CCADET

CENTRO DE CIENCIAS APLICADAS Y DESARROLLO TECNOLÓGICO
UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Desarrollo de Funciones para el Manejo de Periféricos por Medio de un FPGA

Rafael Prieto Meléndez, Alejandro Padrón Godínez,
V. Gerardo Calva Olmos, Alberto A. Herrera Becerra

Diciembre 2011

Proyecto Interno

Resumen

En este trabajo se presenta el desarrollo de una biblioteca de funciones para ser utilizadas en la programación de un FPGA con la finalidad de que éste pueda interactuar con una variedad de periféricos, con el fin de que se puedan desarrollar de manera simple y en poco tiempo aplicaciones en donde se requiera éste tipo de interacción, principalmente para facilitar el desarrollo de interfases de usuario y la interacción con otros sistemas. Siguiendo ésta idea, se desarrollaron módulos para implantar un puerto UART; otro para generar imágenes que se puedan desplegar en un monitor VGA; uno más para poder recibir las señales provenientes de un teclado PS2; un controlador para un display LCD estándar con controlador compatible al HD44780, de 16 x 2 y hasta 40 x 4 caracteres, con interfaz de 4 bits; y un módulo para adquirir datos de un codificador rotatorio. Estos módulos se diseñaron con la idea de tener una interfaz simple y configurable, de tal manera que se pueda acoplar y adaptar de acuerdo a los requerimientos de cada aplicación. Los módulos se desarrollaron y probaron en la tarjeta de desarrollo *Spartan-3E Starter Kit Board* de Xilinx, y para ello se utilizó el lenguaje VHDL, de tal manera que la biblioteca desarrollada pueda ser transportada a cualquier otro FPGA.

Índice

Introducción 3
1. Puerto UART 5
2. Monitor VGA 7
3. Teclado PS2 8
4. Display LCD 10
5. Codificador Rotatorio 12
6. Ejemplos de uso 13
Conclusiones 17
Referencias Bibliográficas 18

Introducción

Entre las líneas de trabajo de nuestro grupo se encuentra el desarrollo de sistemas embebidos, y en este campo de trabajo el uso de los dispositivos lógicos programables es una de las ramas más importantes, siendo los FPGAs el dispositivo programable más utilizado por su versatilidad y capacidad para implantar dispositivos digitales de diversa índole, desde simples bloques lógicos para representar funciones combinacionales, armar circuitos secuenciales, construir complejas máquinas de estados algorítmicas y hasta implantar microprocesadores embebidos en un FPGA.

Un FPGA (del inglés Field Programmable Gate Array) es un dispositivo semiconductor programable de propósito general, que contiene bloques de lógica digital cuya interconexión y funcionalidad puede ser configurada por los usuarios, generalmente mediante un lenguaje de descripción especializado. Para el desarrollo de los módulos que aquí se presentan se utilizó el lenguaje VHDL, y se utilizó como plataforma para el desarrollo la tarjeta *Spartan-3E Starter Kit Board* de Xilinx, la cual cuenta como dispositivo base un FPGA XC3S500E Spartan-3E, el cual contiene más de 10,000 celdas lógicas, además de contar con un CPLD XC2C64A CoolRunner de 64 macroceldas, el cual es otro dispositivo lógico programable que también puede ser aprovechado en el desarrollo de aplicaciones. Se decidió trabajar con ésta tarjeta, ya que además de los dispositivos programables mencionados, cuenta con una gran variedad de componentes funcionales que facilitan enormemente el desarrollo de aplicaciones, tales como memoria RAM y memoria Flash, un oscilador de 50 MHz, un display LCD de 2 x 16 caracteres, un puerto PS2, un puerto VGA, un puerto Ethernet, dos puertos RS-232 de 9 pines (uno DTE y otro DCE), un puerto USB, un convertidor Digital a Analógico con 4 salidas, un convertidor Analógico a Digital con preamplificador con ganancia programable, un codificador rotatorio, switches deslizables y push-button y LEDs, entre otros (Xilinx, 2008), con los cuales es posible desarrollar una gran variedad de aplicaciones sin tener que utilizar otros elementos externos. En la figura 1 se muestra la tarjeta de desarrollo.

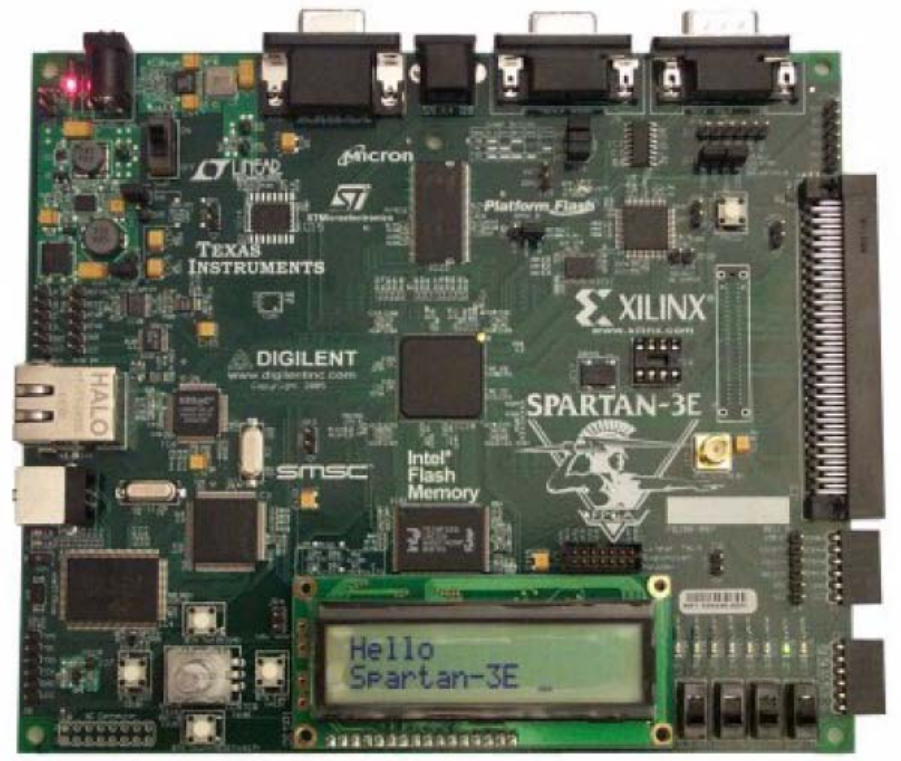

Figura 1. Tarjeta de desarrollo *Spartan-3E Starter Kit Board*.

La idea de desarrollar una biblioteca de bloques funcionales para utilizar éstos dispositivos disponibles en la tarjeta surge del interés de aprovecharlos en el desarrollo de aplicaciones, para lo cual es necesario

contar con los controladores o interfases hacia estos elementos, los cuales hay que programar en su totalidad en el FPGA, y desarrollar estos módulos para cada aplicación resulta muy laborioso, y reutilizar éstos bloques de otros trabajos previos generalmente resulta muy engorroso, ya que generalmente se desarrollan con las características propias de la aplicación y se encuentran entremezclados con otros elementos ajenos al controlador mismo, resultando nuevamente muy complicado separarlos de otros elementos, especialmente si esos módulos fueron desarrollados por un tercero. Las funciones que se desarrollaron fueron creadas para ser módulos totalmente independientes a la aplicación, con una interfase simple y que sean configurables, pensando que éstos puedan ser utilizados no únicamente en esta tarjeta, sino que puedan ser transportados a cualquier otra plataforma basada en un FPGA.

1. Puerto UART

La comunicación serial es una de las formas de intercambio de datos más usados para comunicación entre dispositivos. Uno de los puertos más utilizados para la comunicación serial son los puertos UART, nombre que proviene de las siglas de "**U**niversal **A**synchronous **R**eceiver-**T**ransmitter" (Transmisor-Receptor Asíncrono Universal). La comunicación serial en prácticamente cualquier computadora moderna está basada en este tipo de puertos. En el proceso de transmisión el puerto UART toma los bytes de datos y transmite los bits individuales de forma secuencial, mientras que en el proceso de recepción recibe los bits con los que reensambla los bytes de información transmitida. El puerto UART normalmente no genera y recibe directamente las señales transmitidas, sino que utiliza dispositivos de interfaz para acondicionar apropiadamente las señales eléctricas. Esto permite que seleccionando la interfaz adecuada se puedan establecer comunicaciones utilizando diferentes protocolos de comunicación, tales como los tradicionales RS-232, RS-422 y RS-485, pero también protocolos modernos tales como USB o Bluetooth.

La implantación del puerto UART desarrollada está basada en los módulos UART compactos desarrollados por Chapman (2003), el cual consta de dos módulos, uno para transmisión y otro para recepción, optimizados para la arquitectura, y que utilizan cada uno un buffer de 16 bytes para minimizar el efecto de baja velocidad de transmisión del puerto. La operación de este puerto se complementa con una función generadora de reloj para operación del puerto, el cual genera la base de tiempo de la que depende la velocidad de transmisión. Debido a que este tipo de comunicación es asíncrona se debe generar un sobremuestreo de las señales para compensar las posibles desviaciones de la velocidad nominal de transmisión, por lo que en esta implantación se hace un muestreo a una frecuencia 16 veces mayor que la velocidad de transmisión, con lo que se logra una tolerancia a desviaciones del reloj del 5%. Dado que el sistema de desarrollo opera con un reloj de 50 MHz, por el sobremuestreo requerido se pueden alcanzar con este módulo velocidades de comunicación de hasta 3.125 Mbps.

A continuación se muestra el código generado para implantar el módulo.

Listado 1. Código para un puerto UART

```
------------------------------------------------
-- Libreria:    UART
-- Descripcion: Generador de reloj, buffer y
--              modulos de transmision y recepcion
--              para puerto UART
--                                        *RPM
------------------------------------------------

------------------------------
-- Generador de Reloj
------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity uart_clk is
    Port ( clk : in  STD_LOGIC;
           uartclk : out  STD_LOGIC);
end uart_clk;

architecture Behavioral of uart_clk is
    signal baud_count: integer range 0 to 80 :=0; --
bps
    -- 2604~1200bps; 1302~2400bps;  651~4800bps;
    -- 326~9600bps; 163~19200bps; 81~38400bps;
    -- 54~57600bps; 27~115200bps;
begin
baud_timer: process (clk)
   begin
      if clk'event and clk='1' then
         if baud_count=80 then          -- bps
            baud_count <= 0;
            uartclk <= '1';
         else
            baud_count <= baud_count+1;
            uartclk <= '0';
         end if;
      end if;
   end process baud_timer;
end Behavioral;

------------------------------
-- Transmisor UART con buffer FIFO de 16 bytes
-- 8 bit, no parity, 1 stop bit
------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library unisim;
use unisim.vcomponents.all;

entity uart_tx is
    Port (data_in:in std_logic_vector(7 downto 0);
          write_buffer : in std_logic;
          reset_buffer : in std_logic;
          en_16_x_baud : in std_logic;
            serial_out : out std_logic;
           buffer_full : out std_logic;
     buffer_half_full : out std_logic;
                clk : in std_logic);
    end uart_tx;

architecture macro_level_definition of uart_tx is
component kcuart_tx
    Port(data_in:in std_logic_vector(7 downto 0);
       send_character : in std_logic;
         en_16_x_baud : in std_logic;
           serial_out : out std_logic;
          Tx_complete : out std_logic;
                clk : in std_logic);
    end component;
component bbfifo_16x8
    Port(data_in:in std_logic_vector(7 downto 0);
        data_out:out std_logic_vector(7 downto 0);
```

```
              reset : in std_logic;
              write : in std_logic;
               read : in std_logic;
               full : out std_logic;
          half_full : out std_logic;
       data_present : out std_logic;
                clk : in std_logic);
    end component;
signal fifo_data_out:std_logic_vector(7 downto 0);
signal fifo_data_present : std_logic;
signal fifo_read         : std_logic;
begin
  kcuart: kcuart_tx
  port map (       data_in => fifo_data_out,
            send_character => fifo_data_present,
              en_16_x_baud => en_16_x_baud,
                serial_out => serial_out,
              Tx_complete => fifo_read,
                      clk => clk);

  buf: bbfifo_16x8
  port map (     data_in => data_in,
                data_out => fifo_data_out,
                   reset => reset_buffer,
                   write => write_buffer,
                    read => fifo_read,
                    full => buffer_full,
                half_full => buffer_half_full,
             data_present => fifo_data_present,
                     clk => clk);
end macro_level_definition;

------------------------------
-- Receptor UART con FIFO de 16 bytes
-- 8 bit, no parity, 1 stop bit
------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library unisim;
use unisim.vcomponents.all;

entity uart_rx is
    Port ( serial_in : in std_logic;
        data_out:out std_logic_vector(7 downto 0);
         read_buffer : in std_logic;
        reset_buffer : in std_logic;
        en_16_x_baud : in std_logic;
 buffer_data_present : out std_logic;
         buffer_full : out std_logic;
    buffer_half_full : out std_logic;
                 clk : in std_logic);
    end uart_rx;

architecture macro_level_definition of uart_rx is
component kcuart_rx
    Port ( serial_in : in std_logic;
        data_out:out std_logic_vector(7 downto 0);
         data_strobe : out std_logic;
         en_16_x_baud : in std_logic;
                  clk : in std_logic);
    end component;
component bbfifo_16x8
    Port(data_in: in std_logic_vector(7 downto 0);
       data_out: out std_logic_vector(7 downto 0);
          reset : in std_logic;
          write : in std_logic;
           read : in std_logic;
           full : out std_logic;
      half_full : out std_logic;
   data_present : out std_logic;
            clk : in std_logic);
    end component;

signal uart_data_out:std_logic_vector(7 downto 0);
signal fifo_write   : std_logic;

begin
  kcuart: kcuart_rx
  port map (   serial_in => serial_in,
               data_out => uart_data_out,
            data_strobe => fifo_write,
           en_16_x_baud => en_16_x_baud,
                    clk => clk );

  buf: bbfifo_16x8
  port map (     data_in => uart_data_out,
                data_out => data_out,
                   reset => reset_buffer,
                   write => fifo_write,
                    read => read_buffer,
                    full => buffer_full,
                half_full => buffer_half_full,
             data_present => buffer_data_present,
                     clk => clk);
end macro_level_definition;

------------------------------
-- FIFO 'Bucket Brigade'
-- tamaño 16, dato de 8-bit
------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library unisim;
use unisim.vcomponents.all;

entity bbfifo_16x8 is
    Port(data_in: in std_logic_vector(7 downto 0);
       data_out: out std_logic_vector(7 downto 0);
          reset : in std_logic;
          write : in std_logic;
           read : in std_logic;
           full : out std_logic;
      half_full : out std_logic;
    ata_present : out std_logic;
            clk : in std_logic);
    end bbfifo_16x8;

architecture low_level_definition of bbfifo_16x8 is
signal pointer    : std_logic_vector(3 downto 0);
signal next_count : std_logic_vector(3 downto 0);
signal half_count : std_logic_vector(3 downto 0);
signal count_carry: std_logic_vector(2 downto 0);

signal pointer_zero        : std_logic;
signal pointer_full        : std_logic;
signal decode_data_present : std_logic;
signal data_present_int    : std_logic;
signal valid_write         : std_logic;

attribute INIT : string;
attribute INIT of zero_lut  : label is "0001";
attribute INIT of full_lut  : label is "8000";
attribute INIT of dp_lut    : label is "BFA0";
attribute INIT of valid_lut : label is "C4";

begin
  data_width_loop: for i in 0 to 7 generate
  attribute INIT : string;
  attribute INIT of data_srl : label is "0000";
  begin
     data_srl: SRL16E
     generic map (INIT => X"0000")
     port map(   D => data_in(i),
                CE => valid_write,
               CLK => clk,
                A0 => pointer(0),
                A1 => pointer(1),
                A2 => pointer(2),
                A3 => pointer(3),
                 Q => data_out(i) );
  end generate data_width_loop;

  count_width_loop: for i in 0 to 3 generate
  attribute INIT : string;
  attribute INIT of count_lut : label is "6606";

  begin
     register_bit: FDRE
     port map ( D => next_count(i),
                Q => pointer(i),
```

```
          CE => data_present_int,
           R => reset,
           C => clk);

count_lut: LUT4
generic map (INIT => X"6606")
port map( I0 => pointer(i),
          I1 => read,
          I2 => pointer_zero,
          I3 => write,
           O => half_count(i));

lsb_count: if i=0 generate
begin
  count_muxcy: MUXCY
  port map( DI => pointer(i),
            CI => valid_write,
             S => half_count(i),
             O => count_carry(i));
  count_xor: XORCY
  port map( LI => half_count(i),
            CI => valid_write,
             O => next_count(i));
end generate lsb_count;

mid_count: if i>0 and i<3 generate
begin
  count_muxcy: MUXCY
  port map( DI => pointer(i),
            CI => count_carry(i-1),
             S => half_count(i),
             O => count_carry(i));
  count_xor: XORCY
  port map( LI => half_count(i),
            CI => count_carry(i-1),
             O => next_count(i));
end generate mid_count;

upper_count: if i=3 generate
begin
  count_xor: XORCY
  port map( LI => half_count(i),
            CI => count_carry(i-1),
             O => next_count(i));
```

```
   end generate upper_count;
  end generate count_width_loop;

  zero_lut: LUT4
    generic map (INIT => X"0001")
  port map( I0 => pointer(0),
            I1 => pointer(1),
            I2 => pointer(2),
            I3 => pointer(3),
             O => pointer_zero );

  full_lut: LUT4
    generic map (INIT => X"8000")
  port map( I0 => pointer(0),
            I1 => pointer(1),
            I2 => pointer(2),
            I3 => pointer(3),
             O => pointer_full );

  dp_lut: LUT4
    generic map (INIT => X"BFA0")
  port map( I0 => write,
            I1 => read,
            I2 => pointer_zero,
            I3 => data_present_int,
             O => decode_data_present );
  dp_flop: FDR
  port map ( D => decode_data_present,
             Q => data_present_int,
             R => reset,
             C => clk);

  valid_lut: LUT3
    generic map (INIT => X"C4")
  port map( I0 => pointer_full,
            I1 => write,
            I2 => read,
             O => valid_write );

  full <= pointer_full;
  half_full <= pointer(3);
  data_present <= data_present_int;
end low_level_definition;
```

2. Monitor VGA

Un periférico que consideramos conveniente manejar es un monitor, ya que en muchas aplicaciones es necesario tener una interacción importante con el usuario, y la mejor manera para presentar cantidades considerables de información es a través de un monitor. Para poder enviar información a un monitor se requieren manejar dos aspectos, la generación de las señales de sincronía y la generación de las imágenes que se presentarán en la pantalla. Las señales de sincronía son las que se encargan de hacer el barrido por toda la pantalla y enviar en cada punto de la imagen el valor del pixel que se mostrará. El determinar el valor que tomará cada pixel enviado en responsabilidad de la función para generar las imágenes a mostrar. Las señales de sincronía son funciones que no dependen de lo que se esté desplegando, solo dependen de las características del estándar de video con que se esté trabajando. Existen básicamente tres métodos para la generación de las imágenes (Chu, 2008), el esquema de mapeo de bits, el esquema de mapeo por bloques y el esquema de mapeo por objetos. Cual esquema se utilice depende del tipo de despliegue que requiera cada aplicación. El esquema de mapeo de bits se utiliza principalmente para desplegar imágenes gráficas, mientras que el mapeo por bloques es utilizado principalmente para manejo de texto. El mapeo por objetos es el más simple y se utiliza cuando se utilizan pocos elementos en pantalla. En algunos casos, dependiendo de las características de las imágenes desplegadas, se puede utilizar alguna técnica mixta en que se aplique más de una de las técnicas mencionadas. Para éste conjunto de funciones, se desarrolló una biblioteca para generar las señales de sincronía para una resolución VGA estándar de 640 x 480 pixeles, la cual se presenta a continuación. Debido a los esquemas de generación de imágenes dependen de las características de la aplicación, es difícil generar funciones genéricas que sirvan para cualquier caso, sin embargo se está

trabajando en el desarrollo de funciones que permitan generar imágenes que puedan servir para los casos más comunes de manejo de texto e imágenes.

Listado 2. Código para un controlador para monitor VGA

```
-------------------------------------------------
-- Module Name:    VGASync
-- Descripcion: Generación de señales de sincronia
--              para monitor VGA
--                                          *RPM
-------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity vga_sync is
   port (
      clk, reset: in std_logic;
      hsync, vsync : out std_logic;
      video_on, p_tick: out std_logic;
      pixel_x, pixel_y : out std_logic_vector (9
downto 0)
      );
end vga_sync;

architecture arch of vga_sync is
   -- VGA 640 por 480, parámetros de sincronia
   constant HD: integer:=640; -- h. display area
   constant HF: integer:=16 ; -- h. front porch
   constant HB: integer:=48 ; -- h. back porch
   constant HR: integer:=96 ; -- h. retrace
   constant VD: integer:=480; -- v. display area
   constant VF: integer:=10 ; -- v. front porch
   constant VB: integer:=33 ; -- v. back porch
   constant VR: integer:=2  ; -- v. retrace
   signal mod2_reg, mod2_next: std_logic;
   signal v_count_reg, v_count_next: unsigned(9 downto
0);
   signal h_count_reg, h_count_next: unsigned(9 downto
0);
   signal v_sync_reg, h_sync_reg: std_logic;
   signal v_sync_next, h_sync_next: std_logic;
   signal h_end, v_end, pixel_tick: std_logic;
begin
  process (clk , reset)
  begin
   if reset='1' then
      mod2_reg <= '0';
      v_count_reg <= (others => '0');
      h_count_reg <= (others => '0');
      v_sync_reg <= '0';
      h_sync_reg <= '0';
   elsif (clk'event and clk='1') then
      mod2_reg <= mod2_next;
      v_count_reg <= v_count_next;
      h_count_reg <= h_count_next;
      v_sync_reg <= v_sync_next;
      h_sync_reg <= h_sync_next;
   end if;
end process;

-- mod-2 circu. para generar enable tick de 25 MHz
mod2_next <= not mod2_reg;
-- 25 MHz pixel tick
pixel_tick <= '1' when mod2_reg='1' else '0';
-- status
h_end <= -- end of horizontal counter
   '1' when h_count_reg=(HD+HF+HB+HR-1) else --799
   '0';
v_end <= -- end of vertical counter
   '1' when v_count_reg=(VD+VF+VB+VR-1) else --524
   '0';
-- mod-800 horizontal sync counter
process (h_count_reg,h_end,pixel_tick)
begin
   if pixel_tick='1' then -- 25 MHz tick
      if h_end='1' then
         h_count_next <= (others => '0');
      else
         h_count_next <= h_count_reg+1;
      end if;
   else
      h_count_next <= h_count_reg;
   end if;
end process;

-- mod-525 vertical sync counter
process(v_count_reg,h_end,v_end,pixel_tick)
begin
   if pixel_tick='1' and h_end='1' then
      if (v_end='1') then
         v_count_next <= (others => '0');
      else
         v_count_next <= v_count_reg + 1;
      end if;
   else
      v_count_next <= v_count_reg;
   end if;
end process;

-- horiz. y vert. sync, c/buffer para evitar ruido
h_sync_next <=
   '1' when (h_count_reg>=(HD+HF)) --656
      and (h_count_reg<=(HD+HF+HR-1)) else --751
   '0';
v_sync_next <=
   '1' when (v_count_reg>=(VD+VF)) --490
      and (v_count_reg<=(VD+VF+VR-1)) else --491
   '0';
-- video on/off
video_on <=
   '1' when (h_count_reg<HD) and (v_count_reg<VD) else
   '0';
-- output signal
hsync <= h_sync_reg;
vsync <= v_sync_reg;
pixel_x <= std_logic_vector(h_count_reg);
pixel_y <= std_logic_vector(v_count_reg);
p_tick <= pixel_tick;
end arch;
```

3. Teclado PS2

De manera similar que en el caso del monitor, el uso de un teclado como dispositivo de entrada es la forma más común para recibir datos de un usuario. Por ello se desarrolló esta biblioteca de funciones para manejar un teclado PS2, que es el tipo de teclado estándar más utilizado en la actualidad. El manejador para teclado PS2 que desarrollamos incluye una función para hacer la conversión del “scan-code” que envía el teclado al correspondiente valor ASCII de la tecla presionada. El controlador desarrollado cuenta con una función de filtrado sobre la señal de datos para eliminar los errores que se pudieran producir por el efecto del ruido eléctrico que se puede llegar a producir en el cable del teclado.

Listado 3. Código para un controlador para un teclado PS2

```
------------------------------------------------
-- Libreria:    ps2kbd_pckg
-- Descripcion: Librería de funciones para teclado
--              PS2 con conversión de scan-code a
--              codigo ASCII
--                                          *RPM
------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package ps2kbd_pckg is

   component ps2kbd is
      generic (FilterSize : positive := 8);
      port( clk : in std_logic; -- System Clock
         reset : in std_logic; -- System Reset
         PS2_clk : in std_logic; -- Keyboard Clock
         PS2_data : in std_logic; -- Keyboard Data
         read_ok : in std_logic; -- read scan code
         scan_err : out std_logic; -- error
         scan_done: out std_logic; --scancode listo
         scan_code : out std_logic_vector(7 downto 0)
-- 8 bits Data Out
         );
   end component ps2kbd;

   component PS2ascii is
      port (
        scan_code:in std_logic_vector (7 downto 0);
         caracter: out std_logic_vector(7 downto 0)
         );
   end component PS2ascii;

end ps2kbd_pckg;

--================================================--
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.Numeric_std.all;

entity ps2kbd is
   generic (FilterSize : positive := 8);
   port( clk : in std_logic; -- System Clock
      reset : in std_logic; -- System Reset
      PS2_clk : in std_logic; -- Keyboard Clock
      PS2_data : in std_logic; -- Keyboard Data
      read_ok : in std_logic; -- scan code listo
         scan_err : out std_logic; -- error
         scan_done : out std_logic;--scancode listo
         scan_Code : out std_logic_vector(7 downto 0)
-- 8 bits Data Out
      );
end ps2kbd;

architecture Behavioral of ps2kbd is
   signal PS2_datr : std_logic;
   subtype Filter_t is std_logic_vector(FilterSize-1
downto 0);
   signal filter : Filter_t;
   constant FCERO : Filter_t := (others=>'0');
   constant FUNO : Filter_t := (others=>'1');
   signal fall_clk : std_logic;
   signal bit_cnt : unsigned (3 downto 0);
   signal parity : std_logic;
   signal scan_donei : std_logic;
   signal s_reg : std_logic_vector(8 downto 0);
   signal PS2_clk_f : std_logic;
   Type State_t is (Idle, Shifting);
   signal state : State_t;
begin
   scan_done <= scan_donei;
   -- filtro digital para la senal de reloj
   process (clk, reset)
   begin
      if reset='1' then
         PS2_datr <= '0';
         PS2_clk_f <= '0';
         filter <= (others=>'0');
         fall_clk <= '0';
      elsif rising_edge (clk) then
         PS2_datr <= PS2_data and PS2_data;
      fall_clk <= '0';
         filter <= (PS2_clk and PS2_clk) &
filter(filter'high downto 1);
         if filter = FUNO then
            PS2_clk_f <= '1';
         elsif filter = FCERO then
            PS2_clk_f <= '0';
            if PS2_clk_f = '1' then
               fall_clk <= '1';
            end if;
         end if;
      end if;
   end process;

   -- Lee dato serial.
   process(clk,reset)
   begin
      if reset='1' then
         state <= Idle;
         bit_cnt <= (others => '0');
         s_reg <= (others => '0');
         scan_code <= (others => '0');
         parity <= '0';
         Scan_donei <= '0';
         scan_err <= '0';
      elsif rising_edge (clk) then
         if read_ok='1' then
            scan_donei <= '0';
         end if;
         case state is
            when Idle =>
               parity <= '0';
               bit_cnt <= (others => '0');
               if fall_clk='1' and PS2_datr='0' then --
Start bit
                  scan_err <= '0';
                  state <= Shifting;
               end if;
            when Shifting =>
               if bit_cnt >= 9 then
                  if fall_clk='1' then -- Stop Bit
                     -- Error Paridad o Overflow
                     scan_err <= (not parity) or (not
PS2_datr) or scan_donei;
                     scan_donei <= '1';
                     scan_code <= s_reg(7 downto 0);
                     state <= Idle;
                  end if;
               elsif fall_clk='1' then
                  bit_cnt <= bit_cnt + 1;
                  s_reg <= PS2_datr & s_reg (s_reg'high
downto 1); -- Shift right
                  parity <= parity xor PS2_datr;
               end if;
            when others => -- never reached
               state <= Idle;
         end case;
      end if;
   end process;
end Behavioral;

--================================================--
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.Numeric_std.all;

entity PS2ascii is
   port( scan_code: in std_logic_vector (7 downto 0);
         caracter: out std_logic_vector(7 downto 0)
      );
end PS2ascii;

architecture Behavioral of PS2ascii is
   signal dato: std_logic_vector (7 downto 0);
begin
```

```
   with scan_code select
      dato <=
         -- Letras
         X"41" when X"1C", -- A
         X"42" when X"32", -- B
         X"43" when X"21", -- C
         X"44" when X"23", -- D
         X"45" when X"24", -- E
         X"46" when X"2B", -- F
         X"47" when X"34", -- G
         X"48" when X"33", -- H
         X"49" when X"43", -- I
         X"4A" when X"3B", -- J
         X"4B" when X"42", -- K
         X"4C" when X"4B", -- L
         X"4D" when X"3A", -- M
         X"4E" when X"31", -- N
         X"4F" when X"44", -- O
         X"50" when X"4D", -- P
         X"51" when X"15", -- Q
         X"52" when X"2D", -- R
         X"53" when X"1B", -- S
         X"54" when X"2C", -- T
         X"55" when X"3C", -- U
         X"56" when X"2A", -- V
         X"57" when X"1D", -- W
         X"58" when X"22", -- X
         X"59" when X"35", -- Y
         X"5A" when X"1A", -- Z
         -- Numeros
         X"30" when X"45", -- 0
         X"31" when X"16", -- 1
         X"32" when X"1E", -- 2
         X"33" when X"26", -- 3
         X"34" when X"25", -- 4
         X"35" when X"2E", -- 5
         X"36" when X"36", -- 6
         X"37" when X"3D", -- 7
         X"38" when X"3E", -- 8
         X"39" when X"46", -- 9
         -- Teclado Numerico
         X"30" when X"70", -- 0
         X"31" when X"69", -- 1
         X"32" when X"72", -- 2
         X"33" when X"7A", -- 3
         X"34" when X"6B", -- 4
         X"35" when X"73", -- 5
         X"36" when X"74", -- 6
         X"37" when X"6C", -- 7
         X"38" when X"75", -- 8
         X"39" when X"7D", -- 9
         X"2E" when X"71", -- .
         X"2D" when X"7B", -- -
--       X"2F" when X"E04A", -- /
         X"2B" when X"79", -- +
         X"2A" when X"7C", -- *
--       X"0A" when X"E05A", -- Enter
         -- Simbolos
         X"2E" when X"49", -- .
         X"2C" when X"41", -- ,
         X"2F" when X"4A", -- /
         X"2D" when X"4E", -- -
         X"3D" when X"55", -- =
         X"5B" when X"54", -- [
         X"5D" when X"5B", -- ]
         X"5C" when X"5D", -- '\'
         X"3B" when X"4C", -- ;
         X"27" when X"52", -- '
         X"60" when X"0E", -- `
         -- Control
         X"08" when X"66", -- Backspace
         X"0D" when X"5A", -- Enter
         X"09" when X"0D", -- Tab
         X"1B" when X"76", -- Esc
--       X"7F" when X"E071", -- Del
         -- Otros
         X"20" when others; -- Espacio
   caracter <= dato;
end Behavioral;
```

4. Display LCD

El uso de los displays LCD es muy socorrido en aplicaciones donde la interfaz con el usuario es muy simple y solo se requiere enviar algunos cuantos mensajes de texto. Es por ello que se desarrolló un controlador para manejar este dispositivo. El tipo de display LCD estándar más utilizado es el que utiliza un controlador Hitachi HD44780 o algún otro compatible con él, encontrándose displays de diferentes tamaños, desde 16 caracteres por 2 renglones, hasta 40 caracteres por 4 renglones. Estos displays se pueden manejar con una interfaz de datos de 4 o de 8 bits. El controlador se encarga de hacer la configuración inicial del dispositivo, y durante la operación, de enviar los datos requeridos considerando la temporización establecida para el controlador según su hoja de especificaciones.

Listado 4. Código para un el manejo de un display LCD

```
-------------------------------------------------
-- Libreria:    lcd_pckg
-- Descripcion: Librería para el manejo de display
--              LCD estandar
--                                          *RPM
-------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package lcd_pckg is
   component lcd is
      -- generic ();
      port (
         clk, reset : in std_logic;
         SF_D : out std_logic_vector(3 downto 0);
         LCD_E, LCD_RS, LCD_RW, SF_CE0 : out
std_logic;
         tx_byte : in std_logic_vector(7 downto 0);
         tx_init : in std_logic;
         tx_done : out std_logic;
         init_init : in std_logic;
         init_done : out std_logic;
         lcdrs : in std_logic
      );
   end component lcd;
end lcd_pckg;

--=============================================--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity lcd is
   port (
      clk, reset : in std_logic;
      SF_D : out std_logic_vector(3 downto 0);
      LCD_E, LCD_RS, LCD_RW, SF_CE0 : out std_logic;
      tx_byte : in std_logic_vector(7 downto 0);
      tx_init : in std_logic;
```

```
      tx_done : out std_logic;
      init_init : in std_logic;
      init_done : out std_logic;
      lcdrs : in std_logic
   );
end lcd;

architecture behavior of lcd is
   type tx_sequence is (high_setup, high_hold,
        one_us, low_setup, low_hold, forty_us,
        wait_sync, done);
   signal tx_state : tx_sequence := done;

   type init_sequence is (idle, fifteen_ms, one,
        two, three, four, five, six, seven, eight,
        done);
   signal init_state : init_sequence := idle;

   signal i : integer range 0 to 750000 := 0;
   signal i2 : integer range 0 to 2000 := 0;

   signal SF_D0, SF_D1 : std_logic_vector(3 downto 0);
   signal LCD_E0, LCD_E1 : std_logic;
   signal mux : std_logic;
begin
   SF_CE0 <= '1'; --disable intel strataflash
   LCD_RW <= '0'; --write only
   LCD_RS <= lcdrs;

   -- Envío de un caracter
   transmit : process(clk, reset, tx_init)
   begin
      if(reset='1') then
         tx_state <= done;
      elsif(clk='1' and clk'event) then
         case tx_state is
            when high_setup => --40ns
               LCD_E0 <= '0';
               SF_D0 <= tx_byte(7 downto 4);
               if(i2 = 2) then
                  tx_state <= high_hold;
                  i2 <= 0;
               else
                  tx_state <= high_setup;
                  i2 <= i2 + 1;
               end if;
            when high_hold => --230ns
               LCD_E0 <= '1';
               SF_D0 <= tx_byte(7 downto 4);
               if(i2 = 12) then
                  tx_state <= one_us;
                  i2 <= 0;
               else
                  tx_state <= high_hold;
                  i2 <= i2 + 1;
               end if;
            when one_us =>
               LCD_E0 <= '0';
               if(i2 = 50) then
                  tx_state <= low_setup;
                  i2 <= 0;
               else
                  tx_state <= one_us;
                  i2 <= i2 + 1;
               end if;
            when low_setup =>
               LCD_E0 <= '0';
               SF_D0 <= tx_byte(3 downto 0);
               if(i2 = 2) then
                  tx_state <= low_hold;
                  i2 <= 0;
               else
                  tx_state <= low_setup;
                  i2 <= i2 + 1;
               end if;
            when low_hold =>
               LCD_E0 <= '1';
               SF_D0 <= tx_byte(3 downto 0);
               if(i2 = 12) then
                  tx_state <= forty_us;
                  i2 <= 0;
               else
                  tx_state <= low_hold;
                  i2 <= i2 + 1;
               end if;
            when forty_us =>
               LCD_E0 <= '0';
               if(i2 = 2000) then
                  tx_state <= wait_sync;
                  tx_done <= '1';
                  i2 <= 0;
               else
                  tx_state <= forty_us;
                  i2 <= i2 + 1;
               end if;
            when wait_sync => --40ns
               LCD_E0 <= '0';
               tx_done <= '0';
               if(i2 = 2) then
                  tx_state <= done;
                  i2 <= 0;
               else
                  tx_state <= wait_sync;
                  i2 <= i2 + 1;
               end if;

            when done =>
               LCD_E0 <= '0';
               tx_done <= '0';
               if(tx_init = '1') then
                  tx_state <= high_setup;
                  i2 <= 0;
               else
                  tx_state <= done;
                  i2 <= 0;
               end if;
         end case;
      end if;
   end process transmit;

   -- Configurción inicial del LCD con interfaz
    -- de 4 bits, considerando especificaciones
   power_on_initialize: process(clk, reset,
                        init_init)
   begin
      if(reset='1') then
         init_state <= idle;
         init_done <= '0';
         mux <= '1';
      elsif(clk='1' and clk'event) then
         case init_state is
            when idle =>
               init_done <= '0';
               mux <= '1';
               if(init_init = '1') then
                  init_state <= fifteen_ms;
                  i <= 0;
               else
                  init_state <= idle;
                  i <= i + 1;
               end if;
            when fifteen_ms =>
               mux <= '1';
               init_done <= '0';
               if(i = 750000) then
                  init_state <= one;
                  i <= 0;
               else
                  init_state <= fifteen_ms;
                  i <= i + 1;
               end if;
            when one =>
               SF_D1 <= "0011";
               LCD_E1 <= '1';
               init_done <= '0';
               if(i = 11) then
                  init_state<=two;
                  i <= 0;
               else
                  init_state<=one;
                  i <= i + 1;
               end if;
            when two =>
               LCD_E1 <= '0';
```

```
            init_done <= '0';
            if(i = 205000) then
               init_state<=three;
               i <= 0;
            else
               init_state<=two;
               i <= i + 1;
            end if;
         when three =>
            SF_D1 <= "0011";
            LCD_E1 <= '1';
            init_done <= '0';
            if(i = 11) then
               init_state<=four;
               i <= 0;
            else
               init_state<=three;
               i <= i + 1;
            end if;
         when four =>
            LCD_E1 <= '0';
            init_done <= '0';
            if(i = 5000) then
               init_state<=five;
               i <= 0;
            else
               init_state<=four;
               i <= i + 1;
            end if;
         when five =>
            SF_D1 <= "0011";
            LCD_E1 <= '1';
            init_done <= '0';
            if(i = 11) then
               init_state<=six;
               i <= 0;
            else
               init_state<=five;
               i <= i + 1;
            end if;
         when six =>
            LCD_E1 <= '0';
            init_done <= '0';
            if(i = 2000) then
               init_state<=seven;
               i <= 0;
            else
               init_state<=six;
               i <= i + 1;
            end if;
         when seven =>
            SF_D1 <= "0010";
            LCD_E1 <= '1';
            init_done <= '0';
            if(i = 11) then
               init_state<=eight;
               i <= 0;
            else
               init_state<=seven;
               i <= i + 1;
            end if;
         when eight =>
            LCD_E1 <= '0';
            init_done <= '0';
            if(i = 2000) then
               init_state<=done;
               i <= 0;
            else
               init_state<=eight;
               i <= i + 1;
            end if;
         when done =>
            init_state <= done;
            init_done <= '1';
            mux <= '0';
      end case;
   end if;
end process power_on_initialize;

with mux select
   SF_D <= SF_D0 when '0', --transmite
      SF_D1 when others; --inicializa

with mux select
   LCD_E <= LCD_E0 when '0', --transmite
      LCD_E1 when others; --inicializa
end behavior;
```

5. Codificador Rotatorio

La tarjeta de desarrollo cuenta entre los dispositivos básicos de entrada con una perilla, acoplada a un codificador rotatorio con rotación continua a pasos con 20 posiciones, que puede ser utilizada al interactuar con el usuario como un “control de volumen”, para ajustar el nivel de alguna señal o alguna función similar. El codificador rotatorio entrega dos señales de salida, las cuales corresponden a dos señales en cuadratura que cambian con cada paso de la perilla. Dependiendo de la secuencia con que se presenten los cambios en las dos salidas del codificador se puede reconocer el sentido en que se dio el giro de la perilla en cada caso, logrando con esto el poder implantar un control de intensidad o de desplazamiento con este dispositivo. A continuación se presenta el código correspondiente al controlador. Cabe mencionar que como la salida de las señales del codificador se genera a partir de interruptores mecánicos es necesario filtrar las señales para eliminar los rebotes que se producen en la conmutación de dichos interruptores. El desarrollo de éste módulo está basado en la interface desarrollada por Chapman (2006) para éste dispositivo.

Listado 5. Código para un tomar lectura de un codificador rotatorio

```
----------------------------------------------------
-- Module Name:    RotEncoder
-- Descripcion: Determina el paso y el sentido de
--              giro de un codificador rotacional
--                                              *RPM
----------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity RotEncoder is
   Port (rot_event : out std_logic;
         rot_dir : out std_logic;
         rotary_a : in std_logic;
         rotary_b : in std_logic;
         rotary_press : in std_logic;
         clk : in std_logic);
end RotEncoder;
```

```
architecture Behavioral of RotEncoder is
   signal     rotary_a_in : std_logic;
   signal     rotary_b_in : std_logic;
   signal rotary_press_in : std_logic;
   signal     rotary_in : std_logic_vector(1 downto
0);
   signal       rotary_q1 : std_logic;
   signal       rotary_q2 : std_logic;
   signal delay_rotary_q1 : std_logic;
   signal    rotary_event : std_logic;
   signal     rotary_left : std_logic;
begin
   rotary_filter: process(clk)
   begin
      if clk'event and clk='1' then
         rotary_a_in <= rotary_a;
         rotary_b_in <= rotary_b;
         rotary_press_in <= rotary_press;
         rotary_in <= rotary_b_in & rotary_a_in;

         case rotary_in is
            when "00" => rotary_q1 <= '0';
               rotary_q2 <= rotary_q2;
            when "01" => rotary_q1 <= rotary_q1;
               rotary_q2 <= '0';
            when "10" => rotary_q1 <= rotary_q1;
               rotary_q2 <= '1';
            when "11" => rotary_q1 <= '1';
               rotary_q2 <= rotary_q2;
            when others => rotary_q1 <= rotary_q1;
               rotary_q2 <= rotary_q2;
         end case;
      end if;
   end process rotary_filter;

   direction: process(clk)
   begin
      if clk'event and clk='1' then
         delay_rotary_q1 <= rotary_q1;
         if rotary_q1='1' and delay_rotary_q1='0' then
            rotary_event <= '1';
            rotary_left <= rotary_q2;
         else
            rotary_event <= '0';
            rotary_left <= rotary_left;
         end if;
         rot_event <= rotary_event;
         rot_dir <= rotary_left;
      end if;
   end process direction;
end Behavioral;
```

6. Ejemplos de uso

Para ilustrar la utilización de la biblioteca de módulos desarrollada, a continuación se muestran cuatro ejemplos simples de código en los cuales se hace uso de las diferentes funciones, con la idea de mostrar la forma de instanciar, de interconectar y de hacer uso de los recursos de cada módulo en una aplicación determinada.

En el primer ejemplo se muestra la utilización de los módulos PS2 y UART para implantar una especie de terminal tonta, en la cual la información que se recibe de un teclado PS2 conectado a la tarjeta de desarrollo es enviada de manera serial por un puerto UART. Para observar el funcionamiento del sistema, se puede conectar la tarjeta al puerto serie de una computadora y en un emulador de terminal observar la información recibida.

Listado 6. Código de ejemplo que recibe datos de un teclado y lo envía por puerto serie.

```
-------------------------------------------------
-- Modulo: Prueba de teclado PS/2 y de puerto UART
-- Descripcion: Lee dato del teclado y lo envia
--              por el puerto serie.
--                                              *RPM
-------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Teclado is
   port (
      clk, reset: in std_logic;
      ps2d, ps2c: in std_logic;
      tx: out std_logic;
      tstled: out std_logic
   );
end Teclado;

architecture prueba of Teclado is
   component ps2kbd is
      generic (FilterSize : positive := 8);
      port( clk : in std_logic;
            reset : in std_logic;
            PS2_clk : in std_logic;
            PS2_data : in std_logic;
            read_ok : in std_logic;
            scan_err : out std_logic;
            scan_done : out std_logic;
            scan_Code : out std_logic_vector(7 downto
0));
   end component;
   component PS2ascii is
      port( scan_code: in std_logic_vector (7 downto
0);
            caracter: out std_logic_vector(7 downto
0));
   end component;
   component uart_clk is
      Port ( clk : in  STD_LOGIC;
             uartclk : out  STD_LOGIC);
   end component;
   component uart_tx is
      Port (data_in : in std_logic_vector(7 downto 0);
            write_buffer : in std_logic;
            reset_buffer : in std_logic;
            en_16_x_baud : in std_logic;
            serial_out : out std_logic;
            buffer_full : out std_logic;
            buffer_half_full : out std_logic;
            clk : in std_logic);
   end component;

   constant SP: std_logic_vector (7 downto 0) :=
"00100000"; -- Espacio en ASCII
   signal scan_data, w_data: std_logic_vector (7
downto 0);
   signal scan_done_tick, wr_uart: std_logic;
   signal ascii_code: std_logic_vector (7 downto 0);
   signal baud_clk: std_logic;
   signal tled: std_logic;
```

```
   type state_type is (idle, idle1, send, sendb,
inicio);
   signal state, next_state : state_type;
begin
   -- Receptor PS2
   ps2ctrl: ps2kbd port map (clk, reset, ps2c, ps2d,
'1', open, scan_done_tick, scan_data);
   -- Convertidor Scancode a ASCII
   convA: PS2ascii port map (scan_data, ascii_code);
   -- UART_TX
   uart: uart_tx port map (w_data, wr_uart, reset,
baud_clk, tx, open, open, clk);
   -- UART_clk
   uartclk: uart_clk port map (clk, baud_clk);

   process (clk)
   begin
      if (clk'event and clk = '1') then
         if (reset = '1') then
            state <= inicio;
            tstled <= '0';
         else
            state <= next_state;
            tstled <= tled;
         end if;
      end if;
   end process;

   process (state)
   begin
      if state = send OR state = sendb then
         tled <= '1';
      else
         tled <= '0';
      end if;
   end process;

   process (state, scan_done_tick)
   begin
      wr_uart <= '0';
      w_data <= SP;
      next_state <= state;
      case (state) is
         when idle =>
            if scan_done_tick='1' then
               next_state <= send;
            end if;
         when send =>
            if scan_data = X"F0" then
               next_state <= idle1;
            else
               w_data <= ascii_code;
               wr_uart <= '1';
               next_state <= idle;
            end if;
         when idle1 =>
            if scan_done_tick='1' then
               next_state <= idle;
            end if;
         when sendb =>
            w_data <= SP;
            wr_uart <= '1';
            next_state <= idle;
         when inicio =>
            next_state <= idle;
      end case;
   end process;

end prueba;
```

Como segundo ejemplo ahora se muestra la utilización de los módulos PS2 y LCD, en el cual, de manera similar al ejemplo anterior, se recibe la entrada del teclado, pero en este caso la información recibida es desplegada de manera inmediata en el display de cristal líquido de la tarjeta de desarrollo.

Listado 7. Código de ejemplo que recibe datos de un teclado y lo despliega en el LCD.

```
----------------------------------------------------
-- Modulo: Prueba del LCD
-- Descripcion: Lee dato del teclado y lo muestra
--              en el LCD.
--                                              *RPM
----------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity lcd_main is
   port(
   clk, reset : in std_logic;
   SF_D : out std_logic_vector(3 downto 0);
   LCD_E, LCD_RS, LCD_RW, SF_CE0 : out std_logic;
   LED : out std_logic_vector(7 downto 0);
   PS2C, PS2D : in std_logic );
end lcd_main;

architecture prueba2 of lcd_main is
   component lcd is
      port (clk, reset : in std_logic;
            SF_D : out std_logic_vector(3 downto 0);
            LCD_E, LCD_RS, LCD_RW, SF_CE0 : out
std_logic;
            tx_byte : in std_logic_vector(7 downto 0);
            tx_init : in std_logic;
            tx_done : out std_logic;
            init_init : in std_logic;
            init_done : out std_logic;
            lcdrs : in std_logic);
   end component;
   component ps2kbd is
      generic (FilterSize : positive := 8);
      port( clk : in std_logic;
            reset : in std_logic;
            PS2_clk : in std_logic;
            PS2_data : in std_logic;
            read_ok : in std_logic;
            scan_err : out std_logic;
            scan_done : out std_logic;
            scan_Code : out std_logic_vector(7 downto
0));
   end component;
   component PS2ascii is
      port( scan_code: in std_logic_vector (7 downto
0);
            caracter: out std_logic_vector(7 downto
0));
   end component;

   signal tx_byte : std_logic_vector(7 downto 0);
   signal tx_init : std_logic := '0';
   signal tx_done : std_logic := '0';
   signal lcdrs : std_logic := '0';

   signal init_init, init_done : std_logic := '0';

   signal i3 : integer range 0 to 82000 := 0;
   signal pos : integer range 0 to 80 := 0;

   type display_state is (init, function_set,
entry_set, set_display, clr_display, pause,
                       set_addr, set_addr1, idle,
idle1, send);
   signal cur_state : display_state := init;

   signal scan_done : std_logic;
   signal scan_code, caracter : std_logic_vector(7
downto 0);
```

```
begin
   ULCD: lcd
      port map(clk, reset, SF_D, LCD_E, LCD_RS,
LCD_RW, SF_CE0, tx_byte, tx_init, tx_done,
              init_init, init_done, lcdrs);

   UKBD: ps2kbd port map(clk, reset, PS2C, PS2D, '1',
open, scan_done, scan_code);

   UCNV: PS2ascii port map(scan_code, caracter);

   LED <= tx_byte; -- para diagnostico

   -- controla la secuencia de initializacion
   with cur_state select
      init_init <= '1' when init,
      '0' when others;
   -- selecciona registro
   with cur_state select
      lcdrs <= '0' when
function_set|entry_set|set_display|clr_display|set_ad
dr|set_addr1,
      '1' when others;

   -- Máquina de Estados Principal
   display: process(clk, reset)
   begin
      if(reset='1') then
         cur_state <= function_set;
      elsif(clk='1' and clk'event) then
         case cur_state is
            when init =>
               tx_byte <= "00000000";
               tx_init <= '0';
               if(init_done = '1') then
                  cur_state <= function_set;
               else
                  cur_state <= init;
               end if;
            when function_set =>
               pos <= 0;
               tx_byte <= "00101000";
               tx_init <= '1';
               if(tx_done = '1') then
                  cur_state <= entry_set;
               else
                  cur_state <= function_set;
               end if;
            when entry_set =>
               tx_byte <= "00000110";
               tx_init <= '1';
               if(tx_done = '1') then
                  cur_state <= set_display;
               else
                  cur_state <= entry_set;
               end if;
            when set_display =>
               tx_byte <= "00001100";
               tx_init <= '1';
               if(tx_done = '1') then
                  cur_state <= clr_display;
               else
                  cur_state <= set_display;
               end if;
            when clr_display =>
               tx_byte <= "00000001";
               tx_init <= '1';
               i3 <= 0;
               if(tx_done = '1') then
                  cur_state <= pause;
               else
                  cur_state <= clr_display;
               end if;
            when pause =>
               tx_byte <= "00000000";
               tx_init <= '0';
               if(i3 = 82000) then
                  cur_state <= set_addr;
                  i3 <= 0;
               else
                  cur_state <= pause;
                  i3 <= i3 + 1;
               end if;
            when set_addr =>
               tx_byte <= "10000000";
               tx_init <= '1';
               pos <= 0;
               if(tx_done = '1') then
                  cur_state <= idle;
               else
                  cur_state <= set_addr;
               end if;
            when set_addr1 =>
               tx_byte <= "11000000";
               tx_init <= '1';
               pos <= 64;
               if(tx_done = '1') then
                  cur_state <= idle;
               else
                  cur_state <= set_addr1;
               end if;
            when idle =>
               tx_init <= '0';
               if scan_done='0' then
                  cur_state <= idle;
               elsif scan_code = X"F0" then
                  cur_state <= idle1;
               else
                  cur_state <= send;
                  pos <= pos + 1;
               end if;
            when send =>
               tx_byte <= caracter;
               tx_init <= '1';
               if (tx_done = '1') then
                  if pos = 16 then
                     cur_state <= set_addr1;
                  elsif pos = 80 then
                     cur_state <= set_addr;
                  else
                     cur_state <= idle;
                  end if;
               else
                  cur_state <= send;
               end if;
            when idle1 =>
               tx_init <= '0';
               if scan_done='1' then
                  cur_state <= idle;
               end if;
         end case;
      end if;
   end process display;
end prueba2;
```

En el siguiente ejemplo se muestra la utilización del módulo VGA para generar una imagen siguiendo el esquema de mapeo por objetos. En este caso, la imagen generada se compone de dos partes, en la parte superior se generan las barras cromáticas que se utilizan para la calibración de color ocupando las dos terceras partes de la pantalla, mientras que el la parte inferior de la pantalla se muestra una barra cuyo color es controlado por medio de tres interruptores, cada uno de los cuales controla el encendido de cada uno de los bits de color RGB (rojo, verde y azul), por lo que en pantalla se observa en dicha franja el color correspondiente a la combinación de color dada en los interruptores.

Listado 8. Código de ejemplo que genera una imagen de referencia en un monitor VGA.

```
-------------------------------------------------
-- Modulo: Prueba VGA
-- Descripcion: Salida VGA básica.
--                                          *RPM
-------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity vga_test is
   port (
         clk, reset: in std_logic;
         sw: in std_logic_vector(2 downto 0);
         hsync, vsync: out std_logic;
         rgb: out std_logic_vector(2 downto 0)
         );
end vga_test;

architecture prueba of vga_test is
   component vga_sync is
      port (clk, reset: in std_logic;
            hsync, vsync : out std_logic;
            video_on, p_tick: out std_logic;
            pixel_x, pixel_y : out std_logic_vector (9
downto 0) );
   end component;

   signal rgb_reg : std_logic_vector(2 downto 0);
   signal video_on, p_tick : std_logic;
   signal pixel_x, pixel_y : std_logic_vector (9
downto 0);
begin
   -- instancia circuito VGA sync
   vga_sync_unit : vga_sync
      port map(clk, reset, hsync, vsync, video_on,
p_tick, pixel_x, pixel_y);

   -- buffer rgb
   process (clk, reset)
   begin
      if reset='1' then
         rgb_reg <= ( others => '0') ;
      elsif (clk'event and clk='1') then
         if pixel_y<320 then
            if pixel_x<80 then
               rgb_reg <= "000";
            elseif pixel_x<160 then
               rgb_reg <= "001";
            elseif pixel_x<240 then
               rgb_reg <= "010";
            elseif pixel_x<320 then
               rgb_reg <= "011";
            elseif pixel_x<400 then
               rgb_reg <= "100";
            elseif pixel_x<480 then
               rgb_reg <= "101";
            elseif pixel_x<560 then
               rgb_reg <= "110";
            else
               rgb_reg <= "111";
            end if;
         else
            rgb_reg <= sw;
         end if;
      end if;
   end process;

   rgb <= rgb_reg when video_on='1' else "000";
end prueba;
```

Finalmente, en el siguiente ejemplo se muestra la utilización del módulo de control del codificador rotatorio. Aquí se usa la salida del codificador para mover un motor a pasos, por cada paso que se haga girar al vástago del codificador el motor se moverá un paso en la misma dirección.

Listado 9. Código de ejemplo para controlar el giro de un motor a pasos con el codificador rotatorio.

```
-------------------------------------------------
-- Modulo: Prueba RotatorioMotorPasos
-- Descripcion: Controla la rotación de un motor a
--              pasos con el codificador
--              rotatorio.
--                                          *RPM
-------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity MotPasos is
    Port ( RotA : in  STD_LOGIC;
           RotB : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           Salida : out  STD_LOGIC_VECTOR (3 downto
0));
end MotPasos;

architecture Behavioral of MotPasos is
   signal Posicion : BIT_VECTOR (3 downto 0) :=
"1100";
   signal Paso, Dir : STD_LOGIC;
   component RotEncoder is
      Port (rotary_a : in std_logic;
           rotary_b : in std_logic;
           rot_event : out std_logic;
           rot_dir : out std_logic;
           clk : in std_logic);
   end component;
begin
   U1 : RotEncoder port map (RotA, RotB, Paso, Dir,
clk);

   process (Paso)
      begin
         if(Paso'event and Paso = '1') then
            if Dir='1' then
               Posicion <= Posicion ror 1;
            else
               Posicion <= Posicion rol 1;
            end if;
         end if;
   end process;

   Salida <= To_StdLogicVector(Posicion);

end Behavioral;
```

Conclusiones

En el presente trabajo se muestra el conjunto de funciones de biblioteca que se han desarrollado para controlar periféricos por medio de un FPGA. Aunque estos módulos se han desarrollado para ser utilizados en la tarjeta de desarrollo seleccionada, el código desarrollado en el lenguaje VHDL estándar que puede ser transportado a cualquier otra plataforma FPGA. Aunque en ellas no se cuente con los puertos integrados, basta con el dispositivo a cualquier conjunto de líneas de entrada/salida, las necesarias de acuerdo al periférico utilizado, sin tener que hacer cambios a los módulos, pudiendo configurarlos según se necesite por medio de definición de los parámetros de configuración. De esta manera se tiene que esta biblioteca es portable y de propósito general.

Los bloques presentados se encuentran en un estado de desarrollo avanzado, se han probado y utilizado en aplicaciones de ejemplo para probar su funcionalidad ante diferentes configuraciones, las cuales nos permiten asegurar que están listos para ser utilizados en cualquier aplicación. Sin embargo se sigue trabajando para mejorarlas, por ejemplo, el driver para el monitor se está modificando para trabar con una resolución SVGA de 800 x 600 pixeles y en funciones para facilitar la generación de imágenes para las diferentes técnicas mencionadas. En el caso del display LCD se trabaja en una función para establecer un esquema de mapa de memoria con los caracteres desplegados para facilitar el envío de información. En el caso del teclado PS2 se considera desarrollar funciones adicionales para el manejo simple de las teclas de funciones especiales y se analiza la posibilidad de ampliarla para poder manejar un mouse PS2.

Los periféricos para los que se desarrollaron las funciones son los más comúnmente utilizados y que se pueden implantar fácilmente en cualquier otra plataforma basada en FPGAs aunque no se tengan ya integrados, pero aún queda también como trabajo a futuro el desarrollar funciones para el manejo de los demás recursos con que cuenta la tarjeta de desarrollo con la que hemos estado trabajando, ya que como se estará utilizando en nuestro grupo de trabajo para el desarrollo de futuras aplicaciones, su desarrollo permitirá explotar al máximo los recursos con que se cuenta, aunque algunos de ellos no sean tan generales o no se puedan transportar tan fácilmente a otras plataformas.

Referencias Bibliográficas

Chapman, K. (2003): *UART Transmitter and Receiver Macros*, Xilinx Inc, 13 pp.

_____ (2006): *Rotary Encoder Interface for Spartan-3E Starter Kit*, Xilinx Inc, 10 pp.

Chu, P.P. (2008): *FPGA Prototyping by VHDL Examples*, Wiley Interscience, New Jersey, 471 pp.

Xilinx Inc. (2008): *Spartan-3E FPGA Starter Kit Board User Guide*, Xilinx Inc, 166 pp.