Práctica 12.

Uso de una Interfaz I^2C

En la práctica se utilizará un módulo I2C de código abierto, y se hará la evaluación de su funcionamiento utilizando un acelerómetro ADXL345.

A partir de la documentación y del ejemplo de uso del módulo, en el código siguiente de muestra la estructura para configurar y realizar lecturas del acelerómetro, mostrando el valor leído (Ax) en 10 LEDs.

Complete el código para leer la aceleración de los 3 ejes, y por medio de los switches se pueda elegir el valor a mostrar en los LEDs.

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

entity Acelerometro is
   Port ( clk  : in    STD_LOGIC;
          scl  : inout STD_LOGIC;
          sda  : inout STD_LOGIC;
          leds : out   STD_LOGIC_VECTOR(9 downto 0);
          error: out   STD_LOGIC;
          reset: in    STD_LOGIC);
end Acelerometro;

architecture Behavioral of Acelerometro is
   component i2c_master IS
      PORT(
           clk       : IN     STD_LOGIC;                    --system clock
           reset_n   : IN     STD_LOGIC;                    --active low reset
           ena       : IN     STD_LOGIC;                    --latch in command
           addr      : IN     STD_LOGIC_VECTOR(6 DOWNTO 0); --address of target slave
           rw        : IN     STD_LOGIC;                    --'0' is write, '1' is read
           data_wr   : IN     STD_LOGIC_VECTOR(7 DOWNTO 0); --data to write to slave
           busy      : OUT    STD_LOGIC;                    --indicates transaction in progress
           data_rd   : OUT    STD_LOGIC_VECTOR(7 DOWNTO 0); --data read from slave
           ack_error : BUFFER STD_LOGIC;                    --flag if improper acknowledge from slave
           sda       : INOUT  STD_LOGIC;                    --serial data output of i2c bus
           scl       : INOUT  STD_LOGIC);                   --serial clock output of i2c bus
   end component i2c_master;

   component Divisor is
      Port ( clk : in std_logic;
             div_clk : out std_logic);
   end component Divisor;

   constant adxl345 : std_logic_vector(6 downto 0) := "1010011";   -- 0X53
   constant reg_dataX0 : std_logic_vector(7 downto 0) := X"32";
   constant reg_PwrCtrl : std_logic_vector(7 downto 0) := X"2D";
   signal AX : std_logic_vector(15 downto 0);

   signal i2c_reset_n   : std_logic;
   signal i2c_ena       : std_logic;
   signal i2c_addr      : std_logic_vector(6 downto 0);
   signal i2c_rw        : std_logic;
   signal i2c_data_wr   : std_logic_vector(7 downto 0);
   signal i2c_busy      : std_logic;
   signal i2c_data_rd   : std_logic_vector(7 downto 0);
   signal i2c_ack_error : std_logic;
   signal clk_muestra   : std_logic;
   signal busy_prev     : std_logic;
   signal muestra_prev  : std_logic;

   -- Señales para la maquina de estados
   type estados is (inicia, lee, espera);
   signal edo_actual, edo_sig : estados;

begin
   U1: i2c_master port map (clk, i2c_reset_n, i2c_ena, i2c_addr, i2c_rw, i2c_data_wr, i2c_busy, i2c_data_rd,
i2c_ack_error, sda, scl);
   U2: Divisor port map (clk, clk_muestra);

   i2c_reset_n <= '1';

   -- Proceso para sincronizacion de estados
   process (clk)
   begin
      if (clk'event and clk = '1') then
         if (reset = '1') then
            edo_actual <= inicia;
         else
            edo_actual <= edo_sig;
```

```
      end if;
   end if;
end process;

-- Proceso para la decodificacion de estados
process (edo_actual, clk_muestra, i2c_busy, i2c_data_rd)
   variable busy_cnt : integer range 0 to 15 := 0;
begin
   edo_sig <= edo_actual;  --default is to stay in current state
   case (edo_actual) is
      when inicia =>
         busy_prev <= i2c_busy;                          -- guarda el valor previo de i2c_busy
         IF(busy_prev = '0' AND i2c_busy = '1') THEN     -- detecta flanco positivo de i2c_busy
            busy_cnt := busy_cnt + 1;                    -- cuenta flancos positivos de i2c_busy
         END IF;
         CASE busy_cnt IS                                -- busy_cnt indica el comando actual
            WHEN 0 =>                                    -- busy_cnt=0 : no se ha registrado comando
               i2c_ena <= '1';                              -- inicia la transaccion
               i2c_addr <= adxl345;                         -- fija la direccion del acelerometro
               i2c_rw <= '0';                               -- comando 1 es escritura
               i2c_data_wr <= reg_PwrCtrl;                  -- envia el registro de donde se va a escribir
            WHEN 1 =>                                    -- busy_cnt=1 : comando 1 en proceso, se indica comando 2
               i2c_data_wr <= X"08";                        -- activa al acelerometro
                  i2c_ena <= '1';                              --   asignación para evitar crear latch
                  i2c_addr <= adxl345;                         --    "
                  i2c_rw <= '0';                               --    "
            WHEN 2 =>                                    -- busy_cnt=2 : comando 2 en proceso, fin de transaccion
               i2c_ena <= '0';                              -- desahabilita para terminar transaccion
               IF(i2c_busy = '0') THEN                      -- verifica si escritura de comando 2 esta lista
                  busy_cnt := 0;                            -- reinicia contador para siguiente transaccion
                  edo_sig <= espera;                        -- termina proceso de configuración
               END IF;
                  i2c_addr <= adxl345;                         --   asignación para evitar crear latch
                  i2c_rw <= '0';                               --    "
                  i2c_data_wr <= X"08";                        --    "
            WHEN OTHERS =>
               busy_cnt := 0;                            -- reinicia contador para siguiente transaccion
                  i2c_ena <= '0';                              --   asignación para evitar crear latch
                  i2c_addr <= (others => '0');                 --    "
                  i2c_rw <= '0';                               --    "
                  i2c_data_wr <= (others => '0');              --    "
         END CASE;
      when lee =>
         busy_prev <= i2c_busy;                          -- guarda el valor previo de i2c_busy
         IF(busy_prev = '0' AND i2c_busy = '1') THEN     -- detecta flanco positivo de i2c_busy
            busy_cnt := busy_cnt + 1;                    -- cuenta flancos positivos de i2c_busy
         END IF;
         CASE busy_cnt IS                                -- busy_cnt indica el comando actual
            WHEN 0 =>                                    -- busy_cnt=0 : no se ha registrado comando
               i2c_ena <= '1';                              -- inicia la transaccion
               i2c_addr <= adxl345;                         -- fija la direccion del acelerometro
               i2c_rw <= '0';                               -- comando 1 es escritura
               i2c_data_wr <= reg_dataX0;                   -- envia el registro de donde se va a leer
            WHEN 1 =>                                    -- busy_cnt=1 : comando 1 en proceso, se indica comando 2
               i2c_rw <= '1';                               -- comando 2 es lectura (mismo dispositivo)
                  i2c_ena <= '1';                              --   asignación para evitar crear latch
                  i2c_addr <= adxl345;                         --    "
                  i2c_data_wr <= reg_dataX0;                   --    "
            WHEN 2 =>                                    -- busy_cnt=2 : comando 2 en proceso, se indica comando 3
(mismo anterior)
               IF(i2c_busy = '0') THEN                      -- verifica si lectura de comando 2 esta lista
                  AX(7 DOWNTO 0) <= i2c_data_rd;            -- lee dato de comando 2
               END IF;
                  i2c_ena <= '1';                              --   asignación para evitar crear latch
                  i2c_addr <= adxl345;                         --    "
                  i2c_rw <= '1';                               --    "
                  i2c_data_wr <= reg_dataX0;                   --    "
            WHEN 3 =>                                    -- busy_cnt=3 : comando 3 en proceso, fin de transaccion
               i2c_ena <= '0';                              -- desahabilita para terminar transaccion
               IF(i2c_busy = '0') THEN                      -- verifica si lectura de comando 3 esta lista
                  AX(15 DOWNTO 8) <= i2c_data_rd;           -- lee dato de comando 3
```

```
                busy_cnt := 0;                          -- reinicia contador para siguiente transaccion
                edo_sig <= espera;                      -- termina proceso de lectura de datos
             END IF;
                i2c_addr <= adxl345;                       --   asignación para evitar crear latch
                i2c_rw <= '1';                             --    "
                i2c_data_wr <= reg_dataX0;                 --    "
          WHEN OTHERS =>
             busy_cnt := 0;                             -- reinicia contador para siguiente transaccion
                i2c_ena <= '0';                            -- asignación para evitar crear latch
                i2c_addr <= (others => '0');               --  "
                i2c_rw <= '0';                             --  "
                i2c_data_wr <= (others => '0');            --  "
       END CASE;
      when espera =>
         muestra_prev <= clk_muestra;                   -- guarda el valor previo de clk_muestra
         busy_cnt := 0;                                 -- reinicia contador para siguiente transaccion
         if (muestra_prev = '0' AND clk_muestra = '1') then -- detecta el flanco positivo de clk_muestra
            edo_sig <= lee;
         end if;
            busy_prev <= i2c_busy;                          -- asignación para evitar crear latch
            i2c_ena <= '0';                                 --  "
            i2c_addr <= (others => '0');                    --  "
            i2c_rw <= '0';                                  --  "
            i2c_data_wr <= (others => '0');                 --  "
      when others =>
         edo_sig <= espera;
            busy_prev <= i2c_busy;                          -- asignación para evitar crear latch
            i2c_ena <= '0';                                 --  "
            i2c_addr <= (others => '0');                    --  "
            i2c_rw <= '0';                                  --  "
            i2c_data_wr <= (others => '0');                 --  "
   end case;
end process;

error <= i2c_ack_error;
leds <= AX(9 DOWNTO 0);
end Behavioral;
```