vector(3 downto 0);
  begin
        if e = '1' then
        if qq = "1001" then cont := "0000"; else cont := qq + 1; end if;
        else cont := qq;    end if;
return cont;   end;

signal cont1,cont2                          :std_logic_vector(3 downto 0);
-- cont1 es el contador módulo 10 y cont2 es el siguiente (módulo 16)

signal q4, q3, q2, q1                       :std_logic_vector(3 downto 0);
signal max4, max3, max2, max1,diezseg       :std_logic;
-- q y max serán las salidas de los cuatro contadores divisores por 10
-- diezseg es la onda de temporización de 10" (corresponde a max4)

signal detector     :std_logic_vector(2 downto 1);
signal flanco       :std_logic;
-- detector son dos biestables para detectar el flanco de bajada:
-- cuando detector(2) = 1 y detector(1) = 0, entonces flanco = 1

signal yy           :std_logic;
-- yy es un biestable que se activa cuando diezseg = 1 y contador < 80 o ≥ 90)

begin

Y <= yy;     flanco <= detector(2) and not detector(1);

      max1 <= '1' when q1 = "1001" else '0';
      max2 <= '1' when (q2 = "1001") and  max1 = '1' else '0';
      max3 <= '1' when (q3 = "1001") and  max2 = '1' else '0';
      max4 <= '1' when (q4 = "1001") and  max3 = '1' else '0';

      diezseg <= max4;
```

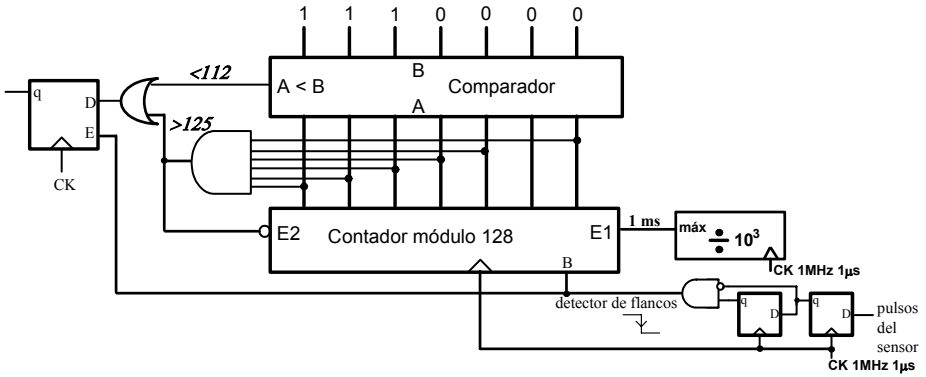


```
RELOJ: process
begin        wait until CK = '1';
detector(1) <= S;                  detector(2) <= detector(1);

-- divisor de frecuencia por 10^4: 4 contadores módulo 10
      q1 <=  mod10('1', q1);                q2 <=  mod10(max1, q2);
      q3 <=  mod10(max2, q3);               q4 <=  mod10(max3, q4);

-- contaje de pulsos del sensor de vueltas del motor
if (( RS = '1' ) or (diezseg = '1')) then   cont1 <= "0000";     cont2 <= "0000";
elsif (flanco = '1') and (cont2 < "1001")  then
            if cont1 = "1001"  then     cont1 <= "0000";     cont2 <= cont2 +1;
            else                        cont1 <= cont1 +1;                end if;
end if;

-- ejecución de las comparaciones al finalizar el intervalo de medida
if diezseg = '1' then
      if (cont2 < "1000") or (cont2 >= "1001") then         yy <= '1';
                  else                                      yy <= '0';     end if;
end if;             end process;       end FRECUENCIA ;
```

La solución anterior tiene un tiempo de respuesta de 10 segundos (lo cual puede ser adecuado para un operador humano); en caso de que la información respecto a la velocidad del motor fuera comunicada a un microprocesador de control, es posible que interesase un tiempo de medida menor.

Pueden conseguirse tiempos mínimos midiendo el período de los pulsos del sensor; es decir, contando el tiempo entre dos pulsos sucesivos, en lugar de contar pulsos en una unidad de tiempo: 1/480 r.p.m. = 60 /480 = 125 ms = **1111101**
1/540 r.p.m. = 60 /540 = 111,1 ms ; 112 ms = **1101111**.

En este caso, para mejorar la resolución, se utiliza un reloj de 1 MHz.



Ahora lo que se cuentan son milisegundos (se mide el período) y dicho contaje se hace directamente en binario y se compara con 125 y con 111; cuando se alcanza el 125 se detiene el contador para evitar que «se dé la vuelta». Las comparaciones se ejecutan al finalizar cada vuelta (flanco de bajada de los pulsos del sensor) y, a la vez, se borra el contador para un nuevo contaje.

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

entity MOTOR2 is   port (  CK, RS, S   : in   std_logic;      Y    : out  std_logic);
end MOTOR2;

architecture PERIODO of MOTOR2 is

function mod10 (e :std_logic; qq :std_logic_vector(3 downto 0)) return std_logic_vector is
  variable cont:std_logic_vector(3 downto 0);
  begin
        if e = '1' then
        if qq = "1001" then cont := "0000"; else cont := qq + 1; end if;
        else cont := qq;    end if;
return cont;   end;

signal contador                       :std_logic_vector(6 downto 0);

signal q3, q2, q1                     :std_logic_vector(3 downto 0);
signal max3, max2, max1, miliseg      :std_logic;

signal detector                       :std_logic_vector(2 downto 1);
signal flanco                         :std_logic;

signal yy                             :std_logic;

begin

Y <= yy;      flanco <= detector(2) and not detector(1);

      max1 <= '1' when q1 = "1001" else '0';
      max2 <= '1' when (q2 = "1001") and  max1 = '1' else '0';
      max3 <= '1' when (q3 = "1001") and  max2 = '1' else '0';          miliseg <= max3;

RELOJ: process
begin        wait until CK = '1';
detector(1) <= S;                 detector(2) <= detector(1);
   q1 <=   mod10('1', q1);        q2 <=   mod10(max1, q2);    q3 <=   mod10(max2, q3);

-- contaje de milisegundos
if (( RS = '1' ) or (flanco = '1')) then                    contador <= "0000000";
elsif (miliseg = '1') and (contador < "1111101")  then      contador <= contador +1;
end if;
```

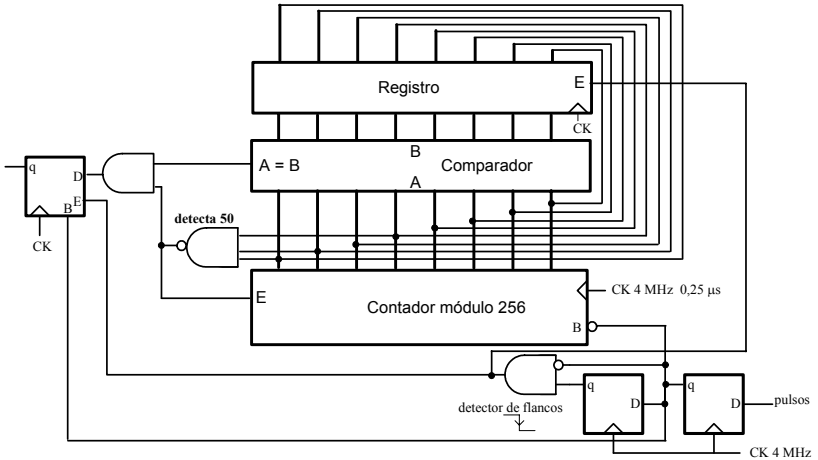


```
-- ejecución de las comparaciones al finalizar cada vuelta del motor
if flanco = '1' then
      if (contador < "1110000") or (contador >= "1111101") then    yy <= '1';
                  else                                             yy <= '0';    end if;
end if;             end process;       end PERIODO ;
```

4. Comparación de anchura de pulsos sucesivos

Interesa disponer de un circuito con una entrada por la que recibirá pulsos de duración variable y una salida que debe avisar cuando los dos últimos pulsos recibidos sean iguales. El circuito será síncrono con un reloj de 4 MHz y el aviso (salida a **1**) se producirá cuando finalice cada pulso hasta que llegue el nuevo pulso; pulsos de duración igual o superior a 50 µs no son tenidos en cuenta.

Con un reloj de 4 MHz (0,25 µs): 50 µs = **110010** µs = **11001000** x 0,25 µs.



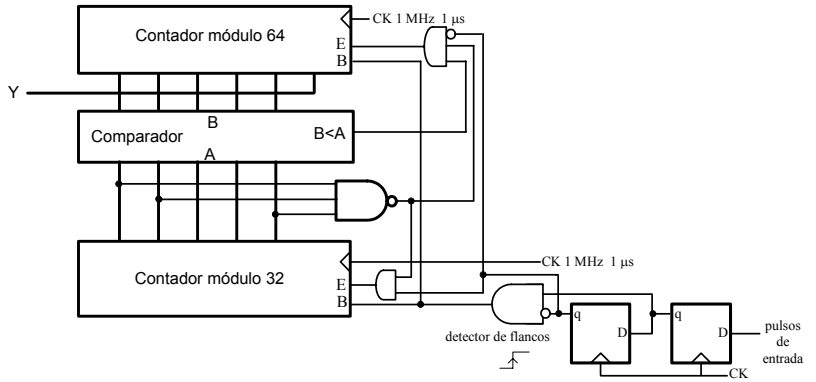
El contador «mide» la anchura de los pulsos y el registro almacena la medida del pulso anterior. La comparación entre contador (pulso de entrada) y registro (pulso anterior) y el almacenamiento en el registro se ejecutan cuando acaba cada pulso (detección de su flanco de bajada). El contador permanece borrado mientras no hay pulso de entrada y, en cambio, el biestable de salida se borra cuando existe pulso de entrada.

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

entity COMPARACION is
port (  CK, RS, P    : in    std_logic;  Y    : out  std_logic);
end COMPARACION ;
```

```
architecture ANCHURA of COMPARACION is

signal contador, registro  :std_logic_vector(7 downto 0);
-- contador y registro son de 8 bits (para abarcar  los 50 µs, medidos con 4 MHz)

signal detector      :std_logic_vector(2 downto 1);
signal flanco        :std_logic;
-- detector son dos biestables para detectar el flanco de bajada:
-- cuando detector(2) = 1 y detector(1) = 0, entonces flanco = 1

signal yy            :std_logic;
-- yy es el biestable de salida

begin

Y <= yy;      flanco <= detector(2) and not detector(1);

RELOJ: process
begin         wait until CK = '1';
detector(1) <= P;                  detector(2) <= detector(1);

-- medida de la anchura de pulso: máximo 50 µs
if (( RS = '1' ) or (detector(1) = '0')) then                  contador <= "00000000";
   elsif contador < "11001000" then                            contador <= contador + 1;
end if;

-- señal de salida: ejecución de la comparación al finalizar un pulso
if detector(1) = '1' then                                          yy <= '0';
   elsif flanco = '1' then
      if (contador < "11001000") and (contador = registro) then yy <= '1';
                  else                                           yy <= '0';    end if;
end if;

-- paso de la medida del pulso al registro para la comparación con el siguiente pulso
if flanco = '1' then    registro <= contador;
end if;
end process;
end ANCHURA;
```


5. Generación de n pulsos

Un sistema síncrono con un reloj de 1 MHz recibe pulsos de diferente duración, separados, al menos, 50 µs; el sistema mide la anchura del pulso recibido (**n** µs) y, si **n** es inferior a 25 µs, genera **n** pulsos (tantos como la propia duración del pulso de entrada) de 1 µs, separados 1 µs cada uno del siguiente.



La primera salida $\mathbf{q_0}$ de un contador completo genera pulsos cuyos semiperíodos (tiempo en **1** y en **0**) son iguales al período del reloj; las salidas siguientes del mismo contador $\mathbf{q_5 \ldots q_1}$ «cuentan» el número de dichos pulsos.

De esta forma, un contador (módulo 32) mide la anchura de los pulsos de entrada y otro contador (módulo 64) genera los pulsos de salida ($\mathbf{q_0}$) y los cuenta. Ambos contadores se borran al inicio de un nuevo pulso de entrada (detección de flanco de subida). La condición para que se generen los pulsos de salida (habilitación del segundo contador) es que haya finalizado el pulso de entrada, que su anchura haya sido inferior a 25 µs y que el número de pulsos de salida sea aún menor que la anchura del pulso de entrada (comparador).

25 µs = **11001** µs.

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

entity PULSOS is

port (  CK, RS,P    : in    std_logic;  Y    : out  std_logic);
end PULSOS;

architecture GENERAR of PULSOS is

signal cont1          :std_logic_vector(4 downto 0);
-- cont1 mide la anchura del pulso de entrada; se detiene si llega a 25 µs

signal cont2          :std_logic_vector(5 downto 0);
-- cont2 genera los pulsos de salida y los cuenta
```

```
signal detector      :std_logic_vector(2 downto 1);
signal flanco        :std_logic;
-- detector son dos biestables para detectar el flanco de subida:
-- cuando detector(2) = 0 y detector(1) = 1, entonces flanco = 1

begin

Y <= cont2(0);      flanco <= not detector(2) and detector(1);

RELOJ: process
begin          wait until CK = '1';
detector(1) <= P;                    detector (2) <= detector(1);
if (RS = '1') or (flanco = '1') then          cont1 <= "00000";              cont2 <= "000000";

-- contaje del primer contador: medida de la anchura del pulso de entrada (máximo 25)
elsif detector(2) = '1' then
        if (cont1 < "11001") then          cont1 <= cont1 + 1;      end if;

-- segundo contador: generación de los pulsos de salida y contaje de los mismos
else    if (cont1 < "11001") and (cont2(5 downto 1) < cont1) then      cont2 <= cont2 + 1;
        end if;
end if;
end process;
end GENERAR ;
```

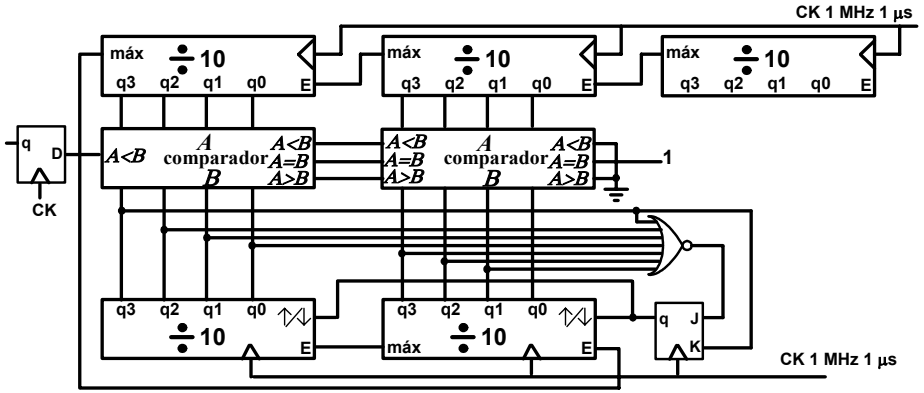
6. Pulsos de anchura progresivamente creciente/decreciente

Interesa un circuito digital que proporcione pulsos de amplitud modulada en la forma siguiente: un pulso cada milisegundo, el primero de ellos de 10 µs y los siguientes con anchura creciente de 20 µs, 30 µs, … (cada uno 10 µs más largo que el anterior), hasta alcanzar los 800 µs; luego los pulsos decrecerán (de 10 en 10 µs) hasta reducirse a 10 µs, momento en el cual volverán a crecer; y así sucesivamente.

Se trata de configurar un módulador de anchura de pulso: un contador que produzca el ciclo on/off y un comparador que compare el estado del contador con una referencia. Dicha referencia ha de ir creciendo, ciclo a ciclo, desde 10 µs hasta 800 µs (con un paso de 10 µs), mediante un contador bidireccional cuyo sentido de contaje sea controlado por un biestable; éste debe ponerse a **0** (borrarse) cuando la referencia llegue a 800 µs en el contaje ascendente y debe pasar a **1** (marcarse) cuando el «descontaje» llega a 10 µs.

Si ambos contadores (el del ciclo y el de la referencia) cuentan en BCD y se refieren a un intervalo base de 10 µs: 10 µs = **0000 0001** x 10 µs, 800 µs = **1000 0000** x 10 µs.

Con reloj de 1 MHz hará falta otro contador década para la temporización en 10 µs. El contador del ciclo PWM será habilitado por esta señal de 10 µs y de módulo 100 para que el ciclo sea de 1 ms (100 x 10 µs); el contador que genera la referencia se habilitará con la salida de máximo del anterior (que determina el final de cada ciclo).




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

entity PWM is    port ( CK, RS    : in    std_logic;  Y    : out  std_logic);
end PWM;

architecture VARIA of PWM is

-- contaje módulo 10, que puede ser bidireccional: hay 5 contadores década en el diseño
-- rs actúa como inicialización, dir como control arriba/abajo y e como habilitación
function mod10 (rs, dir, e :std_logic; qq :std_logic_vector(3 downto 0))
return std_logic_vector is
  variable contador :std_logic_vector(3 downto 0);
  begin
        if rs = '1' then contador := "0000";
        elsif e = '1' then
              if dir = '1' then   if qq = "1001" then contador := "0000";
                                  else contador := qq + 1;                  end if;
              else                if qq = "0000" then contador := "1001";
                                  else contador := qq - 1;                  end if;    end if;
        else contador := qq;    end if;
return contador;   end;

signal cont1, cont2, cont3, cont4, cont5       :std_logic_vector(3 downto 0);
signal max1, max2, ciclo, m_m4                 :std_logic;
-- contadores módulo 10: 1, 2 y 3 generan el ciclo PWM y 4 y 5 la referencia

signal dir, yy                                 :std_logic;
-- 2 biestables: dir controla el contaje up/down de la referencia; yy sincroniza la salida

signal uno                                     :std_logic;
-- señal para establecer un '1'
```

```
begin

Y <= yy;

uno <= '1';

        -- salidas para la interconexión de los 5 contadores
        max1 <= '1' when cont1 = "1001" else '0';
        max2 <= '1' when (cont2 = "1001") and  max1 = '1' else '0';
        ciclo  <= '1' when (cont3 = "1001") and  max2 = '1' else '0';
         -- "ciclo" señala el final del ciclo PWM ( corresponde a max3)

        m_m4 <= '1' when ciclo = '1' and (       ((cont4 = "1001") and (dir = '1'))
                                              or  ((cont4 = "0000") and (dir = '0'))  ) else '0';
        -- "m_m4" es la señal de máximo/mínimo de cont4, necesaria para habilitar cont5

RELOJ: process
begin
wait until CK = '1';

            -- inserción de los 5 contadores
            -- cont1, cont2 y cont3 que forman el contador de ciclo son ascendentes
            cont1 <=   mod10(RS, uno, uno, cont1);
            cont2 <=   mod10(RS, uno, max1, cont2);
            cont3 <=   mod10(RS, uno, max2, cont3);
            -- cont4 y cont5 que generan la referencia son bidireccionales (dir)
            cont4 <=   mod10(RS, dir, ciclo, cont4);
            cont5 <=   mod10(RS, dir, m_m4, cont5);

-- biestable que controla el sentido de variación de la referencia: contaje/descontaje
if (RS = '1') or (cont5 = "1000") then                  dir <= '0';
elsif (cont5 = "0000") and (cont4 <= "0001") then    dir <= '1';
end if;

-- modulación de anchura de pulso: comparación entre contador y referencia
if (cont3 < cont5) or ((cont3 = cont5)  and (cont2 < cont4)) then       yy <= '1';
                      else                                                          yy <= '0';
end if;
end process;
end VARIA;
```


7. Multiplicador de dos números de 64 dígitos

Sea un procesador dedicado específicamente a calcular el producto de dos números binarios **P** y **Q** de 64 dígitos, cuyo resultado **R** será un número binario de hasta 128 dígitos; corresponde al ejemplo de descomposición en parte operativa y de control desarrollado en el capítulo 24 (apartado 3, páginas 107 a 112).

El algoritmo aplicado consiste en recorrer sucesivamente los bits del multiplicador **Q** desde el menos significativo y, cuando el correspondiente bit de **Q** vale **1**, se suma el multiplicando **P** sobre el resultado (esta suma se efectúa sobre la parte más significativa del resultado); luego, se desplaza el resultado hacia la derecha (dividir por 2) para ajustar el valor relativo de cada suma parcial respecto a la siguiente a efectuar.

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

entity MULTIPLICADOR is
    port (    CK, RS, Init      : in    std_logic;
              P, Q              : in    std_logic_vector(63 downto 0);
              Dis               : out    std_logic;
              R                 : out  std_logic_vector(127 downto 0));
end MULTIPLICADOR;

architecture ALGORITMICA of MULTIPLICADOR is

subtype mis_estados is std_logic_vector(4 downto 1);
      -- codificación de los estados: código de un solo uno
      -- conforme a la máquina de estados de la página110
      constant reposo          : mis_estados :="0000";
      constant inicio          : mis_estados :="0001";
      constant intermedio      : mis_estados :="0010";
      constant sumar           : mis_estados :="0100";
      constant desplazar       : mis_estados :="1000";
signal estado   : mis_estados;

signal pp,qq        : std_logic_vector(63 downto 0);
-- registros para los operandos P y Q
signal rr           : std_logic_vector(127 downto 0);
-- registro para el resultado R
signal cont         : std_logic_vector(5 downto 0);
-- contador del número de bit del multiplicador

begin

R <= rr;
```

```
-- PARTE DE CONTROL: máquina algorítmica
CONTROL: process
begin       wait until CK = '1';
if ( RS = '1' ) then                                          estado <= reposo;
else   case estado is
            when reposo =>      if (Init = '1') then          estado <= inicio;       end if;
            when inicio =>                                    estado <= intermedio;
            when intermedio =>  if (qq(0) = '1') then         estado <= sumar;
                                elsif (cont = "111111") then  estado <= reposo;
                                      else                    estado <= desplazar;  end if;
            when sumar =>   if (cont = "111111") then         estado <= reposo;
                                    else                      estado <= desplazar;  end if;
            when desplazar =>                                 estado <= intermedio;
            when others =>
          end case;
end if;
end process;

-- PARTE OPERATIVA: operaciones correspondientes a los diversos estados
OPERACION: process
begin       wait until CK = '1';
Dis <= '0';   -- valor por defecto
if ( RS = '1' ) then  pp <= (others => '0');   qq <= (others => '0');    rr <= (others => '0');
                      cont <= (others => '0');
else      if estado = reposo then      Dis <= '1';                                  end if;
          if estado = inicio then      pp <= P;     cont <= (others => '0');
                                       qq <= Q;     rr <= (others => '0');       end if;
          if estado = sumar then       rr(127 downto 63) <= rr(127 downto 63) + pp;
                                                                                    end if;
          if estado = desplazar then   rr <= '0' & rr(127 downto 1);
                                       qq <= '0' & qq(63 downto 1);
                                       cont <= cont + 1;                          end if;
end if;
end process;
end ALGORITMICA ;
```


8. Raíz cuadrada de un número de 64 dígitos

El resultado **R** será un número binario de hasta 32 dígitos; el algoritmo de cálculo y la correspondiente máquina de estados se encuentran detalladas 24.4 (páginas 116 a 119).

El algoritmo aplicado consiste en recorrer sucesivamente los bits del operando, de dos en dos (comenzando por los más significativos), y restando, cuando se pueda, el resultado parcial **R** con el añadido **01** al final; en cada desplazamiento, se añade un **1** al resultado si se efectúa la resta y un **0** cuando ésta no es posible.

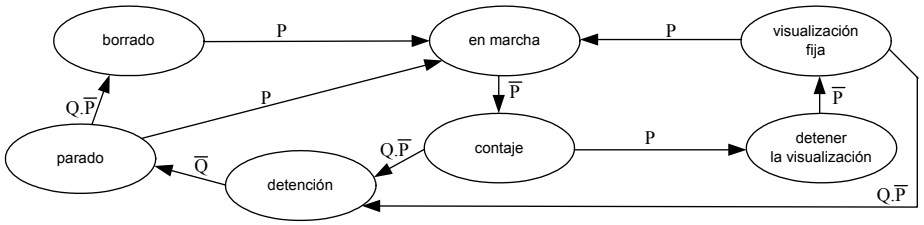
```
library ieee;       use ieee.std_logic_1164.all;       use ieee.std_logic_unsigned.all;

entity RAIZ is
    port (   CK, RS, Init     : in   std_logic;
             P                : in   std_logic_vector(63 downto 0);
             Dis              : out   std_logic;
             R                : out  std_logic_vector(31 downto 0));
end RAIZ ;

architecture CUADRADA of RAIZ is

-- declaración de estados
type mis_estados is (  reposo, inicio, desplazar1,desplazar2,
                       intermedio, restar, desplazar);
signal estado          :mis_estados;

signal pp              : std_logic_vector(63 downto 0);
-- registros para el operando P
signal rr              : std_logic_vector(31 downto 0);
-- registro para el resultado R
signal ss              : std_logic_vector(33 downto 0);
-- registro para el resto actual
signal cont            : std_logic_vector(5 downto 0);
-- contador de número de parejas de bits del operando

begin

R <= rr;

-- PARTE DE CONTROL: máquina algorítmica
CONTROL: process
begin
wait until CK = '1';
if ( RS = '1' ) then                                          estado <= reposo;
```

```
else   case estado is
            when reposo =>       if (Init = '1') then      estado <= inicio;          end if;
            when inicio =>                                 estado <= desplazar1;
            when desplazar1 =>                             estado <= desplazar2;
            when desplazar2 =>                             estado <= intermedio;
            when intermedio =>   if ss < (rr & "01") then  estado <= desplazar;
                                      else                 estado <= restar;          end if;
            when restar =>       if cont(5) = '1' then     estado <= reposo;
                                      else                 estado <= desplazar1;      end if;
            when desplazar =>    if cont(5) = '1' then     estado <= reposo;
                                    else                   estado <= desplazar1;      end if;
            when others =>
        end case;
end if;
end process;


-- PARTE OPERATIVA: operaciones correspondientes a los diversos estados
OPERACION: process
begin
wait until CK = '1';
Dis <= '0';
if ( RS = '1' ) then   pp <= (others => '0');   ss <= (others => '0');    rr <= (others => '0');
                       cont <= (others => '0');
   else     if estado = reposo then      Dis <= '1';                                  end if;
            if estado = inicio then      pp <= P;                  cont <= (others => '0');
                                         ss <= (others => '0');    rr <= (others => '0');
            end if;
            if estado = desplazar1 then  ss <= ss(32 downto 0) & pp(63);
                                         pp <= pp(62 downto 0) & '0';
                                         cont <= cont +1;
            end if;
            if estado = desplazar2 then  ss <= ss(32 downto 0) & pp(63);
                                         pp <= pp(62 downto 0) & '0';
            end if;
            if estado = restar then      ss <= ss – (rr & "01");
                                         rr <= rr(30 downto 0) & '1';              end if;
            if estado = desplazar then   rr <= rr(30 downto 0) & '0';              end if;
end if;
end process;
end CUADRADA;
```

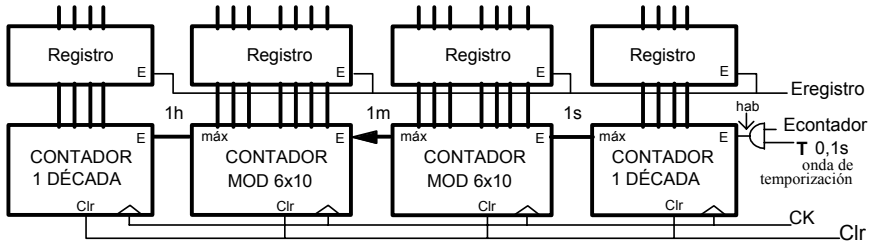

9. Cronómetro controlado por dos pulsadores

Se desea controlar un cronómetro mediante dos pulsadores **P** y **Q** de forma que el contaje de tiempo se inicie al activar **P** y se detenga al pulsar **Q** y un segundo pulso de **Q** sirva para poner a cero el cronómetro; una vez detenido el contaje, si se pulsa **P** se reanuda el mismo. Si mientras el cronómetro está activo se activa **P**, el contaje prosigue pero el visualizador queda detenido en el último valor previo a dicho pulso; se vuelve a la visualización normal del contaje pulsando nuevamente **P**.

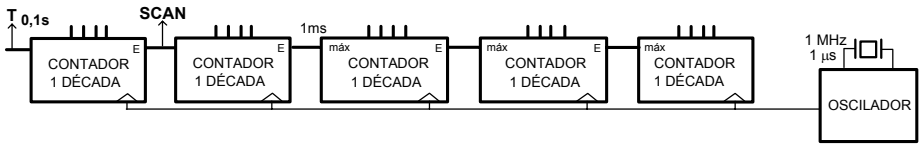
Este ejemplo corresponde al desarrollado en el capítulo 24 (apartado 4, páginas 126 y 127); el control responde al siguiente grafo de estados:



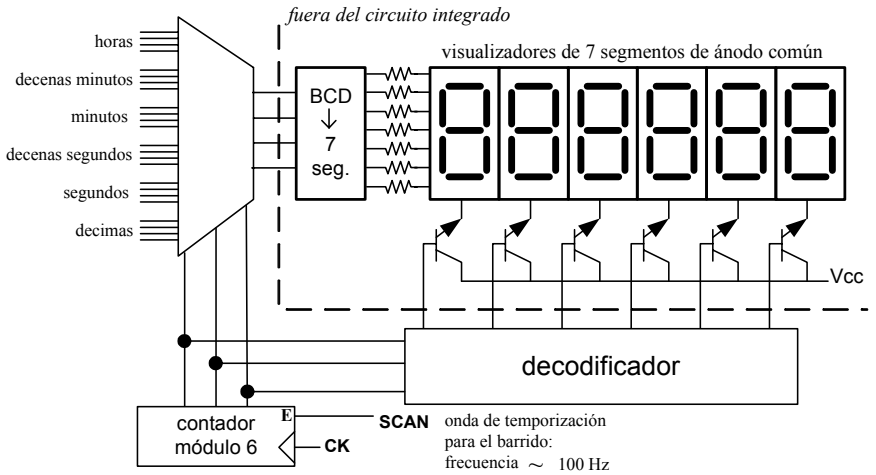
La siguiente figura muestra el diagrama de bloques de la parte operativa:



Como reloj se utilizará un oscilador de precisión con cristal de cuarzo de 1 MHz; por división de esa frecuencia (mediante 5 décadas) se obtendrá la onda de temporización de 0,1 segundo (100 Hz) que es la unidad de tiempo para el contador de la figura anterior. Asimismo, se obtendrá la señal de temporización para el barrido de los visualizadores (véase la figura de la página siguiente) que será de 100 Hz.

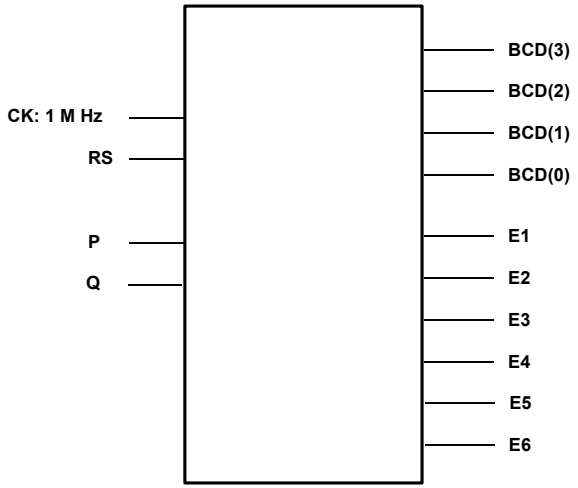


A fin de reducir líneas de conexión (6 cifras BCD requieren 24 líneas, que pasan a ser 42 después de los conversores a 7 segmentos), la representación sobre el visualizador será de tipo dinámico, según el esquema de multiplexado que sigue:



El circuito integrado, supuesto el conversor a 7 segmentos fuera del mismo, tendrá los siguientes terminales:

- entradas: reloj CK, inicialización RS y los dos pulsadores P y Q;
- salidas: 4 comunes para las cifras BCD y 6 para el barrido de los 6 visualizadores.




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

entity CRONOMETRO is
    port (   CK, RS          : in   std_logic;
             P, Q            : in   std_logic;
             BCD             : out   std_logic_vector(3 downto 0);
             E               : out  std_logic_vector(6 downto 1));
end CRONOMETRO;

architecture PULSADORES of CRONOMETRO is

-- contaje módulo 10
function mod10 (e1 :std_logic; q1:std_logic_vector) return std_logic_vector is
    variable contador1 :std_logic_vector(3 downto 0);
    begin
        if e1 = '1' then          if q1 = "1001" then contador1 := "0000";
                                  else contador1 := q1 + 1;          end if;
        else contador1 := q1;                                        end if;
return contador1;   end;

-- contaje módulo 6
function mod6 (e2 :std_logic; q2:std_logic_vector) return std_logic_vector is
    variable contador2 :std_logic_vector(6 downto 4);
    begin
        if e2 = '1' then          if q2 = "101" then contador2 := "000";
                                  else contador2 := q2 + 1;          end if;
        else contador2 := q2;                                        end if;
return contador2;   end;

signal seg, min                          : std_logic_vector(6 downto 0);
signal dec, hor                          : std_logic_vector(3 downto 0);
signal max1, max2, max3, max4,max5       : std_logic;
-- contadores del propio cronómetro (medida de tiempo)

signal regseg, regmin                    : std_logic_vector(6 downto 0);
signal regdec, reghor                    : std_logic_vector(3 downto 0);
-- registros del propio cronómetro (conectados a los contadores anteriores)

-- declaración de estados de la parte de control
type mis_estados is (  borrado, en_marcha, contaje, detencion, parado,
                       detener_vis, vis_detenida);
signal estado                            :mis_estados;

signal Clr, Econtador, Eregistro, hab    :std_logic;
-- señales de salida de la parte de control que actúan sobre el propio cronómetro
```

```
signal div1, div2, div3, div4, div5                 : std_logic_vector(3 downto 0);
signal maxdiv1, maxdiv2, maxdiv3, scan, decima      : std_logic;
-- contadores del divisor de la frecuencia del reloj

signal contscan                                     : std_logic_vector(6 downto 4);
-- contador para el barrido de la representación dinámica (módulo 6)

begin

-- CONTADORES Y REGISTROS DEL PROPIO CRONOMETRO

      hab <= decima and Econtador;      -- habilitación global del contador de tiempo

      -- señal de máximo para conectar los contadores sucesivamente
      max1 <= '1' when dec(3 downto 0) = "1001" and hab   = '1' else '0';
      max2 <= '1' when seg(3 downto 0) = "1001" and max1 = '1' else '0';
      max3 <= '1' when seg(6 downto 4) = "101"  and max2 = '1' else '0';
      max4 <= '1' when min(3 downto 0) = "1001" and max3 = '1' else '0';
      max5 <= '1' when min(6 downto 4) = "101"  and max4 = '1' else '0';

-- inserción de los contadores
CONTADORES: process(RS, CK)
begin
if (RS = '1') or (Clr = '1') then     dec <= (others => '0');          seg <= (others => '0');
                                      min <= (others => '0');          hor <= (others => '0');
elsif CK'event and CK = '1' then
         dec(3 downto 0) <=   mod10(hab, dec(3 downto 0));
         seg(3 downto 0) <= mod10(max1, seg(3 downto 0));
         seg(6 downto 4) <=  mod6(max2, seg(6 downto 4));
         min(3 downto 0) <= mod10(max3, min(3 downto 0));
         min(6 downto 4) <=  mod6(max4, min(6 downto 4));
         hor(3 downto 0) <=  mod10(max5, hor(3 downto 0));
end if;
end process;

-- inserción de los registros
REGISTROS: process(RS, CK)
begin
if RS = '1' then      regdec <= (others => '0');      regseg <= (others => '0');
                      regmin <= (others => '0');      reghor <= (others => '0');
elsif CK'event and CK = '1' then
         if Eregistro = '1' then  regdec <= dec;      regseg <= seg;
                                  regmin <= min;      reghor <= hor;          end if;
end if;
end process;
```

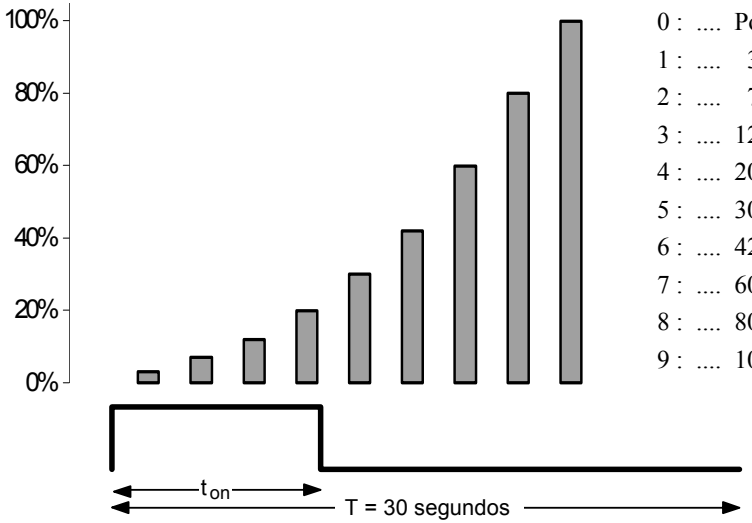


```
-- CONTROL A TRAVÉS DE LOS DOS PULSADORES

-- PARTE DE CONTROL: máquina de estados
CONTROL: process
begin       wait until CK = '1';
if ( RS = '1' ) then                                                  estado <= borrado;
else   case estado is
            when borrado =>       if (P = '1') then       estado <= en_marcha;    end if;
            when en_marcha =>     if (P = '0') then       estado <= contaje;      end if;
            when contaje =>       if (P = '1') then       estado <= detener_vis;
                                  elsif (Q = '1') then    estado <= detencion;    end if;
            when detencion =>     if (Q = '0') then       estado <= parado;       end if;
            when parado =>        if (P = '1') then       estado <= en_marcha;
                                  elsif (Q = '1') then    estado <= borrado;      end if;
            when detener_vis =>   if (P = '0') then       estado <= vis_detenida; end if;
            when vis_detenida =>  if (P = '1') then       estado <= en_marcha;
                                  elsif (Q = '1') then    estado <= detencion;  end if;
            when others =>                                estado <= borrado;
          end case;
end if;
end process;

-- PARTE OPERATIVA: señales correspondientes a los diversos estados
SEÑALES: process
begin
Clr <= '0';     Econtador <= '0';     Eregistro <= '0';       -- valor por defecto
case estado is
         when borrado =>      Clr <= '1';                          Eregistro <= '1';
         when en_marcha =>                 Econtador <= '1';      Eregistro <= '1';
         when contaje =>                   Econtador <= '1';      Eregistro <= '1';
         when detencion =>
         when parado =>
         when detener_vis =>               Econtador <= '1';
         when vis_detenida =>              Econtador <= '1';
         when others =>
end case;
end process;
```

```
-- DIVISIÓN DE FRECUENCIAS DEL RELOJ
        -- señal de máximo para conectar los contadores sucesivamente
        maxdiv1  <= '1' when div1(3 downto 0) = "1001" else '0';
        maxdiv2  <= '1' when div2(3 downto 0) = "1001" and maxdiv1 = '1' else '0';
        maxdiv3  <= '1' when div3(3 downto 0) = "1001" and maxdiv2 = '1' else '0';
        scan     <= '1' when div4(3 downto 0) = "1001" and maxdiv3 = '1' else '0';
        decima   <= '1' when div5 (3 downto 0) = "101"   and scan = '1' else '0';

-- inserción de los divisores por 10
DECADAS: process(RS, CK)          begin
if RS = '1' then            div1 <= (others => '0');   div2 <= (others => '0');
                            div3 <= (others => '0');   div4 <= (others => '0');
                            div5 <= (others => '0');
elsif CK'event and CK = '1' then
          div1 <=   mod10('1', div1);                  div2 <=   mod10(maxdiv1, div2);
          div3  <=   mod10(maxdiv2, div3);             div4  <=   mod10(maxdiv3, div4);
          div5  <=   mod10(scan, div5);
end if;
end process;

-- BARRIDO DINAMICO DE LOS VISUALIZADORES
-- contador que realiza el barrido
MODULO6: process(RS, CK)          begin
if RS = '1' then                  contscan <= (others => '0');
elsif CK'event and CK = '1' then  contscan <=   mod6(scan, contscan);
end if;
end process;

-- multiplexado
MULTIPLEXADO: process(contscan) begin
case contscan is
        when "000" =>       BCD  <= regdec;                  E  <= "000001";
        when "001" =>       BCD  <= regseg(3 downto 0);      E  <= "000010";
        when "010" =>       BCD  <= '0' & regseg(6 downto 4); E  <= "000100";
        when "011" =>       BCD  <= regmin(3 downto 0);      E  <= "001000";
        when "100" =>       BCD  <= '0' & regmin(6 downto 4); E  <= "010000";
        when "101" =>       BCD  <= reghor;                  E  <= "100000";
        when others =>      BCD  <= "0000";                  E  <= "000000";
end case;
end process;

end PULSADORES;
```


10. Control on/off para una placa térmica vitrocerámica

Consideramos una «cocina vitrocerámica» de cuatro «fuegos»; para controlar la potencia suministrada a la resistencia calefactora de cada fuego, se utiliza un ciclo de aproximadamente 30 segundos dividido en 30 pasos de 1" y, para fijar la potencia a suministrar, se emplea un mando circular de control con 10 posiciones (de 0 a 9 en código Gray), que corresponden a las siguientes potencias relativas:



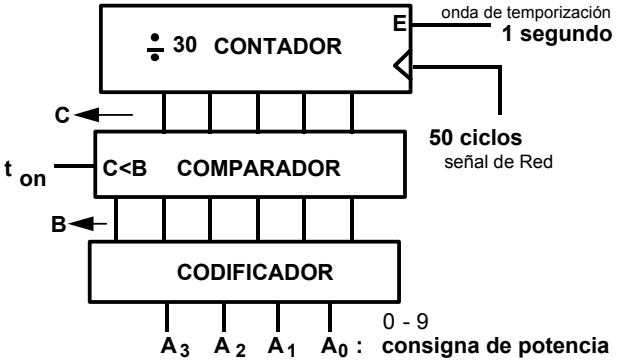
0 : .... Potencia nula
1 : .... 3 %
2 : .... 7 %
3 : .... 12 %
4 : .... 20 %
5 : .... 30 %
6 : .... 42 %
7 : .... 60 %
8 : .... 80 %
9 : .... 100 %

P = (tiempo de on / período) x 100

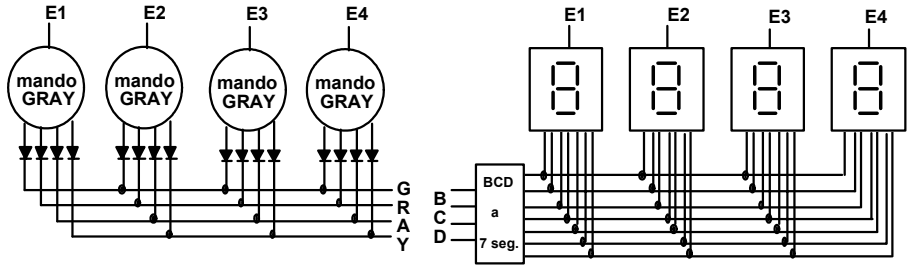
La anterior distribución de potencias (relativas) corresponde, en el caso de un ciclo de 30 unicades, a los siguientes intervalos activos (tiempos de *on*):

0 : .... 0 1 : .... 1 2 : .... 2 3 : .... 4 4 : 6
5 : .... 9 6 : .... 13 7 : .... 18 8 : .... 24 9 : 30

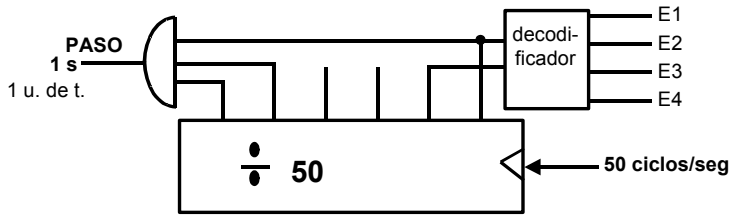
y el esquema circuital del control on/off es el representado en la figura siguiente:



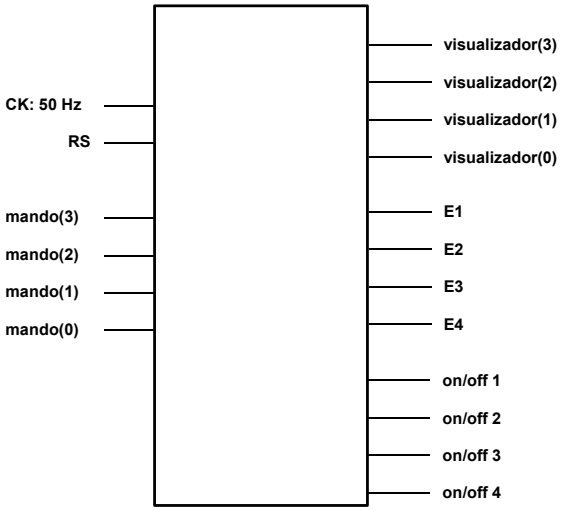
Los niveles de potencia de los cuatro «fuegos», fijados en los mandos rotativos (en codigo Gray) serán leídos dinámicamente, a través de 4 líneas comunes; tales niveles serán representados (en decimal, de 0 a 9) en cuatro visualizadores de siete segmentos, también en forma dinámica, a través de 4 líneas BCD:



El «barrido» para la lectura y representración dinámica será gestionado por el siguiente selector:



De manera que el circuito integrado completo tendrá los siguientes terminales (se aprovecha como reloj la propia señal de la red eléctrica: 50 Hz, 10 ms):



La descripción VHDL se hace mediante un módulo de «control», repetido para los cuatro «fuegos», que ejecuta la modulación de anchura de pulso, y un módulo «común» que realiza la temporización de 1 segundo, la lectura de los mandos y la visualización:

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

entity CONTROL is
   port (      CK,RS,HAB,PASO:   in std_logic;
               CONSIGNA       :   in std_logic_vector( 3 downto 0);
               BCD            :  out std_logic_vector( 3 downto 0);
               ONOFF          :  out std_logic );
end CONTROL ;

architecture ON_OFF of CONTROL is
signal salida              :std_logic;
signal cont, tiempo        :std_logic_vector( 4 downto 0);
signal reg1, reg2          :std_logic_vector( 3 downto 0);

begin
ONOFF <= salida;                BCD <= reg2;

-- validación de la CONSIGNA: actúa cuando HAB = 1
-- se guarda en reg1 el valor recibido
--  y, si el siguiente valor que se recibe es igual, se almacena en reg2
process(RS, CK)   begin
if RS = '1' then   reg1  <= "0000";   reg2  <= "0000";
elsif CK'event and CK = '1' then
        if HAB = '1' then                                  reg1  <= CONSIGNA;   end if;
        if HAB = '1' and (reg1 = CONSIGNA)  then   reg2  <= reg1;   end if;
end if;
end process;

-- contador base de tiempos: módulo 30
-- PASO es una onda de temporización de 1 segundo de período
process(RS, CK)   begin
if RS = '1' then   cont <= "00000";
elsif CK'event and CK = '1' then
        if PASO = '1'  then          if cont = "11101" then      cont <= "00000";
                                              else                  cont <= cont + 1;   end if;
        end if;
end if;                 end process;
```

```
-- conversión CONSIGNA → referencia (tiempo de on)
process(reg2)
begin
case reg2 is    when "0000"     =>    tiempo <= "00000";
                when "0001"     =>    tiempo <= "00001";
                when "0010"     =>    tiempo <= "00011";
                when "0011"     =>    tiempo <= "00101";
                when "0100"     =>    tiempo <= "00111";
                when "0101"     =>    tiempo <= "01010";
                when "0110"     =>    tiempo <= "01111";
                when "0111"     =>    tiempo <= "10100";
                when "1000"     =>    tiempo <= "11000";
                when "1001"     =>    tiempo <= "11110";
                when others     =>    tiempo <= "00000";
end case;
end process;

-- comparador del contador con el tiempo de on
process(RS, CK)   begin
if RS = '1' then   salida <= '0' ;
elsif CK'event and CK = '1' then
       if tiempo > cont then salida <= '1';   else   salida <= '0';   end if;
end if;
end process;

end ON_OFF;
```

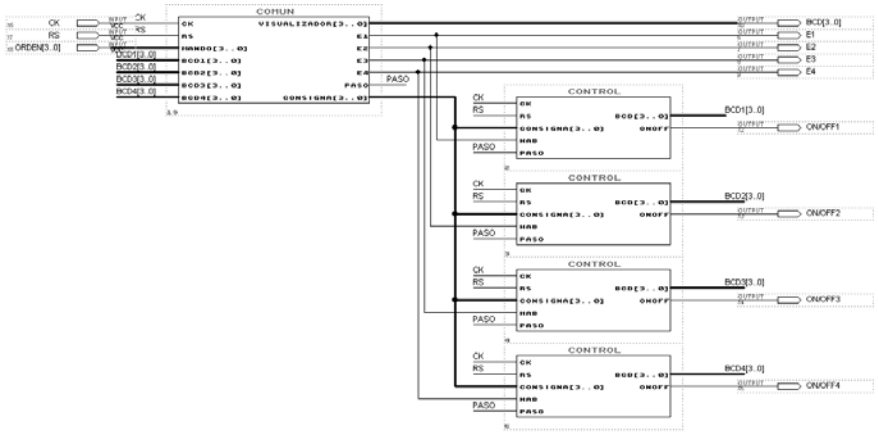
El módulo «común» divide por 50 la frecuencia de reloj, generando una onda de temporización (PASO) de 1 segundo, efectúa la lectura multiplexada (E1, E2, E3, E4) de los cuatro mandos y la escritura multiplexada sobre los cuatro visualizadores.

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

entity COMUN is
   port (  CK,RS                    :   in std_logic;
           mando                    :  in std_logic_vector(3 downto 0);
           BCD1,BCD2,BCD3,BCD4      :  in std_logic_vector(3 downto 0);
           visualizador             : out std_logic_vector(3 downto 0);
           E1,E2,E3,E4              : out std_logic;
           PASO                     : out std_logic;
           CONSIGNA                 : out std_logic_vector(3 downto 0));
end COMUN;
```



```
architecture OPERATIVA of COMUN is
signal ciclo        :std_logic;
signal cont         :std_logic_vector(5 downto 0);
signal EEEE         :std_logic_vector(1 to 4);

begin
PASO <= ciclo;   E1   <= EEEE(1);   E2 <= EEEE(2);   E3 <= EEEE(3);   E4 <= EEEE(4);

-- contador módulo 50 : generación de la señal de 1 segundo (PASO)
process(RS,CK)   begin
if RS = '1' then  cont <= "000000";   ciclo <= '0';
elsif CK'event and CK = '1' then
       if cont = "10111" then      cont <= "000000";          ciclo <= '1';
            else                   cont <= cont + 1;          ciclo <= '0';    end if;
end if;              end process;

-- ciclo de habilitación de los 4 mandos/visualizadores
with cont(1 downto 0) select    EEEE  <=          "1000" when "00",
                                                  "0100" when "01",
                                                  "0010" when "10",
                                                  "0001" when "11",
                                                  "0000" when others;
-- representación dinámica en los 4 visualizadores
with cont(1 downto 0) select    visualizador  <=  BCD1 when "00",
                                                  BCD2 when "01",
                                                  BCD3 when "10",
                                                  BCD4 when "11",
                                                  "0000" when others;
-- conversión de la orden (gray) en consigna (BCD)
process(mando)   begin
case mando is     when "0000"     =>    CONSIGNA <= "0000";
                  when "0001"     =>    CONSIGNA <= "0001";
                  when "0011"     =>    CONSIGNA <= "0010";
                  when "0010"     =>    CONSIGNA <= "0011";
                  when "0110"     =>    CONSIGNA <= "0100";
                  when "0111"     =>    CONSIGNA <= "0101";
                  when "0101"     =>    CONSIGNA <= "0110";
                  when "0100"     =>    CONSIGNA <= "0111";
                  when "1100"     =>    CONSIGNA <= "1000";
                  when "1101"     =>    CONSIGNA <= "1001";
                  when others     =>    CONSIGNA <= "0000";
end case;         end process;      end OPERATIVA;
```

La siguiente figura representa el esquemático correspondiente al circuito completo: conexión de cuatro módulos de «control» junto con un módulo «común».



La descripción VHDL (descripción estructural de conexión de módulos previamente descritos) del esquema anterior es la siguiente:

```
library ieee;    use ieee.std_logic_1164.ALL;
entity VITRO is
  port (    CK,RS                  : in    std_logic;
            mando                  : in    std_logic_vector (3 downto 0);
            visualizador           : out  std_logic_vector (3 downto 0);
            E1,E2,E3,E4                                    : out   std_logic;
            ON_OFF1,ON_OFF2,ON_OFF3,ON_OFF4  : out   std_logic);
end VITRO;

architecture CONEXIONES of VITRO is
   signal CONSIGNA,BCD1,BCD2,BCD3,BCD4     : std_logic_vector (3 downto 0);
   signal PASO                                          : std_logic;
   signal E1interior,E2interior,E3interior,E4interior       : std_logic;

   -- declaración de componentes: 2 componentes, COMUN y CONTROL

component COMUN
    port (  CK,RS                       : in    std_logic;
            mando                       : in    std_logic_vector (3 downto 0);
            BCD1,BCD2,BCD3,BCD4   : in    std_logic_vector (3 downto 0);
            E1,E2 ,E3,E4,PASO           : out   std_logic;
            visualizador                : out   std_logic_vector (3 downto 0);
            CONSIGNA                    : out std_logic_vector(3 downto 0));
  end component;
```



```
component CONTROL
    port (  CK,RS,HAB,PASO          : in    std_logic;
            CONSIGNA                : in    std_logic_vector (3 downto 0);
            ONOFF                   : out   std_logic;
            BCD                     : out   std_logic_vector (3 downto 0));
  end component;

begin

  E1 <= E1interior;   E2 <= E2interior;    E3 <= E3interior;    E4 <= E4interior;

  -- conexión de los módulos: 1 módulo COMUN y 4 módulos CONTROL

U1 : COMUN
    port map (   CK=>CK,  RS => RS,  PASO => PASO,  CONSIGNA => CONSIGNA,
                 BCD1 => BCD1,  BCD2 => BCD2,  BCD3 => BCD3,  BCD4 => BCD4,
                 E1 => E1interior,  E2 => E2interior,  E3 => E3interior,   E4 => E4interior,
                mando => mando, visualizador  => visualizador  );

U2 : CONTROL
    port map (   CK => CK,  RS => RS,  CONSIGNA => CONSIGNA, PASO=>PASO,
                 HAB => E1interior,  BCD => BCD1,  ONOFF => ON_OFF1  );

U3 : CONTROL
    port map (   CK => CK,  RS => RS,  CONSIGNA => CONSIGNA, PASO => PASO,
                 HAB => E2interior, BCD => BCD2,  ONOFF => ON_OFF2  );

U4 : CONTROL
    port map (   CK => CK,  RS => RS,  CONSIGNA => CONSIGNA,  PASO => PASO,
                 HAB => E3interior,  BCD => BCD3,  ONOFF => ON_OFF3  );

U5 : CONTROL
    port map (   CK => CK,  RS => RS,  CONSIGNA => CONSIGNA,  PASO => PASO,
                 HAB => E4interior,  BCD => BCD4,  ONOFF => ON_OFF4  );

end CONEXIONES;
```