

Temas Específicos de Electrónica Digital I

Comunicación USB 2.0 para aplicaciones científicas basadas en FPGA

Edwin Barragán

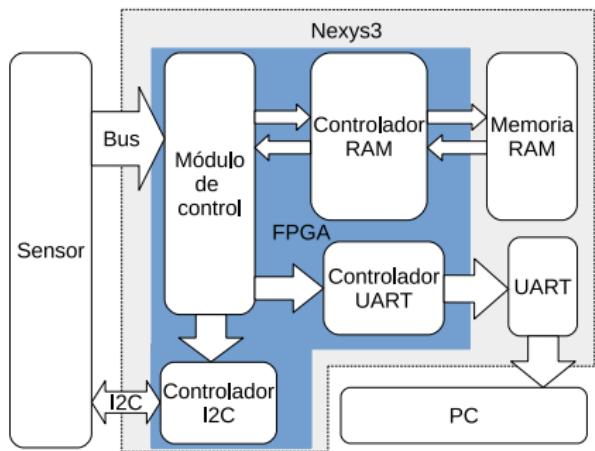
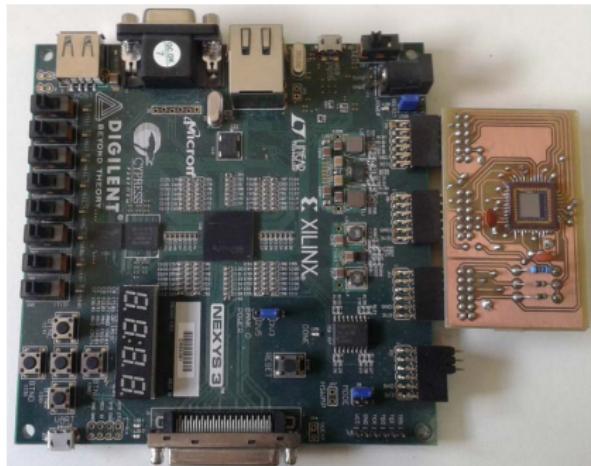
edwin.barragan@cab.cnea.gov.ar

Universidad Nacional de San Juan
Facultad de Ingeniería

3 de junio de 2019

Una comunicación USB para aplicaciones científicas basadas en FPGA

Preámbulo



Agenda

- 1 Introducción
- 2 Implementación
- 3 Evaluación y validación
- 4 Resultados y conclusiones

Agenda

1 Introducción

- Motivación
- Objetivos
- Bus Serial Universal

2 Implementación

- Arquitectura del sistema
- Configuración del puente
- Circuito sintetizado
- Circuito de interconexión

Agenda

3 Evaluación y validación

- Test benches de VHDL
- Elementos de VHDL utilizados para depuración
- Depuración de firmware del puente
- Biblioteca de PC
- Programas de prueba

4 Resultados y conclusiones

- Pruebas de robustez y resultados
- Conclusiones
- Trabajo futuro

Agenda

1 Introducción

- Motivación
- Objetivos
- Bus Serial Universal

La producción de información científica

- Los avances en las escalas de integración de circuitos permiten desarrollar sensores que recolectan mayor volumen de datos.
- Los nuevos sensores necesitan nuevos circuitos adicionales que les permitan adquirir datos y controlar su funcionamiento.
- El empleo de FPGA es muy útil para sintetizar circuitos digitales de alta velocidad.
- Los datos deben ser procesados para transformarse en información.
- Los datos se deben transmitir desde los sistemas generadores a los sistemas procesadores.

La necesidad de una comunicación entre un FPGA y una PC

- Las computadoras son herramientas muy útiles para procesar datos.
- Los FGPA pueden operar a altas velocidades y realizar procesos en paralelo.
- Es de utilidad una comunicación entre las PC y las aplicaciones que utilizan FPGA para la implementación de circuitos.
- USB es una opción robusta, con ancho de banda suficiente para transmitir imágenes e incorporada en cualquier PC moderna.

Agenda

1 Introducción

- Motivación
- **Objetivos**
- Bus Serial Universal

Objetivos

- Objetivo General
 - ▶ Realizar una comunicación entre un FPGA y una PC mediante USB 2.0
- Objetivos Particulares
 - ▶ Comprender el funcionamiento del kit de desarrollo CY3684 y el framework provisto por Cypress.
 - ▶ Configurar el chip CY7C68014A, incorporado en el kit de desarrollo anterior.
 - ▶ Sintetizar un circuito en VHDL que sea capaz de interactuar con las memorias FIFO de la interfaz.
 - ▶ Sintetizar circuitos de prueba para Test Bench.
 - ▶ Validar el funcionamiento.

Agenda

1 Introducción

- Motivación
- Objetivos
- Bus Serial Universal

USB - Bus Serial Universal

El Bus Serial Universal o USB es un sistema de comunicación pensado, en su concepción original, para conectar periféricos a una PC.

Los objetivos perseguidos por norma son:

- Conexión de teléfonos a la PC.
- Facilidad de uso.
- Proveer un puerto de expansión para periféricos.

USB - Bus Serial Universal

El Bus Serial Universal o USB es un sistema de comunicación pensado, en su concepción original, para conectar periféricos a una PC.

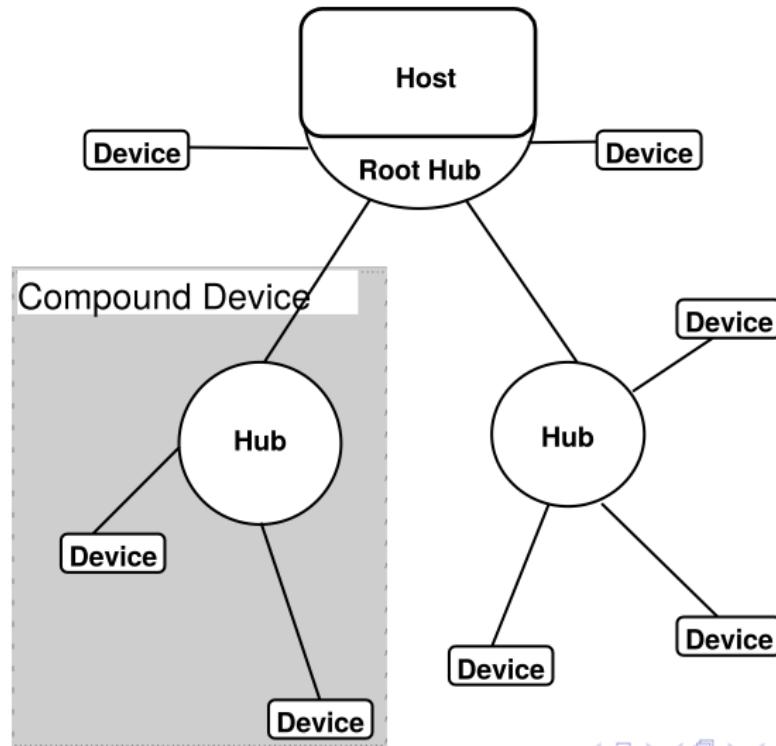
Los objetivos perseguidos por norma son:

- Conexión de teléfonos a la PC.
- Facilidad de uso.
- Proveer un puerto de expansión para periféricos.
- Mayor rendimiento
- Mayor ancho de banda

La respuesta a esta demanda fue agregar dos nuevas velocidades de operación: 12 y 480 Mbit/s.

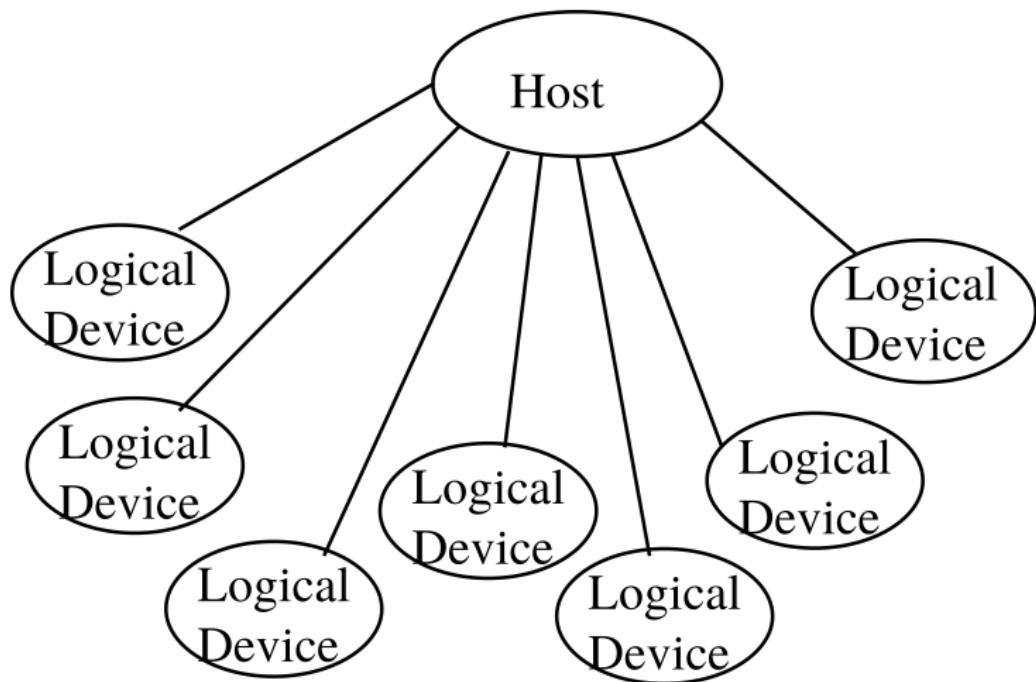
USB - Topología

- Física

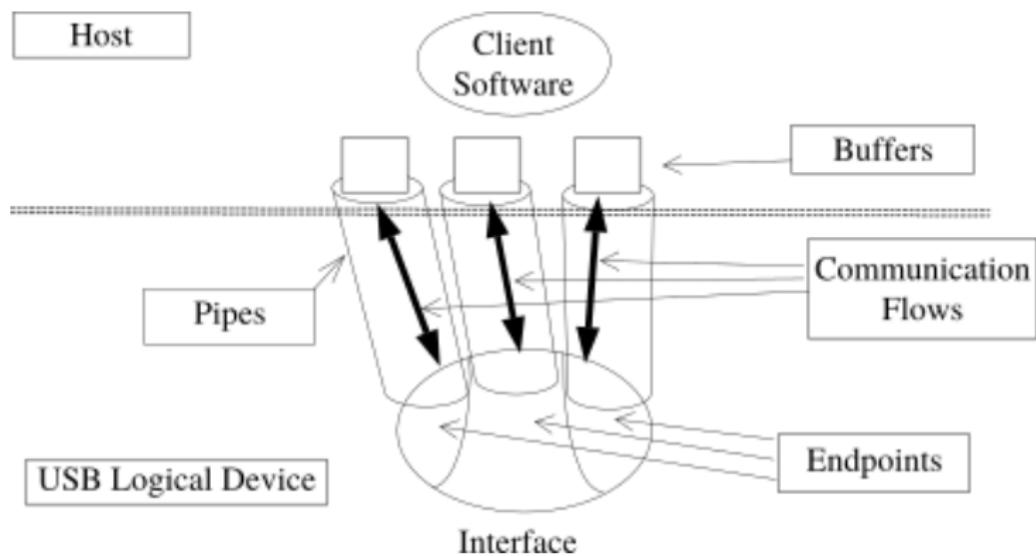


USB - Topología

- Lógica



USB - Flujo de datos

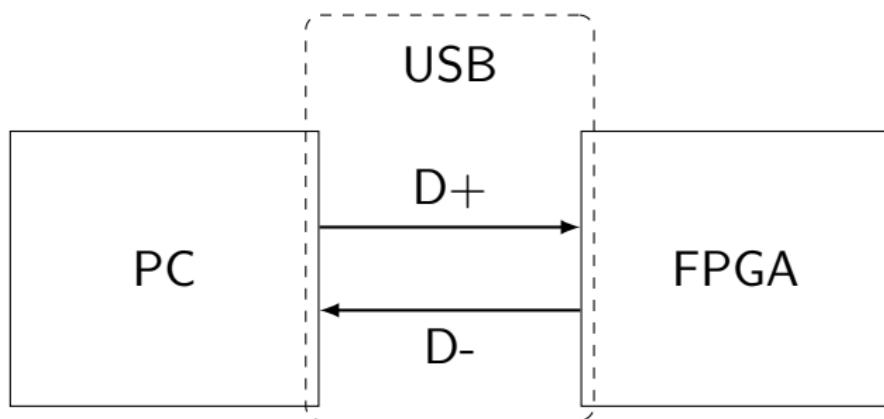


Agenda

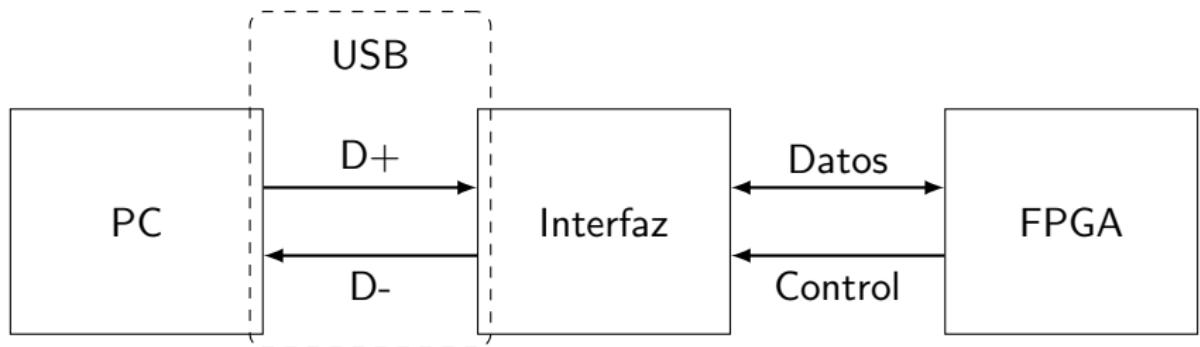
2 Implementación

- Arquitectura del sistema
- Configuración del puente
- Circuito sintetizado
- Circuito de interconexión

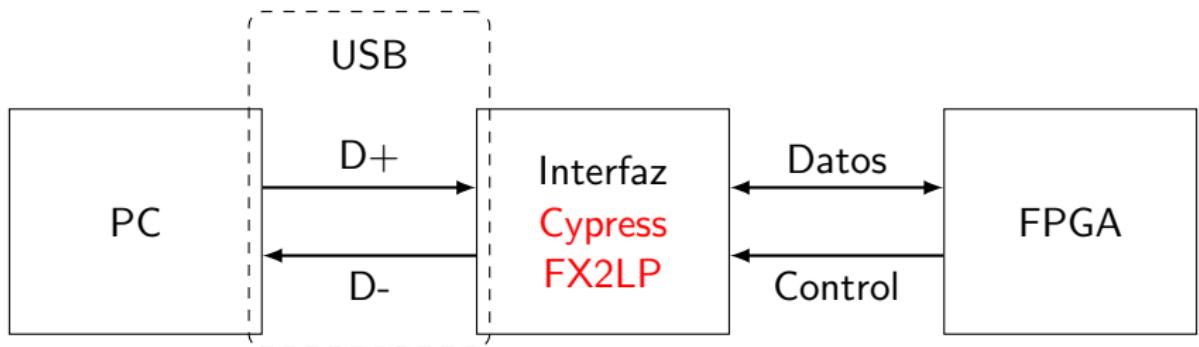
Arquitectura del sistema realizado



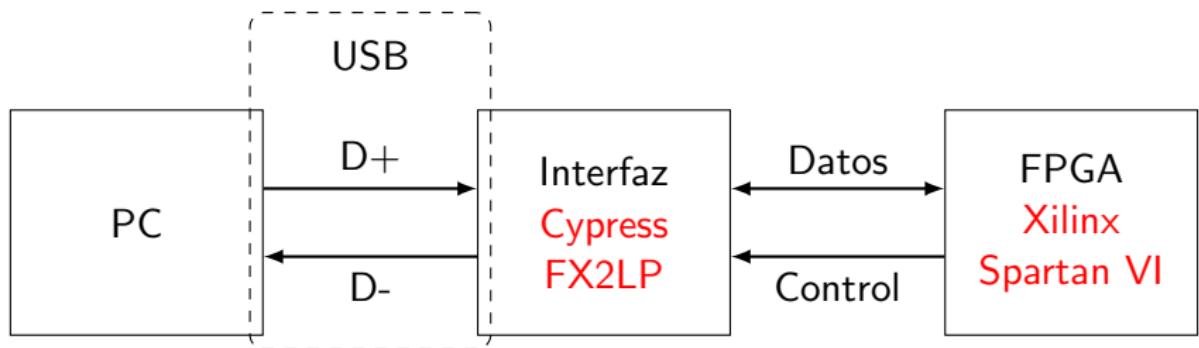
Arquitectura del sistema realizado



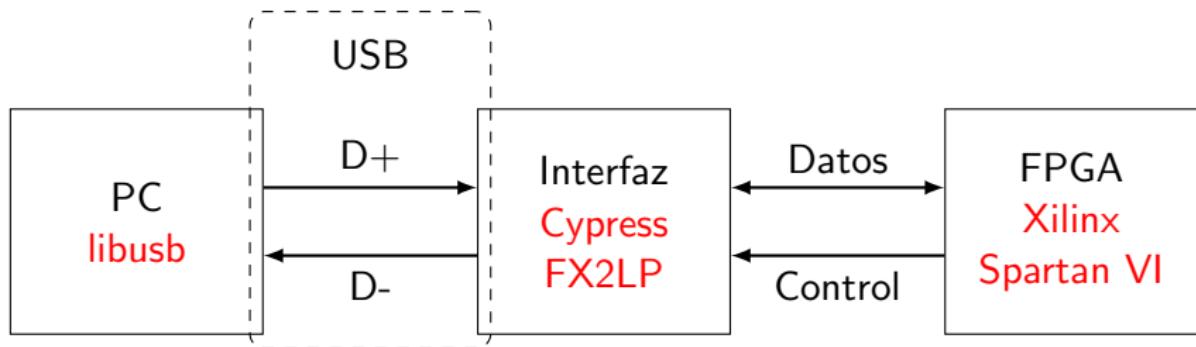
Arquitectura del sistema realizado



Arquitectura del sistema realizado



Arquitectura del sistema realizado

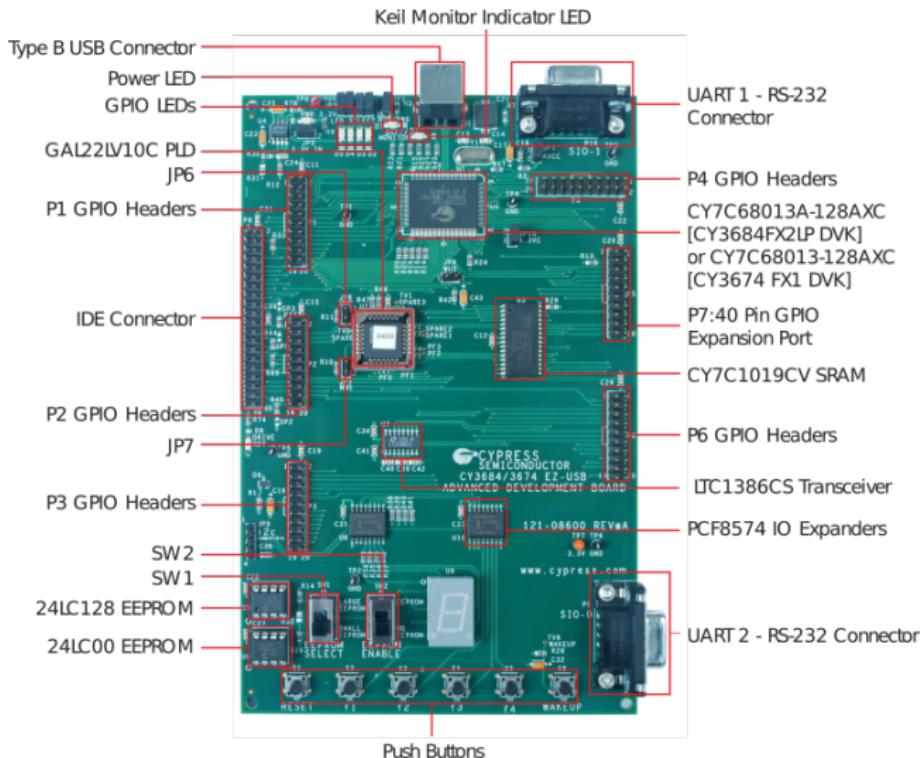


Agenda

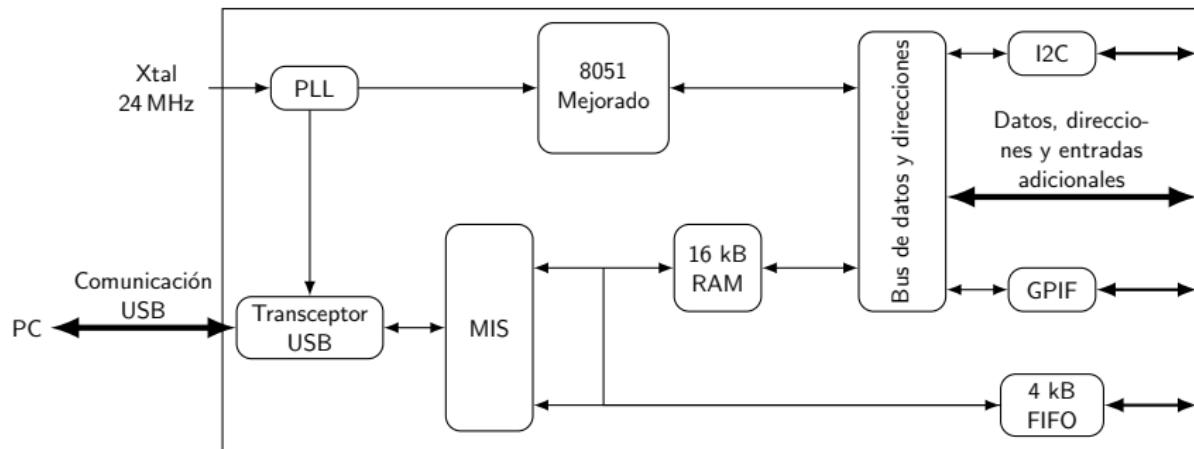
2 Implementación

- Arquitectura del sistema
- **Configuración del puente**
- Circuito sintetizado
- Circuito de interconexión

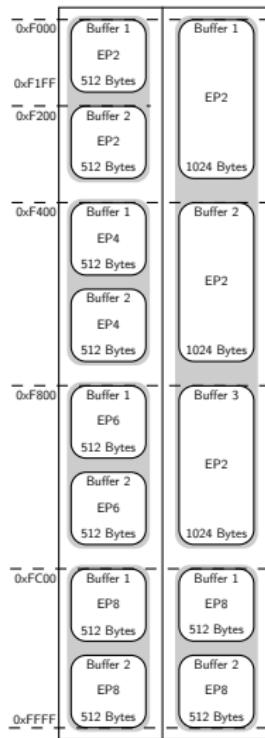
Placa de desarrollo utilizada para la interfaz



El circuito integrado FX2LP



Configuración del dispositivo USB

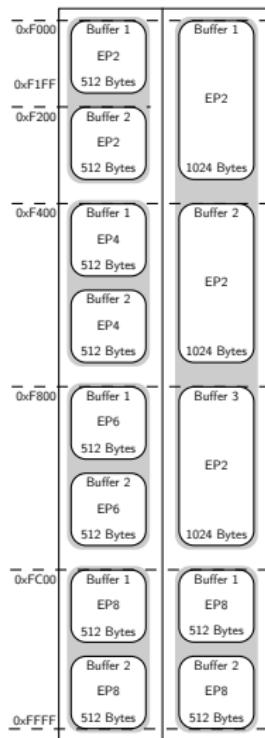


La configuración de las tuberías de comunicación se definieron con la finalidad de obtener el mayor ancho de banda posible a la entrada.

- Entrada:

- Extremo EP2
- Transferencias Isocrónicas
- 1024 Bytes máximo por transferencia
- 3 Buffers

Configuración del dispositivo USB



La configuración de las tuberías de comunicación se definieron con la finalidad de obtener el mayor ancho de banda posible a la entrada.

- Salida:

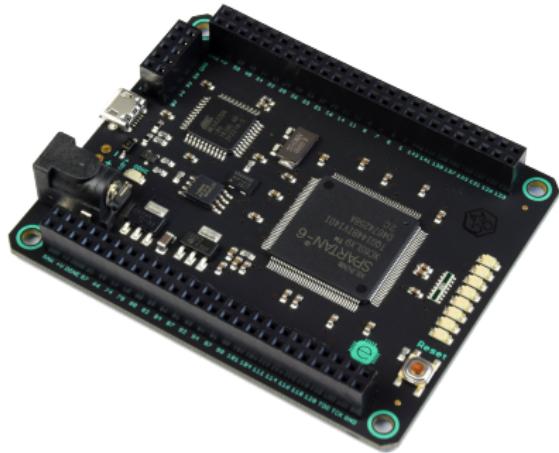
- ▶ Extremo EP8
- ▶ Transferencia por bultos
- ▶ 512 Bytes por transferencia
- ▶ 2 Buffers

Agenda

2 Implementación

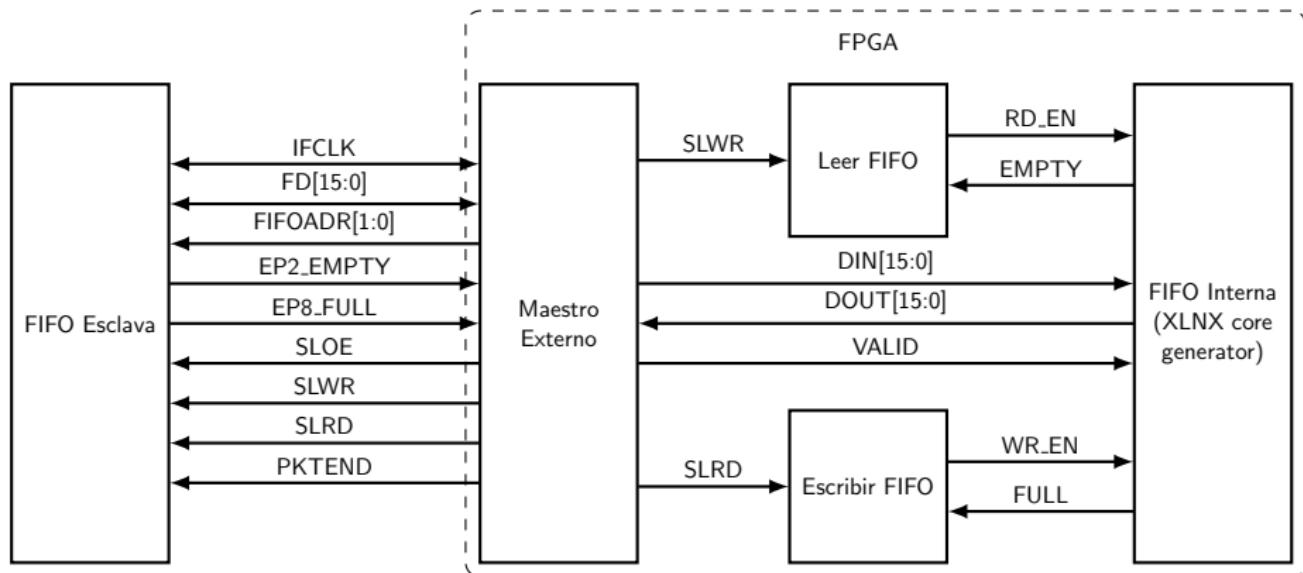
- Arquitectura del sistema
- Configuración del puente
- **Circuito sintetizado**
- Circuito de interconexión

La placa de desarrollo MOJO v3

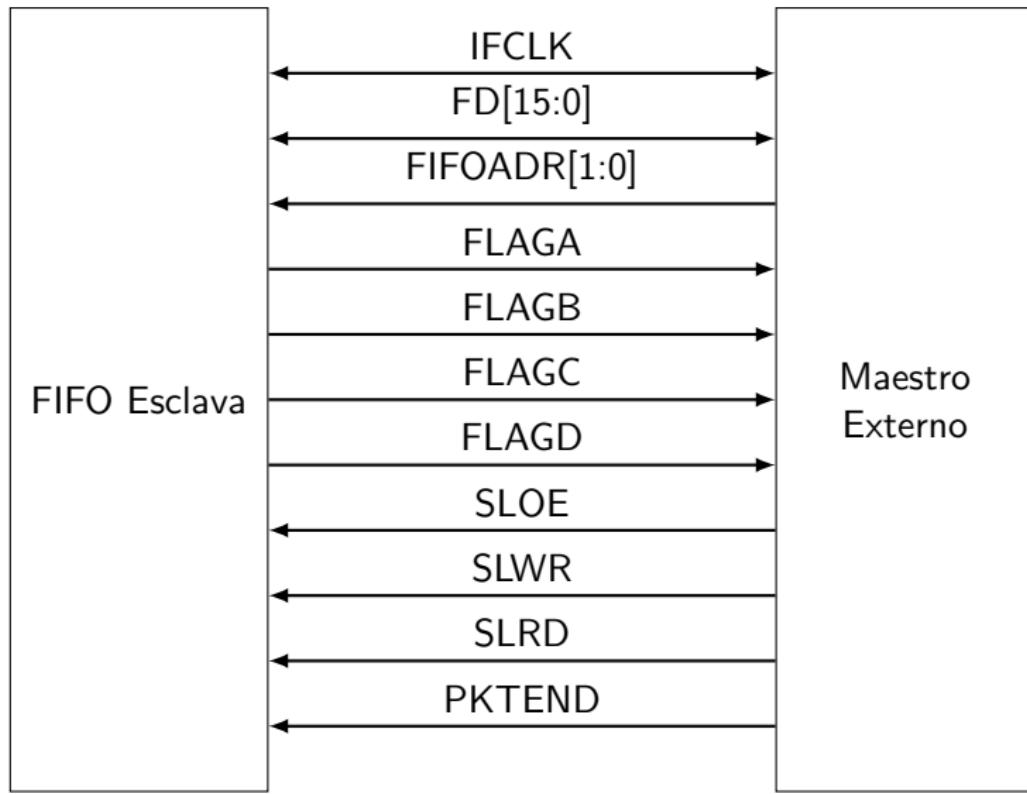


- FPGA Spartan 6 XC6SLX9 de Xilinx
- 84 pines IO digitales
- 8 entradas analógicas
- 8 LEDs de propósito general
- 1 pulsador de propósito general
- Regulador de voltaje de entrada de 4.8V - 12V
- ATmega32U4 para configurar la FPGA y leer los pines analógicos
- Bootloader compatible con Arduino
- Memoria flash para almacenar la configuración de la FPGA
(programación persistente)

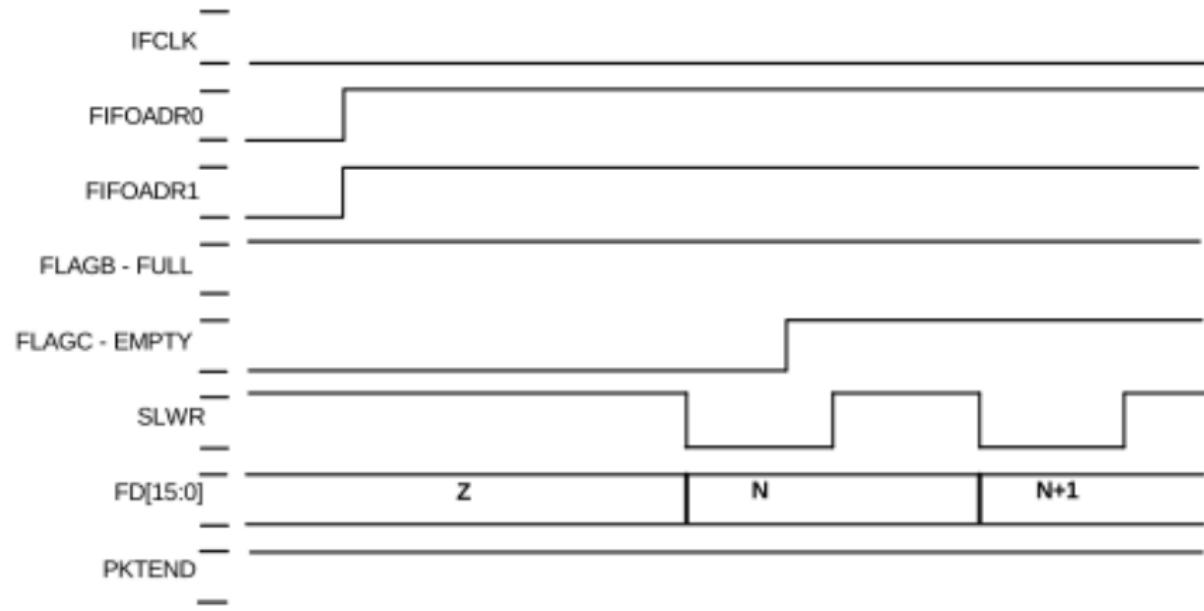
Estructura interna FPGA



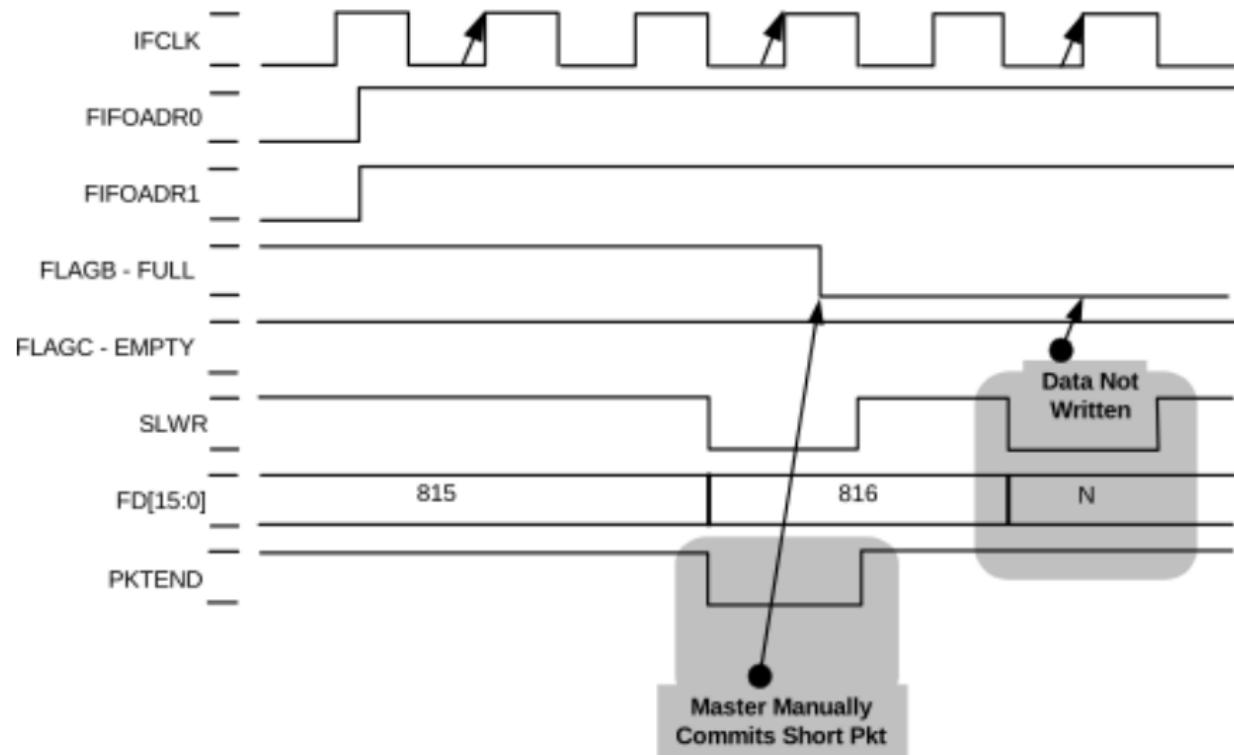
Interfaz - FPGA



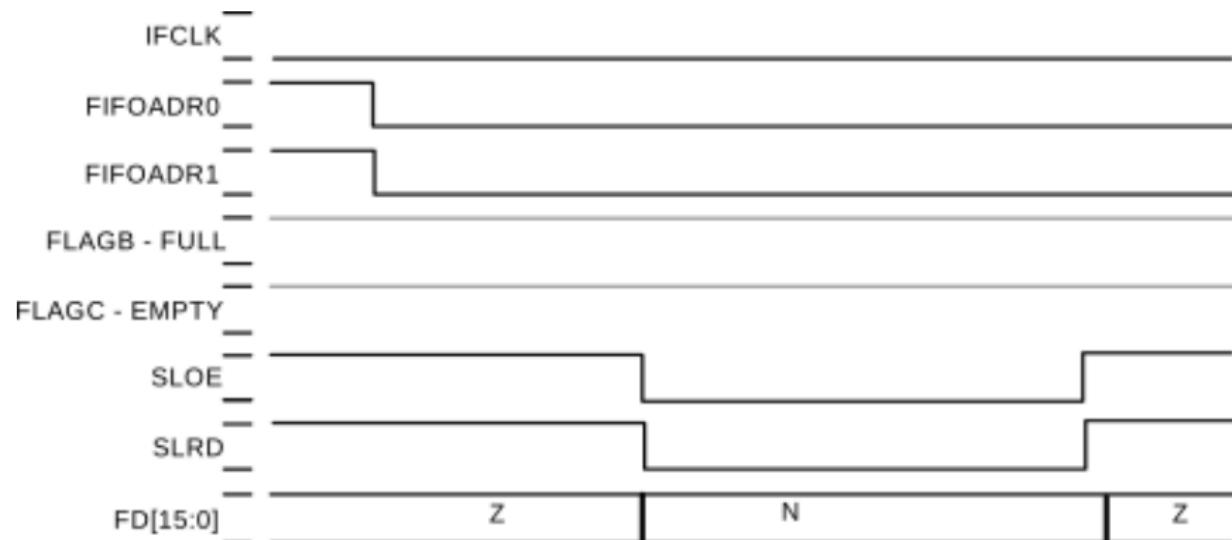
Operaciones en la FIFO - Lectura asíncrona



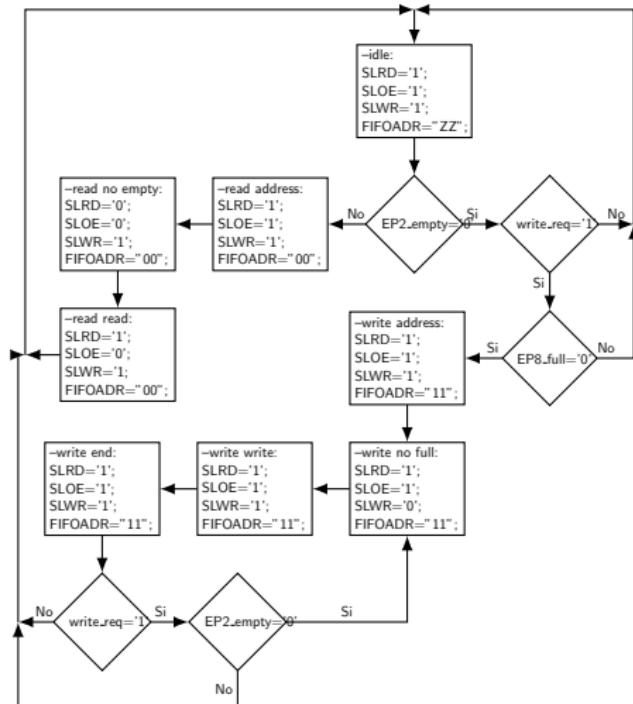
Operaciones en la FIFO - Lectura asíncrona



Operaciones en la FIFO - Escritura asíncrona



Máquina de estados algorítmica de la interfaz



Implementación de la máquina de estados de la interfaz

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

entity fx2lp_interface is
generic(
    in_ep_addr: std_logic_vector(1 downto 0) := "00";
    out_ep_addr: std_logic_vector(1 downto 0) := "11";
    port_width: integer := 16
);
```

Implementación de la máquina de estados de la interfaz

```
port(
    clk : in std_logic;                                — entrada de reloj
    reset : in std_logic;
    — desde y hacia la interfaz
    fdata : inout std_logic_vector(port_width-1 downto 0);— datos FIFO
    faddr : out std_logic_vector(1 downto 0);— canal FIFO
    slrd : out std_logic;                               — señal de lectura
    slwr : out std_logic;                               — señal de escritura
    flaga : in std_logic;                             — EP2_full—>write_full_flag
    flagb : in std_logic;                             — EP8_empty—>read_empty_flag
    flagc : in std_logic;                             — EP8_full—>read_full_flag
    flagd : in std_logic;                             — EP2_empty—>write_empty_flag
    sloe : out std_logic;                            — habilitacion de salida
    pktend: out std_logic;
    — desde y hacia el sistema
    send_req : in std_logic;                         — pedido de envio de datos
    rx_data : out std_logic_vector(port_width-1 downto 0);
    data_to_tx : in std_logic_vector(port_width-1 downto 0)
);
end fx2lp_interface;
```

Implementación de la máquina de estados de la interfaz

```
architecture Behavioral of fx2lp_interface is

signal slwr_int    : std_logic := '1';
signal slrd_int    : std_logic := '1';
signal sloe_int    : std_logic := '1';
signal pktend_int  : std_logic := '1';
signal faddr_int   : std_logic_vector(1 downto 0) := "ZZ";
signal fdata_out    : std_logic_vector(port_width-1 downto 0);
signal fdata_in    : std_logic_vector(port_width-1 downto 0);
signal rd_eflag    : std_logic := '1';
signal wr_fflag    : std_logic := '1';

signal sys_clk      : std_logic;

-- señales de temporización
signal count3 : natural range 0 to 4 := 0;
signal count2 : natural range 0 to 3 := 0;
signal trig3 , trig2 : std_logic := '0';
```

Implementación de la máquina de estados de la interfaz

```
-- maquina de estados de la interfaz
type if_fsm is
(
  idle ,
  read_addr , read_no_empty , read_read ,
  write_addr , write_no_full , write_write , write_end
);
signal curr_state , next_state : if_fsm := idle;
```

Implementación de la máquina de estados de la interfaz

```
begin
    -- conexión de señales internas hacia los puertos
    sys_clk <= clk;

    slwr <= slwr_int;
    slrd <= slrd_int;
    sloe <= sloe_int;
    faddr <= faddr_int;
    pktend <= pktend_int;

    wr_fflag <= not flaga;
    rd_eflag <= not flagb;

    rx_data <= fdata_in;
    fdata_out <= data_to_tx;
```

Implementación de la máquina de estados de la interfaz

```
-- señalización
with curr_state select
    faddr_int <= out_ep_addr when read_addr | read_no_empty
        | read_read,
        in_ep_addr when write_addr | write_no_full | write_write
        | write_end,
        (others => 'Z') when others;
    slwr_int <= '0' when next_state = write_write else '1';
    slrd_int <= '0' when curr_state = read_no_empty else '1';
    pktend_int <= ((not rd_eflag) or send_req);
with curr_state select
    sloe_int <= '0' when read_addr | read_read | read_no_empty ,
        '1' when others;
with curr_state select
    fdata <= fdata_out when write_no_full | write_write | write_end
        | write_addr,
        (others => 'Z') when others;
with curr_state select
    fdata_in <= fdata when read_no_empty | read_read | read_addr ,
        fdata_in when others;
```

Implementación de la máquina de estados de la interfaz

```
--implementacion de la maquina de estados
interfaz_fsm: process(curr_state , wr_fflag , rd_eflag , send_req)
begin
    case curr_state is
        when idle =>
            if rd_eflag = '0' then
                next_state <= read_addr;
            elsif send_req = '1' then
                if wr_fflag = '0' then
                    next_state <= write_addr;
                else
                    next_state <= idle;
                end if;
            else
                next_state <= idle;
            end if;
        when read_addr =>
            next_state <= read_no_empty;
        when read_no_empty =>
            next_state <= read_read;
        when read_read =>
            if rd_eflag = '0' then
                next_state <= read_addr;
            else
                next_state <= idle;
            end if;
        when write_addr =>
            next_state <= write_no_full;
        when write_no_full =>
            next_state <= write_write;
        when write_write =>
            next_state <= write_end;
```

Implementación de la máquina de estados de la interfaz

```
--implementacion de la maquina de estados
interfaz_fsm: process(curr_state , wr_fflag , rd_eflag , send_req)
begin
    case curr_state is
        when write_end =>
            if send_req = '1' then
                if rd_eflag = '1' then
                    next_state <= write_no_full;
                else
                    next_state <= idle;
                end if;
            else
                next_state <= idle;
            end if;
        when others =>
            next_state <= idle;

    end case;
end process interfaz_fsm;
```

Implementación de la máquina de estados de la interfaz

```
-- temporizaciones
counter3: process(sys_clk, reset, trig3)
begin
  if reset = '0' then
    count3 <= 0;
  elsif rising_edge(sys_clk) then
    if count3 > 0 then
      count3 <= count3 - 1;
    elsif trig3 = '1' then
      count3 <= 4;
    end if;
  end if;
end process counter3;

trig3 <= '1' when (next_state = write_write) else '0';
```

Implementación de la máquina de estados de la interfaz

```
counter2: process(sys_clk, reset, trig2)
begin
    if reset = '0' then
        count2 <= 0;
    elsif rising_edge(sys_clk) then
        if count2 > 0 then
            count2 <= count2 - 1;
        elsif trig2 = '1' then
            count2 <= 3;
        end if;
    end if;
end process counter2;

with next_state select
    trig2 <= '1' when read_no_empty | read_read | write_no_full | write_full;
    '0' when others;
```

Implementación de la máquina de estados de la interfaz

```
global_fsm_clk: process (sys_clk , reset)
begin
  if reset = '0' then
    curr_state <= idle;
  elsif rising_edge(sys_clk) then
    if count2 = 0 and count3 = 0 then
      curr_state <= next_state;
    end if;
  end if;
end process global_fsm_clk;

end Behavioral;
```

Instanciación de la interfaz en el top

```
entity fx2lp_interface_top is
generic(
    in_ep_addr: std_logic_vector(1 downto 0) := "00";
    out_ep_addr: std_logic_vector(1 downto 0) := "11";
    port_width: integer := 16
);
port(
    — señales que van y vienen desde y hacia el FX2LP
    fdata : inout std_logic_vector(port_width-1 downto 0); — datos FIFO
    faddr : out std_logic_vector(1 downto 0); — canal FIFO
    slrd : out std_logic; — señal de lectura
    slwr : out std_logic; — señal de escritura
    — EP2 streaming in (to pc)
    flaga : in std_logic; — EP2_full—>write_full_flag
    flagb : in std_logic; — EP8_empty—>read_empty_flag
    — EP8 bulk out (from pc)
    flagc : in std_logic; — EP8_full—>read_full_flag
    flagd : in std_logic; — EP2_empty—>write_empty_flag
    sloe : out std_logic; — señal de habilitacion de salida
    pktend : out std_logic;
    — reloj
    clk_out : out std_logic; — salida de reloj
    — señales desde hacia mojo
    clk_in : in std_logic; — entrada de reloj
    button : in std_logic; — activo en bajo
    led : out std_logic_vector(7 downto 0)
);
end fx2lp_interface_top;
```

Instanciación de la interfaz en el top

```
architecture fx2lp_interface_arq of fx2lp_interface_top is
-- declaracion de componentes
COMPONENT fx2lp_interface
GENERIC(
    in_ep_addr: std_logic_vector(1 downto 0) := "00";
    out_ep_addr: std_logic_vector(1 downto 0) := "11";
    port_width: integer := 16
);
PORT(
    clk : IN std_logic;
    reset : IN std_logic;
    flaga : IN std_logic;
    flagb : IN std_logic;
    flagc : IN std_logic;
    flagd : IN std_logic;
    send_req : IN std_logic;
    data_to_tx : IN std_logic_vector(15 downto 0);
    fdata : INOUT std_logic_vector(15 downto 0);
    faddr : OUT std_logic_vector(1 downto 0);
    slrd : OUT std_logic;
    slwr : OUT std_logic;
    sloe : OUT std_logic;
    pktend : OUT std_logic;
    rx_data : OUT std_logic_vector(15 downto 0)
);
END COMPONENT;
```

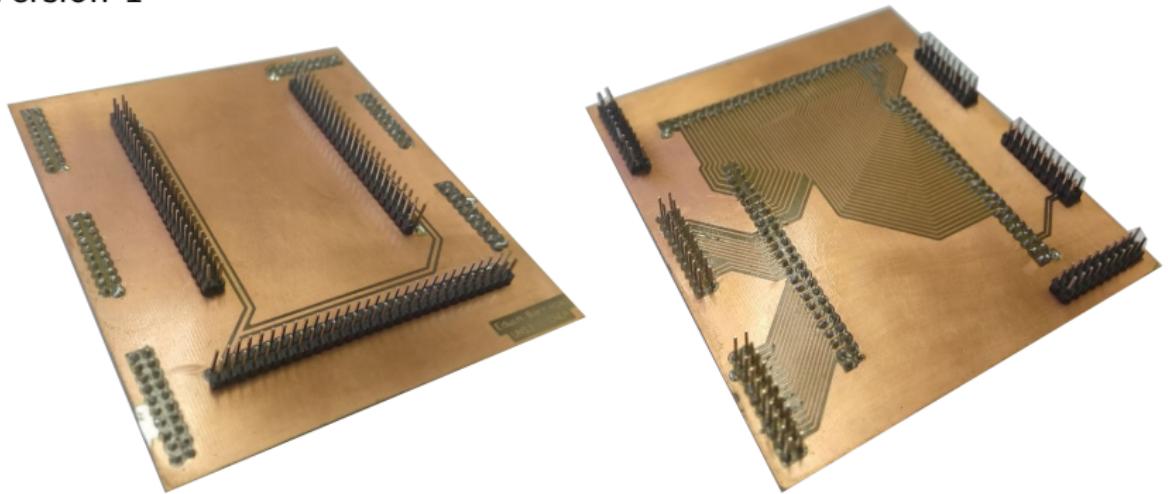
Agenda

2 Implementación

- Arquitectura del sistema
- Configuración del puente
- Circuito sintetizado
- Circuito de interconexión

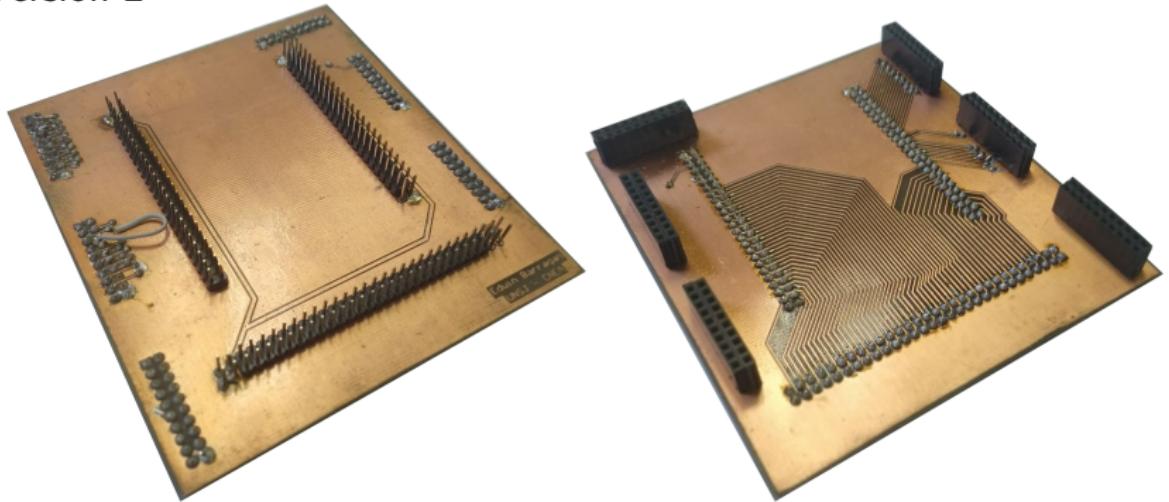
Circuito de interconexión

- Versión 1



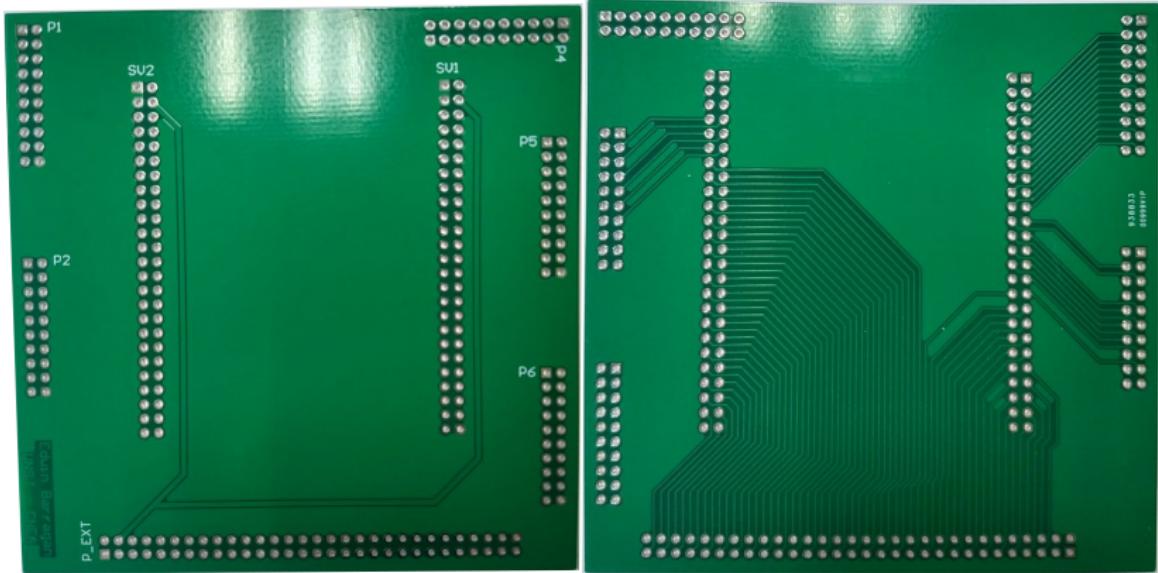
Círculo de interconexión

- Versión 2



Círculo de interconexión

- Versión 3



Sistema completo



Agenda

3 Evaluación y validación

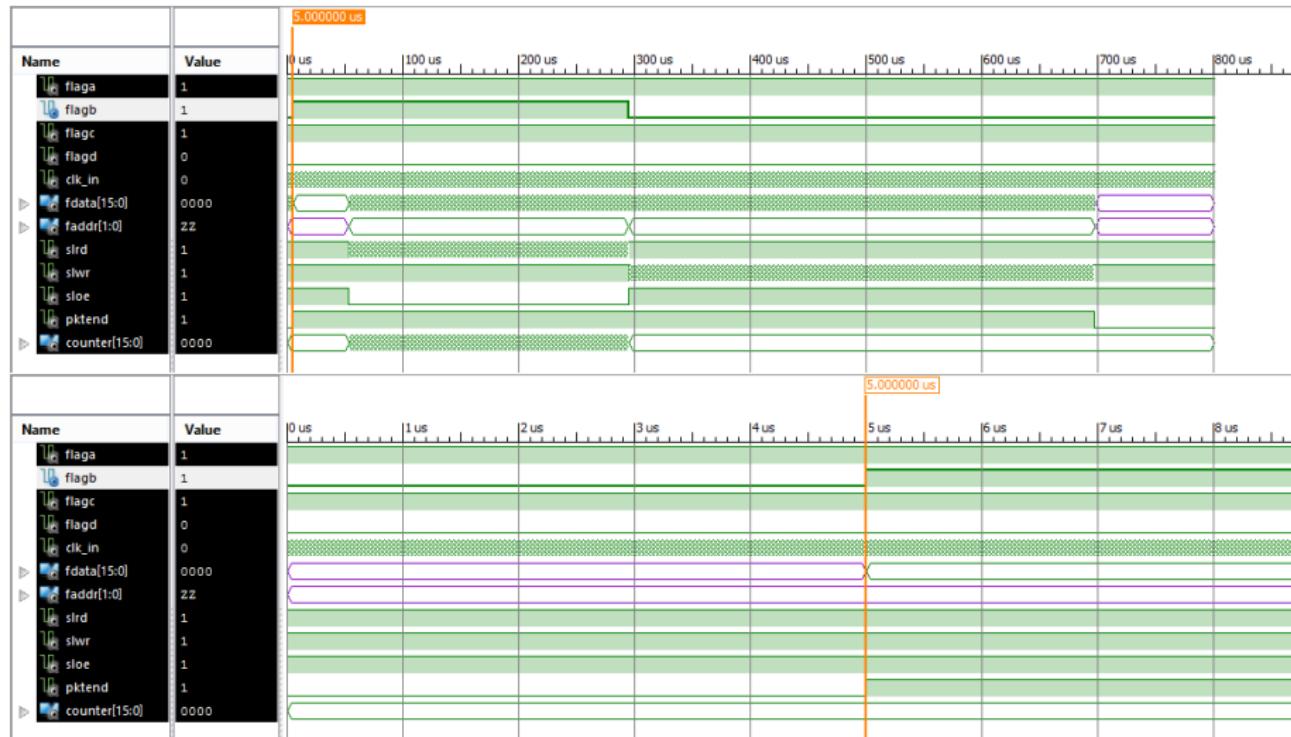
● Test benches de VHDL

- Elementos de VHDL utilizados para depuración
- Depuración de firmware del puente
- Biblioteca de PC
- Programas de prueba

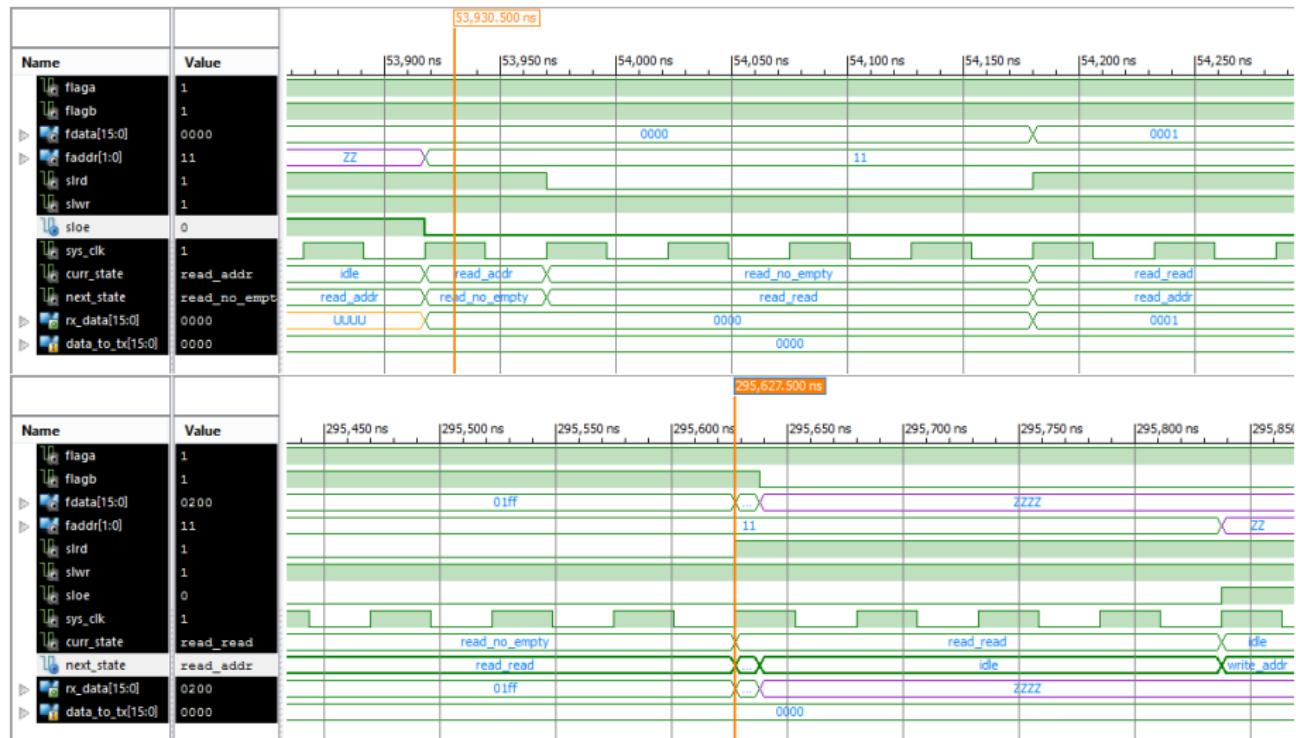
Test bench

El sistema fue simulado con el objetivo de corroborar el correcto funcionamiento del sistema, cómo así también detectar fallas en la descripción realizada y obtener una forma de depuración de lo realizado.

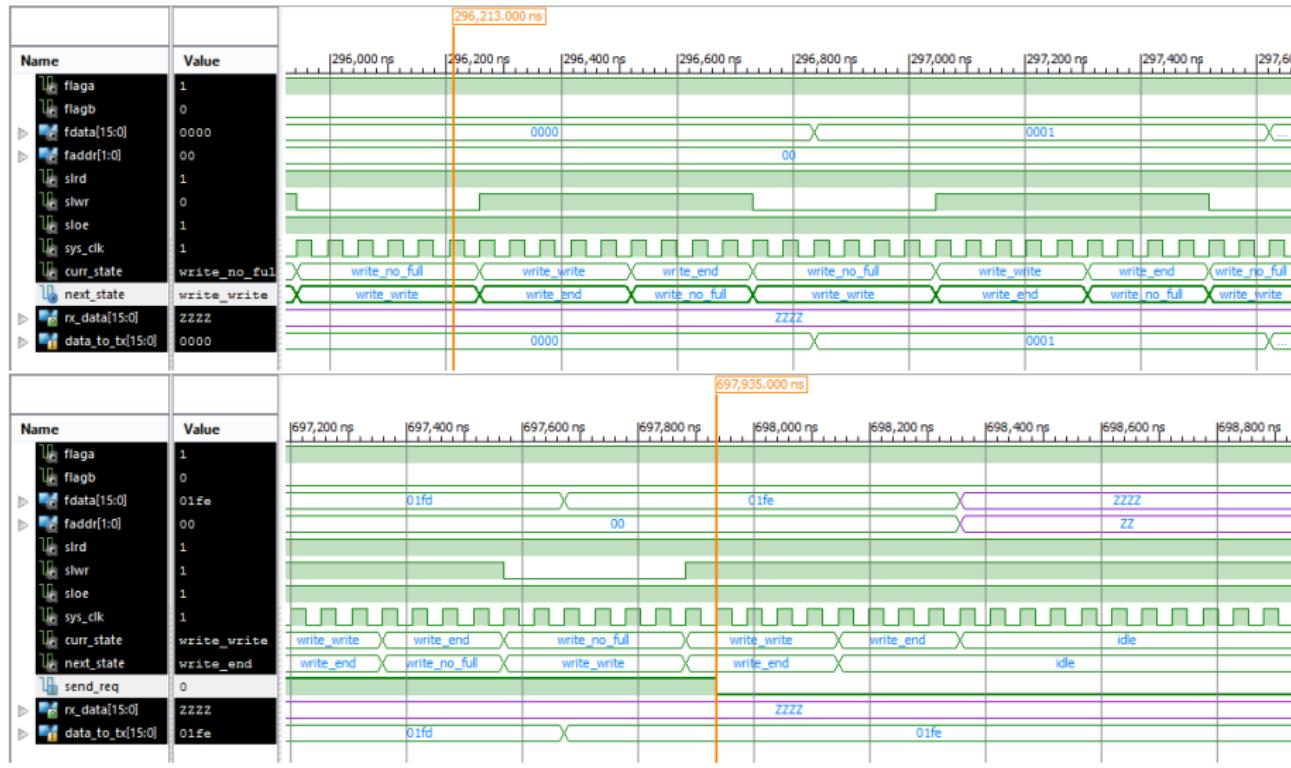
Test bench - Top



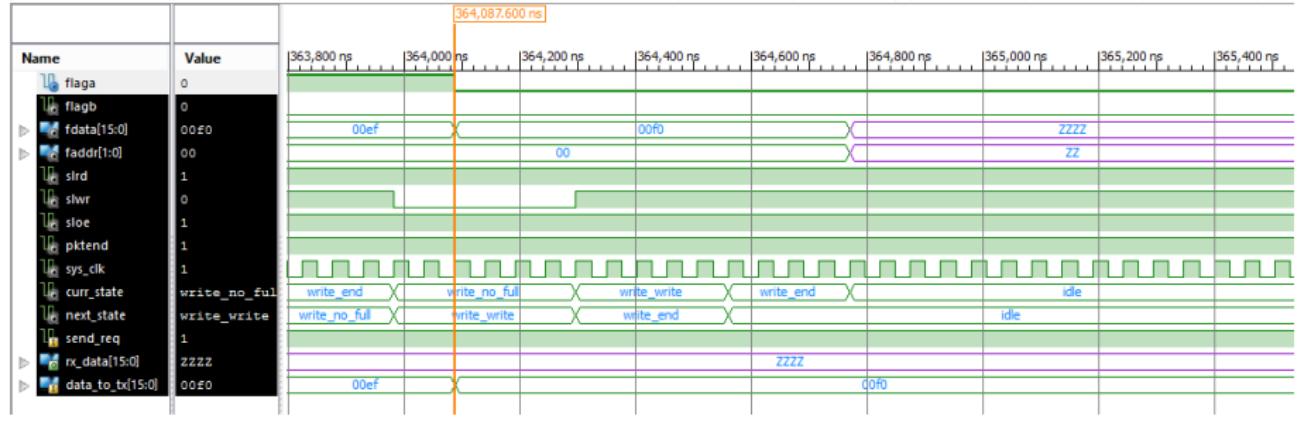
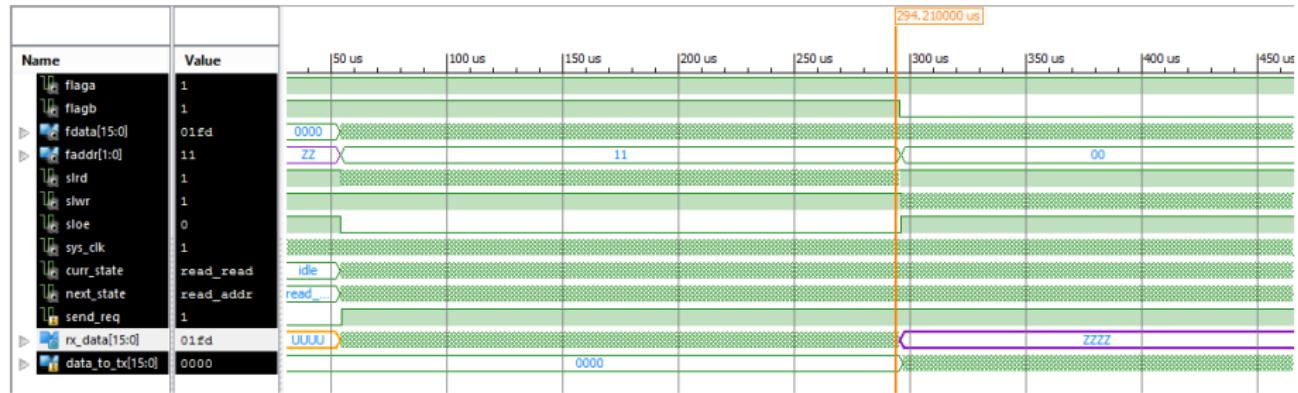
Test bench - Lectura de la interfaz



Test bench - Escritura en la interfaz



Test bench - Escritura en la interfaz



Agenda

3 Evaluación y validación

- Test benches de VHDL
- Elementos de VHDL utilizados para depuración
- Depuración de firmware del puente
- Biblioteca de PC
- Programas de prueba

Depuración de la interfaz en VHDL

- Se utilizó un registro para elaborar un eco de a dos bytes. Permitió corroborar el correcto funcionamiento de la lectura y escritura de las memorias FIFO de la interfaz de Cypress.
- Se utilizó una memoria FIFO generada con el software “Core Generator” de Xilinx para lograr un correcto funcionamiento de las señales de la interfaz escrita en VHDL.
- Además, se utilizó un PLL como generador de reloj, ya que si bien se pensaba hacer sincrónico, por errores de diseño esto no fue posible.

Agenda

3 Evaluación y validación

- Test benches de VHDL
- Elementos de VHDL utilizados para depuración
- **Depuración de firmware del puente**
- Biblioteca de PC
- Programas de prueba

Debug Cypress

- Se utilizó el framework provisto por Cypress para ser compilado con SDCC. No dió los frutos esperados.
- Se utilizó el framework provisto por Cypress para ser compilado con C51 de Keil. No funcionó en primera instancia.
- Se programó el puerto UART del 8051 para rastrear el error.
- Se logró que el framework trabaje adecuadamente solo con el puerto UART funcionando.
- Se utilizaron los LEDs indicadores para corroborar el llenado y vaciamiento de los extremos.
- Se aprovecharon los botones indicadores para corroborar el correcto funcionamiento de los extremos.

Agenda

3 Evaluación y validación

- Test benches de VHDL
- Elementos de VHDL utilizados para depuración
- Depuración de firmware del puente
- **Biblioteca de PC**
- Programas de prueba

libusb-1.0

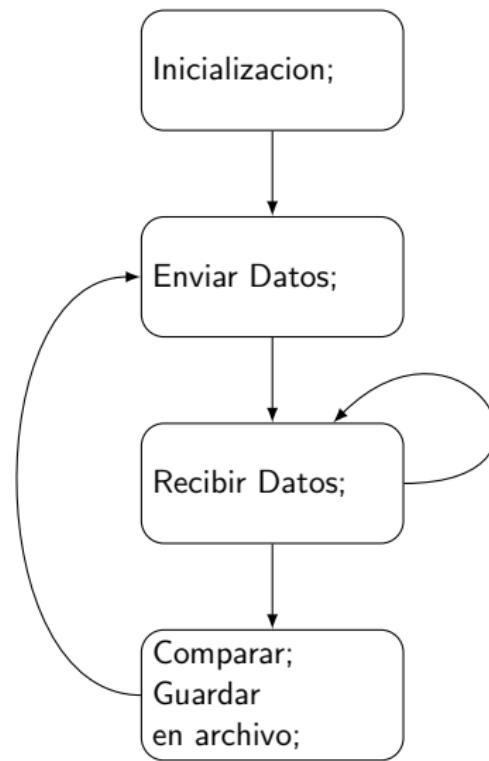
- Se utilizó la biblioteca libusb-1.0 para la elaboración de un programa de computadoras, escrito en C, que permita leer y escribir paquetes de datos desde la PC.
- Esta biblioteca es de código abierto y es soportada por todos los sistemas operativos actualmente en uso. Esto brinda la reutilización del código generado.

Agenda

3 Evaluación y validación

- Test benches de VHDL
- Elementos de VHDL utilizados para depuración
- Depuración de firmware del puente
- Biblioteca de PC
- Programas de prueba

Esquemas de prueba



Agenda

4 Resultados y conclusiones

- Pruebas de robustez y resultados
- Conclusiones
- Trabajo futuro

Resultados de la prueba de robustez de la comunicación

- Se dejó correr el sistema por más de 24 horas logrando la correcta transmisión y recepción de 388.191.289 paquetes de 128 bytes cada uno.
- Lo anterior equivale a una comunicación que cumple las especificaciones de robustez de la norma USB.
- La tasa de transferencia de datos es de 12 Mbps, lo que equivale a una comunicación USB full-speed.
- No obstante, esta prueba no mide en forma correcta el ancho de banda de entrada que se puede llegar a lograr

Agenda

4 Resultados y conclusiones

- Pruebas de robustez y resultados
- Conclusiones
- Trabajo futuro

Conclusiones

- Se elaboró un sistema de comunicación USB que permite conectar una FPGA con una PC, mediante una interfaz comercial.
- El sistema elaborado permite leer y escribir datos en forma robusta, es decir, sin perder datos.
- El sistema elaborado alcanzó una tasa de transferencia de 12 Mbps.
- Se consolidaron las técnicas y herramientas aprendidas que permiten desarrollar y depurar sistemas en VHDL, C para microcontroladores y C para computadoras.
- Se desarrollaron competencias sobre las formas de prueba y validación de los sistemas implementados y sus componentes.

Agenda

4 Resultados y conclusiones

- Pruebas de robustez y resultados
- Conclusiones
- Trabajo futuro

Lo que falta...

- Realizar una prueba de máxima transferencia de datos.
- Elaborar la documentación necesaria para que la interfaz de comunicación pueda ser utilizada por otros desarrolladores

Consultas

Consultas y sugerencias.

Muchas gracias.

Material Adicional

— Company:
— Engineer:
—
— Create Date: 17:23:43 05/29/2019
— Design Name:
— Module Name: fx2lp_interface — Behavioral
— Project Name:
— Target Devices:
— Tool versions:
— Description:
—
— Dependencies:
—
— Revision:
— Revision 0.01 — File Created
— Additional Comments:
—

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

entity fx2lp_interface is
generic(
    in_ep_addr: std_logic_vector(1 downto 0) := "00";
    out_ep_addr: std_logic_vector(1 downto 0) := "11";
    port_width: integer := 16
);
port(
    clk : in std_logic;                                — entrada de reloj
    reset : in std_logic;                               — desde y hacia la interfaz
    fdata : inout std_logic_vector(port_width-1 downto 0); — datos FIFO
    faddr : out std_logic_vector(1 downto 0);           — canal FIFO
    slrd : out std_logic;                             — señal de lectura
```

```

slwr : out std_logic;           — señal de escritura
flaga : in std_logic;          — EP2_full—>write_full_flag
flagb : in std_logic;          — EP8_empty—>read_empty_flag
flagc : in std_logic;          — EP8_full—>read_full_flag
flagd : in std_logic;          — EP2_empty—>write_empty_flag
sloe : out std_logic;          — habilitacion de salida
pktend: out std_logic;
— desde y hacia el sistema
send_req :in std_logic;        — pedido de envio de datos
rx_data :out std_logic_vector(port_width-1 downto 0);
data_to_tx :in std_logic_vector(port_width-1 downto 0)
);
end fx2lp_interface;

```

architecture Behavioral of fx2lp_interface is

```

signal slwr_int : std_logic := '1';
signal slrd_int : std_logic := '1';
signal sloe_int : std_logic := '1';
signal pktend_int : std_logic := '1';
signal faddr_int : std_logic_vector(1 downto 0) := "ZZ";
signal fdata_out : std_logic_vector(port_width-1 downto 0);
signal fdata_in : std_logic_vector(port_width-1 downto 0);
signal rd_eflag : std_logic := '1';
signal wr_fflag : std_logic := '1';

signal sys_clk : std_logic;

```

— señales de temporizacion

```

signal count3 : natural range 0 to 4 := 0;
signal count2 : natural range 0 to 3 := 0;
signal trig3, trig2 : std_logic := '0';

```

— maquina de estados de la interfaz

type if_fsm is

```

(
    idle,
    read_addr, read_no_empty, read_read,
    write_addr, write_no_full, write_write, write_end
);
signal curr_state, next_state : if_fsm := idle;

begin
    — conexion de señales internas hacia los puertos
    sys_clk <= clk;

    slwr <= slwr_int;
    slrd <= slrd_int;
    sloe <= sloe_int;
    faddr <= faddr_int;
    pktend <= pktend_int;

    wr_fflag <= not flaga;
    rd_eflag <= not flagb;

    rx_data <= fdata_in;
    fdata_out <= data_to_tx;

    — señalización
    with curr_state select
        faddr_int <= out_ep_addr when read_addr | read_no_empty
            | read_read,
            in_ep_addr when write_addr | write_no_full | write_write
            | write_end,
            (others => 'Z') when others;
    slwr_int <= '0' when next_state = write_write else '1';
    slrd_int <= '0' when curr_state = read_no_empty else '1';
    pktend_int <= ((not rd_eflag) or send_req);
    with curr_state select
        sloe_int <= '0' when read_addr | read_read | read_no_empty,
        ...

```

```
'1' when others;
with curr_state select
  fdata <= fdata_out when write_no_full | write_write | write_end
    | write_addr,
  ('others' => 'Z') when others;
with curr_state select
  fdata_in <= fdata      when read_no_empty | read_read | read_addr,
  fdata_in  when others;
```

—implementacion de la maquina de estados

```
interfaz_fsm: process(curr_state, wr_fflag, rd_eflag, send_req)
begin
  case curr_state is
    when idle =>
      if rd_eflag = '0' then
        next_state <= read_addr;
      elsif send_req = '1' then
        if wr_fflag = '0' then
          next_state <= write_addr;
        else
          next_state <= idle;
        end if;
      else
        next_state <= idle;
      end if;
    when read_addr =>
      next_state <= read_no_empty;
    when read_no_empty =>
      next_state <= read_read;
    when read_read =>
      if rd_eflag = '0' then
        next_state <= read_addr;
      else
        next_state <= idle;
      end if;
```

```
when write_addr =>
  next_state <= write_no_full;
when write_no_full =>
  next_state <= write_write;
when write_write =>
  next_state <= write_end;
when write_end =>
  if send_req = '1' then
    if rd_eflag = '1' then
      next_state <= write_no_full;
    else
      next_state <= idle;
    end if;
  else
    next_state <= idle;
  end if;
when others =>
  next_state <= idle;
end case;
end process interfaz_fsm;
```

— temporizaciones

```
counter3: process(sys_clk, reset, trig3)
begin
  if reset = '0' then
    count3 <= 0;
  elsif rising_edge(sys_clk) then
    if count3 > 0 then
      count3 <= count3 - 1;
    elsif trig3 = '1' then
      count3 <= 4;
    end if;
  end if;
end process counter3;
```

```
trig3 <= '1' when (next_state = write_write) else '0';

counter2: process(sys_clk, reset, trig2)
begin
    if reset = '0' then
        count2 <= 0;
    elsif rising_edge(sys_clk) then
        if count2 > 0 then
            count2 <= count2 - 1;
        elsif trig2 = '1' then
            count2 <= 3;
        end if;
    end if;
end process counter2;

with next_state select
    trig2 <= '1' when read_no_empty | read_read | write_no_full | write_end,
    '0' when others;

global_fsm_clk: process (sys_clk, reset)
begin
    if reset = '0' then
        curr_state <= idle;
    elsif rising_edge(sys_clk) then
        if count2 = 0 and count3 = 0 then
            curr_state <= next_state;
        end if;
    end if;
end process global_fsm_clk;

end Behavioral;

library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.numeric_std.all;
library unisim;
use unisim.vcomponents.all;

entity fx2lp_interface_top is
generic(
    in_ep_addr: std_logic_vector(1 downto 0) := "00";
    out_ep_addr: std_logic_vector(1 downto 0) := "11";
    port_width: integer := 16
);
port(
    — senales que van y vienen desde y hacia el FX2LP
    fdata : inout std_logic_vector(port_width-1 downto 0); — datos FIFO
    faddr : out std_logic_vector(1 downto 0); — canal FIFO
    slrd : out std_logic; — senal de lectura
    slwr : out std_logic; — senal de escritura
    — EP2 streaming in (to pc)
    flaga : in std_logic; — EP2_full—>write_full_flag
    flagb : in std_logic; — EP8_empty—>read_empty_flag
    — EP8 bulk out (from pc)
    flagc : in std_logic; — EP8_full—>read_full_flag
    flagd : in std_logic; — EP2_empty—>write_empty_flag
    sloe : out std_logic; — senal de habilitacion de salida
    pktend : out std_logic;
    — reloj
    clk_out : out std_logic; — salida de reloj
    — senales desde hacia mojo
    clk_in : in std_logic; — entrada de reloj
    button : in std_logic; — activo en bajo
    led : out std_logic_vector(7 downto 0)
);
end fx2lp_interface_top;

architecture fx2lp_interface_arq of fx2lp_interface_top is
    — declaracion de componentes
```

```
COMPONENT fx2lp_interface
  GENERIC(
    in_ep_addr: std_logic_vector(1 downto 0) := "00";
    out_ep_addr: std_logic_vector(1 downto 0) := "11";
    port_width: integer := 16
  );
  PORT(
    clk : IN std_logic;
    reset : IN std_logic;
    flaga : IN std_logic;
    flagb : IN std_logic;
    flagc : IN std_logic;
    flagd : IN std_logic;
    send_req : IN std_logic;
    data_to_tx : IN std_logic_vector(15 downto 0);
    fdata : INOUT std_logic_vector(15 downto 0);
    faddr : OUT std_logic_vector(1 downto 0);
    slrd : OUT std_logic;
    slwr : OUT std_logic;
    sloe : OUT std_logic;
    pktend : OUT std_logic;
    rx_data : OUT std_logic_vector(15 downto 0)
  );
END COMPONENT;
```

```
COMPONENT fifo_generator_v9_3
  PORT (
    rst : IN STD_LOGIC;
    wr_clk : IN STD_LOGIC;
    rd_clk : IN STD_LOGIC;
    din : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
    wr_en : IN STD_LOGIC;
    rd_en : IN STD_LOGIC;
    dout : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
```

```
full : OUT STD_LOGIC;
empty : OUT STD_LOGIC;
valid : OUT STD_LOGIC
);
END COMPONENT;

component clk_wiz_v3_6
port(
    CLK_IN1 : in std_logic;
    CLK_OUT1: out std_logic;
    CLK_OUT2: out std_logic;
    CLK_OUT3: out std_logic;
    CLK_OUT4: out std_logic;
    RESET : in std_logic;
    LOCKED : out std_logic
);
end component;

— declaracion de señales
— relojes
    — reloj usado en el sistema
signal sys_clk : std_logic := '0';
    — salidas de pll
signal pll_0 : std_logic := '0';
signal pll_90 : std_logic := '0';
signal pll_180 : std_logic := '0';
signal pll_270 : std_logic := '0';
signal locked : std_logic := '0';

— cables de utilidad
signal write_req : std_logic := '0';
signal slwr_sig : std_logic := '0';
signal slrd_sig : std_logic := '0';

— cables para registro de eco
```

```
— signal d_reg, q_reg
— signal c_reg, rst_reg, ce_reg

— cables necesarios para la memoria fifo
signal fifo_full      : std_logic;
signal fifo_empty     : std_logic;
signal wr_en          : std_logic;
signal rd_en          : std_logic;
signal valid          : std_logic;
signal dout           : std_logic_vector(port_width-1 downto 0);
signal din            : std_logic_vector(port_width-1 downto 0);
— señales de sistema
    —init
signal reset          : std_logic := '0';

        —para depuracion
signal debug_clk
signal counter
signal checksum

        —temporizaciones
signal count3
signal count2
signal cont
signal rst_cont
signal trig3, trig2

— Maquinas de Estados: xc6slx9-2tqg144
    —maquina de estados de la interfaz
type if_fsm is
(
    idle,
    read_addr, read_no_empty, read_read,
    write_addr, write_no_full, write_write, write_end
);
```

```
signal curr_state, next_state
--debug
signal num_state
--debug

type fifo_wr_states is
(
    idle, fifo_wren, fifo_nwren
);
signal fifo_wr_cst:      fifo_wr_states := idle;
signal fifo_wr_nst:      fifo_wr_states := idle;

type fifo_rd_states is
(
    idle, fifo_rden, fifo_nrden
);
signal fifo_rd_cst:      fifo_rd_states := idle;
signal fifo_rd_nst:      fifo_rd_states := idle;

begin
--debuggin leds
    — fisically connected. Don't comment!!!
led(7 downto 0) <= (others => '0');

addr_y : ODDR2
port map
(
    D0      => '1',
    D1      => '0',
    CE      => '1',
    C0      => pll_180,
    C1      => (not pll_180),
    R       => '0',
    — clk out buffer
```

```
S      => '0',
Q      => clk_out
);

—instanciaciones
— interfaz
interface: fx2lp_interface PORT MAP(
    clk => sys_clk,
    reset => reset,
    fdata => fdata,
    faddr => faddr,
    slrd => slrd_sig,
    slwr => slwr_sig,
    flaga => flaga,
    flagb => flagb,
    flagc => flagc,
    flagd => flagd,
    sloe => sloe,
    pktend => pktend,
    send_req => write_req,
    rx_data => din,
    data_to_tx => dout
);

—fifo
fifo : fifo_generator_v9_3
PORT MAP (
    rst => not reset,
    wr_clk => sys_clk,
    rd_clk => sys_clk,
    din => din,
    wr_en => wr_en,
    rd_en => rd_en,
    dout => dout,
    full => fifo_full,
```

```

    empty => fifo_empty ,
    valid => valid
);

pll : clk_wiz_v3_6
port map(
    CLK_IN1 => clk_in ,
    CLK_OUT1 => pll_0 ,
    CLK_OUT2 => pll_90 ,
    CLK_OUT3 => pll_180 ,
    CLK_OUT4 => pll_270 ,
    RESET => '0',
    LOCKED => locked
);

```

—REVISAR TODO ESTO

```

wr_req_ff : process (sys_clk , slwr_sig , fifo_empty)
begin
    if rising_edge(sys_clk) then
        if fifo_empty = '0' then
            write_req <= '1';
        else
            if slwr_sig = '1' then
                write_req <= '0';
            else
                write_req <= write_req;
            end if;
        end if;
    end if;
end process wr_req_ff;
—REVISAR TODO ESTO

```

— reloj
 $sys_clk \leqslant pll_0$;

```

— conexiones de señales internas hacia el exterior
slwr <= slwr_sig;
slrd <= slrd_sig;
sloe <= sloe_int;
faddr <= faddr_int;
pktend <= pktend_int;

write_full_flag <= flaga;
write_empty_flag <= flagd;
read_full_flag <= flagc;
read_empty_flag <= flagb;

reset <= '1' when rst_cont = 0 else '0';

— control signaling
with curr_state select
    faddr_int <=    out_ep_addr when read_addr | read_no_empty | read_read,-- |
                                in_ep_addr when write_addr | write_no_empty | write_read,
                                (others => 'Z') when others;

    slwr_int <=      '0' when next_state = write_write else
                      '1';

    slrd_int <=      '0' when curr_state = read_no_empty else
                      '1';

    pktend_int <= (read_empty_flag or write_req);

    with curr_state select
        sloe_int <=      '0' when read_read | read_no_empty,
        sloe_int <=      '0' when read_addr | read_read | read_no_empty,
                           '1' when others;

— debug
    fdata_out(15 downto 8) <= counter(7 downto 0);

```

```

      fdata_out(7 downto 0) <= counter(15 downto 8);
      debug

      with curr_state select
          fdata <= dout      —dout->fdata_out
          when write_no_full | write_write | write_end | write_
          (others => 'Z') when others;

      with curr_state select
          fdata_in <= fdata   when read_no_empty | read_read | read_addr,
          fdata_in when others;

      — fifo seignales
      wr_en <= '1' when fifo_wr_cst = fifo_wren else '0';

      rd_en <= '1' when fifo_rd_cst = fifo_rden else '0';

      — Implementacion de las maquinas de estado de la interfaz
      interfaz_fsm: process(curr_state, write_full_flag, read_empty_flag, write_req)
      begin
          case curr_state is
              when idle =>
                  if read_empty_flag = '1' then
                      next_state <= read_addr;
                  elsif write_req = '1' then
                      if write_full_flag = '1' then
                          next_state <= write_addr;
                      else
                          next_state <= idle;
                      end if;
                  else
                      next_state <= idle;
                  end if;

              when read_addr =>

```

```
next_state <= read_no_empty;

when read_no_empty =>
    next_state <= read_read;

when read_read =>
    if read_empty_flag = '1' then
        next_state <= read_addr;
    else
        next_state <= idle;
    end if;

when write_addr =>
    next_state <= write_no_full;

when write_no_full =>
    next_state <= write_write;

when write_write =>
    next_state <= write_end;

when write_end =>
    if write_req = '1' then
        if read_empty_flag = '0' then
            next_state <= write_no_full;
        else
            next_state <= idle;
        end if;
    else
        next_state <= idle;
    end if;

when others =>
    next_state <= idle;
end case;
```

```
end process interfaz_fsm;

— maquina de estados de lectura fifo
read_fifo_fsm : process(fifo_rd_cst, fifo_empty, slwr_sig)
begin
    case fifo_rd_cst is
        when idle =>
            if slwr_sig = '0' then
                if fifo_empty ='0' then
                    fifo_rd_nst <= fifo_rdlen;
                else
                    fifo_rd_nst <= idle;
                end if;
            else
                fifo_rd_nst <= idle;
            end if;

        when fifo_rdlen =>
            fifo_rd_nst <= fifo_nrden;

        when fifo_nrden =>
            if slwr_sig ='1' then
                fifo_rd_nst <= idle;
            end if;

        when others =>
            fifo_rd_nst <= idle;
    end case;
end process read_fifo_fsm;

— maquina de estados escritura fifo
write_fifo_fsm: process(fifo_wr_cst, slrd_sig, fifo_full)
begin
    case fifo_wr_cst is
        when idle =>
```

```

        if slrd_sig = '0' then
            if fifo_full = '0' then
                fifo_wr_nst <= fifo_wren;
            else
                fifo_wr_nst <= idle;
            end if;
        else
            fifo_wr_nst <= idle;
        end if;

when fifo_wren =>
    fifo_wr_nst <= fifo_nwren;

when fifo_nwren =>
    if slrd_sig = '1' then
        fifo_wr_nst <= idle;
    end if;

when others =>
    fifo_wr_nst <= idle;
end case;
end process write_fifo_fsm;

```

— temporizaciones

```

counter3: process(sys_clk, reset, trig3)
begin
    if reset = '0' then
        count3 <= 0;
    elsif rising_edge(sys_clk) then
        if count3 > 0 then
            count3 <= count3 - 1;
        elsif trig3 = '1' then
            count3 <= 4;
        end if;
    end if;

```

```
end process counter3;

trig3 <= '1' when (next_state = write_write) else '0';

counter2: process(sys_clk, reset, trig2)
begin
    if reset = '0' then
        count2 <= 0;
    elsif rising_edge(sys_clk) then
        if count2 > 0 then
            count2 <= count2 - 1;
        elsif trig2 = '1' then
            count2 <= 3;
        end if;
    end if;
end process counter2;

with next_state select
    trig2 <=      '1' when read_no_empty | read_read | write_no_full | write_en
                    '0' when others;

— reloj fsm
global_fsm_clk: process (sys_clk, reset)
begin
    if reset = '0' then
        curr_state <= idle;
        fifo_rd_cst <= idle;
        fifo_wr_cst <= idle;
    elsif rising_edge(sys_clk) then
        fifo_rd_cst <= fifo_rd_nst;
        fifo_wr_cst <= fifo_wr_nst;
        if count2 = 0 and count3 = 0 then
            curr_state <= next_state;
        end if;
    end if;
```

```
end process global_fsm_clk;

reloj_lento: process(pll_0)
begin
    if(rising_edge(pll_0))then
        cont <= cont - 1;
        if cont = 0 then
            cont <= 10000000;
            debug_clk <= not debug_clk;
        end if;
    end if;
end process reloj_lento;

init_RST: process(sys_clk)
begin
    if rst_cont /= 0 then
        if rising_edge(sys_clk) then
            rst_cont <= rst_cont - 1;
        end if;
    end if;
end process init_RST;

end fx2lp_interface_arq;
```

```
#Created by Constraints Editor (xc6slx9-tqg144-2) - 2017/11/24
```

```
NET "clk_in" TNM_NET = "clk_in";
```

```
TIMESPEC TS_clk_in = PERIOD "clk_in" 50 MHz HIGH 50 %;
```

```
NET "clk_in" LOC = P56;
```

```
NET "clk_in" IOSTANDARD = LVTTL;
```

```
NET "button" LOC = P38;
```

```
NET "button" IOSTANDARD = LVTTL;
```

```
#net "clk_out" loc = p | iostandard = lvttl;  
NET "fdata[15]" LOC = P50;  
NET "fdata[15]" IOSTANDARD = LVTTL;  
NET "fdata[14]" LOC = P51;  
NET "fdata[14]" IOSTANDARD = LVTTL;  
NET "fdata[13]" LOC = P40;  
NET "fdata[13]" IOSTANDARD = LVTTL;  
NET "fdata[12]" LOC = P41;  
NET "fdata[12]" IOSTANDARD = LVTTL;  
NET "fdata[11]" LOC = P34;  
NET "fdata[11]" IOSTANDARD = LVTTL;  
NET "fdata[10]" LOC = P35;  
NET "fdata[10]" IOSTANDARD = LVTTL;  
NET "fdata[9]" LOC = P32;  
NET "fdata[9]" IOSTANDARD = LVTTL;  
NET "fdata[8]" LOC = P33;  
NET "fdata[8]" IOSTANDARD = LVTTL;  
NET "fdata[7]" LOC = P29;  
NET "fdata[7]" IOSTANDARD = LVTTL;  
NET "fdata[6]" LOC = P30;  
NET "fdata[6]" IOSTANDARD = LVTTL;  
NET "fdata[5]" LOC = P26;  
NET "fdata[5]" IOSTANDARD = LVTTL;  
NET "fdata[4]" LOC = P27;  
NET "fdata[4]" IOSTANDARD = LVTTL;  
NET "fdata[3]" LOC = P23;  
NET "fdata[3]" IOSTANDARD = LVTTL;  
NET "fdata[2]" LOC = P24;  
NET "fdata[2]" IOSTANDARD = LVTTL;  
NET "fdata[1]" LOC = P21;  
NET "fdata[1]" IOSTANDARD = LVTTL;  
NET "fdata[0]" LOC = P22;  
NET "fdata[0]" IOSTANDARD = LVTTL;  
  
NET "faddr[1]" LOC = P9;
```

```
NET "faddr[1]" IOSTANDARD = LVTTL;  
NET "faddr[0]" LOC = P8;  
NET "faddr[0]" IOSTANDARD = LVTTL;  
  
NET "slrd" LOC = P16;  
NET "slrd" IOSTANDARD = LVTTL;  
NET "slwr" LOC = P17;  
NET "slwr" IOSTANDARD = LVTTL;  
NET "flaga" LOC = P12;  
NET "flaga" IOSTANDARD = LVTTL;  
NET "flagb" LOC = P14;  
NET "flagb" IOSTANDARD = LVTTL;  
  
NET "flagc" LOC = P15;  
NET "flagc" IOSTANDARD = LVTTL;  
NET "flagd" LOC = P11;  
NET "flagd" IOSTANDARD = LVTTL;  
NET "sloe" LOC = P6;  
NET "sloe" IOSTANDARD = LVTTL;  
  
NET "pktend" LOC = P10;  
NET "pktend" IOSTANDARD = LVTTL;  
#NET "clk" TNM_NET = clk;  
#TIMESPEC TS_clk = PERIOD "clk" 50 MHz HIGH 50%;  
#  
#NET "clk" LOC = P56 | IOSTANDARD = LVTTL;  
#NET "rst_n" LOC = P38 | IOSTANDARD = LVTTL;  
#  
#NET "cclk" LOC = P70 | IOSTANDARD = LVTTL;  
#  
NET "led[0]" LOC = P134;  
NET "led[0]" IOSTANDARD = LVTTL;  
NET "led[1]" LOC = P133;  
NET "led[1]" IOSTANDARD = LVTTL;  
NET "led[2]" LOC = P132;
```

```
NET "led[2]" IOSTANDARD = LVTTL;
NET "led[3]" LOC = P131;
NET "led[3]" IOSTANDARD = LVTTL;
NET "led[4]" LOC = P127;
NET "led[4]" IOSTANDARD = LVTTL;
NET "led[5]" LOC = P126;
NET "led[5]" IOSTANDARD = LVTTL;
NET "led[6]" LOC = P124;
NET "led[6]" IOSTANDARD = LVTTL;
NET "led[7]" LOC = P123;
NET "led[7]" IOSTANDARD = LVTTL;
#
#NET "spi_mosi" LOC = P44 | IOSTANDARD = LVTTL;
#NET "spi_miso" LOC = P45 | IOSTANDARD = LVTTL;
#NET "spi_ss" LOC = P48 | IOSTANDARD = LVTTL;
#NET "spi_sck" LOC = P43 | IOSTANDARD = LVTTL;
#NET "spi_channel<0>" LOC = P46 | IOSTANDARD = LVTTL;
#NET "spi_channel<1>" LOC = P61 | IOSTANDARD = LVTTL;
#NET "spi_channel<2>" LOC = P62 | IOSTANDARD = LVTTL;
#NET "spi_channel<3>" LOC = P65 | IOSTANDARD = LVTTL;
#
#NET "avr_tx" LOC = P55 | IOSTANDARD = LVTTL;
#NET "avr_rx" LOC = P59 | IOSTANDARD = LVTTL;
#NET "avr_rx_busy" LOC = P39 | IOSTANDARD = LVTTL;
```

— Trabajo Final de Ingeniería Electrónica
— Universidad Nacional de San Juan
— Autor: Edwin Barragán
— Asesores: Cristian Sisterna
— Martín Pérez
— Marcelo Segura

— Create Date: 14:13:35 07/21/2017

— Design Name: FX2LP – FPGA Interface
— Module Name: /home/lechuzin/Facultad/Trabajo Final/lechuzing/Cyclon6/interface_test_bench
— Project Name: Cyclon6
— Target Device: Cyclon6
— Tool versions:
— Description:
—
— VHDL Test Bench Created by ISE for module: fx2lp_interface_top
—
— Dependencies:
—
— Revision:
— Revision 0.01 – File Created
— Additional Comments:
—
— Notes:
— This testbench has been automatically generated using types std_logic and
— std_logic_vector for the ports of the unit under test. Xilinx recommends
— that these types always be used for the top-level I/O of a design in order
— to guarantee that the testbench will bind correctly to the post-implementation
— simulation model.

```
LIBRARY ieee;  
USE ieee.std_logic_1164.ALL;  
use ieee.std_logic_unsigned.all;
```

— Uncomment the following library declaration if using
— arithmetic functions with Signed or Unsigned values
— USE ieee.numeric_std.ALL;

```
ENTITY interface_test_bench IS  
END interface_test_bench;
```

```
ARCHITECTURE behavior OF interface_test_bench IS
```

— Component Declaration for the Unit Under Test (UUT)

```
COMPONENT fx2lpi-interface_top
PORT(  

    fdata : INOUT std_logic_vector(15 downto 0);  

    faddr : OUT std_logic_vector(1 downto 0);  

    slrd : OUT std_logic;  

    slwr : OUT std_logic;  

    flaga : IN std_logic;  

    flagb : IN std_logic;  

    flagc : IN std_logic;  

    flagd : IN std_logic;  

    sloe : OUT std_logic;  

    pktend : OUT std_logic;  

    clk_in : IN std_logic;  

    clk_out : OUT std_logic;  

    button : IN std_logic;  

    send_req : IN std_logic;  

    data_out : OUT std_logic_vector(15 downto 0);  

    data_in : IN std_logic_vector(15 downto 0)  

);
END COMPONENT;
```

— Inputs

```
signal flaga : std_logic := '0';  

signal flagb : std_logic := '0';  

signal flagc : std_logic := '0';  

signal flagd : std_logic := '0';  

signal clk_in : std_logic := '0';  

signal button : std_logic := '0';  

signal send_req : std_logic := '0';  

signal data_in : std_logic_vector(15 downto 0) := (others => '0');
```

— BiDirs

```
signal fdata : std_logic_vector(15 downto 0);

— Outputs
signal faddr : std_logic_vector(1 downto 0);
signal slrd : std_logic;
signal slwr : std_logic;
signal sloe : std_logic;
signal pktend : std_logic;
signal clk_out : std_logic;
signal data_out : std_logic_vector(15 downto 0);

— Clock period definitions
constant clk_in_period : time := 21 ns;
constant clk_out_period : time := 21 ns;

— Data generator
signal counter: std_logic_vector(15 downto 0) := x"0000";
```

BEGIN

```
— Instantiate the Unit Under Test (UUT)
uut: fx2lp_interface_top PORT MAP (
    fdata => fdata,
    faddr => faddr,
    slrd => slrd,
    slwr => slwr,
    flaga => flaga,
    flagb => flagb,
    flagc => flagc,
    flagd => flagd,
    sloe => sloe,
    pktend => pktend,
    clk_in => clk_in,
    clk_out => clk_out,
    button => button,
```

```
send_req => send_req,
data_out => data_out,
data_in => data_in
);

— Clock process definitions
clk_in_process :process
begin
    clk_in <= '0';
    wait for clk_in_period/2;
    clk_in <= '1';
    wait for clk_in_period/2;
end process;

clk_out_process :process
begin
    clk_out <= '0';
    wait for clk_out_period/2;
    clk_out <= '1';
    wait for clk_out_period/2;
end process;

— Start up reset
button <= '0', '1' after 100 ns;

— flags signaling. All them are low-active
flags_proc: process
begin
    flaga <= '1'; — ep2full
    flagb <= '0'; — ep8empty
    flagc <= '1'; — ep8full
    flagd <= '0'; — ep2empty
    wait until button = '1';

    wait until falling_edge(clk_in);
```

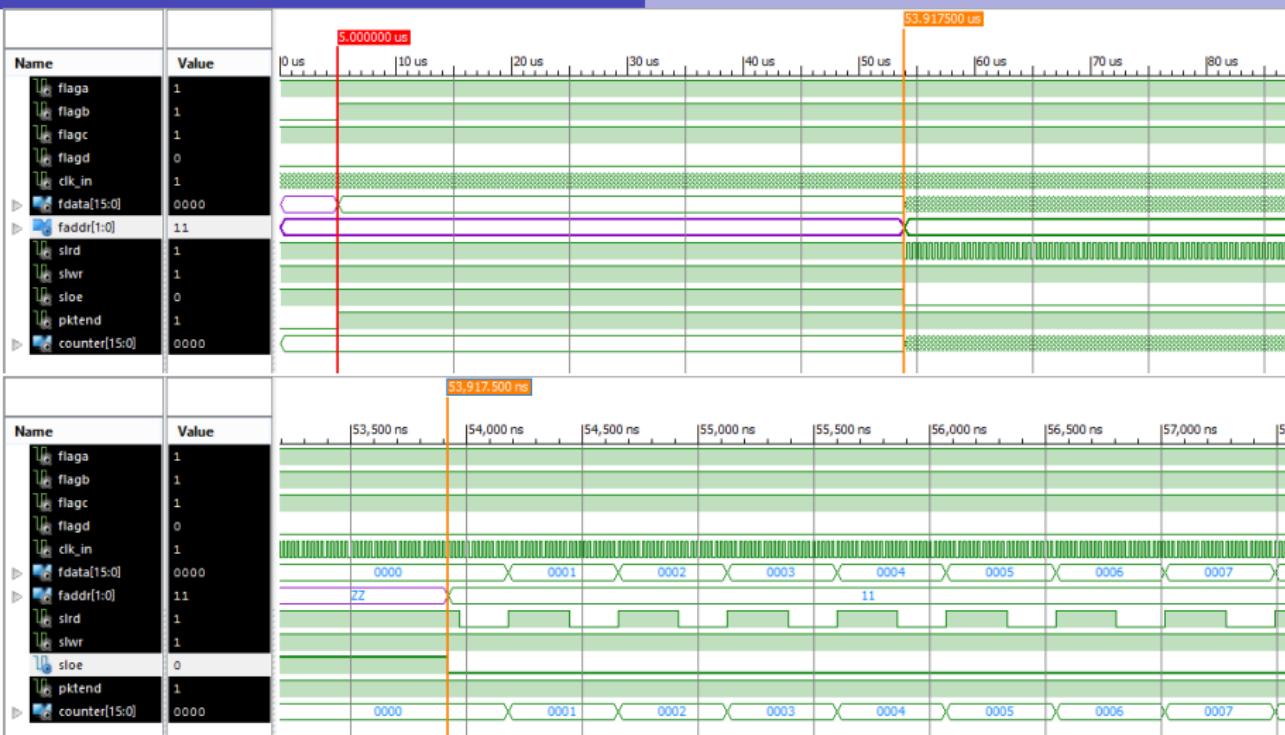
```
flagb <= '1';
wait until counter = x"0060";
wait until falling_edge(clk_in);

flagb <= '0';
wait until pktend = '0';
end process;

— Data counter
data_proc: process(slrd)
begin
    if button = '0' then
        counter <= x"0000";
    elsif rising_edge(slrd)then
        counter <= counter + 1;
    end if;
end process;

— Stimulus process
with flagb select
    fdata <= counter when '1',( others => 'Z')when others;

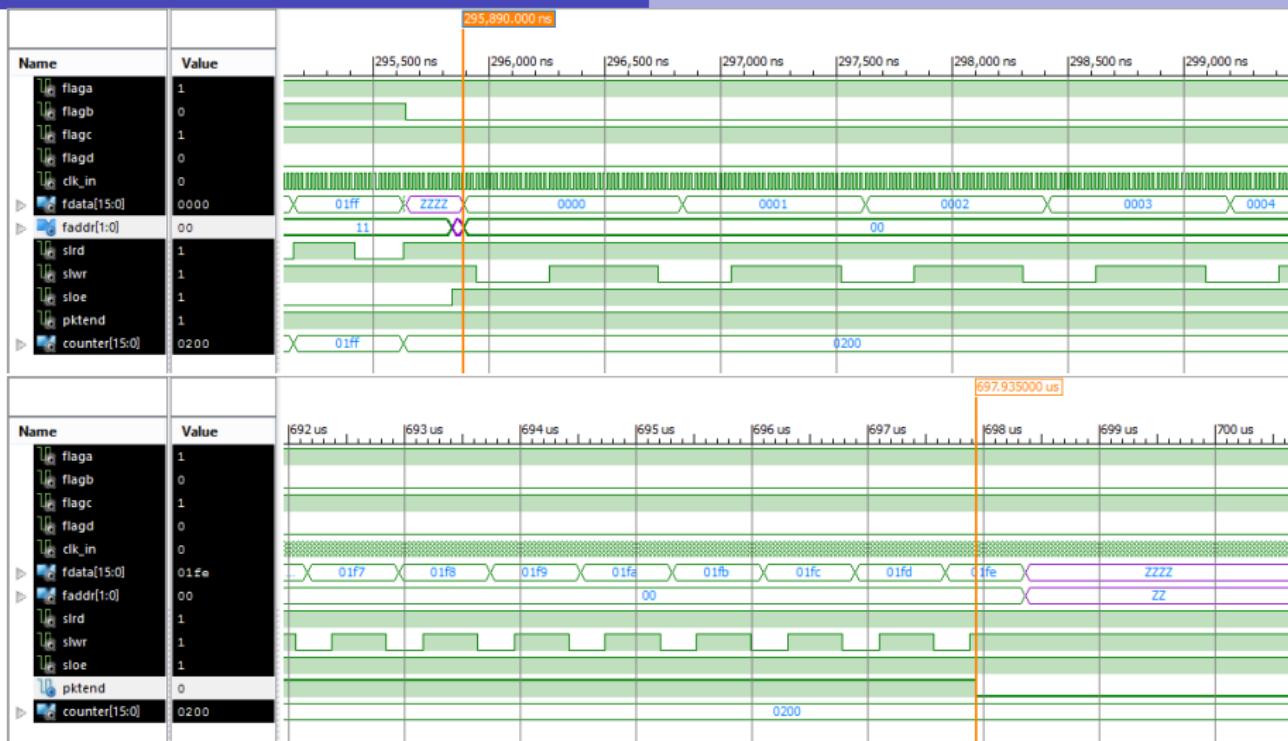
END;
```



Resultados y conclusiones

Trabajo futuro





```
#pragma noiv // Do not generate interrupt vectors
```

```
#include "fx2.h"
#include "fx2regs.h"
#include "syncdly.h" // SYNCDELAY macro
#include "FX2LPSerial.h"
```

```
#include "leds.h"

extern BOOL GotSUD;           // Received setup data flag
extern BOOL Sleep;
extern BOOL RwuEn;
extern BOOL Selfpwr;

#define BTN_ADDR          0x20
#define LED_ADDR          0x21

BYTE xdata Digit[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82,
                      0xf8, 0x80, 0x98, 0x88, 0x83, 0xc6, 0xa1, 0x86,
                      0x8e};

// Dummy—Read these values to turn LED's on and off
//xdata volatile const BYTE D2ON      _at_ 0x8800;
//xdata volatile const BYTE D2OFF     _at_ 0x8000;
//xdata volatile const BYTE D3ON      _at_ 0x9800;
//xdata volatile const BYTE D3OFF     _at_ 0x9000;
//xdata volatile const BYTE D4ON      _at_ 0xA800;
//xdata volatile const BYTE D4OFF     _at_ 0xA000;
//xdata volatile const BYTE D5ON      _at_ 0xB800;
//xdata volatile const BYTE D5OFF     _at_ 0xB000;

BYTE Configuration;           // Current configuration
BYTE AlternateSetting = 0;    // Alternate settings



---


// Task Dispatcher hooks
// The following hooks are called by the task dispatcher.


---


```

```
WORD mycount = 0;
WORD blinktime = 0;
BYTE inblink = 0x00;
```

```
BYTE outblink = 0x00;
WORD blinkmask = 0; // HS/FS blink rate

BYTE refresh = 0;

void TD_Init(void) // Called once at startup
{
    BYTE dum;

    // turn off the 4 LED's
    dum = D2OFF;
    dum = D3OFF;
    dum = D4OFF;
    dum = D5OFF;

    // set the CPU clock to 48MHz
    //CPUCS = ((CPUCS & ~bmCLKSPD) | bmCLKSPD1);
    // 48 MHz //commented 'cos are configured by Serial_Init
    FX2LPSerial_Init();
    FX2LPSerial_XmitString("Serial_Port_Initialized");
    FX2LPSerial_XmitChar('\n');
    FX2LPSerial_XmitChar('\n');

    // set the slave FIFO interface to 48MHz
    //set s-fifo to internal clk, no_output,
    //no_inverted clk, no_async, no_gstate,
    //slavefifo(11) = 0xC3
    IFCONFIG = 0xcb; //para hacerlo async.
    //Error de diseño.
    //Corregir en VHDL previamente

    SYNCDELAY;
    REVCTL = 0x00; /*Chip Revision Control Register: poniendo
                    esto en 0x01 me deja manipular los
                    paquetes, pero debo mover paquete por > < < < < > < < < > < < < >
```

*paquete a cada buffer...
no logro tomar ningun tipo de flag y mucho
menos pude mover el buffer a la memoria fifo*

*Por otro lado, poniendo el bit 0x02 logro
desactivar el auto-armado de los endopoints
y puedo cambiar de modo autoout a manualout
sin necesidad de perder datos.*/*

SYNCDELAY;

//Pin Flags Configuration

```
PINFLAGSAB = 0xBC;           // FLAGA <- EP2 Full Flag
                            // FLAGD <- EP2 Empty Flag
SYNCDELAY;
PINFLAGSCD = 0x8F;           // FLAGC <- EP8 Full Flag
                            // FLAGB <- EP8 Empty Flag
```

EP1OUTCFG = 0xA0;

SYNCDELAY;

EP1INCFG = 0xA0;

SYNCDELAY;

EP4CFG &= 0x7F;

SYNCDELAY;

EP6CFG &= 0x7F;

SYNCDELAY;

EP8CFG = 0xA0; //EP8: DIR=OUT, TYPE=BULK, SIZE=512, BUF=2x

SYNCDELAY;

EP2CFG = 0xD2; //EP2: DIR=IN, TYPE=ISOC, SIZE=1024, BUF=3x

SYNCDELAY;

FIFORESET = 0x80;

SYNCDELAY;

FIFORESET = 0x02;

SYNCDELAY;

FIFORESET = 0x04;

```
SYNCDELAY;
FIFORESET = 0x06;
SYNCDELAY;
FIFORESET = 0x08;
SYNCDELAY;
FIFORESET = 0x00;
SYNCDELAY;

EP8FIFOCFG = 0x00;
SYNCDELAY;
EP2FIFOCFG = 0x00;
SYNCDELAY;

//setting on auto mode. rising edge is necessary
EP8FIFOCFG = 0x11;
SYNCDELAY;
EP2FIFOCFG = 0x0D;
SYNCDELAY;

// We want to get SOF interrupts
USBIE |= bmSOF;
EIEX4 = 1;
INTSETUP |= (INT4IN | bmAV4EN);
EXIF &= ~0x40;
EP8FIFOIE = 0x03;

FX2LPSerial_XmitString(" Bridge_Configured\n");
}

void TD_Poll(void)      // Called repeatedly while the device is idle
{
    BYTE dum;

    if(EP8FIFOFLGS & bmBIT1)//ep8 fifo empty
    {
```

```
dum = D4ON;  
}  
else  
{  
    dum = D4OFF;  
}  
  
if(EP8FIFOFLGS & bmBIT0)//ep8 fifo full//  
//(EP2468STAT & bmBIT6)//ep8 empty  
{  
    dum = D3ON;  
}  
else  
{  
    dum = D3OFF;  
}  
  
if(EP2FIFOFLGS & bmBIT1)//ep2 fifo empty  
{  
    dum = D2ON;  
}  
else  
{  
    dum = D2OFF;  
}  
}
```

```
BOOL TD_Suspend(void) //Called before the device goes into  
//suspend mode  
{  
    return(TRUE);
```

}

```
BOOL TD_Resume(void)      // Called after the device resumes
{
    return(TRUE);
}
```

```
// Device Request hooks
// The following hooks are called by the end point 0 device
// request parser.
```

```
BOOL DR_GetDescriptor(void)
{
    FX2LPSerial_XmitString(" Getting_Descriptor:~ ");
    return(TRUE);
}
```

```
BOOL DR_SetConfiguration(void) //Called when a Set Configuration
                                //command is received
{
    FX2LPSerial_XmitString(" Setting_Config\n\n");
    Configuration = SETUPDAT[2];
    return(TRUE);           // Handled by user code
}
```

```
BOOL DR_GetConfiguration(void) //Called when a Get Configuration
                                //command is received
{
    FX2LPSerial_XmitString(" Getting_Config\n\n");
    EP0BUF[0] = Configuration;
    EP0BCH = 0;
    EP0BCL = 1;
    return(TRUE);           // Handled by user code
```

```
}

BOOL DR_SetInterface(void)      //Called when a Set Interface
                                //command is received
{
    FX2LPSerial_XmitString(" Setting_Interface\n\n");
    AlternateSetting = SETUPDAT[2];
    return(TRUE);           // Handled by user code
}

BOOL DR_GetInterface(void)      //Called when a Set Interface
                                //command is received
{
    FX2LPSerial_XmitString(" Getting_Interface\n\n");
    EP0BUF[0] = AlternateSetting;
    EP0BCH = 0;
    EP0BCL = 1;
    return(TRUE);           // Handled by user code
}

BOOL DR_GetStatus(void)
{
    FX2LPSerial_XmitString(" Getting_Status\n\n");
    return(TRUE);
}

BOOL DR_ClearFeature(void)
{
    FX2LPSerial_XmitString(" Clearing_Features\n\n");
    return(TRUE);
}

BOOL DR_SetFeature(void)
{
```

```
FX2LPSerial_XmitString("Setting_Features\n\n");
return(TRUE);
}

BOOL DR_VendorCmnd( void )
{
    return(TRUE);
}

//-----  

// USB Interrupt Handlers
// The following functions are called by the USB interrupt jump
// table.
//-----  

// Setup Data Available Interrupt Handler
void ISR_Sudav(void) interrupt 0
{
    FX2LPSerial_XmitString("SUDAv_ISR\n");
    GotSUD = TRUE;           // Set flag
    EZUSB_IRQ_CLEAR();
    USBIRQ = bmSUDAV;        // Clear SUDAV IRQ
}

// Setup Token Interrupt Handler
void ISR_Sutok(void) interrupt 0
{
    FX2LPSerial_XmitString("SUTok_ISR\n");
    EZUSB_IRQ_CLEAR();
    USBIRQ = bmSUTOK;        // Clear SUTOK IRQ
}

void ISR_Sof(void) interrupt 0
{
```

```
BYTE dum;
EZUSB_IRQ_CLEAR();
USBIRQ = bmSOF;           // Clear SOF IRQ
blinktime++;
if(blinktime&blinkmask)
    dum=D5OFF;           // ~1/sec blinking LED
else
    dum=D5ON;

if(--inblink == 0)
    dum = D4OFF;
if(--outblink == 0)
    dum = D3OFF;
}

void ISR_Ures(void) interrupt 0
{
    BYTE dum;

    // Whenever we get a USB Reset, we should revert to
    // full speed mode
    FX2LPSerial_XmitString("URst_ISR\n");
    pConfigDscr = pFullSpeedConfigDscr;
    ((CONFIGDSCR *xdata *)pConfigDscr)->type = CONFIG_DSCR;
    pOtherConfigDscr = pHIGH_SPEED_CONFIG_DSCR;
    ((CONFIGDSCR *xdata *)pOtherConfigDscr)->type =
        OTHERSPEED_DSCR;

    EZUSB_IRQ_CLEAR();
    USBIRQ = bmURES;       // Clear URES IRQ
    dum = D2OFF;           // Turn off high-speed LED
    blinkmask = 0x0400;     // 2 sec period for FS
}

void ISR_Susp(void) interrupt 0
```

```
{  
    Sleep = TRUE;  
    EZUSB_IRQ_CLEAR();  
    USBIRQ = bmSUSP;  
}  
  
void ISR_Highspeed(void) interrupt 0  
{  
    blinkmask = 0x1000;      // 1 sec period for HS  
    if (EZUSB_HIGHSPEED())  
    {  
        pConfigDscr = pHighSpeedConfigDscr;  
        ((CONFIGDSCR *xdata *) pConfigDscr)->type =  
            CONFIG_DSCR;  
        pOtherConfigDscr = pFullSpeedConfigDscr;  
        ((CONFIGDSCR *xdata *) pOtherConfigDscr)->type =  
            OTHERSPEED_DSCR;  
    }  
    else  
    {  
        pConfigDscr = pFullSpeedConfigDscr;  
        pOtherConfigDscr = pHighSpeedConfigDscr;  
    }  
  
    EZUSB_IRQ_CLEAR();  
    USBIRQ = bmHSGRANT;  
}  
void ISR_Ep8eflag(void) interrupt 0  
{  
    EXIF &= ~bmBIT6;  
    EP8FIFOIRQ = 0x02;  
    FX2LPSerial_XmitHex2(EP8BCH);  
    FX2LPSerial_XmitHex2(EP8BCL);  
    FX2LPSerial_XmitString("\nEP8: -Estoy -vacío\n");  
}
```

```
}

void ISR_Ep8fflag(void) interrupt 0
{
    EXIF &= ~0x40;
    EP8FIFOIRQ = 0x01;
    FX2LPSerial_XmitString("EP8:_Estoy_lleno\n");
}
```

```
///
// Name      : tfUSBCheck.cpp
// Author    : Edwin Barragan
// Version   :
// Copyright : My final degree work
// Description : Hello World in C++, Ansi-style
///
```

```
//TODO hay que hacer un script que programe el dispositivo
//TODO todavia no realiza comunicacion alguna
```

```
//Pasos a seguir segun
```

```
/*https://www.dreamincode.net/forums/topic/148707-introduction-to-using-libusb-10/page--p--885389--hl--USB--fromsearch--1#entry885389*/
```

```
/* initialize the library by calling the function libusb_init and creating a session
 * Call the function libusb_get_device_list to get a list of connected devices. This creates
 * Discover the one and open the device either by libusb_open or libusb_open_device_with_vid
 * Clear the list you got from libusb_get_device_list by using libusb_free_device_list
 * Claim the interface with libusb_claim_interface (requires you to know the interface number)
 * Do desired I/O
 * Release the device by using libusb_release_interface
 * Close the device you opened before, by using libusb_close
```



* Close the session by using libusb_exit*/

```
#include "tfUSBCheck.h"
bool to_exit = 1;
bool to_send = 1;
bool to_receive = 0;
bool to_check = 0;
bool to_log = 0;
int actual_length = 1;

USBChecker::USBChecker(void){
    ctx = NULL;
    dev = NULL;
    dscr = NULL;
    handler = NULL;
    transfer_in = NULL;
    transfer_out = NULL;
}

void USBChecker::closeChecker(void){
    if(transfer_out)
        libusb_cancel_transfer(transfer_out);
    if(transfer_in)
        libusb_cancel_transfer(transfer_in);
    libusb_release_interface(handler, INTERFACE);
}

USBChecker::USBChecker(libusb_context* context, libusb_device* device, libusb_device_handle* handle)
{
    ctx = context;
    dev = device;
    handler = handle;
    dscr = NULL;
    transfer_in = NULL;
```

```
        transfer_out = NULL;
}

void USBChecker::init_myusb_device(void) throw(int){

    int is_not_ok;
    libusb_device **list;
    ssize_t cont;
    ssize_t dev_idx;

    // initialize the library by calling the function libusb_init and creating a session
    // inicializacion del usb
    is_not_ok = libusb_init(&ctx);
    if(is_not_ok)
    {
        err_timestamp();
        cerr << "No funciona libusb" << endl;
        throw(NO_LIBUSB);
    }
    libusb_set_debug(ctx, LIBUSB_LOG_LEVEL_DEBUG); //todos los msjs seran mostrados en

    // Call the function libusb_get_device_list to get a list of connected devices.
    // This creates an array of libusb_device containing all USB devices connected to the
    // discover devices
    cont = libusb_get_device_list(ctx, &list);
    // Loop through all these devices and check their options
    for(dev_idx = 0; dev_idx < cont; dev_idx++)
    {
        libusb_device *aux_dev = list[dev_idx];
        libusb_device_descriptor aux_dscr;

        //getting descriptor
        is_not_ok = libusb_get_device_descriptor(aux_dev, &aux_dscr);
    }
}
```

```
if(is_not_ok)
{
    err_timestamp();
    cerr << "No_puede_acceder_a_un_descriptor" << endl;
}

// Discover the one
if(aux_dscr.idVendor == CY_VID)
{
    if(aux_dscr.idProduct == CY_FX2LP_PID)
    {
        dev = aux_dev;
        dscr = &aux_dscr;
    }
    else
    {
        err_timestamp();
        cerr << "El_dispositivo_no_se_encontraba_programado" << endl;
        throw(NO_PROGR);
    }
}

if(dev == NULL)
{
    cerr << "No_se_encontraba_el_dispositivo" << endl;
    throw(NO_DOI);
}

// Discover the one and open the device either by
// libusb_open or
// libusb_open_device_with_vid_pid(when you know vendor and product id of the device)
// to open the device
//open my device
```

```
handler = libusb_open_device_with_vid_pid(ctx, CY_VID, CY_FX2LP_PID);

// Clear the list you got from libusb_get_device_list by using libusb_free_device_list
libusb_free_device_list(list, 1);
//devices discovered
// Claim the interface with
//libusb_claim_interface (requires you to know the interface numbers of device)
    //first i have to detach the device driver to the kernel.
    //doing this automatically
is_not_ok = libusb_set_auto_detach_kernel_driver(handler, 1);
if(is_not_ok)
{
    cerr << "detached_failed" << endl;
    throw(DRIVER_BLOCKED);
}
libusb_claim_interface(handler, INTERFACE);
libusb_set_interface_alt_setting(handler,0,0);
//para debug
out_timestamp();
cout << "Tengo control sobre la placa" << endl;
}

void /*USBChecker::*/close_myusb_device(int ignored){
    cout << "Sali con la señal" << ignored << endl;
    to_exit = 0;
}

void USBChecker::dataGenerator(unsigned char* data){
    char number;
    unsigned char block = 0, bit;
    uint row;
    uint i;
    char rowParity = 0, colParity = 0;
    const char mask = 0x01;
```

```
while( block <= MAX_OUT_DATA / 8)
{
    i = block * 8;
    colParity = 0;
    for(row = i; row < i + 7; row++)
    {
        rowParity = 0;
        number = char(rand() & 0x7f);
        data[row] = number << 1;
        for(bit = 0; bit < 7; bit++)
        {
            rowParity = rowParity ^ (number & mask);
            number = number >> 1;
        }
        if(rowParity)
            data[row] = data[row] | 0x01;
        else
            data[row] = data[row] & 0xF7;
        colParity = colParity ^ data[row];
    }
    data[row] = colParity;
    block++;
}
//para debug
out_timestamp();
cout << "Genera datos aleatorios:" << endl;
for(i = 0; i < MAX_OUT_DATA; i++)
{
    if(i %16 == 15)
        cout << (int)data[i] << endl;
    else
        cout << (int)data[i] << '\t';
}
```

```
void USBChecker::dataChecker(unsigned char* data, vector<err> *errores){  
    unsigned char block = 0, bit;  
    uint row;  
    char mask = 0x01;  
  
    cout << "Estoy_en_el_checker" << endl;  
    while(block < actual_length/8)  
    {  
        err error;  
        error.block = 0;  
        error.row = 0;  
        error.check = 0;  
        uint i = block * 8;  
        char colParity = 0;  
        for(row = i; row < i + 8; row++)  
        {  
            char rowParity = 0;  
            char aux_data = data[row];  
            for(bit = 0; bit < 8; bit++)  
            {  
                rowParity = rowParity ^ (aux_data & mask);  
                aux_data >>= 1;  
            }  
            if(!(rowParity))  
            {  
                error.row = row;  
                errores->push_back(error);  
            }  
            colParity ^= data[row];  
        }  
        if(!colParity)  
        {  
            error.block = block;  
            error.check = colParity;  
        }  
    }  
}
```

```
                errores->push_back(error);
            }
        }
        to_log = 1;
        cout << "sali_del_Checker" << endl;
    }

void USBChecker::sendData(unsigned char* data){
    transfer_out = libusb_alloc_transfer(0);
//    int esta=0;
    libusb_fill_bulk_transfer(transfer_out , handler , OUT_EP , data , MAX_OUT_DATA , send_cb , NULL
//    if(libusb_bulk_transfer(handler , OUT_EP , data , MAX_OUT_DATA,&esta , 60000))
//        cout << "buenisimo" << endl;
//    else
//        cout << "malisimo" << endl;
    int c = libusb_submit_transfer(transfer_out );
    out_timestamp();
    cout << "Mande_datos_" << c << endl;
}

void USBChecker::send_cb(struct libusb_transfer* transfer){
    out_timestamp();
    cout << "Estoy_en_el_callback_de_envio" << endl;
    cout << transfer->status << endl;
    switch(transfer->status)
    {
        case LIBUSB_TRANSFER_COMPLETED:
            to_receive = 1;
            break;
        case LIBUSB_TRANSFER_ERROR:
            break;
        case LIBUSB_TRANSFER_TIMED_OUT:
            err_timestamp();
            cerr << "El_dispositivo_no_responde." << endl;
    }
}
```

```

        to_exit = 0;
        //timers libres. no ocurriria
        break;
    case LIBUSB_TRANSFER_CANCELLED:
        //no programado para que ocurra
        break;
    case LIBUSB_TRANSFER_STALL:
        //no programado para que ocurra
        break;
    case LIBUSB_TRANSFER_NO_DEVICE:
        //si se desconecta el dispositivo
        err_timestamp();
        cerr << "El dispositivo se desconecto. Se cierra el programa." << endl;
        to_exit = 0;
        return;
    case LIBUSB_TRANSFER_OVERFLOW:
        err_timestamp();
        cerr << "Falló por overflow. Revisar código de generación de datos." << endl;
        to_exit = 0;
        break;
    default:
        break;
}
libusb_free_transfer(transfer);
}

void USBChecker::receiveData(unsigned char* data){
int pktSize;
int isoPkts = 1;
transfer_in = libusb_alloc_transfer(isoPkts);
pktSize = libusb_get_max_iso_packet_size(dev, IN_EP);
libusb_fill_iso_transfer(transfer_in, handler, IN_EP, data, pktSize, isoPkts, receive_cb, NULL);
libusb_set_iso_packet_lengths(transfer_in, pktSize);
// transfer_in->flags = LIBUSB_TRANSFER_FREE_BUFFER | LIBUSB_TRANSFER_FREE_TRANSFER;
int c = libusb_submit_transfer(transfer_in);
```

```
out_timestamp();
cout << "Intento_recibir_datos_ " << c << endl;
cout << "Tengo_ " << pktSize << " _como_tamano_de_paquete" << endl;
}

void USBChecker::receive_cb(struct libusb_transfer* transfer){
    out_timestamp();
    cout << "Estoy_en_el_callback_de_recepcion" << endl;
    cout << transfer->status << endl;
    switch(transfer->status)
    {
        case LIBUSB_TRANSFER_COMPLETED:
            actual_length = transfer->actual_length;
            for(int i; i < 512; i++)
            {
                if(i %16 == 15)
                    cout << transfer->buffer[i] << endl;
                else cout << transfer->buffer[i] << '\t';
            }
            to_check = 1;
            libusb_free_transfer(transfer);
            break;
        case LIBUSB_TRANSFER_ERROR:
            break;
        case LIBUSB_TRANSFER_TIMED_OUT:
            err_timestamp();
            cerr << "El_dispositivo_no_responde." << endl;
            to_exit = 0;
            //timers libres. no ocurriria
            break;
        case LIBUSB_TRANSFER_CANCELLED:
            //no prgramado para que ocurra
            break;
        case LIBUSB_TRANSFER_STALL:
            //no programado para que ocurra
```

```
        break;
    case LIBUSB_TRANSFER_NO_DEVICE:
        //si se desconecta el dispositivo
        err_timestamp();
        cerr << "El dispositivo se desconecto. Se cierra el programa." << endl;
        to_exit = 0;
        return;
    case LIBUSB_TRANSFER_OVERFLOW:
        err_timestamp();
        cerr << "Falló por overflow. Revisar código de generación de datos." << endl;
        to_exit = 0;
        break;
    default:
        break;
}
}

void out_timestamp(void){
    time_t timer = time(NULL);
    tm *loctimer = localtime(&timer);
    cout << "-----" << endl;
    cout << loctimer->tm_year << "_";
    cout << loctimer->tm_mon << "_";
    cout << loctimer->tm_mday << "_";
    cout << loctimer->tm_hour << ":" ;
    cout << loctimer->tm_min << ":" ;
    cout << loctimer->tm_sec << endl;
    cout << "-----" << endl;
}

void err_timestamp(void){
    time_t timer = time(NULL);
    tm* loctimer = localtime(&timer);
    cerr << loctimer->tm_year << "_";
    cerr << loctimer->tm_mon << "_" ;
```

```
cerr << loctimer->tm_mday << " ";
cerr << loctimer->tm_hour << ":" ;
cerr << loctimer->tm_min << ":" ;
cerr << loctimer->tm_sec << '\t';
}

int main() {
    time_t pre_timer;
    time_t pos_timer;

    struct timeval tv;
    libusb_context* ctx = NULL;
    libusb_device* dev = NULL;
    libusb_device_handle* handler = NULL;

    static unsigned char buffer_in[MAX_IN_DATA];
    static unsigned char buffer_out[MAX_OUT_DATA];
    vector<err> errs;

    struct sigaction sigIntHandler;

    sigIntHandler.sa_handler = close_myusb_device;
    sigemptyset(&sigIntHandler.sa_mask);
    sigIntHandler.sa_flags = 0;

    sigaction(SIGKILL, &sigIntHandler, NULL);
    sigaction(SIGTERM, &sigIntHandler, NULL);
    sigaction(SIGSTOP, &sigIntHandler, NULL);
    sigaction(SIGQUIT, &sigIntHandler, NULL);

    freopen("errlog","a", stderr); // redirijo el stderr a un archivo de registro de errores
                                    // la funcion freopen
                                    // mismo archivo

    freopen("datalog","a", stdout);
```

```
USBChecker checker(ctx, dev, handler);
// init the device
try
{
    checker.init_myusb_device();
}
catch(int e)
{
    err_timestamp();
    switch(e)
    {
        case NO_DOI:// 901 //No device of interest
            cerr << "No se encontro dispositivo... Conecte la placa Cypress" << endl;
            break;
        case NO_PROGR:// 902 //No programmed Cypress devices
            cerr << "La placa Cypress no se encuentra programada" << endl;
            break;
        case DRIVER_BLOCKED:// 903 //Driver couldn't be detach
            cerr << "No se puede alcanzar control de la placa" << endl;
            break;
        case NO_LIBUSB: // 904 // LIBUSB is not working
            cerr << "Nada que hacer... Final del programa" << endl;
            fclose(stderr); // libero stderr
            fclose(stdout); // libero stdout
            return(904);
    }
    to_exit = 0;
}
// init finished

while(actual_length != 0)
{
    checker.receiveData(buffer_out);
    tv.tv_sec = 0;
```

```
tv.tv_usec = 10000;
libusb_handle_events_timeout_completed(ctx,&tv,NULL);
}

to_check = 0;

//ASYNC Transfer
while(to_exit)
{
    if(to_send) //send_flag;
    {
        cout << "Entre aqui" << endl;
        checker.dataGenerator(buffer_out);
        checker.sendData(buffer_out);
        to_send = 0;
    }
    if(to_receive)
    {
        checker.receiveData(buffer_in);
        to_receive = 0;
    }
    if(to_check)
    {
        checker.dataChecker(buffer_in, &errs);
        to_check = 0;
    }
    if(to_log)
    {
        out_timestamp();
        cout << "OUT_TRANSFER:" << endl;
        for(uint i = 0; i < sizeof(buffer_out); i++)
        {
            if(i%16 != 15)
                cout <<(int) buffer_out[i] << '\t';
            else
                cout << (int) buffer_out[i] << endl;
        }
    }
}
```

```

    }
    cout << "IN_TRANSFER:" << endl;
    for(uint i = 0; i < sizeof(buffer_in); i++)
    {
        if(i %16 != 15)
            cout <<(int) buffer_in[i] << '\t';
        else
            cout <<(int) buffer_in[i] << endl;
    }
    if(errs.size() == 0)
        cout << "No_hubo_errores" << endl;
    else
    {
        cout << "Errores:" << endl;
        uint j = 0;
        for(uint i = 0; i < errs.size(); i++)
        {
            if(errs.at(i).row != 0)
            {
                cout << "At_row:" << errs.at(i).row << endl;
                j++;
            }
            else
                cout << "In_block:" << errs.at(i).block << endl;
        }
        cout << "Tasa_de_error=" << errs.size()/sizeof(buffer_out);
    }
    to_send = 1;
    cout << "le_dije_que_haga_mas_envio_de_datos" << endl;
    to_log = 0;
    to_exit = 1;
}
tv.tv_sec = 0;
tv.tv_usec = 10000;

```

```
    libusb_handle_events_timeout_completed(ctx,&tv,NULL);
}

// deinit begins
// Release the device by using libusb_release_interface
checker.closeChecker();
//libusb_release_interface(handler,0);

// Close the device you opened before, by using libusb_close
libusb_close(handler); //cierro el dispositivo

// Close the session by using libusb_exit*/
libusb_exit(ctx); // desconfiguro el contexto y libero la memoria

fclose(stderr); // libero stderr
fclose(stdout); // libero stdout

return 0;
}

#ifndef TFUSBCHECK_H_
#define TFUSBCHECK_H_

#include <iostream>
#include <fstream>
#include <libusb-1.0/libusb.h>
#include <cstdio>
#include <csignal>
#include <cstdlib>
#include <vector>
#include <ctime>

//errores: 9 es error, xx es el identificador del error
#define NO_DOI 901 //No device of interest
```

```
#define NO_PROGR          902      //No programmed Cypress devices
#define DRIVER_BLOCKED    903      //Driver couldn't be detach
#define NO_LIBUSB          904      //LIBUSB is not working

//Cypress data
#define CY_VID             0x04b4
#define CY_FX2LP_PID       0x1003
#define OUT_EP              (LIBUSB_ENDPOINT_OUT | 8)
#define IN_EP               (LIBUSB_ENDPOINT_IN | 2)
#define INTERFACE           0

// ASCII
#define ESC                 0x1B

#define MAX_OUT_DATA       256
#define MAX_IN_DATA         512

using namespace std;

typedef struct{
    char row;
    char block;
    char check;
} err;

class USBChecker
{
private:
    struct libusb_context* ctx;
    struct libusb_device* dev;
    struct libusb_device_descriptor *dscr;
    struct libusb_device_handle* handler;
    struct libusb_transfer *transfer_out;
    struct libusb_transfer *transfer_in;
```

```
public:  
    USBChecker(void);  
    USBChecker(libusb_context*, libusb_device*, libusb_device_handle*);  
    ~USBChecker(void){}  
  
    void init_myusb_device(void) throw(int);  
    void get_usb_information(void);  
    void closeChecker(void);  
  
    void dataGenerator(unsigned char*);  
    void dataChecker(unsigned char*,vector<err>*);  
  
    void sendData(unsigned char*);  
  
    void receiveData(unsigned char*);  
  
    void static receive_cb(struct libusb_transfer*);  
    void static send_cb(struct libusb_transfer*);  
  
};  
  
void static close_myusb_device(int ignored);  
void err_timestamp(void);  
void out_timestamp(void);  
  
#endif //TFUSBCHECK.H
```