

# Temas Específicos de Electrónica Digital I

Comunicación USB 2.0 para aplicaciones científicas basadas en FPGA

Edwin Barragán

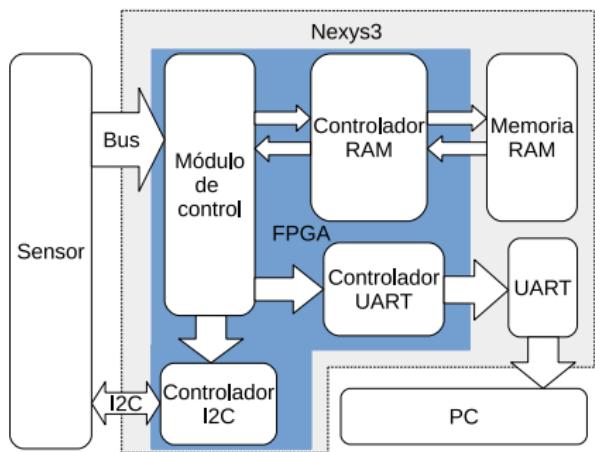
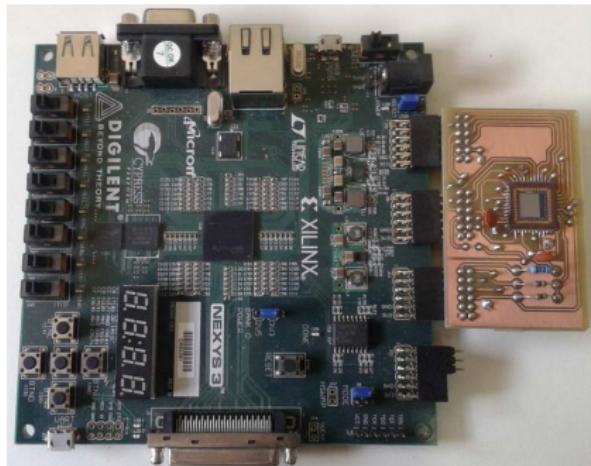
[edwin.barragan@cab.cnea.gov.ar](mailto:edwin.barragan@cab.cnea.gov.ar)

Universidad Nacional de San Juan  
Facultad de Ingeniería

1 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.

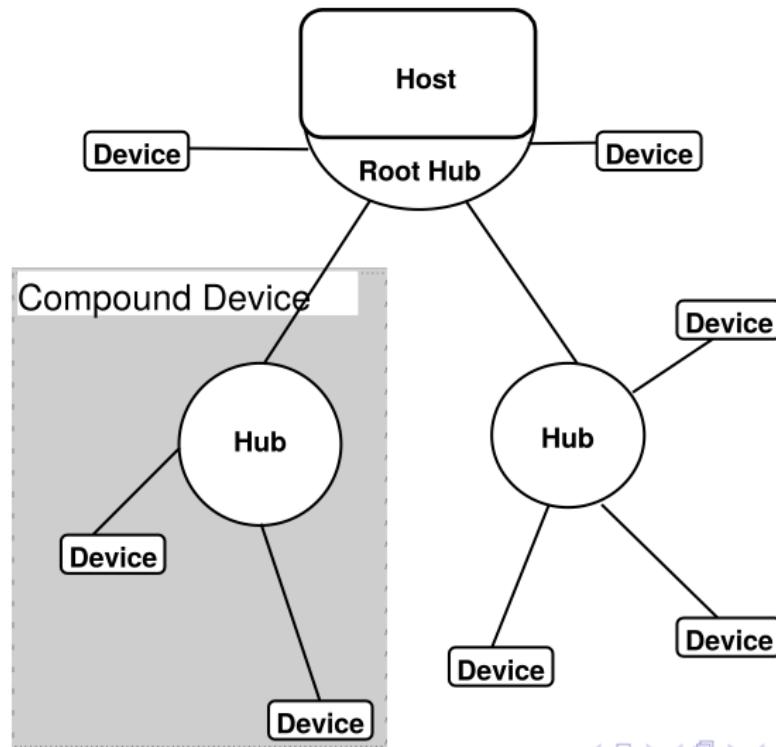
Los objetivos perseguidos por norma son:

- 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 una nueva velocidad de 480 Mbit/s.

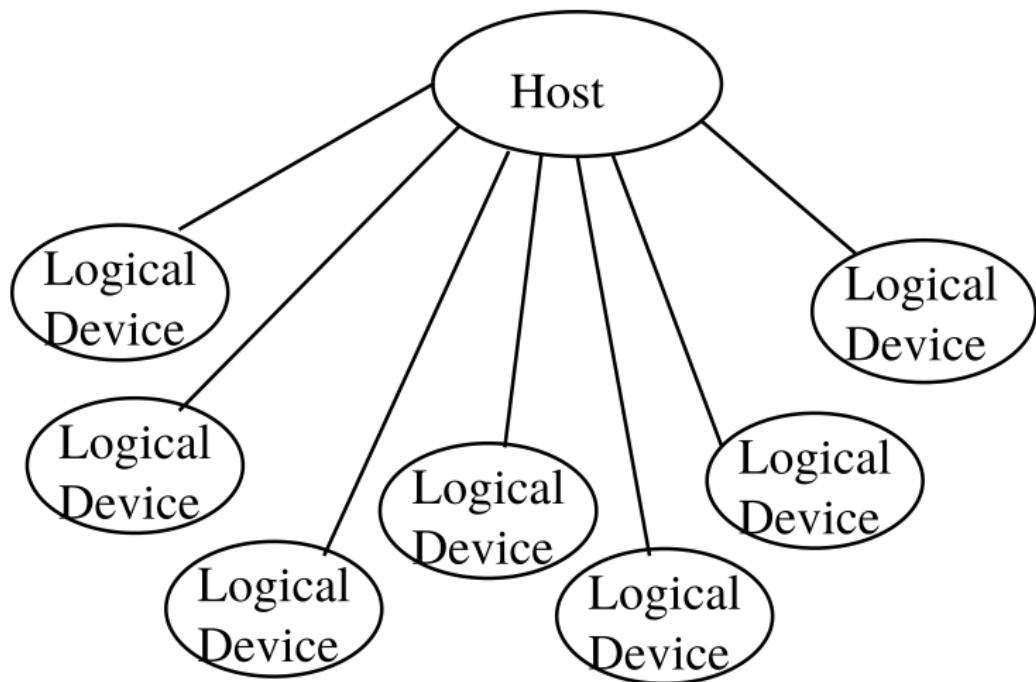
# USB - Topología

- Física

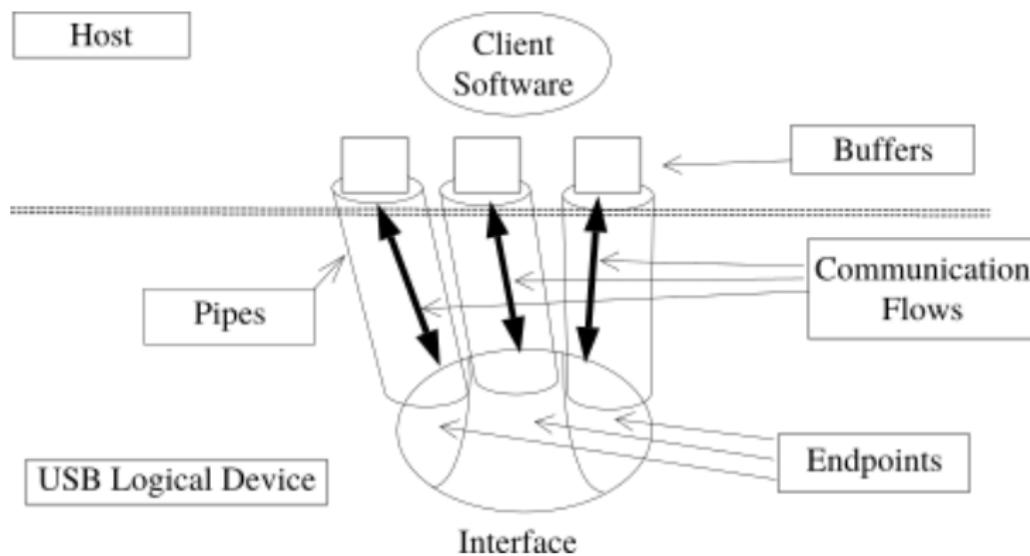


# USB - Topología

- Lógica



# USB - Flujo de datos

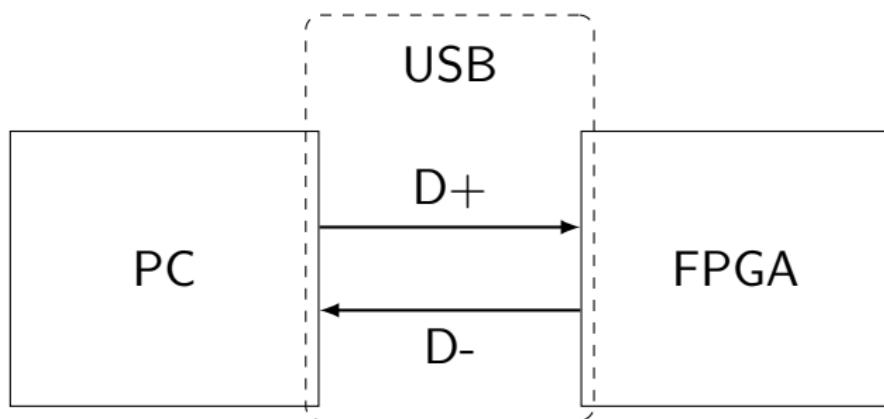


# Agenda

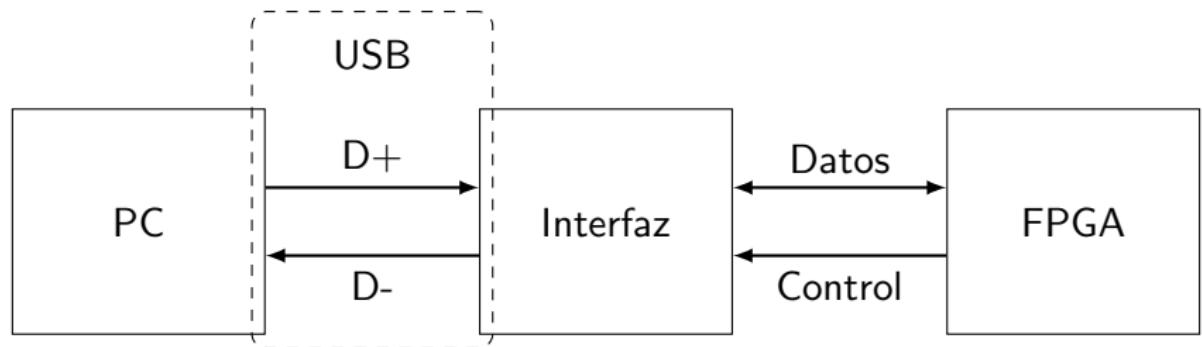
## 2 Implementación

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

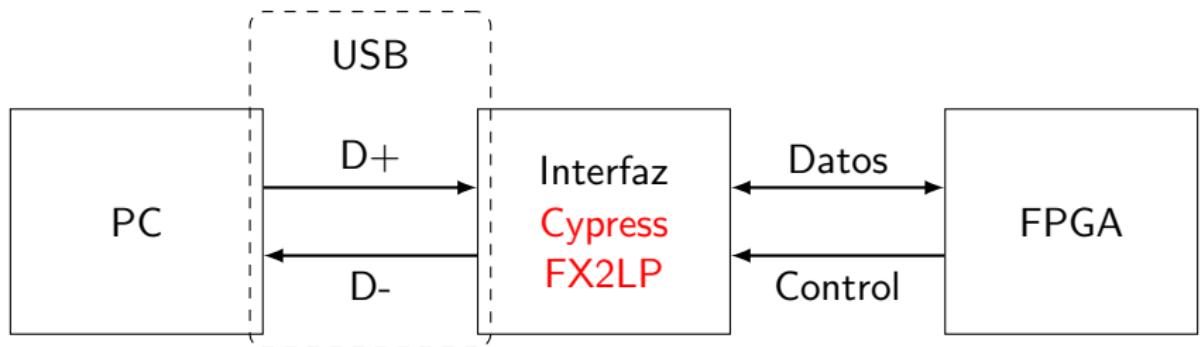
# Arquitectura del sistema realizado



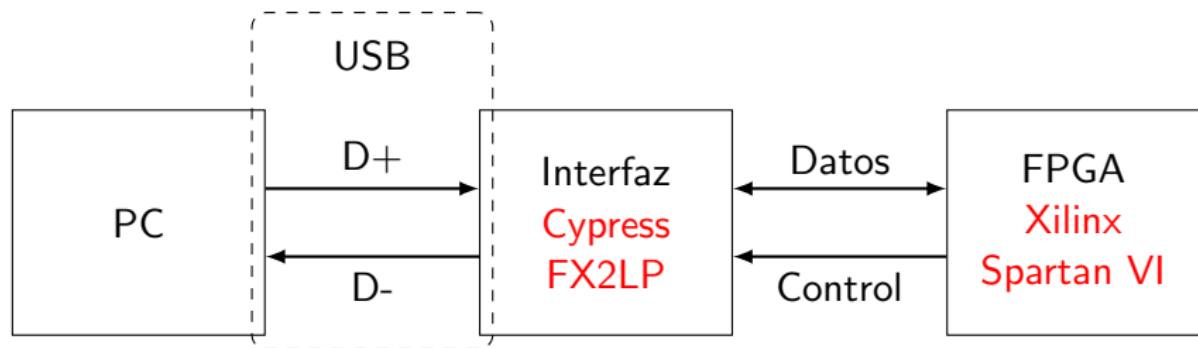
# Arquitectura del sistema realizado



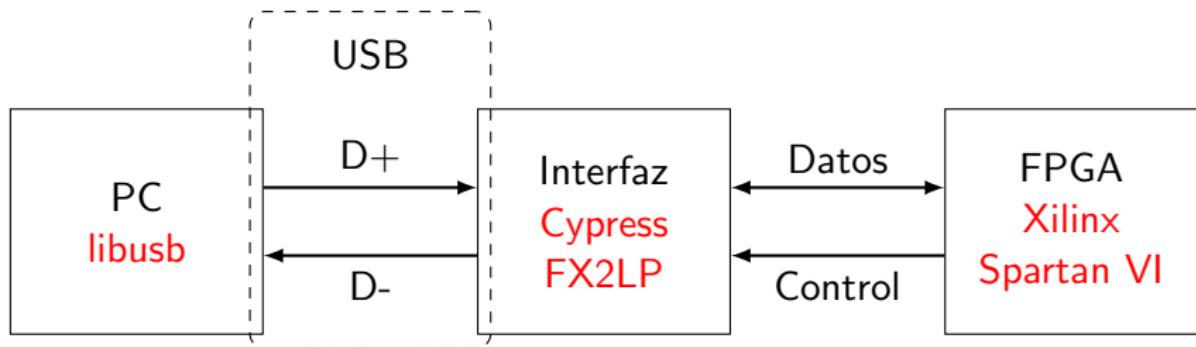
# Arquitectura del sistema realizado



# Arquitectura del sistema realizado



# Arquitectura del sistema realizado

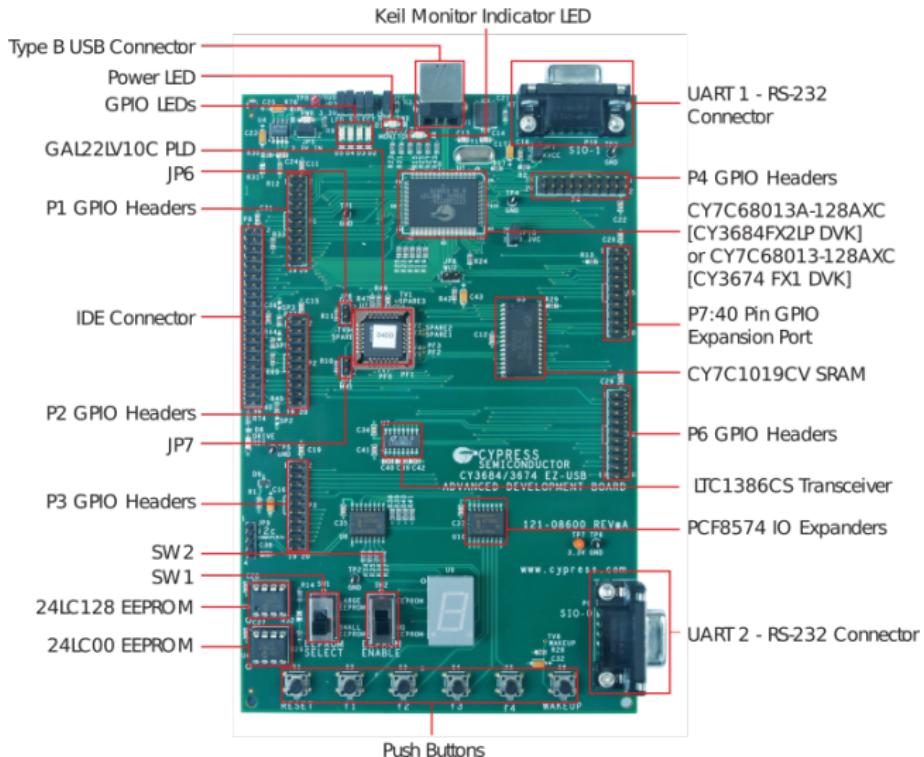


# Agenda

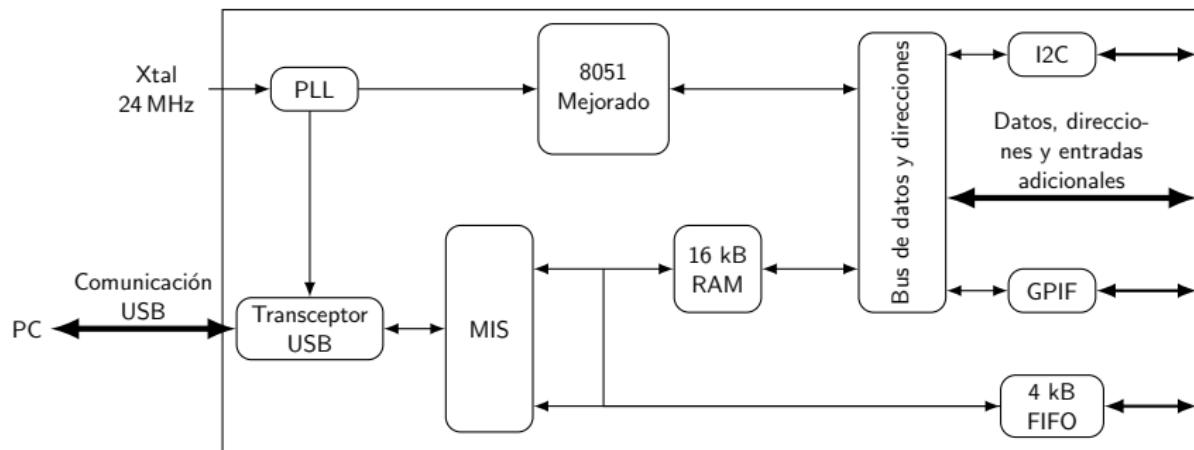
## 2 Implementación

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

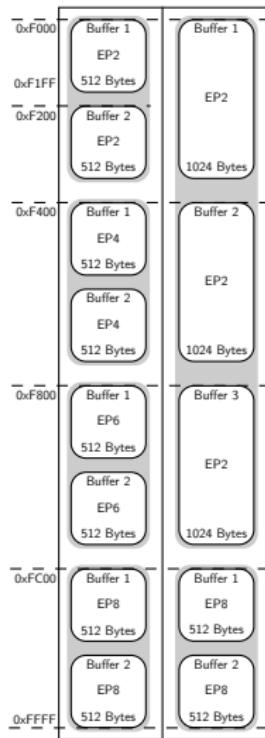
# Placa de desarrollo utilizada para la interfaz



# El circuito integrado FX2LP



# Configuración del dispositivo USB

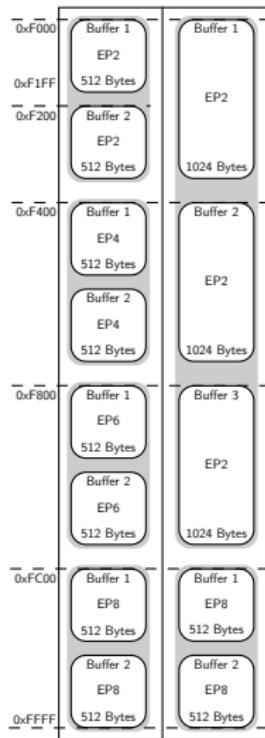


La configuración de las tuberías de comunicación se definieron con la finalidad de obtener el mayor ancho de banda posible a la entrada.

- Entrada:

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

# Configuración del dispositivo USB

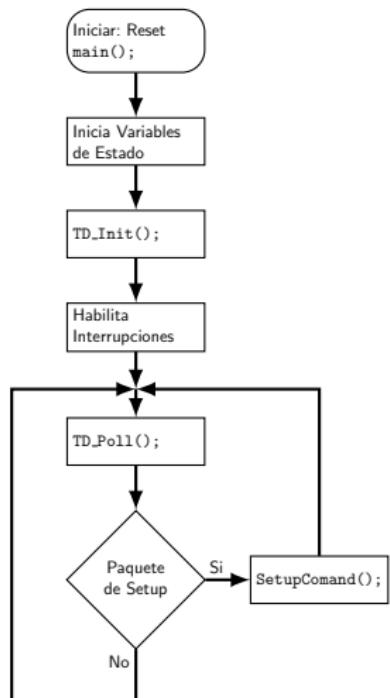


La configuración de las tuberías de comunicación se definieron con la finalidad de obtener el mayor ancho de banda posible a la entrada.

- Salida:

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

# Cypress Software Development Kit



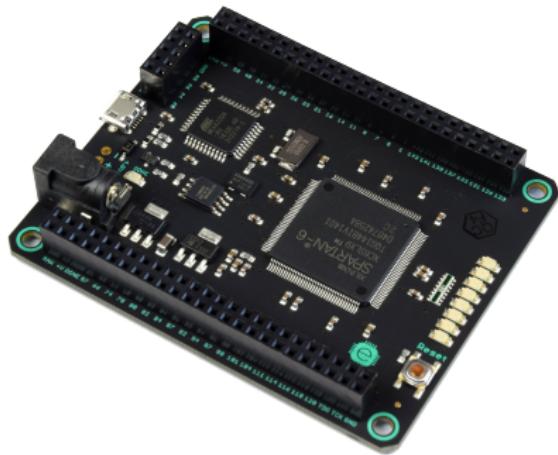
- El programa inicia al comienzo del main();
- Inicia todas las variables de estado
- TD\_Init ejecuta la configuración del usuario
- El usuario debe configurar las interrupciones que desea habilitar. Al salir de TD\_Init, el programa habilita todas (EA=1)
- Finalmente, se ejecuta las instrucciones que el usuario programa de manera específica
- Además, se ejecutan las funciones que controlan el USB.

# Agenda

## 2 Implementación

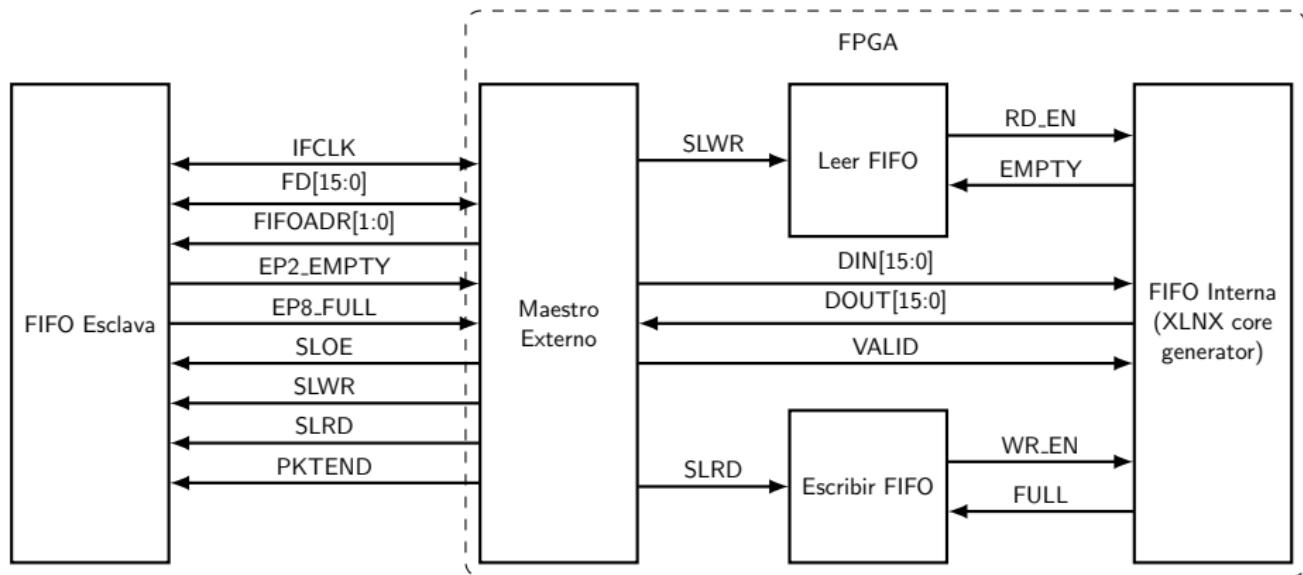
- Arquitectura del sistema
- Configuración del puente
- **Circuito sintetizado**
- Circuito de interconexión

# La placa de desarrollo MOJO v3

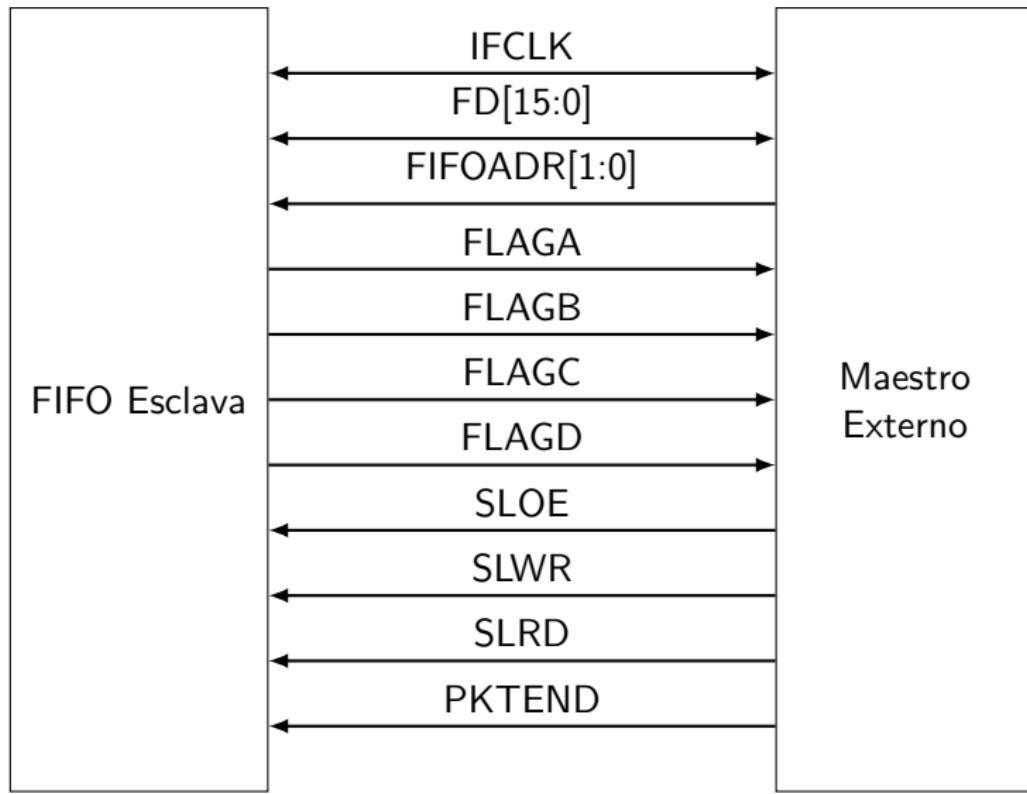


- FPGA Spartan 6 XC6SLX9 de Xilinx
- 84 pines IO digitales
- 8 entradas analógicas
- 8 LEDs de propósito general
- 1 pulsador de propósito general
- Regulador de voltaje de entrada de 4.8V - 12V
- ATmega32U4 para configurar la FPGA y leer los pines analógicos
- Bootloader compatible con Arduino
- Memoria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

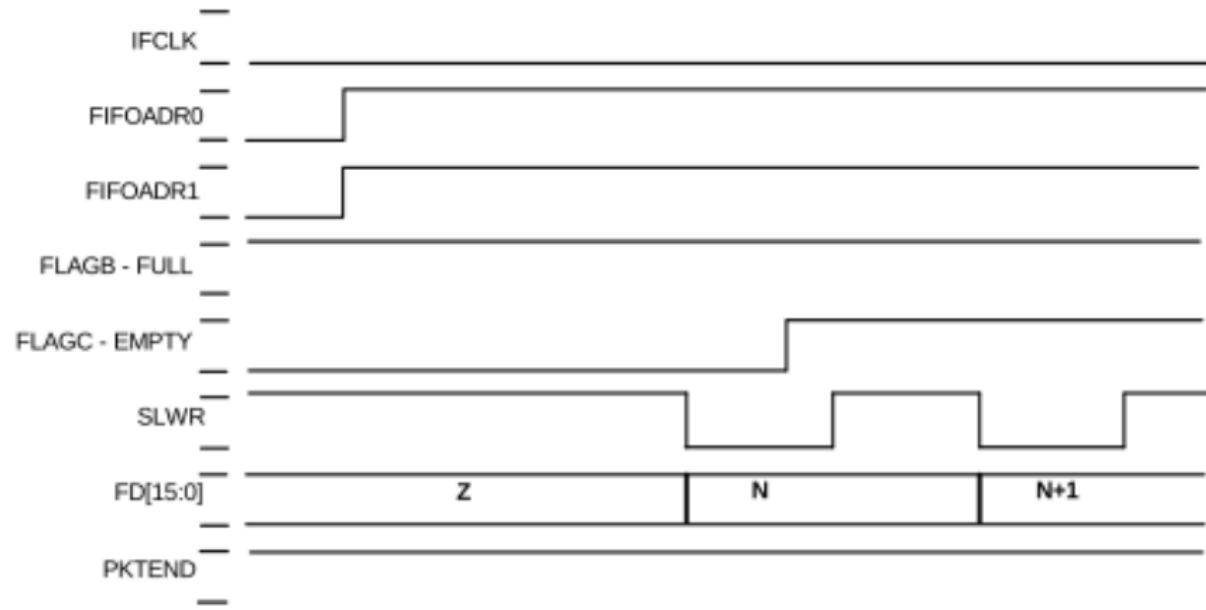
# Estructura interna FPGA



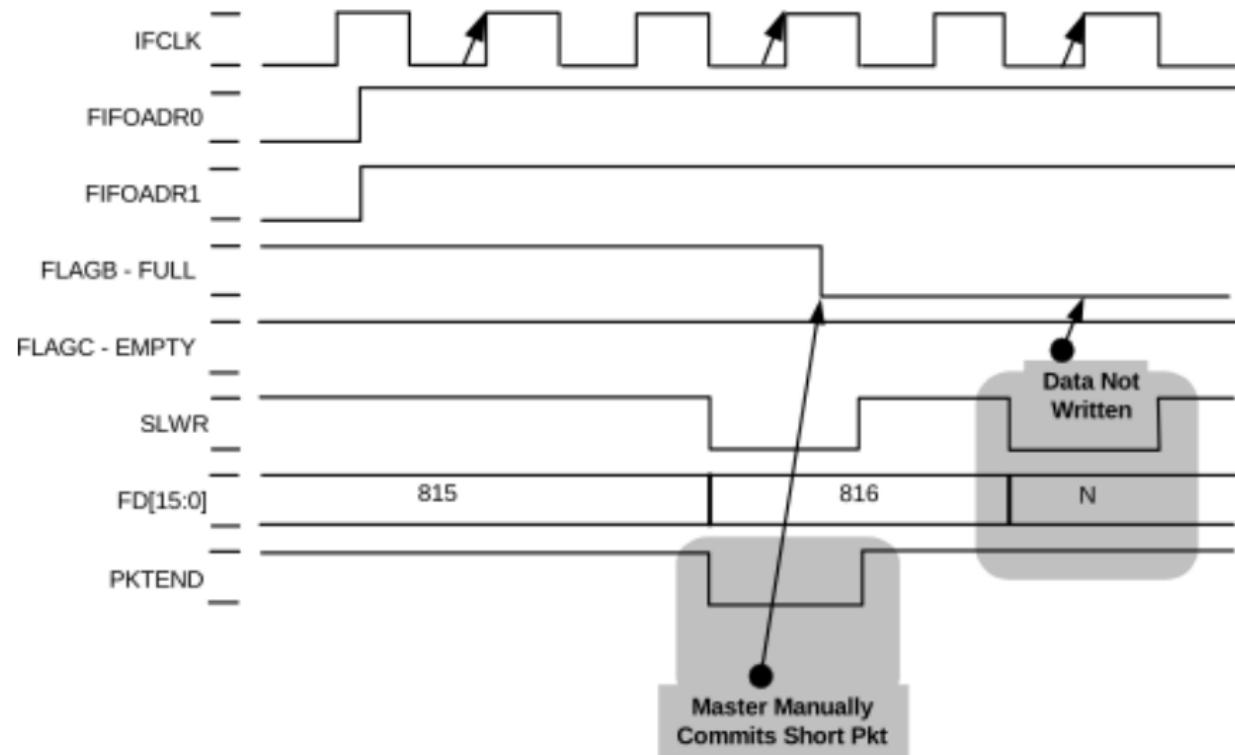
# Interfaz - FPGA



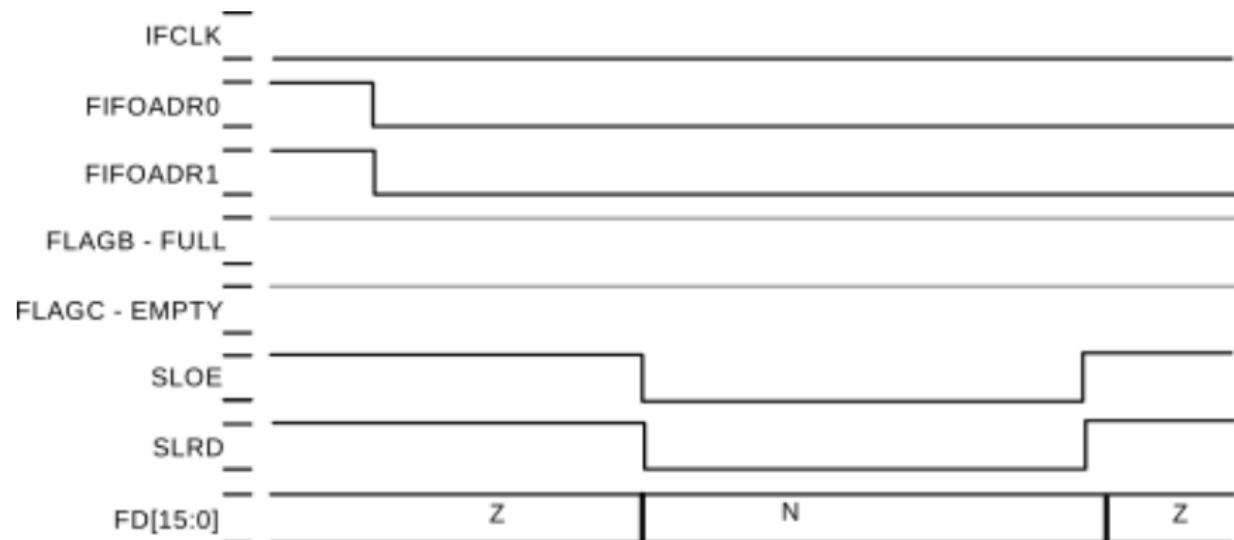
# Operaciones en la FIFO - Lectura asíncrona



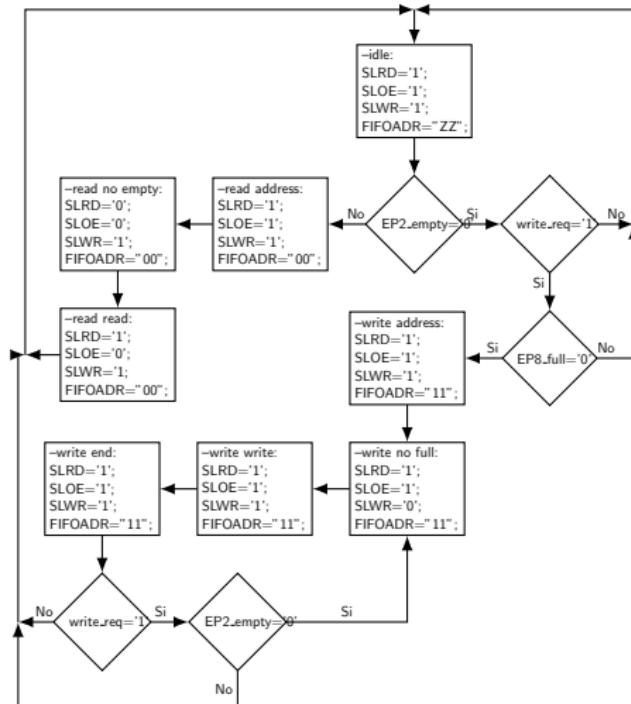
# Operaciones en la FIFO - Lectura asíncrona



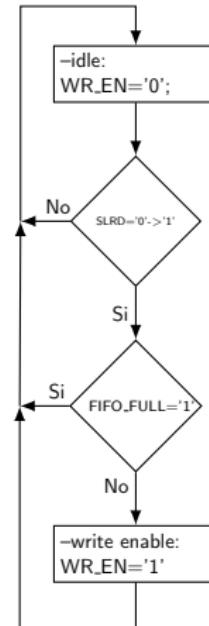
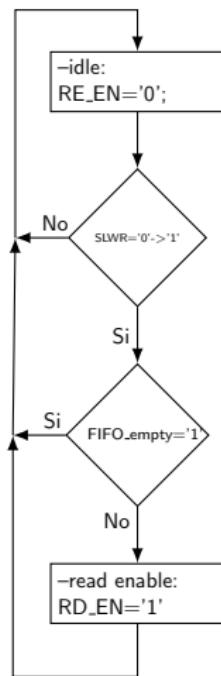
# Operaciones en la FIFO - Escritura asíncrona



# Máquina de estados algorítmica de la interfaz



# Maquinas de estado hacia la memoria FIFO interna



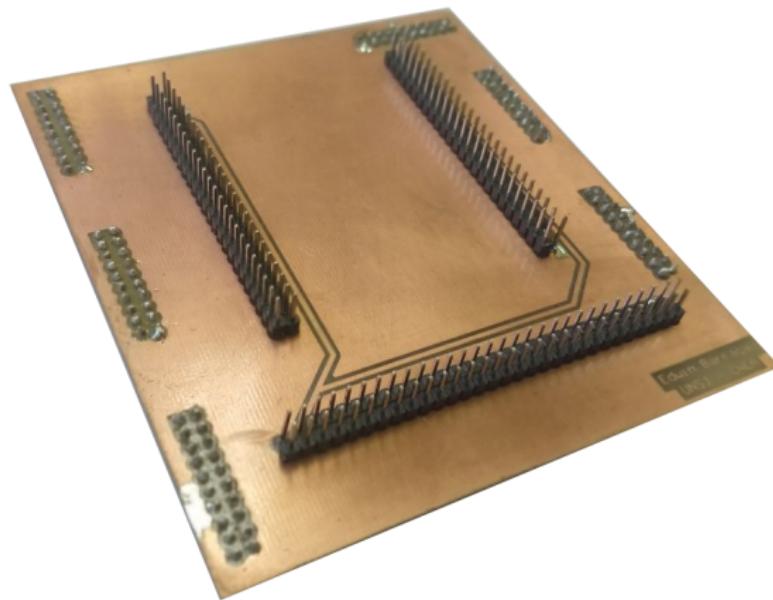
# Agenda

## 2 Implementación

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

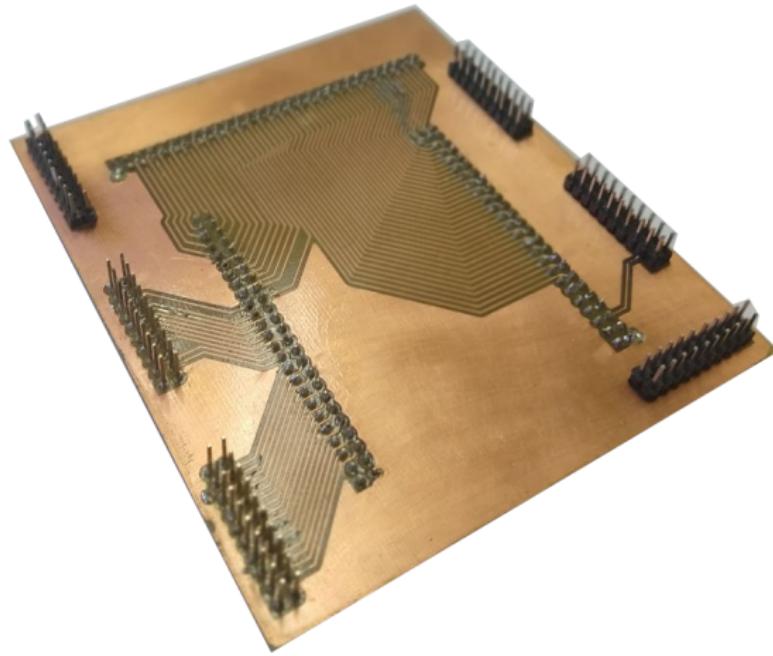
# Círculo de interconexión

- Versión 1



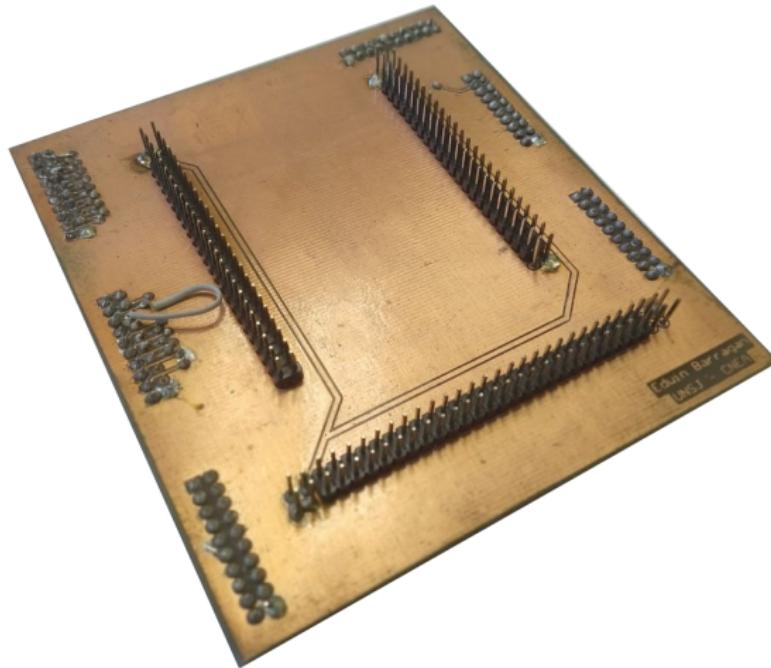
# Circuito de interconexión

- Versión 1



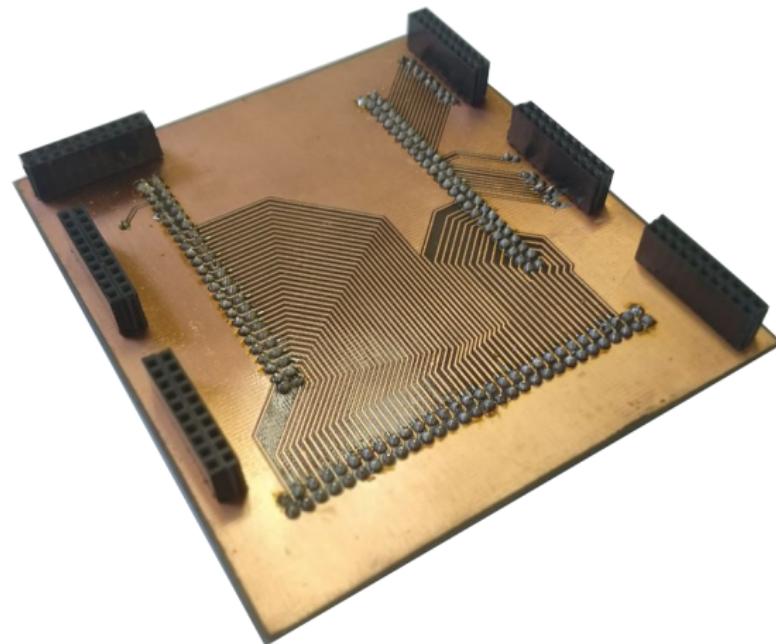
# Círculo de interconexión

- Versión 2



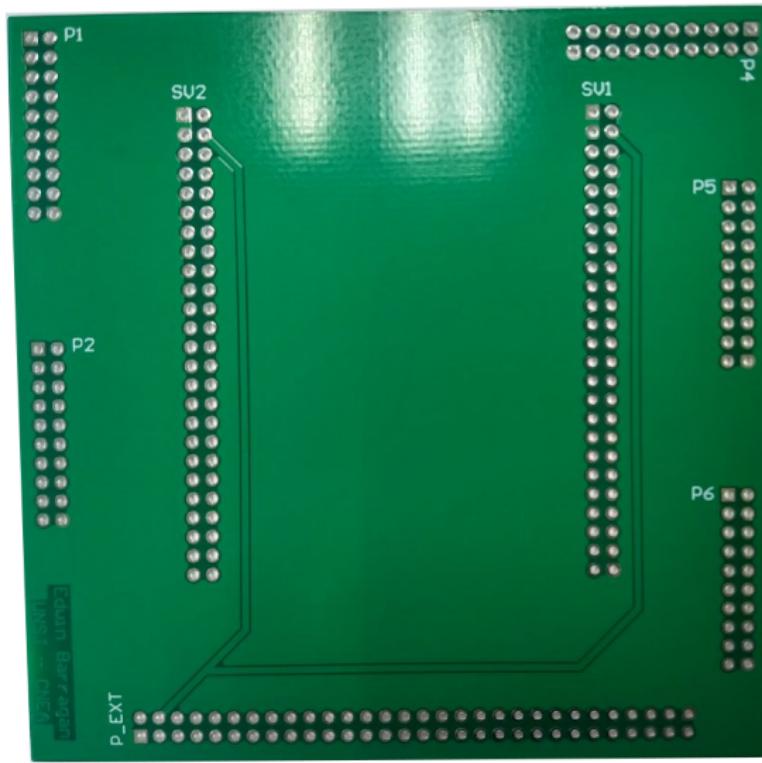
# Círculo de interconexión

- Versión 2



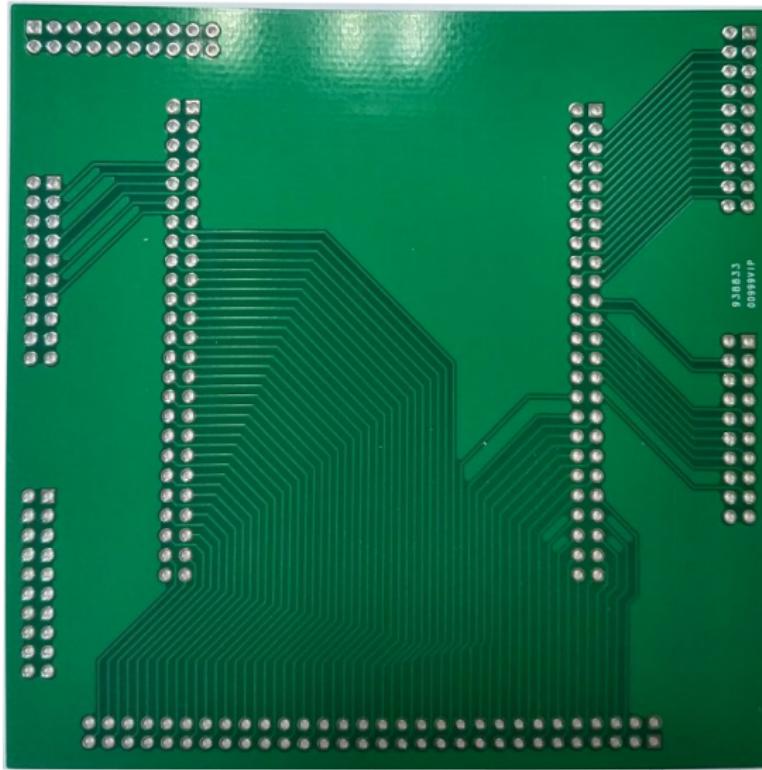
# Círculo de interconexión

- Versión 3



# Circuito de interconexión

- Versión 3



# Sistema completo



# Agenda

## 3 Evaluación y validación

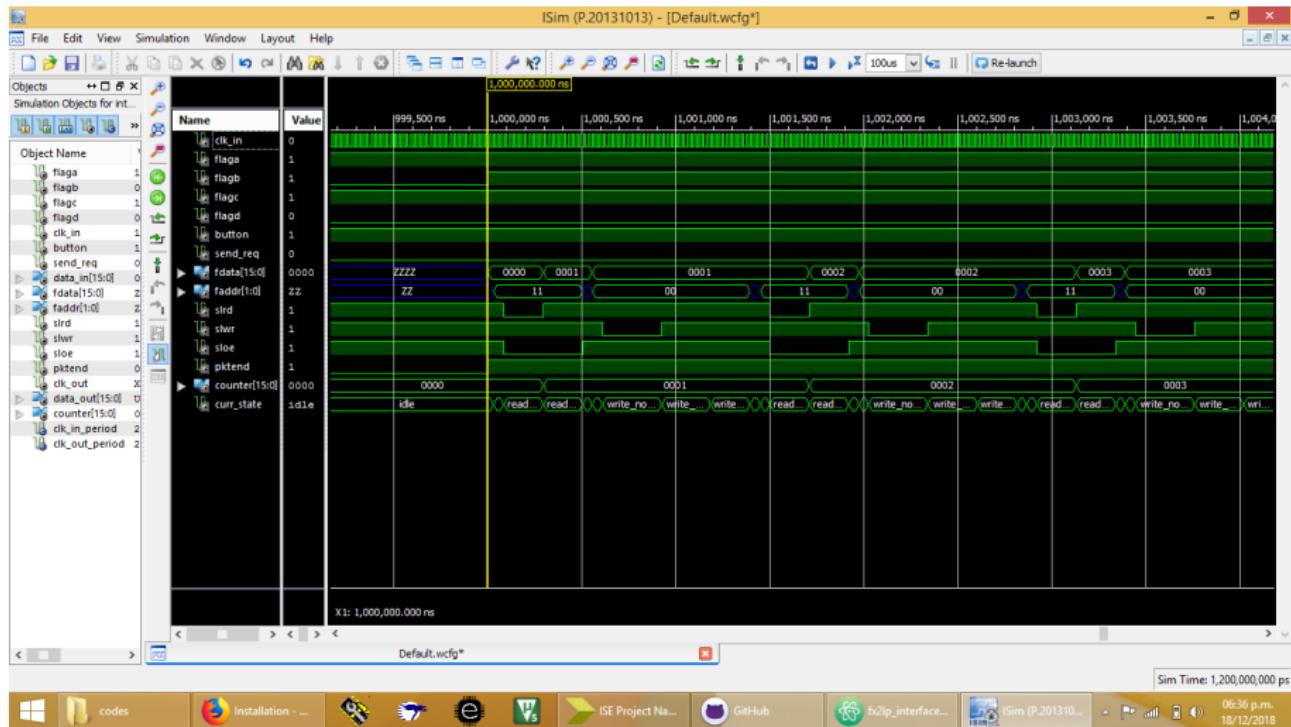
### ● Test benches de VHDL

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

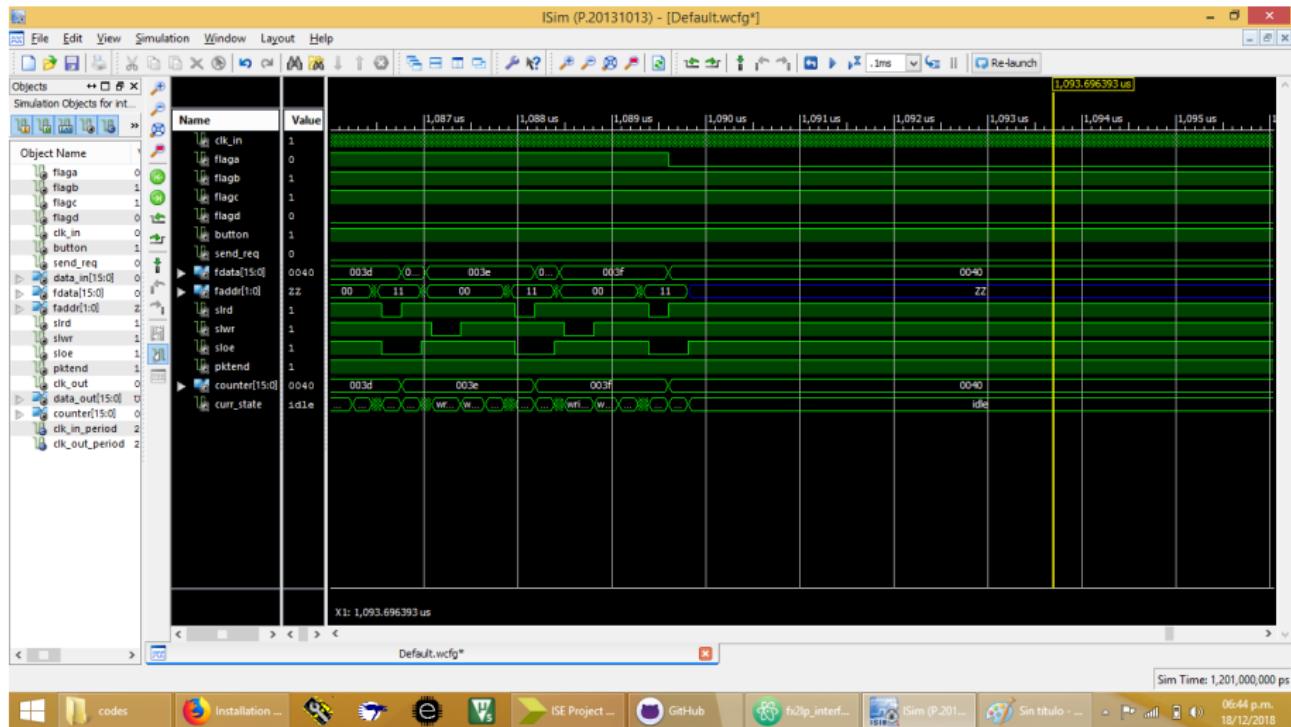
# Test bench

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.

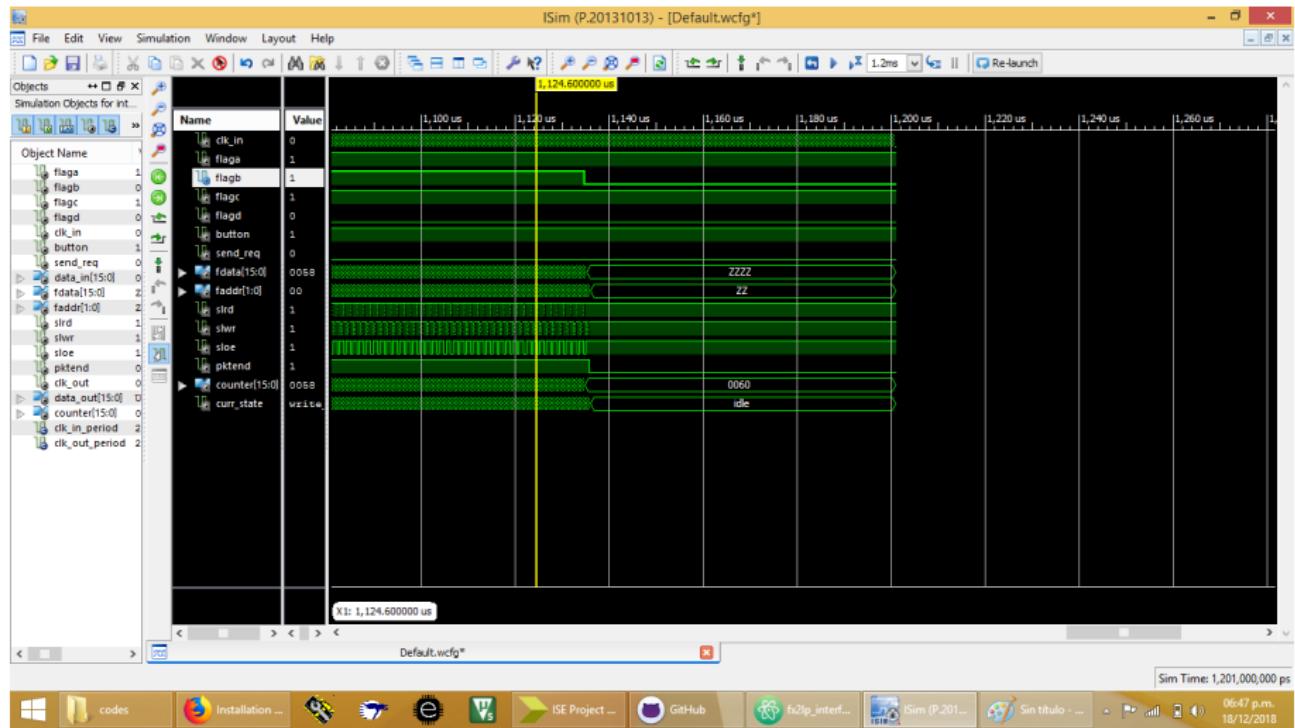
# Test bench



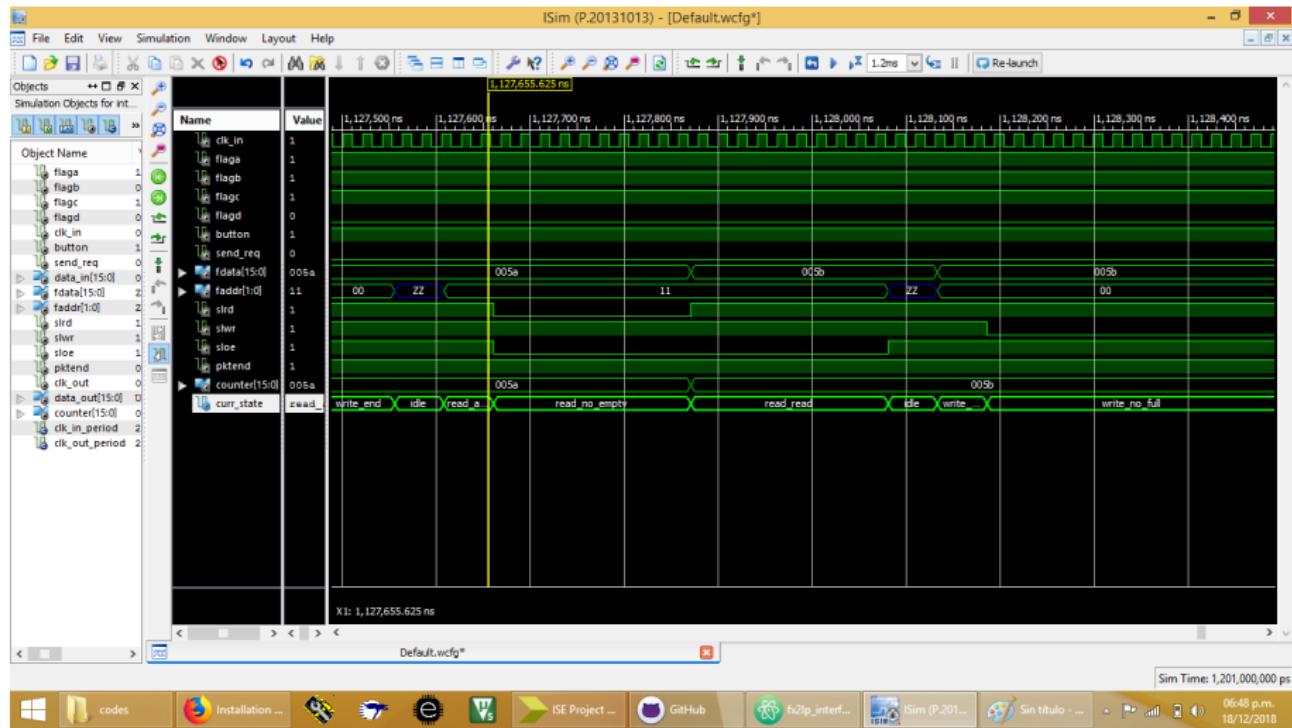
# Test bench



# Test bench



# Test bench



# 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.

# 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 aprendió sobre las técnicas y herramientas 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

```
#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;

EP8FIFO CFG = 0x00;
SYNCDELAY;
EP2FIFO CFG = 0x00;
SYNCDELAY;

//setting on auto mode. rising edge is necessary
EP8FIFO CFG = 0x11;
SYNCDELAY;
EP2FIFO CFG = 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 = pHIGHSpeedConfigDscr;  
    <img alt="Navigation icons" data-bbox="670 930 985 955"/>
```

```
((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 =
```

```
    }
    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 _vacio\n");

}

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

{}