Práctica 2 de SED

Por Simón Mateo de Pedraza

Contenido

Tarea 1: ejecutar el testbench

Ejecutando el testbench que aparece en el guion comprobamos el correcto funcionamiento del decodificador BCD a 7 segmentos. En caso de correcto funcionamiento debe salir un mensaje de "Simulación terminada"

```
# EXECUTION:: FAILURE: Simulacin finalizada. Test superado.
# EXECUTION:: Time: 320 ns,  Iteration: 0,  Instance: /decoder_tb,  Process: tb.
# KERNEL: Stopped at time 320 ns + 0.
# VSIM: Simulation has finished.
Done
```

Como se puede observar, el decoder funciona correctamente.

Tarea 2: entidad top

Crear una entidad que contenga el decodificador y muestre el número decodificado en un display en uno de los grupos de 4 en un lado de la placa, así como el display seleccionado para mostrar el número en otro en el otro grupo.

Utilizar una arquitectura con descripción estructural.

```
ENTITY top IS
 PORT ( code : IN std_logic_vector(3 DOWNTO 0);
         digsel : IN std_logic_vector(3 DOWNTO 0);
         digctrl : OUT std_logic_vector(3 DOWNTO 0);
         segment : OUT std_logic_vector(6 DOWNTO 0)
 );
END top;

architecture Structural of top is
    COMPONENT decoder
        PORT (
            code : IN std_logic_vector(3 DOWNTO 0);
            led : OUT std_logic_vector(6 DOWNTO 0)
        );
    END COMPONENT;

    signal a: std_logic_vector(3 downto 0);

begin
    Inst_decoder: decoder PORT MAP (
    code => code,
    led => segment
    );

    a<=digsel;
    digctrl<=a;

end Structural;
```

Al principio los displays no funcionaban como se esperaba, pero asignando correctamente las señales en el fichero de restricciones se corrige este fallo.

Tarea 4: comportamiento del módulo de sincronización

Siendo el código dado para módulo de sincronización el siguiente

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

entity SYNCHRNZR is
  port (
    CLK      : in  std_logic;
    ASYNC_IN : in  std_logic;
    SYNC_OUT : out std_logic
  );
end SYNCHRNZR;

architecture BEHAVIORAL of SYNCHRNZR is
  signal sreg : std_logic_vector(1 downto 0);
begin
  process (CLK)
  begin
     if rising_edge(CLK) then
       sync_out <= sreg(1);
       sreg <= sreg(0) & async_in;
     end if;
  end process;
end BEHAVIORAL;
```

Se observa que llega la señal de reloj CLK y una señal asíncrona ASYNC_IN. Se desea obtener a la salida una señal síncrona con el reloj SYNC_OUT.

La forma con la que se consigue esto es la siguiente:

1) Almacenar el valor de la señal asíncrona actual y el anterior en la señal sreg.
2) Asegurar que a la salida siempre aparece el valor anterior de la señal asíncrona
3) Actualizar los valores presente y pasado de la señal asíncrona almacenados cada flanco de subida del reloj

De esta manera se consigue coordinar los cambios en la señal de salida con el reloj.

Tarea 5: comportamiento del detector de flanco

Siendo el código dado para el detector de flanco el siguiente

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

entity EDGEDTCTR is
  port (
    CLK      : in  std_logic;
    SYNC_IN  : in  std_logic;
    EDGE     : out std_logic
  );
end EDGEDTCTR;

architecture BEHAVIORAL of EDGEDTCTR is
  signal sreg : std_logic_vector(2 downto 0);
begin
  process (CLK)
  begin
    if rising_edge(CLK) then
      sreg <= sreg(1 downto 0) & SYNC_IN;
    end if;
  end process;

  with sreg select
    EDGE <= '1' when "100",
            '0' when others;
end BEHAVIORAL;
```

De nuevo encontramos una señal de reloj y una de sincronización.

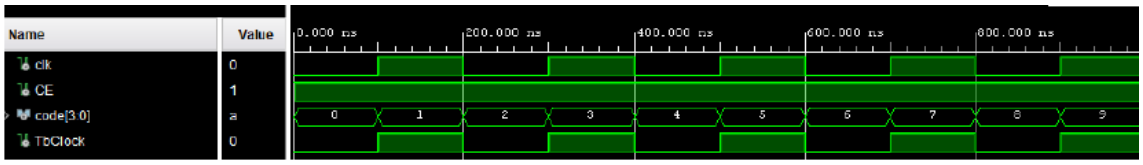
En este caso, se almacenan los valores de las posiciones anteriores de la señal síncrona de entrada. Las posiciones anteriores se van almacenando en los bits 1 y 2 de la señal sreg, mientras que en el bit 0 se almacena el valor actual. Esta actualización se realiza en cada flanco de subida del reloj.

Gracias a este código se consigue que solo se genere una señal de salida cuando se pulse el botón (genera 1 y los 2 ciclos siguientes 0), indicando que se ha producido un flanco de bajada en el botón y no es una señal debida a los rebotes.

Tarea 6: arquitectura del contador 0-9

Se ha optado por una arquitectura behavioral, colocando la señal de reloj en la lista de sensibilidad del process.

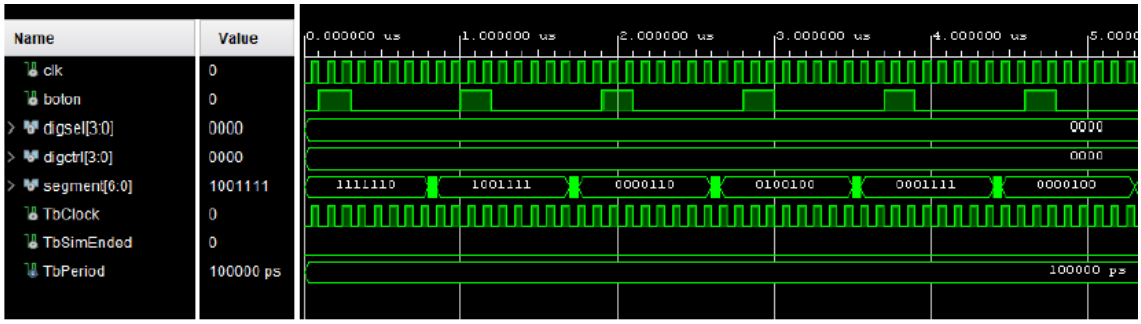
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;


entity counter is
 PORT (
 clk : in std_logic;
 CE : in std_logic;
 code : out std_logic_vector(3 downto 0)
 );
end counter;

architecture Behavioral of counter is
begin
process(CLK)
    variable acum : std_logic_vector(3 downto 0) := "0000";

   begin
        if CE = '1' then
            code <= acum;
            acum := std_logic_vector(to_unsigned(to_integer(unsigned( acum )) + 1, 4));
        end if;
    end process;
end Behavioral;
```



Tarea 7: implementación de los módulos

```
ENTITY top IS
 PORT (  clk : IN std_logic;
         boton : IN std_logic;
         digsel : IN std_logic_vector(3 DOWNTO 0);
         digctrl : OUT std_logic_vector(3 DOWNTO 0);
         segment : OUT std_logic_vector(6 DOWNTO 0)
 );
END top;

architecture Structural of top is
    COMPONENT decoder
        PORT (
            code : IN std_logic_vector(3 DOWNTO 0);
            led : OUT std_logic_vector(6 DOWNTO 0)
        );
    END COMPONENT;

    component SYNCHRNZR is
         port (
             CLK : in std_logic;
             ASYNC_IN : in std_logic;
             SYNC_OUT : out std_logic
         );
    end component;

    component EDGEDTCTR is
         port (
             CLK : in std_logic;
             SYNC_IN : in std_logic;
             EDGE : out std_logic
         );
    end component;

    component counter is
         PORT (
         clk : in std_logic;
         CE : in std_logic;
         code : out std_logic_vector(3 downto 0)
         );
    end component;

    signal a: std_logic_vector(3 downto 0);
    signal d: std_logic_vector(3 downto 0);
    signal b, c: std_logic;
```

```
begin
    Inst_decoder: decoder PORT MAP (
    code => d,
    led => segment
    );

    Inst_synchrnzr: synchrnzr PORT MAP(
    clk => clk,
    async_in => boton,
    sync_out => b
    );

    Inst_edgedtctr: edgedtctr PORT MAP(
    clk => clk,
    sync_in => b,
    edge => c
    );

    Inst_counter: counter PORT MAP(
    clk => clk,
    ce => c,
    code => d
    );

    a<=digsel;
    digctrl<=a;

end Structural;
```



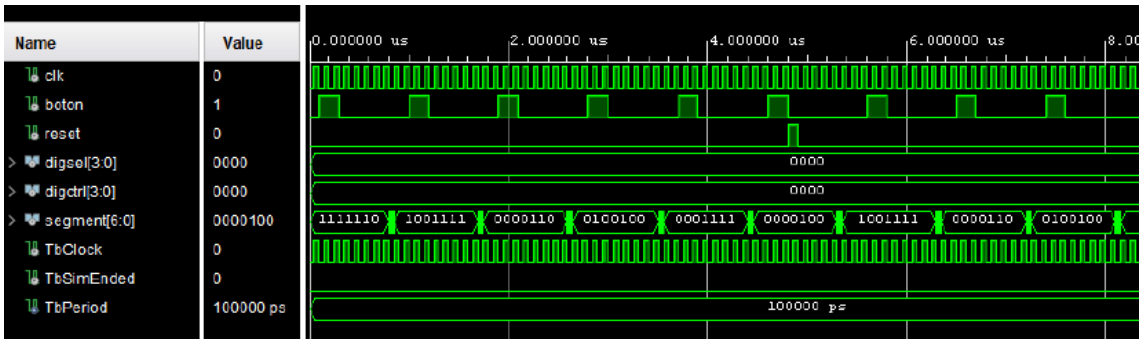
Tarea 8: reset

Se debe modificar el contador

```
entity counter is
 PORT (
 clk : in std_logic;
 CE : in std_logic;
 reset : in std_logic;
 code : out std_logic_vector(3 downto 0)
 );
end counter;

architecture Behavioral of counter is
begin
process(CLK, reset)
    variable acum : std_logic_vector(3 downto 0) := "0000";

   begin
        if CE = '1' then
            code <= acum;
            acum := std_logic_vector(to_unsigned(to_integer(unsigned( acum )) + 1, 4));
        end if;
        if reset = '1' then
            acum := "0000";
        end if;
    end process;
end Behavioral;
```

Se debe modificar el instanciamiento del contador en top e incluir una señal de entrada para el reset.

```
ENTITY top IS
 PORT (  clk : IN std_logic;
         boton, reset : IN std_logic;
         digsel : IN std_logic_vector(3 DOWNTO 0);
         digctrl : OUT std_logic_vector(3 DOWNTO 0);
         segment : OUT std_logic_vector(6 DOWNTO 0)
 );
END top;


   Inst_counter: counter PORT MAP(
   clk => clk,
   ce => c,
   reset => reset,
   code => d
   );
```



Conclusión

En esta práctica tomamos un poco de contacto con la programación estructural, así como en instanciar varias entidades como parte de una entidad top. Práctica que estoy seguro será de ayuda a la hora de realizar el trabajo de la asignatura.

Sin embargo, por esto mismo la práctica tiene una duración más que considerable, que no sería tan problemático si no coincidiera con la temporada de primeros parciales de las asignaturas de cuarto. Quizás daría tiempo a hacerse mejor si se empezase por la parte de microcontroladores en el laboratorio y en lugar de VHDL, ya que en diciembre no hay parciales de las asignaturas de cuarto (al menos este año)