Escuela Politécnica Superior,
Grado en Informática

Diseño Automático de Sistemas
Prof: Luis Pérez Miguel

Práctica 1:

Debouncer de la señal de un boton

UNIVERSIDAD
NEBRIJA

José María Fernández Gómez
Marcos Pardo Zapico

1. Diseño del sistema

1.1 top_practica1.vhd

```
    LED_register <='0';
  ELSIF rising_edge(clk100Mhz) THEN
    -- ------------------------------------------------------------------------
    -- Registrar valor
    -- ------------------------------------------------------------------------
    LED_register <= state_LED;
  END IF;
END PROCESS;
-- PROCESS to toggle LED
toggleLED : PROCESS (LED_register, Toggle_LED)
BEGIN
  -- ------------------------------------------------------------------------
  state_LED <= LED_register XOR Toggle_LED;
  -- ------------------------------------------------------------------------
END PROCESS;
-- Connect LED_register to the output
LED <= LED_register;
```

En primer lugar, inicializamos el estado del led a 0 para que este apagado y a continuación lo cargamos en registro auxiliar para poder modificar su valor de forma dinámica dentro de un process en el cual vamos a evaluar el estado actual del led y la salida del debouncer y por medio de un XOR iremos alternando el estado actual del led. En caso de que este apagado y en el debouncer no haya cambios esto de mantendrá apagado, en caso de que este apagado y del debouncer salga una señal de encendido, el led se encenderá. En los casos contrarios sucederá lo mismo de manera análoga.
(1 0 ->1, 1 1 ->0).

1.2 synchronizer.vhd

El fichero synchronizer.vhd **no requiere de ninguna modificación** se trata de un sincronizador como los que serán explicados en clase. La salida de este (BTN_sync) es una señal síncrona con el sistema. Sin embargo, dicha salida fluctúa siguiendo los rebotes del pulsador y por tanto debe ser aplicado el circuito antirrebotes.

1.3 debouncer.vhd

```
    -- ------------------------------------------------------------------------------------
    -- Declarar un tipo para los estados de la fsm usando type
    -- ------------------------------------------------------------------------------------
    type state_type is (IDLE, BTN_PRS, VALID, BTN_UNPRS); --crea lo estados
    signal current_state, next_state :state_type;

    signal count : unsigned(c_counter_width-1 downto 0);
    signal time_elapsed ,debounced_aux: std_logic;
```

Hemos declarado un conjunto de tipos de estado que representan algunos de los estados que puede pasar nuestra FSM y dos señales auxiliares de este mismo tipo que representan el estado actual de nuestra maquina y el siguiente que va a tomar.
A continuación declaramos una señal count que va a actuar como contador de ciclos desde la ultima vez que se activo la señal de salida del debouncer.
La dos ultimas señales son auxiliares que emplearemos para comprobar cuando salto por ultima vez nuestro debouncer y para poder asignar nuestra señal de salida, final de debouncer ya que al tratarse de una señal de salida no se puede modificar directamente su valor, se tiene que hacer con una señal.

```
begin
    --Timer
    process (clk, rst_n)
    begin
    -- ------------------------------------------------------------------------------
        -- Completar el timer que genera la señal de time_elapsed para trancionar en
        -- las máquinas de estados
        -- ------------------------------------------------------------------------------
        if(rst_n = '0') then
            count <= (others => '0');
        elsif (rising_edge(clk)) then
            time_elapsed <= '0';
            if(current_state=BTN_PRS) then
                if ((count < c_cycles) and (next_state /= VALID))then
                    count<= count +1;
                else
                    time_elapsed <= '1';
                    count <= (others => '0');
                end if;
            elsif(current_state=BTN_UNPRS) then
                if (count < c_cycles) and (next_state /= VALID)then
                    count<= count +1;
                else
                    time_elapsed <= '1';
                    count <= (others => '0');
                end if;
            elsif((current_state=IDLE) or (current_state=VALID) and (next_state = BTN_PRS or next_state= BTN_UNPRS )) then
                count <= count+1;
            end if;
        end if;
    end process;
```

Este código es parte de un proceso que se desencadena por una señal de reloj y una señal de reinicio. Se utiliza para implementar un mecanismo antirrebote para la pulsación de un botón. El proceso contiene una declaración if-else, que verifica si el botón está presionado o no y luego incrementa un contador. Si el contador alcanza un cierto número de ciclos, se reinicia y una variable llamada 'time_elapsed' se establece en '1'. Esta variable se utiliza luego para determinar si la pulsación del botón ha sido válida o no.

```
process (clk, rst_n)
begin

    if(rst_n = '0') then
        current_state <= IDLE;
        debounced <='0';
    elsif (rising_edge(clk)) then
        current_state <= next_state;
        debounced <= debounced_aux;
    end if;

end process;
```

Esto es la lógica que va a seguir la maquina de estados para cambiar de estado. Al tener un reset asíncrono el comportamiento se encuentra fuera de la comprobación del reloj, y en caso de que este salte, el estado de la máquina será IDLE. Por último la señal final la reasignaremos hacia nuestra señal auxiliar debounced_aux dentro de la comprobación de los ciclos de reloj.

```
process (ena,sig_in, current_state,time_elapsed)
begin
    debounced_aux<='0';
    case current_state is
        when IDLE =>
            if(sig_in = '1') then
                next_state <= BTN_PRS;
            else
                next_state <= current_state;
            end if;
        when VALID =>
            if (ena = '0') then
                next_state <= IDLE;
            elsif (sig_in = '0') then
                next_state <= BTN_UNPRS;
            else
                next_state <= current_state;
            end if;
            when BTN_PRS =>
            if( ena = '0') then
                next_state <= IDLE;
            elsif (time_elapsed = '0') then
                next_state <= current_state;
            elsif ((time_elapsed = '1') and (sig_in = '0')) then
                next_state <= IDLE;
            elsif ((sig_in = '1') and (time_elapsed = '1')) then
                next_state <= VALID;
                debounced_aux<='1';
            else
                next_state <= current_state;
            end if;
        when BTN_UNPRS =>
            if ( time_elapsed = '0') then
                next_state <= current_state;
            elsif ((ena = '0') or (time_elapsed = '1')) then
                next_state <= IDLE;
            else
                next_state <= current_state;
            end if;
        when others =>
            next_state <= IDLE;
    end case;

end process;
```

Aquí se ha codificado por medio de un condicional case el comportamiento de la maquina en función del esquema proporcionado. Lo que se realiza en este apartado son comprobaciones de las señales de control enable(ENA), sig_in(la señal a debouncear) time_elapsed(la señal que indica el ultimo cambio realizado) y current_state((señal importante ya que es una máquina de Moore y necesitamos que nuestra señal final se actualice en función de dicho estado.

2. Verificación

A continuación, se muestra una foto de la simulación del Vivado de nuestro sistema completo, se observa cómo tras el primer cambio en la señal del botón, se deja pasar un tiempo de 1000 nanosegundos antes de cambiar el valor del LED, mostrando así el correcto funcionamiento de nuestro sistema.