Laboratorio 04: Contador decimal con display de 7-segmentos

Por:
Ian Gabriel Cañas Fernández, 1092228

.
.

Contador decimal con display de 7-segmentos
Utilizando y dominando los procesos para poder simular una combinación de dígitos variados.

Objetivos:

- Diseñar un sistema digital que despliegue un contador decimal de cuatro dígitos (0 – 9999) utilizando 4 displays de 7-segmentos.

Procedimiento:
En el presente laboratorio, vamos a crear un contador decimal que va a presentar la numeración en los displays disponibles en la FPGA. Solo que, al estar estos en paralelo entre sí, si intentamos presentar varios dígitos al mismo tiempo, no se podría, pues al encender varios displays, siempre presentan el mismo símbolo solicitado, por lo que vamos a crear un divisor de frecuencia, para que cada dígito se muestre en su momento.

Presentamos el código empezando por las librerías y los puertos a ser utilizados:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;

entity Lab04 is
Port (
        reloj: in std_logic; -- Reloj

        Disp: out std_logic_vector (7 downto 0); -- Vector que establecerá cuáles displays encender
        Num7Seg: out std_logic_vector (6 DOWNTO 0); -- Código del número correspondiente al 7 Segmentos

        Reinicio: in std_logic; -- Botón de reinicio de conteo
        GoTo990, GoTo9990: in std_logic -- Botones que me dirigen a 990 o 9990
        );
end Lab04;
```

Podemos notar que solo hay tres puertos de entradas, estos son el reloj interno de la tarjeta, que es un cristal que cambia cíclicamente su estado. Y dos botones cuyas funciones son reiniciar y adelantar el conteo.

Luego de la entidad y librerías establecemos variables con las que vamos a trabajar:

```
architecture Behavioral of Lab04 is
    constant NumMax: INTEGER := 49999;
    constant NumMax2: INTEGER := 49999999;

    signal conteo: INTEGER range 0 to NumMax;
    signal conteo2: INTEGER range 0 to NumMax2;

    signal clck_state, clck_state100: std_logic;

    signal unidad, decena, centena, miles: std_logic_vector (3 downto 0) := "0000";
    signal Cambio: std_logic_vector(1 downto 0) := "00";
```

Estas variables a partir de los procesos puestos van a crear la combinación necesaria para el conteo. Es decir, crear los números, establecer las frecuencias de muestreo y conteo, etc.

Continuamos con la función previamente realizada en el laboratorio 02 que, al ser llamada con el parámetro del dígito, va a devolver la combinación correspondiente al display de ánodo común.

```
function BcdTo7Seg (
    BCD: in std_logic_vector(3 downto 0)) -- Código binario del dígito
    return std_logic_vector is

    variable Cod7Seg: std_logic_vector (6 downto 0);

    begin
    with BCD select -- Selección de la combinación de ánodos en el display.
        Cod7Seg :=    ("0000001") WHEN ("0000"),
            ("1001111") WHEN ("0001"),
            ("0010010") WHEN ("0010"),
            ("0000110") WHEN ("0011"),
            ("1001100") WHEN ("0100"),
            ("0100100") WHEN ("0101"),
            ("0100000") WHEN ("0110"),
            ("0001111") WHEN ("0111"),
            ("0000000") WHEN ("1000"),
            ("0000100") WHEN ("1001"),
            ("1111111") WHEN OTHERS;
    return Cod7Seg; -- Devolución del respectivo valor.
end function;
```

Proseguimos agregando dos procesos que van a generar una señal de cambio cada milisegundo y cada segundo, dichas señales se conocerán como $clck\_state$ y $clck\_state100$ respectivamente, cuyo estado va a oscilar entre 0 y 1 exclusivamente.

```
Reconteo: process(reloj) -- Va a "mandar una señal" a clock_state cada 1 ms
begin
    if reloj'event and reloj = '1' then
        if conteo < NumMax then
            conteo <= conteo + 1;
        else
            clck_state <= not clck_state;
            conteo <= 0;
        end if;
    end if;
end process;

CentiSec: process (reloj) -- Va a "mandar una señal" a clock_state100 cada 1 s
begin
    if reloj'event and reloj = '1' then
        if conteo2 < NumMax2 then
            conteo2 <= conteo2 + 1;
        else
            clck_state100 <= not clck_state100;
            conteo2 <= 0;
        end if;
    end if;
end process;
```

El cambio del estado de $clck\_state$, que es de un milisegundo, va a ser detectado por un proceso que se va a encargar de alternar el display a ser encendido en dicho momento.

```
cambiador: process(clck_state)
begin
    if clck_state'event and clck_state = '1' then
        if Cambio < "11" then
            Cambio <= Cambio + 1;
        else
            Cambio <= "00";
        end if;
    end if;
end process;
```

Paralelamente, el cambio de estado de $clck\_state100$ va a ser detectado por otro proceso, para hacer un conteo, solo que hemos introducido dos botones, uno que va a reiniciar el conteo y otro que lo va a llevar a 990.

Los botones anteriormente mencionados se encuentran en la lista de sensibilidad del proceso, por lo que no es necesario esperar al próximo cambio de segundero para hacer el cambio.

Inclusive, los botones del proceso de conteo tienen prioridad absoluta frente a la numeración, pues va a rehacer el conteo.

```
Contador: process(clck_state100, reinicio, GoTo990, GoTo9990)
begin

   if reinicio = '1' then
        unidad <= "0000";
        decena <= "0000";
        centena <= "0000";
        miles <= "0000";
    elsif GoTo990 = '1' then
        unidad <= "0000";
        decena <= "1001";
        centena <= "1001";
        miles <= "0000";
    elsif GoTo9990 = '1' then
        unidad <= "0000";
        decena <= "1001";
        centena <= "1001";
        miles <= "1001";
    elsif clck_state100'event and clck_state100 = '1' then

        if unidad /= "1001" then
            unidad <= unidad + 1;
        else
            unidad <= "0000";
            if decena /= 9 then
                decena <= decena + 1;
            else
                decena <= "0000";
                if centena /= "1001" then
                    centena <= centena + 1;
                else
                    centena <= "0000";
                    if miles /= "1001" then
                        miles <= miles + 1;
                    else
                        miles <= "0000";
                    end if;
                end if;
            end if;
        end if;
   end if;
end process;
```

Finalmente nos encontraremos con el proceso que va a detectar cada cambio establecido, sea el cambio de display mostrado o el nuevo valor de cada dígito y que, inmediatamente va a encargarse de encender exclusivamente ese display y llamar a la función que va a traducir el valor de binario al decimal a ser representado con la combinación de ánodos mostrados a inicios del programa.

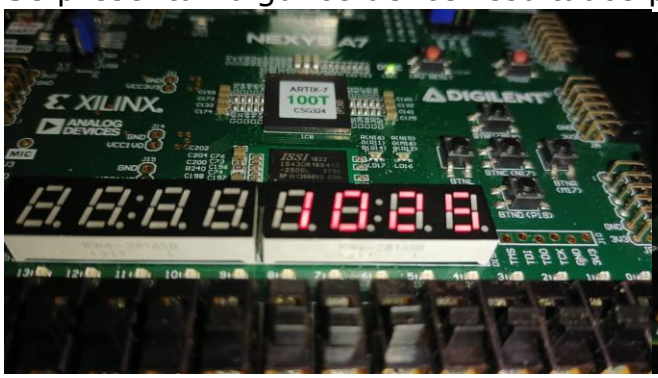
```
Muestra_display: process(clck_state, unidad, decena, centena, miles)
begin
    if Cambio = "00" then
        Disp <= "11111110";
        Num7Seg <= BcdTo7Seg(unidad);
    elsif Cambio = "01" then
        Disp <= "11111101";
        Num7Seg <= BcdTo7Seg(decena);
    elsif Cambio = "10" then
        Disp <= "11111011";
        Num7Seg <= BcdTo7Seg(centena);
    else --Cambio = "11" then
        Disp <= "11110111";
        Num7Seg <= BcdTo7Seg(miles);
    end if;
end process;
```

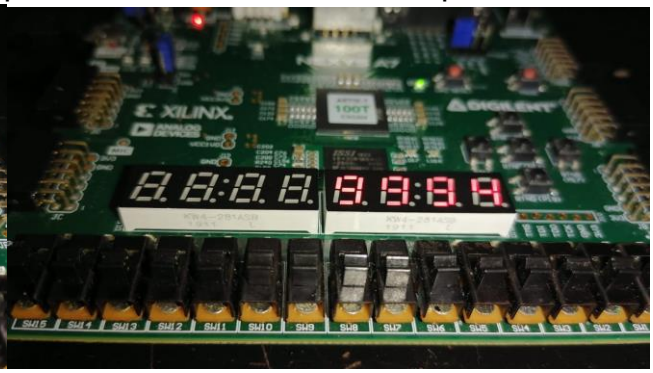
Resultados:

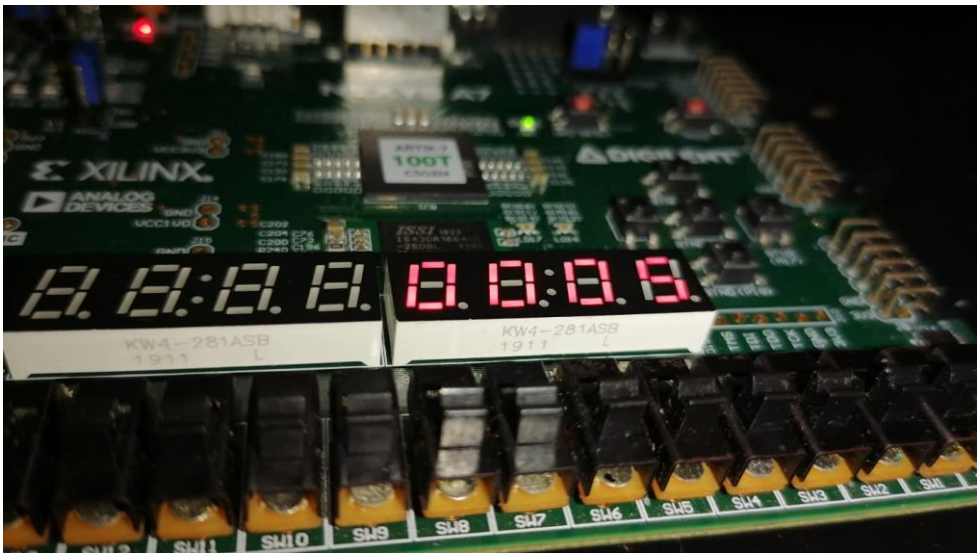
Los resultados obtenidos son los esperados; se presentan números decimales en los displays trabajados en el orden establecido, cumple fielmente con la regla de conteo y acarreo para la cantidad de dígitos necesarios.

Además, la FPGA, inmediatamente se pulsa el botón de reinicio, inicia el conteo desde 0. Claramente, al ser pulsado el botón de GoTo990 o de GoTo9990, el conteo inicia en 990 y 9990 tal y como se le solicita, ignorando el estado anterior del conteo. Una observación interesante es que automáticamente llega a 9999, reinicia al conteo desde 0.

Se presentan algunos de los resultados presentados en varios tiempos.







Análisis:
Se pudo comprender el comportamiento de los procesos y las funciones en VHDL, conociendo sus prioridades y evitando un proceso cíclico indeseado. Es necesario reconocer que es mejor tener varios procesos simples que uno muy complejo, además, en la confección e implementación del código, se pudo observar que un mal uso de las sentencias secuenciales puede provocar errores muy grandes o pesados para la tarjeta.

Nota: Se aconseja pensar los procesos secuenciales de una manera más física que digital.