CCADET

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

Informe Técnico
Herramientas para Sistemas Embebidos Formados por FPGAs y Microcontroladores

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

Diciembre 2014

Proyecto Interno

Financiamiento interno

Título: Herramientas para Sistemas Embebidos Formados por FPGAs y Microcontroladores

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

Afiliación: Centro de Ciencias Aplicadas y Desarrollo Tecnológico, Universidad Nacional Autónoma de México.

Resumen

En este trabajo se presentan algunos desarrollos que complementan 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 SPI; uno para poder recibir las señales provenientes de un mouse PS2; un generador de señales PWM para controlar la intensidad de energía transmitida a dispositivos eléctricos tales como motores de DC y lámparas LED; y se adaptó el módulo de sincronía de video VGA para generar de resolución SVGA. Adicionalmente se describen los mecanismos utilizados para establecer la comunicación entre un FPGA y un microcontrolador. También se describe la forma de establecer un puerto USB por medio de un microcontrolador. Los módulos se desarrollaron y probaron en la tarjeta de desarrollo *Spartan-3E Starter Kit Board* de Xilinx, y fueron desarrollados en el lenguaje VHDL.

Índice

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.

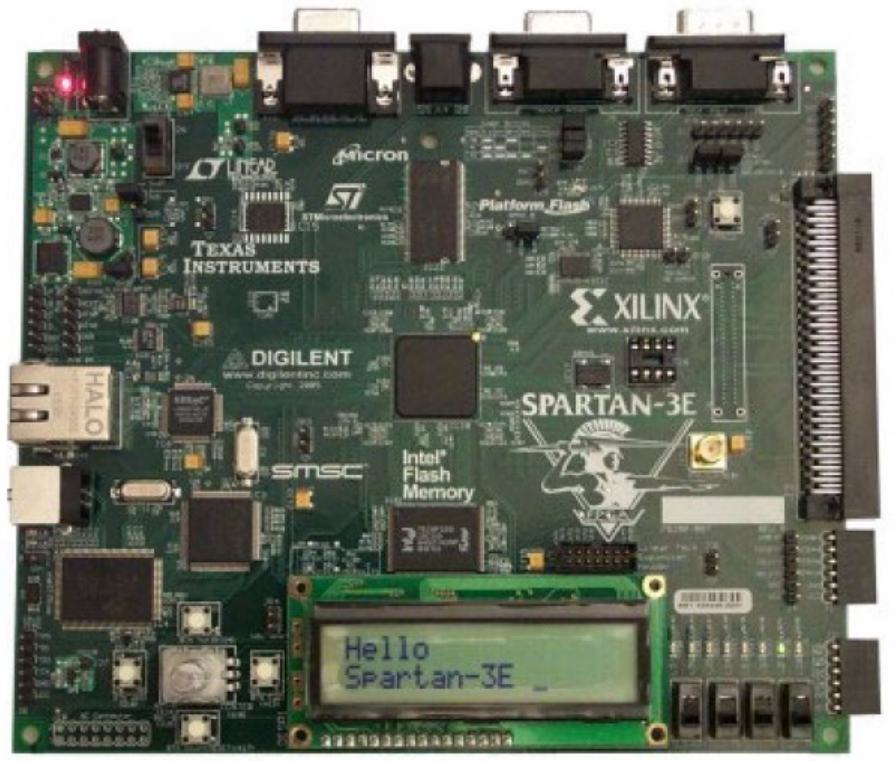

Figura 1. Tarjeta de desarrollo *Spartan-3E Starter Kit Board*.

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 SPI

La comunicación serial es una de las formas de intercambio de datos más usados para comunicación entre dispositivos. Existen diversos protocolos de comunicación serial, y en particular existen diversos protocolos orientados a la comunicación entre circuitos integrados, siendo los más comunes los protocolos SPI e I2C; la mayoría de los sensores con interfaz serial cuentan con la capacidad de utilizar éstos dos protocolos. Se decidió desarrollar el módulo para el puerto SPI debido a que éste protocolo permite comunicaciones a mayor velocidad ya que éste protocolo no requiere de direccionar a los dispositivos con que se comunica, se elimina el tiempo requerido para enviar éste dato en cada paquete de información. El protocolo utiliza tres líneas para la comunicación, entrada serial, salida serial y la señal de reloj, más una línea de habilitación para cada dispositivo que se conecte en el bus de comunicación. Dependiendo del tipo de dispositivo con que se haga la comunicación, se puede omitir la línea de entrada o salida serial si no son utilizadas (Ball, 2002).

El diseño de este módulo está basado en un registro de corrimiento universal, en el cual se almacena tanto el dato a enviar como el dato recibido. A continuación se muestra el código generado para implantar el módulo.

Listado 1. Código para un puerto SPI

```
------------------------------------------------
-- Libreria:    SPI
-- Descripcion: Módulo para comunicación SPI y
--              modulos de soporte (registro de
--              corrimiento universal y generador
--              de reloj)
------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity spi is
   Generic(
      n     : integer := 16;
      cpol  : STD_LOGIC := '0';
      cpha  : STD_LOGIC := '0';
      div   : integer := 500000
       );
   Port(
      clk     : in  STD_LOGIC;
     reset    : in  STD_LOGIC;
      --datos
      data_in : in  STD_LOGIC_VECTOR(0 to n - 1);
      data_out: out STD_LOGIC_VECTOR   (0 to n - 1);
      --control
      init_t  : in STD_LOGIC;
      trans : out STD_LOGIC;
      --protocolo
      spi_mosi: out STD_LOGIC;
      spi_miso: in  STD_LOGIC;
      spi_sclk: out STD_LOGIC;
      dac_cs  : out STD_LOGIC--;
         );
end spi;

------------------------------------------------

architecture Behavioral of spi is

component shift_reg_univ is
   generic (
         n : integer := 4
         );
    Port (
      clk  : in  STD_LOGIC;
      reset: in  STD_LOGIC;
      ctrl : in  STD_LOGIC_VECTOR ( 1 downto 0 );
      d    : in  STD_LOGIC_VECTOR ( n - 1 downto 0);
      q : out STD_LOGIC_VECTOR ( n - 1 downto 0)
      );
end component;

component clk_pol_ena is
Generic (
         div  : integer := 4 ;
         cpol : STD_LOGIC := '0'
        );
    Port (
        clk     : in  std_logic;
        e_clk   : in  std_logic;
        clk_out : out std_logic
        );
end component;

------------------------------------------------

signal s_clk  : STD_LOGIC;
signal s_d : STD_LOGIC_VECTOR ( n - 1 downto 0);
signal s_q : STD_LOGIC_VECTOR ( n - 1 downto 0);
signal f_d : STD_LOGIC;
signal clk_out  : STD_LOGIC;
type state_type is (E_idle,E_mode,E_carga,E_send);
signal state, next_state : state_type;
signal done     : STD_LOGIC;
signal ctrl : STD_LOGIC_VECTOR ( 1 downto 0 ) :=
"00";
signal e_clk  : STD_LOGIC;
signal sel_mux  : STD_LOGIC;
signal load     : STD_LOGIC;
signal trans_i     : STD_LOGIC;
signal data_in :STD_LOGIC_VECTOR(0 to n - 1) :=
"0110001101111111";
signal data_in :STD_LOGIC_VECTOR(0 to n - 1) :=
"1111111111000110";

begin
   shift : shift_reg_univ
         generic map   (n)
         Port map   (s_clk,reset,ctrl,s_d,s_q);
   clock : clk_pol_ena
         generic map (div,cpol)
         port map   (clk,e_clk,clk_out);

   --procesos
   spi_mosi <= s_q(n - 1);
   spi_sclk <= clk_out;

   s_d <= data_in(0 to n - 2) & f_d when sel_mux ='1'
   else data_in;

   s_clk <= clk_out when sel_mux ='1' else
         load;

   data_out <= (others => '0') when sel_mux ='1' else
```

```
          s_q;

process (clk_out)
begin
   if clk_out'event and clk_out = '1' then
     f_d <= spi_miso;
   end if;
end process;

process (s_clk,trans_i)
variable nbit : integer := 0;
begin
   if trans_i = '0' then
     done <= '0';
      nbit := 0;
   elsif s_clk'event and s_clk='0' then
     if nbit = n - 1 then
          nbit := 0;
          done <= '1';
        else
          nbit := nbit + 1;

     end if;
   end if;
end process;

process (clk,reset)   --- sincroniza las salidas
   begin
   if reset = '1' then
      state <= E_idle;
      trans <= '0';
   elsif clk'event and clk = '1' then
       state   <= next_state;
      trans <= trans_i;

   end if;
 end process;

process (state) --- decodificacion de las salidas
   begin
      case (state) is
         when E_idle =>
            ctrl    <= "00";
            sel_mux <= '0';
            load    <= '0';
            e_clk   <= '0';
            trans_i <= '0';
            dac_cs  <= '1';
         when E_mode =>
            ctrl    <= "11";
            sel_mux <= '0';
            load    <= '1';
            e_clk   <= '0';
            trans_i <= '0';
            dac_cs  <= '1';
         when E_carga =>
            ctrl    <= "11";
            sel_mux <= '0';
            load    <= '0';
            e_clk   <= '0';
            trans_i <= '0';
            dac_cs  <= '1';
         when E_send =>
            ctrl    <= "01";
            sel_mux <= '1';
            load    <= '0';
            e_clk   <= '1';
            trans_i <= '1';
            dac_cs  <= '0';
      end case;
end process;

process (state,init_t,done)
   begin
   next_state <= state;
     case (state) is
     when E_idle =>
        if init_t = '1' then
           next_state <= E_mode;
        end if;

     when E_mode =>
          next_state <= E_carga;

     when E_carga =>
          next_state <= E_send;

     when E_send =>
        if done = '1' then
             next_state <= E_idle;
        end if;

     end case;
   end process;
end Behavioral;

--------------------------------------
-- Registro de corrimiento universal
--------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity shift_reg_univ is
   generic (
        n : integer := 4
        );

    Port (
      clk  : in  STD_LOGIC;
      reset: in  STD_LOGIC;
      ctrl : in  STD_LOGIC_VECTOR ( 1 downto 0 );
      d    : in  STD_LOGIC_VECTOR ( n - 1 downto 0);
      q    : out STD_LOGIC_VECTOR ( n - 1 downto 0)
      );
end shift_reg_univ;

architecture Behavioral of shift_reg_univ is

   signal r_reg : STD_LOGIC_VECTOR ( n - 1 downto 0);
  signal r_next : STD_LOGIC_VECTOR ( n - 1 downto 0);

begin

   process(clk,reset)
   begin
      if(reset = '1') then
         r_reg <= (others => '0');
      elsif(clk'event and clk = '0') then
         r_reg <= r_next;
      end if;
   end process;

   with ctrl select
      r_next <=
      r_reg                              when "00",--no
opera
      r_reg(n - 2 downto 0) & d(0)     when "01",--
shift left
      d(n - 1) & r_reg(n - 1 downto 1) when "10",--
shift right
      d                                  when  others;--
load

   q <= r_reg;

end Behavioral;

--------------------------------------
-- Generador de reloj
--------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity clk_pol_ena is
   Generic (
         div  : integer := 4 ;
         cpol : STD_LOGIC := '0'
        );
   Port (
        clk     : in  std_logic;
        e_clk   : in  std_logic;
        clk_out : out std_logic
```

```
        );
end clk_pol_ena;

architecture Behavioral of clk_pol_ena is

   signal count : integer range 0 to div - 1  := 0;
   signal reg  : std_logic := '0';

begin

process(clk,e_clk)
begin
   if e_clk = '1' then
      if  clk'event and clk = '1'  then
         if count =(div - 1) then
            count <= 0;
            reg <= not reg;
         else
            count <= count + 1;
         end if;
      end if;
   else
      if cpol = '0' then
         reg <= '0';
      else
         reg <= '1';
      end if;
   end if;
end process;

clk_out <= reg;

end Behavioral;
```

2. Mouse PS2

Dado que el mouse, como dispositivo de entrada, es de los elementos de interacción con el usuario más utilizados, se desarrolló éste módulo para completar el módulo para el uso de un teclado PS2 que se desarrolló anteriormente. Por ello se desarrolló esta biblioteca para manejar un mouse PS2, que es de uno de los tipos de mouse estándar más utilizado en la actualidad. Los mouses con interfaz RS232 se pueden utilizar por medio del controlador UART desarrollado previamente, y los mouses con interfaz USB se pueden utilizar con la interfaz descrita más adelante. El controlador determina el desplazamiento en los ejes X y Y, así como el estado de los botones del mouse (soporta mouses de hasta 3 botones).

Listado 2. Código para un controlador para un mouse PS2

```
-------------------------------------------------
-- Module Name: MouseFPGA
-- Descripcion: Controlador para mouse con
--              interfaz PS2
-------------------------------------------------
library ieee;
use ieee. std_logic_1164.all;
use ieee. numeric_std.all;

entity mousefpga is
   port(
      clk ,reset: in std_logic;
      ps2d, ps2c : inout std_logic ;
      xm, ym: out std_logic_vector (8 downto 0);
      btnm: out std_logic_vector(2 downto 0 );
      m_done_tick : out std_logic
      );
end mousefpga;

architecture Behavioral of mousefpga is
   constant STRM: std_logic_vector (7 downto 0) :=
"11110100";
   -- stream command F4
   type state_type is (init1 ,init2 ,init3 ,pack1
,pack2 ,pack3 ,done);
   signal state_reg ,state_next : state_type;
   signal rx_data: std_logic_vector (7 downto 0) ;
   signal rx_done_tick ,tx_done_tick : std_logic;
   signal wr_ps2 : std_logic;
   signal x_reg ,y_reg : std_logic_vector(8 downto 0);
   signal x_next ,y_next : std_logic_vector(8 downto 0
);
   signal btn_reg ,btn_next : std_logic_vector(2
downto 0);

begin
   --instanciacion de modulos
   ps2_rxtx_unit : entity work.ps2_rxtx (Behavioral)
         port map(clk => clk,
         reset => reset,
         wr_ps2 => wr_ps2,
         din => STRM,
         dout => rx_data,
         ps2d => ps2d,
         ps2c => ps2c,
         rx_done_tick => rx_done_tick,
         tx_done_tick => tx_done_tick);

   -- registros de estado y datos
   process (clk, reset)
   begin
      if reset = '1' then
         state_reg <= init1;
         x_reg <= (others => '0');
         y_reg <= (others => '0');
         btn_reg <= (others => '0');
      elsif(clk'event and clk='1') then
         state_reg <= state_next ;
         x_reg <= x_next;
         y_reg <= y_next;
         btn_reg <= btn_next;
      end if;
   end process;

   -- secuencia de estados
   process (state_reg, rx_done_tick, tx_done_tick,
x_reg, y_reg, btn_reg, rx_data)
   begin
      wr_ps2 <= '0';
      m_done_tick <= '0';
      x_next <= x_reg;
      y_next <= y_reg;
      btn_next <= btn_reg ;
      state_next <= state_reg;
      case state_reg is
         when init1 =>
            wr_ps2 <= '1';
            state_next <= init2 ;
         when init2 => --espera para completar envio
            if tx_done_tick = '1' then
               state_next <= init3;
            end if ;
         when init3 => --espera por acknowledge
            if rx_done_tick = '1' then
               state_next <= pack1 ;
            end if ;
         when pack1 => --espera por primer paquete de
datos
```

```
            if rx_done_tick = '1' then
               state_next <= pack2 ;
               y_next(8) <= rx_data(5) ;
               x_next(8) <= rx_data(4) ;
               btn_next <= rx_data(2 downto 0);
            end if;
         when pack2 => --wait for 2 nd data packet
            if rx_done_tick = '1' then
               state_next <= pack3;
               x_next(7 downto 0) <= rx_data;
            end if;
         when pack3 => --wait for 3 rd data packet
            if rx_done_tick = '1' then
               state_next <= done;
               y_next(7 downto 0) <= rx_data;
            end if;
         when done =>
            m_done_tick <= '1';
            state_next <= pack1;
      end case;
   end process;

   xm <= x_reg;

   ym <= y_reg;

   btnm <= btn_reg;

end Behavioral;
```

3. Video resolución SVGA

El monitor es un periférico que es muy conveniente de manejar, 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. Previamente desarrollamos un generador de señales de sincronía para manejar un monitor con resolución VGA (640 x 480 pixeles), pero consideramos que para ciertas aplicaciones es conveniente poder manejar despliegues en video con mayor resolución. Por esta razón desarrollamos el módulo para generar señales de video con resolución Súper-VGA (SVGA) de 800 x 600 pixeles, el cual se presenta a continuación. Para la generación de las imágenes a desplegar se puede seguir utilizando cualquiera de los mecanismos descritos para la resolución VGA con la única modificación del tamaño máximo de los elementos a desplegar que en este caso es mayor.

Listado 3. Código para un controlador para video SVGA

```
----------------------------------------------------
-- Module Name: SVGASync
-- Descripcion: Generación de señales de sincronia
--              para monitor SVGA (800 x 600)
----------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity svga_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 svga_sync;

architecture arch of svga_sync is
   -- Parametros SVGA 800 x 600 sync
   constant HD: integer:=800; -- h. display area
   constant HF: integer:=48 ; -- h. front porch
   constant HB: integer:=56 ; -- h. back porch
   constant HR: integer:=120; -- h. retrace
   constant VD: integer:=600; -- v. display area
   constant VF: integer:=23 ; -- v. front porch
   constant VB: integer:=37 ; -- v. back porch
   constant VR: integer:=6  ; -- v. retrace

   signal mod1_reg, mod1_next: std_logic; -- sync
counters
   signal v_count_reg, v_count_next: unsigned(9 downto
0);
   signal h_count_reg, h_count_next: unsigned(9 downto
0);
   -- output buffer
   signal v_sync_reg, h_sync_reg: std_logic;
   signal v_sync_next, h_sync_next: std_logic;
   -- status signal
   signal h_end, v_end, pixel_tick: std_logic;
begin
   -- registers
process (clk)
begin
   if reset='1' then
      mod1_reg <= '1';
   else
      mod1_reg <= not clk;
   end if;
end process;

process (clk , reset)
begin
   if reset='1' then
      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
      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;
```

```
mod1_next <= not mod1_reg;
-- 50 MHz pixel tick
pixel_tick <= '1' when mod1_reg='1' else '0';
-- status
h_end <=          -- final horizontal counter
   '1' when h_count_reg=(HD+HF+HB+HR-1) else   -- 1039
   '0';
v_end <= -- end of vertical counter
   '1' when v_count_reg=(VD+VF+VB+VR-1) else   -- 665
   '0';

-- mod-1040 horizontal sync counter
process (h_count_reg,h_end,pixel_tick)
begin
   if pixel_tick='1' then   -- 50 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-666 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;

-- horizontal & vertical sync buffer
h_sync_next <=
   '1' when (h_count_reg>=(HD+HF))   -- 864
     and (h_count_reg<=(HD+HF+HR-1)) else   -- 983
   '0';
v_sync_next <=
   '1' when (v_count_reg>=(VD+VF))   -- 623
     and (v_count_reg<=(VD+VF+VR-1)) else   -- 628
   '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;
   p_tick <= mod1_reg;
end arch;
```

4. Control PWM

La señal PWM (Pulse Width Modulation – modulación por ancho de pulso) si bien tiene sus orígenes como un tipo de modulación en los sistemas de comunicaciones digitales, su uso más frecuente es en los sistemas de control, en donde es muy utilizada en una gran variedad de aplicaciones. Un ejemplo de uso de estos sistemas es en el control de los servomotores de modelismo, en los cuales la posición del rotor dentro de su rango de giro (típicamente 180°) depende del ancho de pulso de la señal de control. En los servomotores estándar la posición varía linealmente respecto al ancho del pulso que varía en el rango de 1 a 2 mS en una señal con período de 20 mS, es decir, se utiliza una señal PWM de 50 Hz cuyo ciclo de trabajo varíe entre el 5% y el 10%. Pero quizá su uso más importante está en el uso de control de potencia eléctrica. Dado que es relativamente difícil construir fuentes de voltaje variable, especialmente de muy alta potencia, se utilizan los circuitos PWM de frecuencia alta (en relación a la velocidad de respuesta del dispositivo a controlar) de tal manera que el voltaje efectivo que ven los dispositivos equivale al voltaje promedio que reciben, el cual está dado por el producto del voltaje de alimentación multiplicado por el ciclo de trabajo. Esto se ocupa mucho por ejemplo en los motores de corriente directa para regular su velocidad, ya que ésta es proporcional al voltaje con que es alimentado, y al utilizar un PWM el voltaje con el que trabaja el motor, y por tanto la velocidad a la que gira corresponde al voltaje promedio que recibe. Otro ejemplo de su uso cada vez más común es para controlar la intensidad de las lámparas de LEDs, ya que en estos dispositivos la intensidad de la luz varía linealmente respecto a la corriente que circula a través de ellos, pero como la corriente no varía de forma lineal respecto al voltaje, sino exponencialmente como en cualquier otro diodo, y hacer fuentes de corriente variables es más complicado, se utilizan circuitos PWM en donde la potencia luminosa que ve el ojo nuevamente corresponde a la potencia promedio dada por el ciclo de trabajo, únicamente teniendo el cuidado que la frecuencia a la que cambia el PWM sea mayor al tiempo de retención del ojo humano para que no sea perceptible el parpadeo de la luz (25 Hz o mayor es suficiente para ser imperceptible y no causar fatiga visual). El módulo desarrollado para ésta función se muestra a continuación. En éste módulo se utiliza una palabra de 16 bits para definir el ancho del pulso, donde el ciclo de trabajo está dado por `(limite * 100 / 65536)`, por lo que un valor de 0 corresponderá a un ciclo de trabajo de 0% (siempre apagado) y un valor de 65535 a un ciclo de trabajo del 99.998% (siempre encendido).

Listado 4. Código para la generación de una señal PWM

```
-------------------------------------------------
-- Libreria:    PWM
-- Descripcion: Librería para la generación de una
--              señal modulada por ancho de pulso
--              (PWM)
--                                          *RPM
-------------------------------------------------

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

entity pwm is
   port (clk: in STD_LOGIC;
         limite: in STD_LOGIC_VECTOR(15 downto 0);
         pwmout: out STD_LOGIC);
end pwm;

architecture Behavioral of pwm is

begin

   cPWM: process(clk)
      variable cuenta: STD_LOGIC_VECTOR(15 downto
0):=X"0000";
   begin
      if rising_edge(clk) then
         cuenta := cuenta + 1;
         if cuenta>limite then
            pwmout <= '1';
         else
            pwmout <= '0';
         end if;
      end if;
   end process;

end Behavioral;
```

5. Comunicación con microcontroladores y puerto USB

Para que un sistema embebido se pueda comunicar con el exterior, la forma más utilizada es hacerlo por medio de un puerto USB, que en la actualidad se ha convertido en el puerto estándar de mayor uso. Si bien es cierto que, por el hecho de ser un dispositivo digital, es posible desarrollar el código de un transceptor USB para insertarlo en un FPGA, la complejidad y por tanto el espacio que ocuparía este dispositivo en un FPGA hace que no sea la mejor opción. Además, dependiendo del modo de transmisión y de la velocidad de transferencia, se requiere hacer un acoplamiento de voltajes de precisión, a niveles de voltaje que no sin comunes en los sistemas basados en FPGAS, además de que también se requiere acoplar impedancias, y todo esto se traduce en tener que agregar dispositivos externos al FPGA. Es por ello que en la práctica, aunque existen diversas implantaciones de puertos USB para FPGAs, por el costo que esto representa, resulta más económico utilizar un microcontrolador que cuente entre sus periféricos una interfaz USB. Esta es la solución más utilizada en la actualidad en el desarrollo de sistemas con interfaz USB, ya que en estos microcontroladores al ser programables son más versátiles que los simples transceptores que se utilizaban antiguamente, y por su tamaño y costo reducido, en la actualidad son la mejor opción para implantar éste tipo de puertos.

Al desarrollar un sistema embebido que utilice dos elementos de procesamiento, como es el caso de usar un FPGA y un microcontrolador en un mismo diseño, provoca que surja un nuevo aspecto a considerar, la comunicación entre los dos elementos, así como la sincronización de las operaciones realizadas entre ambos. Esta situación se resuelve estableciendo cual va a ser el protocolo de comunicación entre ambos dispositivos, para lo cual se puede utilizar ya sea una comunicación serial, como las ya desarrolladas (UART o SPI), o bien establecer una comunicación dedicada en paralelo. El tipo de comunicación que sea más conveniente de utilizar depende de la cantidad de información que se necesita intercambiar y de la velocidad con que esta se presenta. Si el volumen de información es alto entonces se recomienda usar un puerto paralelo dedicado, para lo cual se puede usar en los microcontroladores cualquier puerto de acceso paralelo, que dependiendo del microcontrolador y el puerto interno utilizado puede ser desde 4 y hasta 16 bits, y en el FPGA solamente se define un puerto con el tamaño adecuado, y se pueden definir líneas de sincronización, tal como se hace para manejar los módulos de la biblioteca de funciones desarrollada. Si el volumen de datos a intercambiar no es muy grande, o la velocidad de transferencia requerida no es demasiado alta, se puede utilizar algún tipo de puerto serial, aprovechando que en todos los microcontroladores existen cuentan con esta clase de puertos entre sus periféricos.

A continuación se muestra el código genérico para utilizar un puerto USB en un microcontrolador PIC 18Fx550 de Microchip, utilizando un compilador CCS PCWH.

Listado 5. Código para el uso de un puerto USB en un microcontrolador

```
////////////////////////////////////////////////////////////////////
/// Modulo: PicUSB.h                                             ///
///                                                              ///
/// Este ejemplo muestra como configurar el dispositivo USB y sus ///
/// descriptores. Los únicos cambios con respecto a su versión    ///
/// original (usb_desc_scope.h) han sido realizados en el apartado ///
/// start device descriptors, concretamente el vendor y product id ///
/// y en apartado de start string descriptors, para definir el    ///
/// nombre del dispositivo y la compañia.                        ///
///                                                              ///
/// Realizado con el compilador CCS PCWH 4.018                   ///
///                                                              ///
////////////////////////////////////////////////////////////////////

#IFNDEF __USB_DESCRIPTORS__
#DEFINE __USB_DESCRIPTORS__

#include <usb.h>

///   start config descriptor
   #DEFINE USB_TOTAL_CONFIG_LEN      32 //config+interface+class+endpoint

   //configuration descriptor
   char const USB_CONFIG_DESC[] = {
   //config_descriptor for config index 1
         USB_DESC_CONFIG_LEN,     //length of descriptor size
         USB_DESC_CONFIG_TYPE,         //constant CONFIGURATION (0x02)
         USB_TOTAL_CONFIG_LEN,0,  //size of all data returned for this config
         1,      //number of interfaces this device supports
         0x01,                //identifier for this configuration.
         0x00,                //index of string descriptor for this configuration
         0xC0,                //bit 6=1 if self powered, bit 5=1 if supports remote wakeup, bits
0-4 reserved and bit7=1
         0x32,                //maximum bus power required (maximum milliamperes/2)  (0x32 =
100mA)

   //interface descriptor 0 alt 0
         USB_DESC_INTERFACE_LEN,  //length of descriptor
         USB_DESC_INTERFACE_TYPE,      //constant INTERFACE (0x04)
         0x00,                //number defining this interface (IF we had more than one
interface)
         0x00,                //alternate setting
         2,                   //number of endpoints, not counting endpoint 0.
         0xFF,                //class code, FF = vendor defined
         0xFF,                //subclass code, FF = vendor
         0xFF,                //protocol code, FF = vendor
         0x00,                //index of string descriptor for interface

   //endpoint descriptor
         USB_DESC_ENDPOINT_LEN, //length of descriptor
         USB_DESC_ENDPOINT_TYPE,     //constant ENDPOINT (0x05)
         0x81,              //endpoint number and direction (0x81 = EP1 IN)
         0x02,              //transfer type supported (0 is control, 1 is iso, 2 is bulk, 3 is
interrupt)
         USB_EP1_TX_SIZE & 0xFF, USB_EP1_TX_SIZE >> 8,         //maximum packet size supported
//USB_EP1_TX_SIZE,0x00,         //maximum packet size supported
         0x01,              //polling interval in ms. (for interrupt transfers ONLY)

   //endpoint descriptor
         USB_DESC_ENDPOINT_LEN, //length of descriptor
         USB_DESC_ENDPOINT_TYPE,     //constant ENDPOINT (0x05)
         0x01,              //endpoint number and direction (0x01 = EP1 OUT)
         0x02,              //transfer type supported (0 is control, 1 is iso, 2 is bulk, 3 is
interrupt)
         USB_EP1_RX_SIZE & 0xFF, USB_EP1_RX_SIZE >> 8,         //maximum packet size supported
//USB_EP1_RX_SIZE,0x00,         //maximum packet size supported
         0x01,              //polling interval in ms. (for interrupt transfers ONLY)
  };
```

```
   //****** BEGIN CONFIG DESCRIPTOR LOOKUP TABLES ********
   #define USB_NUM_HID_INTERFACES   0
   //the maximum number of interfaces seen on any config
   #define USB_MAX_NUM_INTERFACES   1
   //define how many interfaces there are per config.  [0] is the first config, etc.
   const char USB_NUM_INTERFACES[USB_NUM_CONFIGURATIONS]={1};

   #if (sizeof(USB_CONFIG_DESC) != USB_TOTAL_CONFIG_LEN)
      #error USB_TOTAL_CONFIG_LEN not defined correctly
   #endif

///   start device descriptors
   //device descriptor
   char const USB_DEVICE_DESC[] ={
         USB_DESC_DEVICE_LEN,          //the length of this report
         0x01,                //constant DEVICE (0x01)
         0x10,0x01,           //usb version in bcd
         0x00,                //class code (if 0, interface defines class.  FF is vendor defined)
         0x00,                //subclass code
         0x00,                //protocol code
         USB_MAX_EP0_PACKET_LENGTH,   //max packet size for endpoint 0. (SLOW SPEED SPECIFIES 8)
         0xd8,0x04,           //vendor id (0x04D8 is Microchip)
         0x01,0x00,           //product id
         0x01,0x00,           //device release number
         0x01,                //index of string description of manufacturer.
         0x02,                //index of string descriptor of the product
         0x00,                //index of string descriptor of serial number
         USB_NUM_CONFIGURATIONS   //number of possible configurations
   };

///   start string descriptors

//the offset of the starting location of each string.
const char USB_STRING_DESC_OFFSET[]={0,4,12};

#define USB_STRING_DESC_COUNT sizeof(USB_STRING_DESC_OFFSET)

char const USB_STRING_DESC[]={
   //string 0 -->serial number
         4, //length of string index
         USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
         0x09,0x04,   //Microsoft Defined for US-English
   //string 1 --> la compañia del producto
         8, //length of string index
         USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
         'A',0,
         'B',0,
         'C',0,
   //string 2 --> nombre del dispositivo
         16, //length of string index
         USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
         'U',0,
         'N',0,
         'A',0,
         'M',0,
         '.',0,
         ' ',0
};
#ENDIF

////////////////////////////////////////////////////////////////////
/// Modulo: PicUSB.c                                             ///
/// Para una configuración estándar, permite obtener y enviar    ///
/// datos usando el puerto USB en modo Full-Speed                ///
///                                                              ///
/// Realizado con el compilador CCS PCWH 4.018                   ///
////////////////////////////////////////////////////////////////////
#include <18F4550.h>
#device ADC=8
#fuses HSPLL,NOWDT,NOPROTECT,LVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN
```

```
  #use delay(clock=48000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)

#define USB_HID_DEVICE     FALSE             //deshabilitamos el uso de las directivas HID
#define USB_EP1_TX_ENABLE  USB_ENABLE_BULK   //turn on EP1(EndPoint1) for IN bulk/interrupt
transfers
#define USB_EP1_RX_ENABLE  USB_ENABLE_BULK   //turn on EP1(EndPoint1) for OUT bulk/interrupt
transfers
#define USB_EP1_TX_SIZE    64                //size to allocate for the tx endpoint 1 buffer
#define USB_EP1_RX_SIZE    8                 //size to allocate for the rx endpoint 1 buffer

//#define USB_CON_SENSE_PIN PIN_B2

#include <pic18_usb.h>   //Microchip PIC18Fxx5x Hardware layer for CCS's PIC USB driver
#include <PicUSB.h>      //Configuración del USB y los descriptores para este dispositivo
#include <usb.c>         //handles usb setup tokens and get descriptor reports


////////////////////////////////////////////////////////////////////////////
//
// Ejemplo genérico de uso del módulo de comunicación USB
// Se considera que hay 2 LEDs conectados (verde y rojo a los pines B6 y B7
// respectivamente). El LED rojo indica que el puerto esta activo y el verde
// que el puerto fue enumerado por el host.
// El código aqui presentado corresponde a una plantilla de propósito general
// al cual se le puede adicionar el código correspondiente a la aplicacion
// se este desarrollando.
//
////////////////////////////////////////////////////////////////////////////

void main(void) {

   output_low(PIN_B6);              //enciende led rojo
   output_high(PIN_B7);

   printf("Inicializando\n");
   usb_init();                      //inicializacion del puerto USB
   printf("Inicializacion terminada\n");

   usb_task();                      //habilita periferico usb e interrupciones
   printf("Esperando enumeracion\n");
   usb_wait_for_enumeration();      //esperamos hasta que el Pic sea configurado por el host

   output_low(PIN_B7);
   output_high(PIN_B6);                    //encendemos led verde

   while (TRUE)
   {
      if(usb_enumerated())          //si el Pic está enumerado
      {
         printf("Enumerado:OK\n");

////////////////////////////////////////////////////////////////////////////
//
// Aqui se agrega el codigo que depende de la aplicacion deseada, en donde
// se pueden usar las funciones que se ejemplifican a continuacion para el
// intercambio de información.
//
//       if (usb_kbhit(1))   // verifica si el endpoint de salida contiene datos del host
//
//       usb_get_packet(1, recibe, n_bytes);   // lee datos del endpoint
//
//       usb_put_packet(1, envia, n_bytes, USB_DTS_TOGGLE); // envia datos al endpoint
//
////////////////////////////////////////////////////////////////////////////
      }
   }
}
```

6. Ejemplos de uso

Para ilustrar la utilización de los módulos desarrollados, 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 mouse PS2, utilizando también el módulo para el control del display LCD. Con este programa se reciben los valores de desplazamiento del mouse, mismos que se van acumulando y el resultado es desplegado en el display LCD; los valores ahí observados corresponderían a las coordenadas en las que se encontraría el puntero si éste fuera desplegado en un monitor.

Listado 6. Código de ejemplo que lee la posición del mouse y lo despliega en el display LCD.

```
----------------------------------------------------
-- Modulo: Prueba de mouse PS/2
-- Descripcion: Lee estado del mouse y despliega
--              la posicion del puntero en el
--              display LCD.
----------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_signed.ALL;

entity app_mouse_lcd is
port(
   clk, reset : in std_logic;
   ps2d, ps2c : inout std_logic;
   SF_D : out std_logic_vector(3 downto 0);
   carac :out std_logic_vector(7 downto 0);
   LCD_E, LCD_RS, LCD_RW, SF_CE0  : out std_logic
   );
end app_mouse_lcd;

architecture Behavioral of app_mouse_lcd 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 mousefpga is
      port(
      clk ,reset: in std_logic;
      ps2d, ps2c : inout std_logic ;
      xm, ym: out std_logic_vector (8 downto 0);
      btnm: out std_logic_vector(2 downto 0 );
      m_done_tick : out std_logic
      );
   end component;

   component convertTostringNumbers is
      port    (
         clk, reset : in std_logic;
         numberToconvert : in std_logic_vector(10
downto 0);
         initConvert : in std_logic;
         tx_char: in std_logic;
         char : out std_logic_vector(7 downto 0);
         doneNum : out std_logic
         );
   end component;

  --lista de señales de interconeccion  y señales de
interfaz
  signal value_coorX: std_logic_vector(10 downto 0)
:= "00000000000";
  signal value_coorY: std_logic_vector(10 downto 0)
:= "00000000000";
  signal mux : std_logic;

  --Señales de interconexion para el convertidor
  signal xm: std_logic_vector(8 downto 0);
  signal ym: std_logic_vector(8 downto 0);
  signal btnm: std_logic_vector(2 downto 0);
  signal m_done_tick: std_logic;
  --Señales de interconexion para el lcd
  signal tx_byte :  std_logic_vector(7 downto 0);
  signal tx_init :  std_logic;
  signal tx_done :  std_logic;
  signal init_init : std_logic;
  signal init_done :  std_logic;
  signal lcdrs : std_logic := '0';
  --Señales de interconexion para el mouse
  signal numberToconvert : std_logic_vector(10 downto
0) := "00000000000";
  signal initConvert : std_logic;
  signal tx_char : std_logic;
  signal char : std_logic_vector(7 downto 0);
  signal doneNum : std_logic;

  type display_state is ( init, function_set,
     entry_set, set_display, clr_display,
     pause,set_addr,send_open_parentesis,
     init_conver,idle,init_tx,send
      --,send_close_parentesis, idle
     );
  signal state_reg, state_next : display_state
:=init;

begin

  mouse : mousefpga port map(clk, reset, ps2d,   ps2c,
xm, ym, btnm, m_done_tick);

  display : 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);

  convertidor : convertTostringNumbers port map(clk,
reset, numberToconvert,initConvert, tx_char, char,
doneNum);

-- controla la secuencia de initializacion
   with state_reg select
      init_init <= '1' when init,
                   '0' when others;

--selecciona registro
   with state_reg select
         lcdrs <= '0' when function_set | entry_set |
set_display | clr_display | set_addr,
                  '1' when others;

with mux select
```

```
        numberToconvert<= value_coorX when '0',
      -- Transmite character
        value_coorY  when others;       -- Inicializa

process(clk, reset)
begin
   if reset = '1' then
   state_reg <= init ;
   elsif ( clk'event and clk = '1') then
   state_reg <= state_next;
   end if ;
end process;

carac <= char;

-- Máquina de Estados Principal
process(state_reg, tx_done , doneNum , init_done )
variable int : integer range 0 to 18200000 := 0;
begin
   initConvert <= '0';
   tx_char <= '0';
   tx_init <= '0';
   state_next <= state_reg;

   case state_reg is

         when  init =>
            tx_byte <= "00000000";
            tx_init <= '0';
            if(init_done = '1') then
               state_next <= function_set;
            else
               state_next <= init;
            end if;

         when  function_set =>
               tx_byte <= "00101000";
               tx_init <= '1';
               if(tx_done = '1') then
                  state_next <= entry_set;
               else
                  state_next <= function_set;
               end if;

         when  entry_set =>
               tx_byte <= "00000110";
               tx_init <= '1';
               if(tx_done = '1') then
                  state_next <= set_display;
               else
                  state_next <= entry_set;
               end if;

         when  set_display =>
               tx_byte <= "00001100";
               tx_init <= '1';
               if(tx_done = '1') then
                  state_next <= clr_display;
               else
                  state_next <= set_display;
               end if;

         when  clr_display =>
               tx_byte <= "00000001";
               tx_init <= '1';
               if(tx_done = '1') then
                  state_next <= pause;
               else
                  state_next <= clr_display;
               end if;
         when pause =>
               tx_byte <= "00000000";
               tx_init <= '0';
               if(int = 82000) then
                  state_next <= set_addr;
                  int := 0;
               else
                  state_next <= pause;
                  int := int + 1;
               end if;
         when set_addr =>
               tx_byte <= "10000000";
               tx_init <= '1';
               if(tx_done = '1') then
                  state_next <= send_open_parentesis;
               else
                  state_next <= set_addr;
               end if;
         when send_open_parentesis =>
               tx_byte <= X"28";
               tx_init <= '1';
               if (tx_done = '1') then
                  state_next <= init_conver;
               else
                  state_next <= send_open_parentesis;
               end if;

         when init_conver =>
               initConvert <= '1';
               tx_init <= '0';
               state_next <= init_tx;

         when init_tx =>
               tx_char <= '1';
               tx_init <= '0'  ;
               state_next <= idle;

         when send =>
               tx_byte <= char;
               tx_init <= '1';
               if (tx_done = '1' ) then
                  if (doneNum = '0') then
                     state_next <= idle;
                     tx_char <= '1';
                  else
                     state_next <=send_close_parentesis;
                  end if;
               else
                  state_next <= send;
               end if;

            when send_close_parentesis =>
               tx_byte <= X"29";
               tx_init <= '1';
               if (tx_done = '1') then
                  state_next <= idle;
               else
                  state_next <= send_close_parentesis;
               end if;

         when idle =>
                     state_next <= idle;

      end case;

end process ;

process (reset,m_done_tick)
begin
   if reset = '1' then
      value_coorX <= "00000000000";
      value_coorY <= "00000000000";
   elsif (m_done_tick = '1') then
      value_coorX <= value_coorX + "00" & xm;
      value_coorY <= value_coorY + "00" & ym;
   end if;
end process;

end Behavioral;

end prueba;
```

En la figura 2 se observa la tarjeta de desarrollo con el mouse conectado y en el display mostrando las coordenadas del cursor.

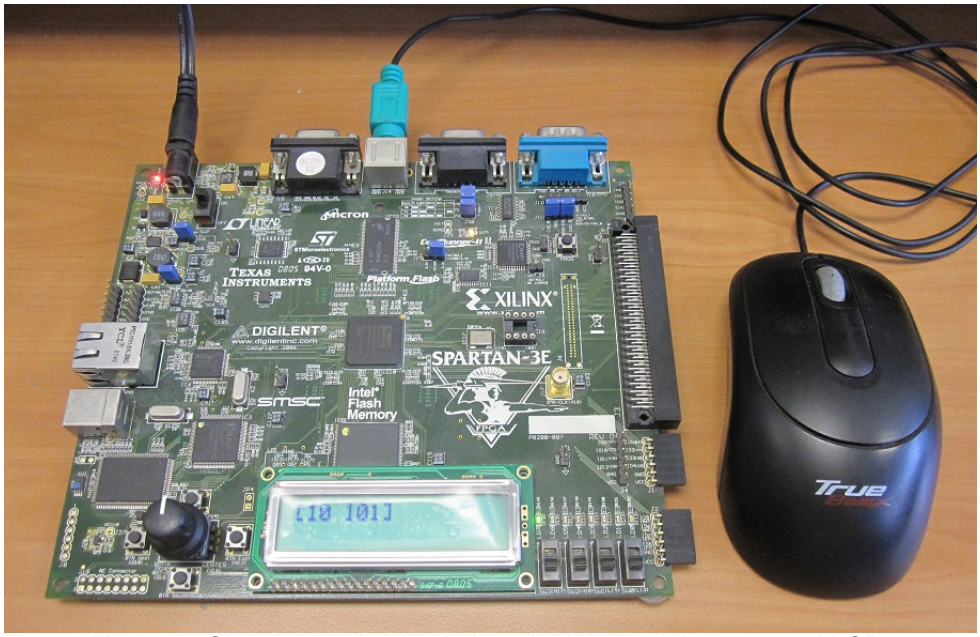

Figura 2. Sistema corriendo el ejemplo del uso del mouse y el LCD.

En el siguiente ejemplo se muestra la utilización del módulo SVGA 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 la parte superior de la pantalla con barras verticales seguidas de barras horizontales, mientras que en 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 7. Código de ejemplo que genera una imagen de referencia en un monitor SVGA.

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

entity svga_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 svga_test;

architecture prueba of svga_test is
   component svga_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 SVGA sync
   svga_sync_unit : svga_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<450 then
            if pixel_x<100 then
               rgb_reg <= "000";
            elsif pixel_x<200 then
               rgb_reg <= "001";
            elsif pixel_x<300 then
               rgb_reg <= "010";
            elsif pixel_x<400 then
               rgb_reg <= "011";
            elsif pixel_x<500 then
               rgb_reg <= "100";
            elsif pixel_x<600 then
               rgb_reg <= "101";
            elsif pixel_x<700 then
               rgb_reg <= "110";
            else
               rgb_reg <= "111";
            end if;
         elsif pixel_y<490 then
               rgb_reg <= "001";
         elsif pixel_y<530 then
```

```
      rgb_reg <= "010";
elsif pixel_y<570 then
      rgb_reg <= "100";
else
   rgb_reg <= sw;
end if;
```

```
    end if;
  end process;

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

En la figura 3 se muestra la imagen con el patrón cromático de referencia generado en éste ejemplo, manejando un monitor con resolución SuperVGA.

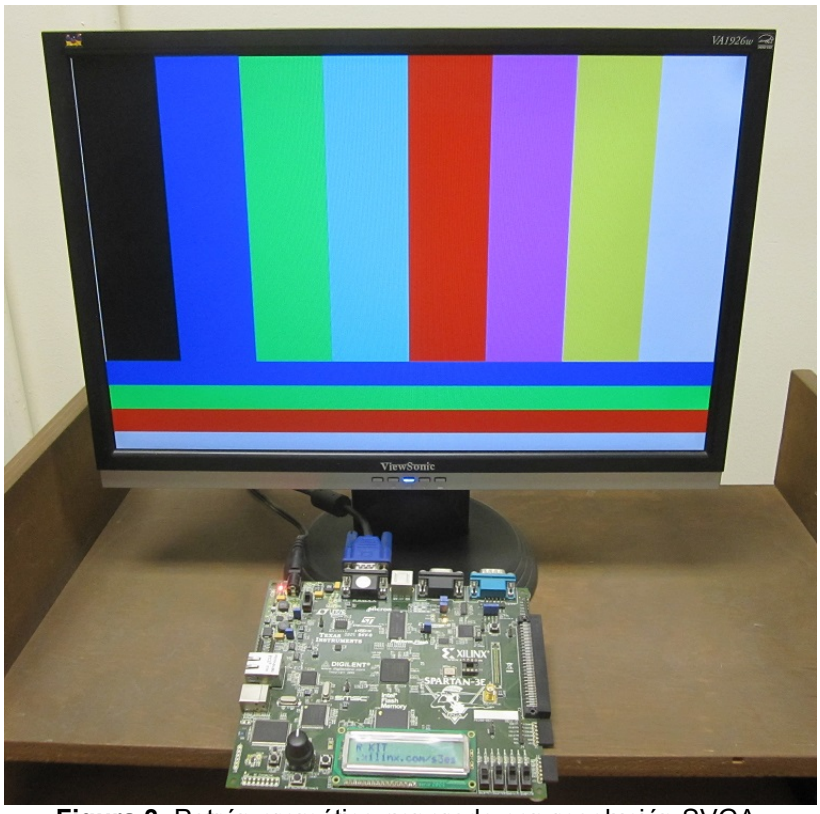

Figura 3. Patrón cromático generado con resolución SVGA.

En el siguiente ejemplo se muestra la utilización del módulo PWM para controlar la intensidad de encendido de 8 LEDs a partir de la entrada proveniente de 2 botones, uno para incrementar la intensidad y otro para disminuirla. Los LEDs de salida se dividen en 2 bloques, la mitad de ellos muestra la intensidad de acuerdo a la salida del PWM y la otra mitad muestran el complemento de la intensidad, es decir, mientras la mitad de los LEDs incrementa su brillo la otra mitad disminuye el brillo en la misma proporción. Si en vez de utilizar LEDs normales se utilizan LEDs bicolor conectando a uno de los ánodos la señal directa del PWM y al otro cátodo la salida complementaria, lo que se verá en estos LEDs es el cambio en la tonalidad de encendido. Ésta es la técnica que se utiliza en los LEDs RGB para generar luces con diferentes tonos de color utilizando éste tipo de LEDs.

Listado 8. Código de ejemplo para controlar la intensidad de encendido de un conjunto de LEDs.

```
-------------------------------------------------
-- Modulo: Prueba PWM
-- Descripcion: Control de intensidad de LEDs por
--              medio de una señal PWM.
--                                          *RPM
-------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

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

entity leds is
   port (reloj, sube, baja: in STD_LOGIC;
         leds: out STD_LOGIC_VECTOR(7 downto 0));
end leds;
```

```
architecture Behavioral of leds is
   component pwm is
   port (clk: in STD_LOGIC;
         limite: in STD_LOGIC_VECTOR(15 downto 0);
         pwmout: out STD_LOGIC);
   end component;

   signal clk: STD_LOGIC;
   signal salida: STD_LOGIC;
   signal limite: STD_LOGIC_VECTOR(7 downto
0):="10000000";
begin
   -- instancia generador PWM
   pwm_unit : pwm
      port map(clk, limite, salida);

   divisor: process (reloj)
      variable cuenta: STD_LOGIC_VECTOR(27 downto
0):=X"0000000";
   begin
      if rising_edge(reloj) then
         if cuenta=X"48009E0" then
            cuenta:=X"0000000";
         else
            cuenta:=cuenta+1;
         end if;
      end if;
      clk <= cuenta(10);
   end process;

   dLimite: process(sube, baja)
      variable cuenta: STD_LOGIC_VECTOR(7 downto
0):="10000000";
   begin
      if sube='1' then
         if cuenta<245 then
            cuenta := cuenta + 10;
         end if;
      end if;
      if baja='1' then
         if cuenta>10 then
            cuenta := cuenta - 10;
         end if;
      end if;
      limite <= cuenta;
   end process;

   with salida select
      leds <= "00001111" when '0',
              "11110000" when '1';

end Behavioral;
```

En la figura 4 se observa el control de la intensidad de encendido de los LEDs en la tarjeta de desarrollo.

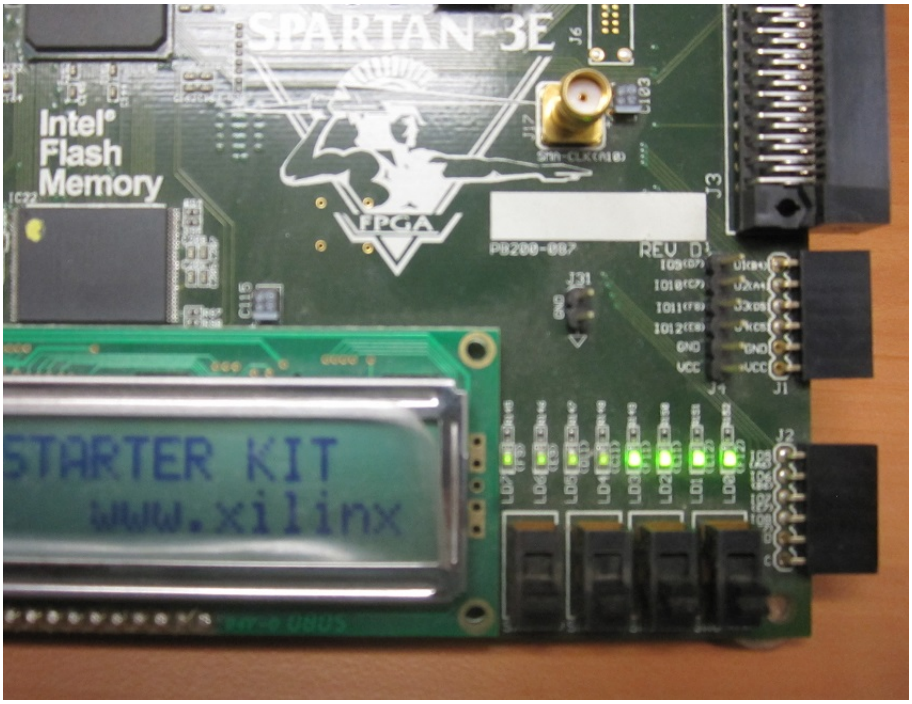

Figura 4. Ejemplo del control de intensidad de los LEDs con un módulo PWM.

Finalmente, en el siguiente ejemplo se muestra una aplicación en la cual se utiliza la comunicación USB, tomando como base el código genérico mostrado anteriormente y considerando que, por las razones mencionadas, se utiliza un microcontrolador PIC18F4550 como transeiver USB. Para intercambiar información con el FPGA, la comunicación se hace por medio de un puerto UART como el desarrollado previamente. En este ejemplo, una vez que el sistema está enumerado, espera a recibir un paquete de 3 bytes, en donde el primer byte define un modo de operación. En el primer modo, el modo “suma”, toma cada uno de los dos bytes restantes como un operando numérico, los suma y regresa un paquete con el resultado. En el modo “LED” toma el segundo byte para determinar si enciende o apaga alguno de los dos LEDs.

Listado 9. Código de ejemplo de la comunicación USB.

```
////////////////////////////////////////////////////////////////////
// Modulo: PicUSB.c
// Ejemplo de uso de la comunicación USB.
// Al conectar el PiC a la PC enciende el Led Rojo hasta que el
// dispositivo haya sido configurado por la PC, en ese momento
// enciende el Led Verde. Esperaremos hasta que se reciba un
// paquete de datos. Comprobamos el primer byte del paquete para
// comprobar si queremos entrar al modo Suma, donde se realizará
// una suma de dos operandos que se reciben en los dos bytes
// restantes del paquete; una vez realizada la suma se envía un
// paquete con el resultado. Si entramos al modo Led se comprueba
// el segundo byte del paquete para determinar si se apagan o si se
// enciende alguno de los leds.
//
// Realizado con el compilador CCS PCWH 4.018
////////////////////////////////////////////////////////////////////

#include <18F4550.h>
#fuses HSPLL,NOWDT,NOPROTECT,LVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN
#use delay(clock=48000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)

#define USB_HID_DEVICE     FALSE             //deshabilitamos el uso de las directivas HID
#define USB_EP1_TX_ENABLE  USB_ENABLE_BULK   //turn on EP1(EndPoint1) for IN bulk/interrupt
transfers
#define USB_EP1_RX_ENABLE  USB_ENABLE_BULK   //turn on EP1(EndPoint1) for OUT bulk/interrupt
transfers
#define USB_EP1_TX_SIZE    64                //size to allocate for the tx endpoint 1 buffer
#define USB_EP1_RX_SIZE    8                 //size to allocate for the rx endpoint 1 buffer

//#define USB_CON_SENSE_PIN PIN_B2           // optional conection sense pin

#include <pic18_usb.h>   //Microchip PIC18Fxx5x Hardware layer for CCS's PIC USB driver
#include <PicUSB.h>      //Configuración del USB y los descriptores para este dispositivo
#include <usb.c>         //handles usb setup tokens and get descriptor reports

#define modo      recibe[0]
#define param1    recibe[1]
#define param2    recibe[2]
#define resultado envia[0]

void main(void) {

   int8 recibe[3];                 //declaracion de variables
   int8 envia[1];

   output_low(PIN_B6);
   output_high(PIN_B7);

   printf("Inicializando\n");
   usb_init();                     //inicializamos el USB
   printf("Inicializacion terminada\n");

   usb_task();                     //habilita periferico usb e interrupciones
   printf("Esperando enumeracion\n");
   usb_wait_for_enumeration();     //esperamos hasta que el Pic sea configurado por el host

   output_low(PIN_B7);
   output_high(PIN_B6);            //encendemos led verde

   while (TRUE)
   {
      if(usb_enumerated())         //si el Pic está enumerado
      {
         printf("Enumerado:OK\n");
         if (usb_kbhit(1))         //si el endpoint de salida contiene datos del host
         {
```

```
            usb_get_packet(1, recibe, 3); //lee paquete de 3 bytes del EP1

            if (modo == 0)          // Modo_Suma
            {
               resultado = param1 + param2;  //hacemos la suma

               usb_put_packet(1, envia, 1, USB_DTS_TOGGLE); //envía paquete de 1 byte del EP1
            }

            if (modo == 1)          // Modo_Led
            {
               if (param1 == 0) {output_low(PIN_B6); output_low(PIN_B7);} //apagamos los leds
               if (param1 == 1) {output_high(PIN_B6); output_low(PIN_B7);} //encendemos led verde
               if (param1 == 2) {output_low(PIN_B6); output_high(PIN_B7);} //encendemos led rojo
            }
         }
      }
   }
}
```

Conclusiones

En el presente trabajo se muestra el conjunto de funciones desarrolladas para complementar la biblioteca de funciones 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 mejorar y ampliar ésta biblioteca de funciones.

Referencias Bibliográficas

Ball, S. R. (2002): *Embedded Microprocessor Systems. Real World Design*, Newnes, Maryland, 362 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.