Temas Específicos de Electrónica Digital I

Comunicación USB 2.0 para aplicaciones cientificas basadas en FPGA

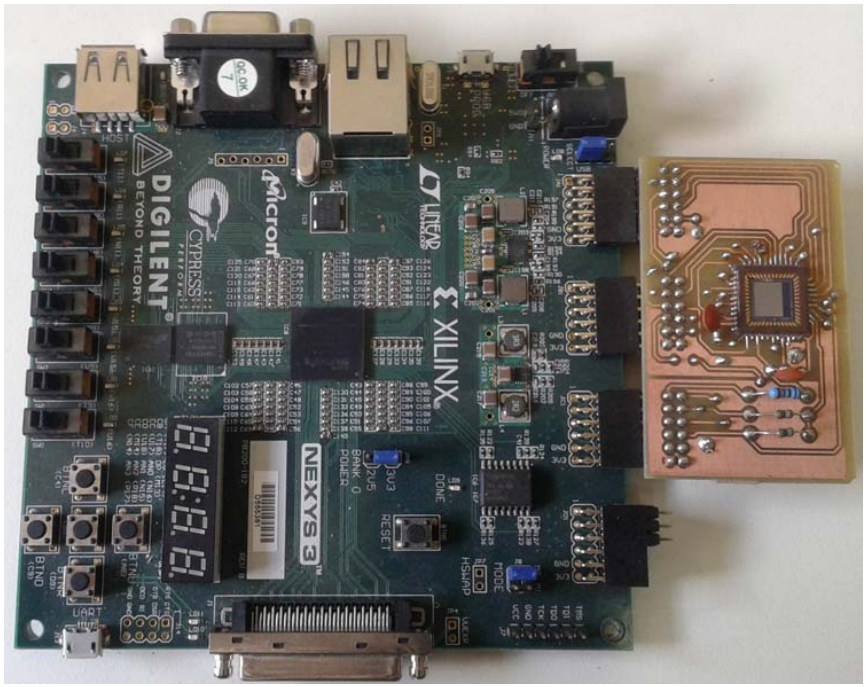
Edwin Barragán
edwin.barragan@cab.cnea.gov.ar

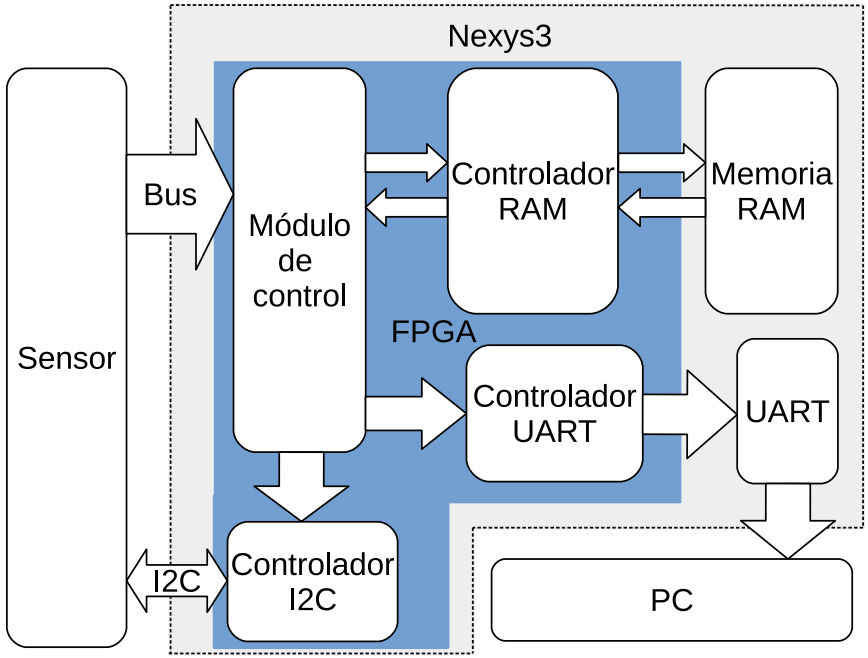
Universidad Nacional de San Juan
Facultad de Ingeniería

16 de mayo de 2019

Una comunicación USB para aplicaciones científicas basadas en FPGA

Preámbulo





Agenda

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

4. Resultados y conclusiones
   - Robustez
   - Tasa máxima de Transferencia
   - Trabajo futuro

Agenda

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.
- La utilización de FPGA es muy útil para sintetizar circuitos digitales.
- Los datos deben ser procesados para transformase 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 FGPAs pueden operar a altas velocidades y utilizar puertos paralelos.
- Es de utilidad una comunicación entre las PCs 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

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

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

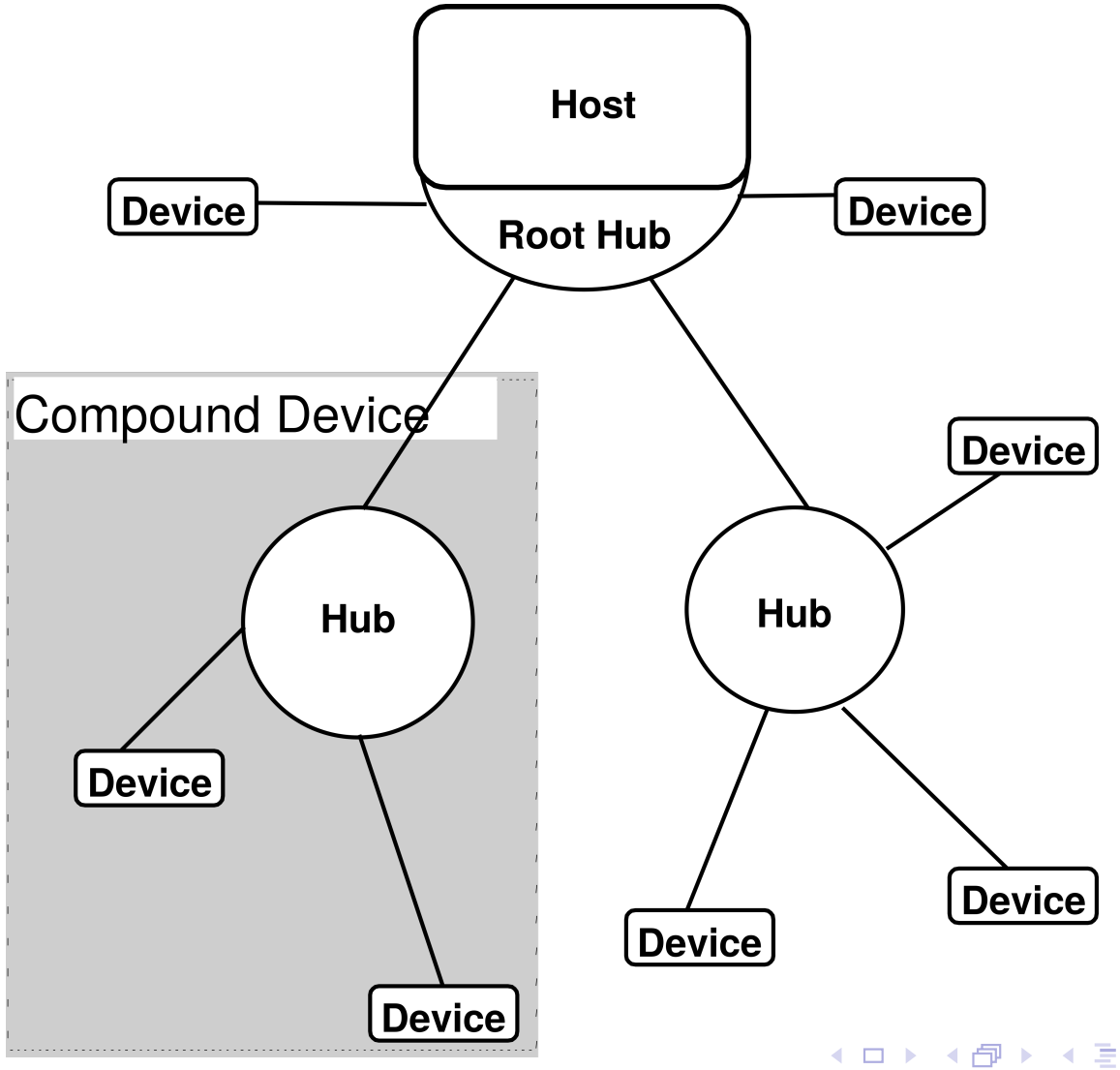
El Bus Serial Universal o USB es un sistema de comunicación pensado, en su concepción original, para conectar periféricos a una PC.
Los objetivos perseguidos por norma son:

- 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.

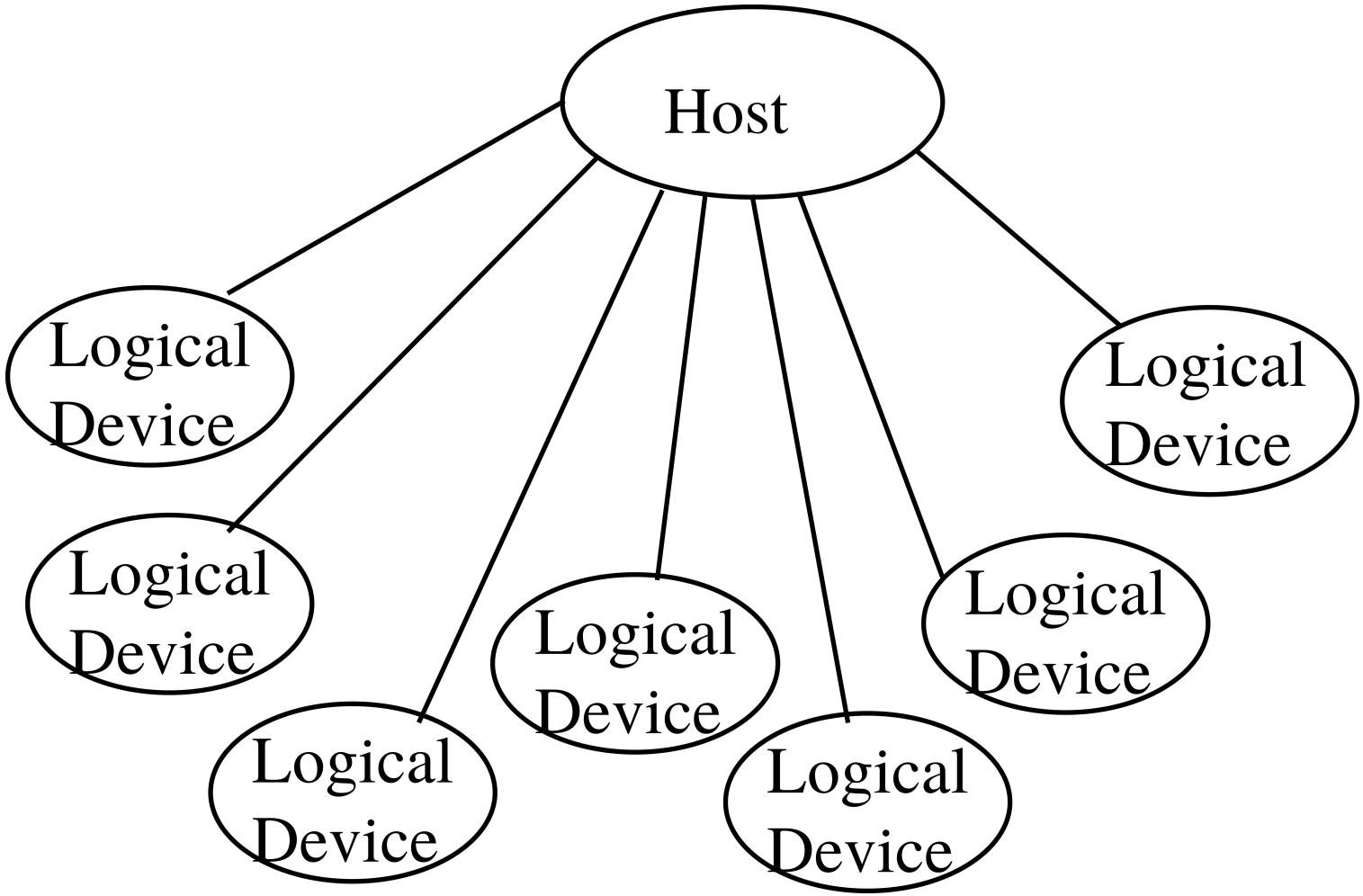
USB - Topología

- Física

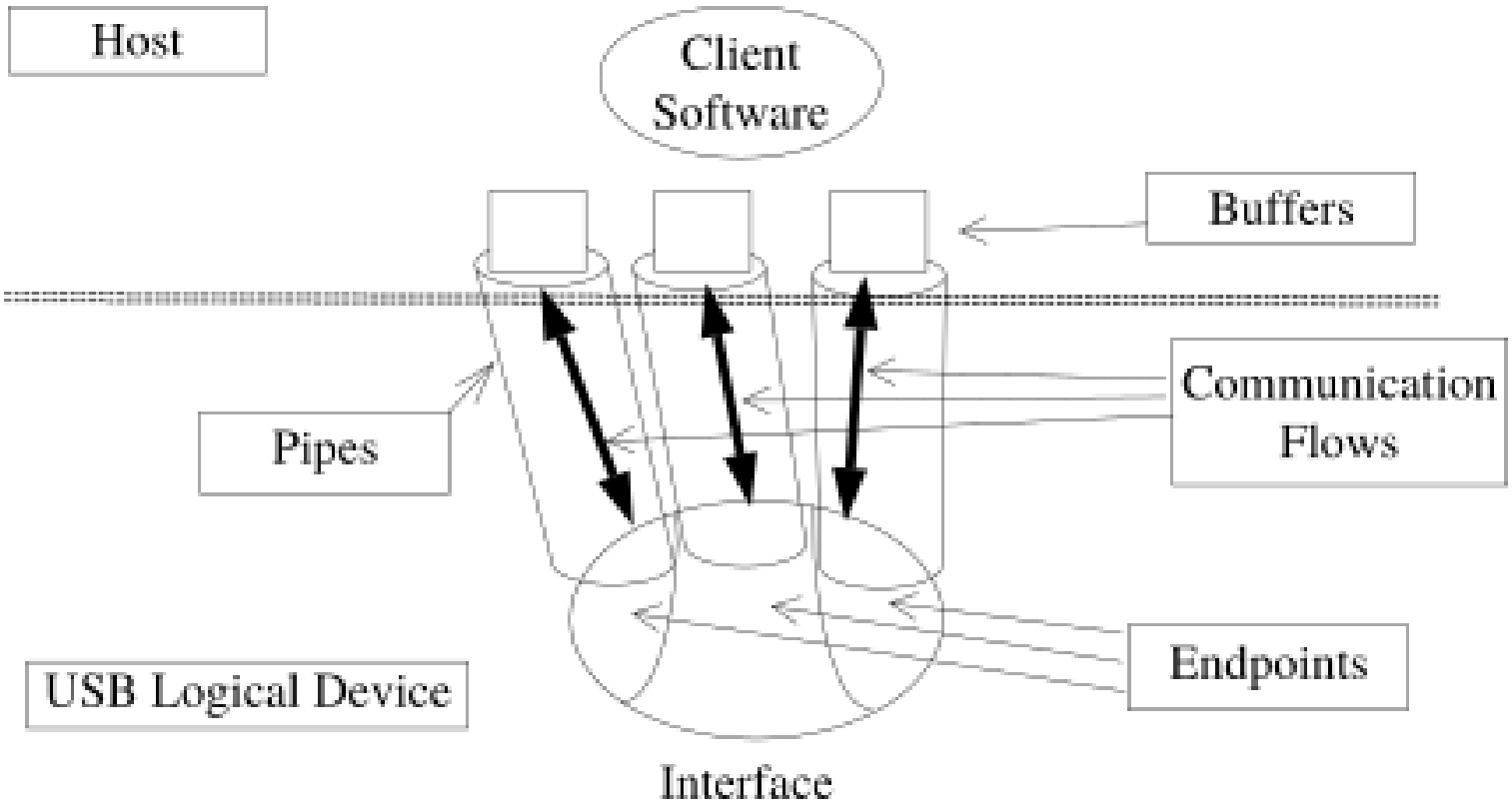


USB - Topología

- Lógica



USB - Flujo de datos

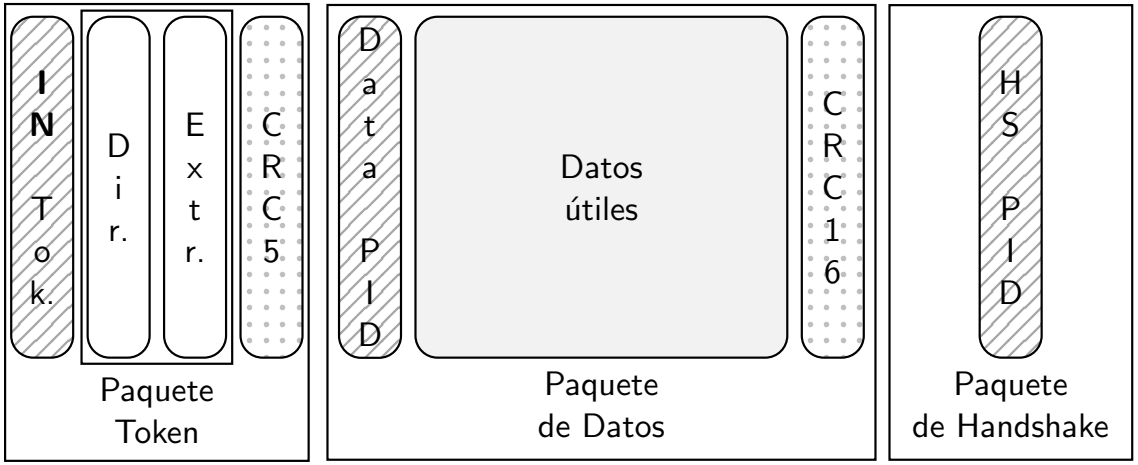


USB - Tipos de paquetes

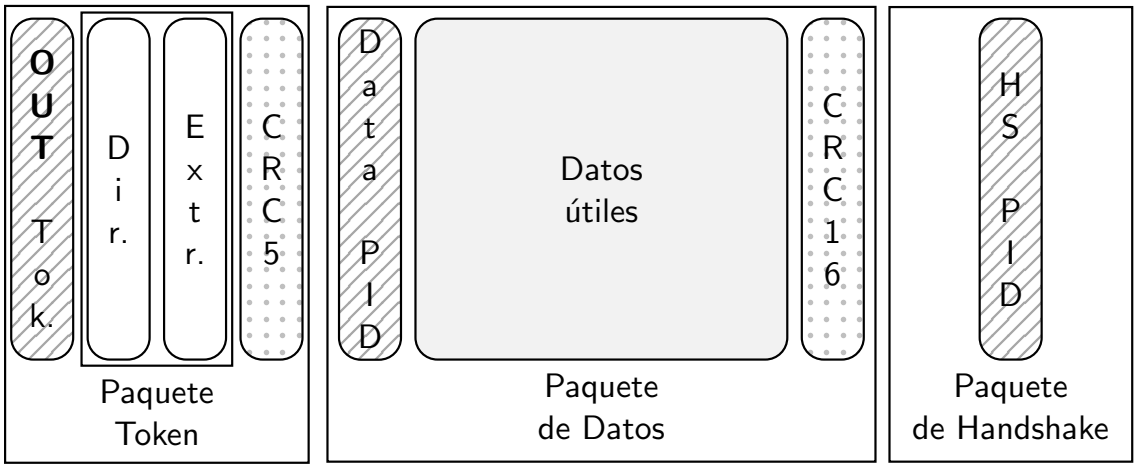
- Paquetes Token
- Paquetes de Datos
- Paquetes de Handshake
- Paquetes Especiales

USB - Trama de paquetes

- Entrada de Paquetes al Host



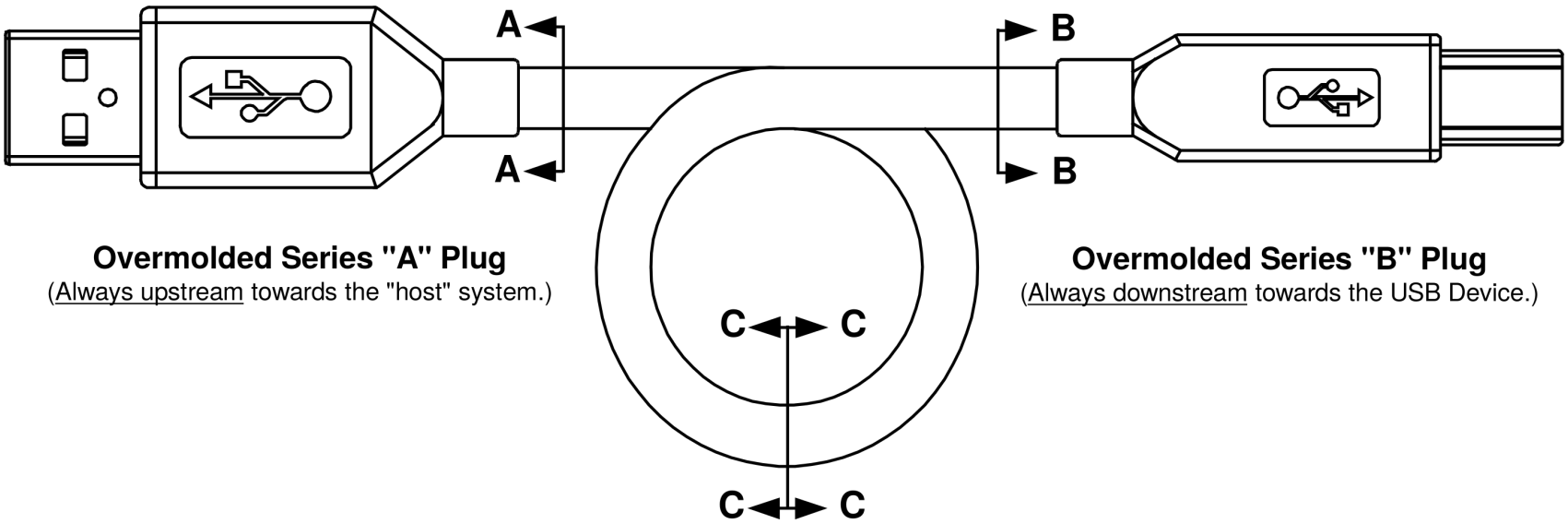
- Salida de Paquetes hacia periféricos



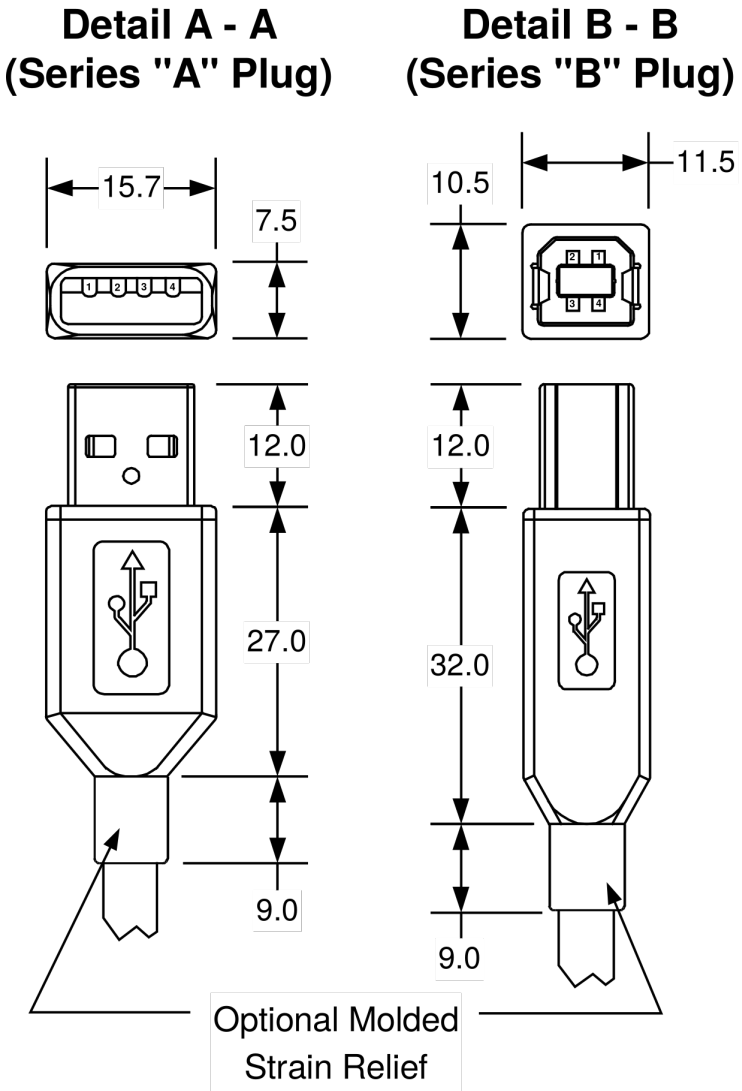
USB - Conexión mecánica

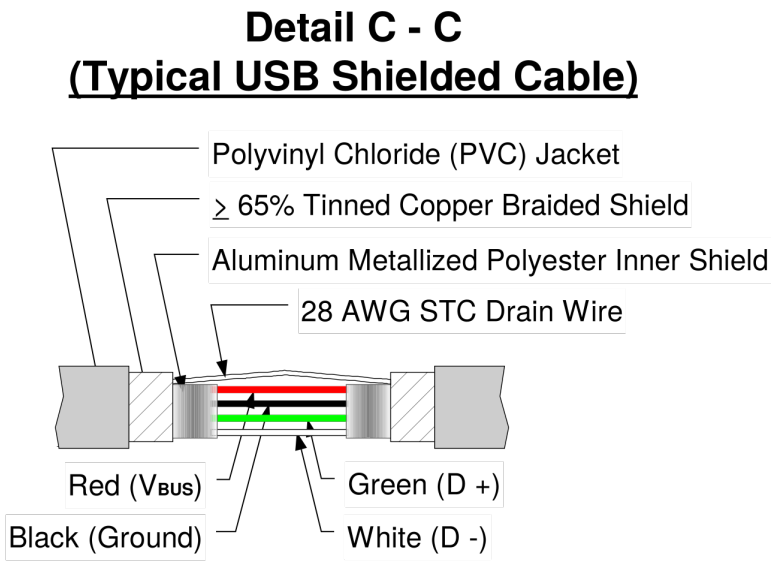
- La conexión mecánica posee dos pares de conductores: uno de alimentación (V_{BUS} y GROUND) y otro de datos diferenciales(D+ y D-). Además posee diferentes tipos de fichas de conexión.
- Los conectores poseen diferencia en los extremos, a fin de identificar el extremo que va hacia el Host (tipo A) y el extremo que va hacia los dispositivos (tipo B, mini USB, micro USB)

USB - Conexión mecánica



USB - Conexión mecánica





All dimensions are in millimeters (**mm**) unless otherwise noted.

Dimensions are **TYPICAL** and are for general reference purposes only.

USB - Tipo de transferencias

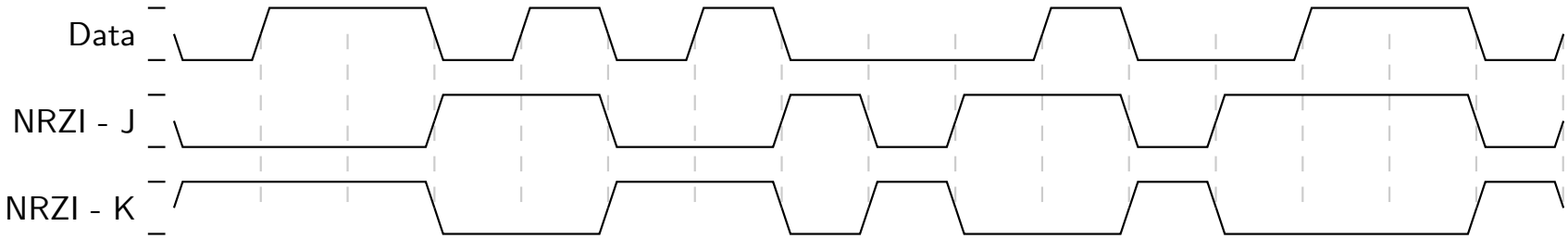
Existen 4 tipos de transferencia los cuales difieren en cómo es transmitida la información, la dirección que posee, el tamaño máximo, acceso al bus, tiempos de latencia, manejo de errores y la secuencia de requerimiento de datos

- Transferencias de Control
- Transferencias de Interrupción
- Transferencias de Bultos
- Transferencias Isocrónicas

USB - Especificaciones eléctricas

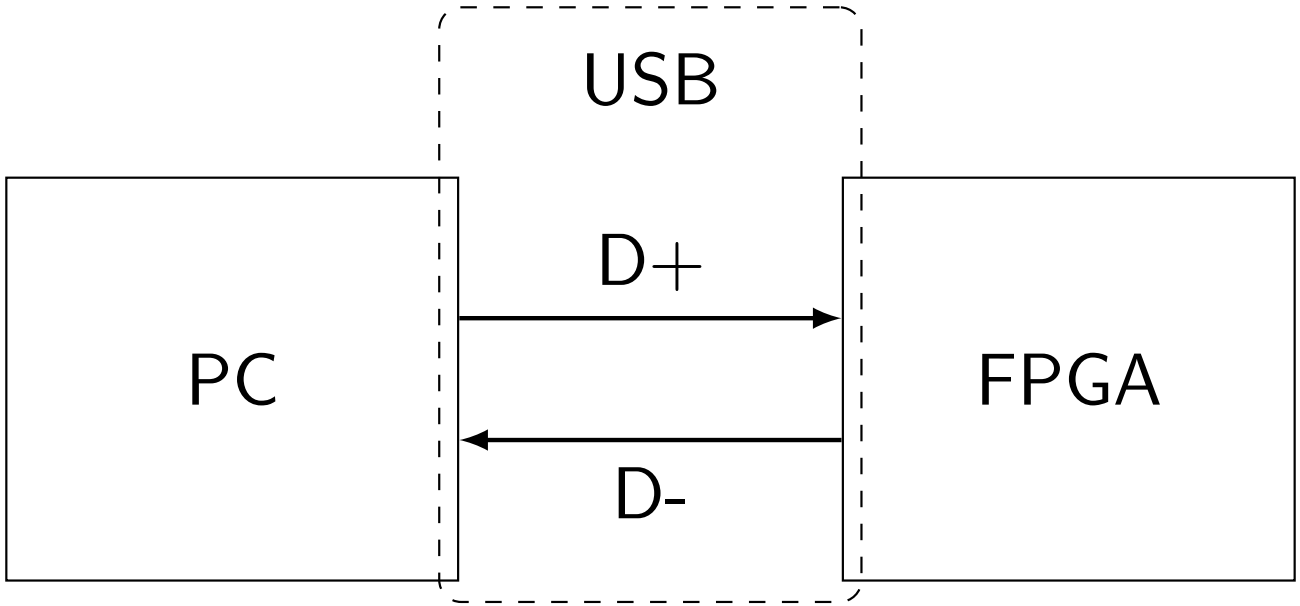
- Existen 3 velocidades de señalización posibles: 480 Mbit/s denominada high-speed, 12 Mbit/s para full-speed y 1.5 Mbit/s con low-speed.
- Se utiliza señal diferencial con un esquema de codificación NRZI (inversión de no retorno a zero).
- Los conductores de energía, V_{BUS} y GROUND poseen 5 y 0 V respectivamente.
- Los conductores de datos son diferenciales y están polarizados de forma tal que pueda ser identificada la velocidad de operación y la conexión/desconexión de dispositivos.

USB - Codificación NRZI

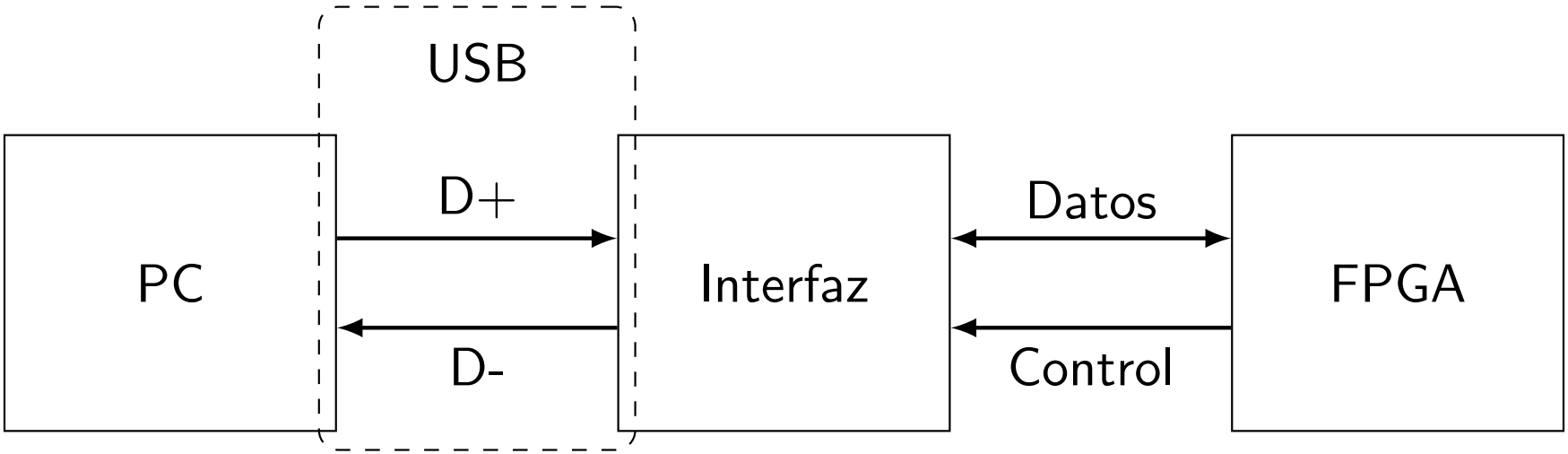


Agenda

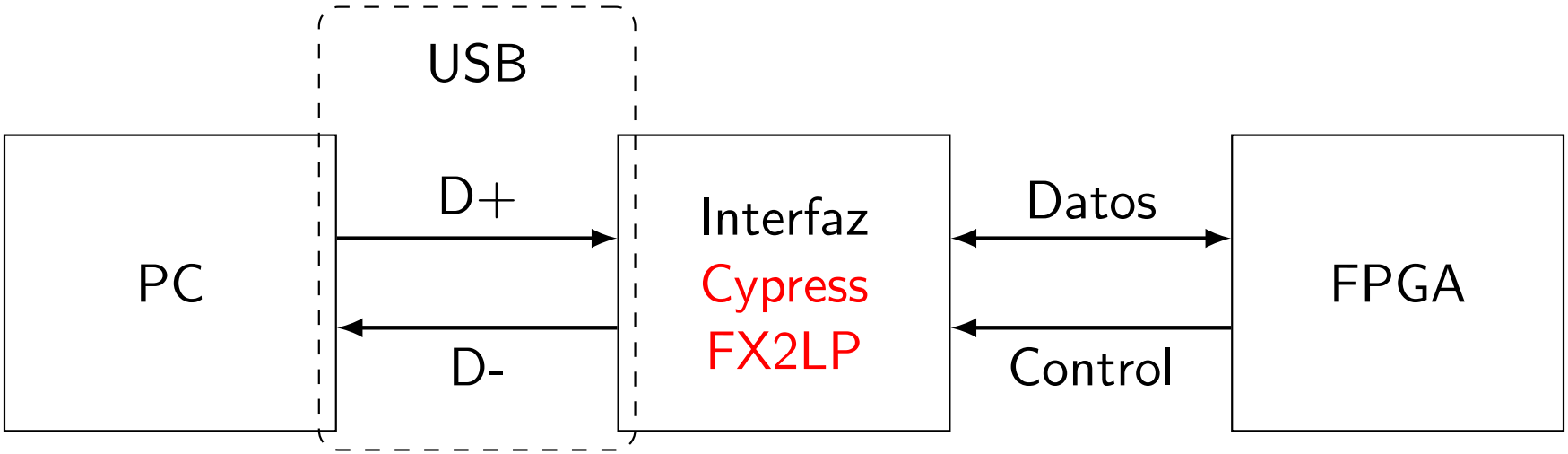
Arquitectura del sistema realizado



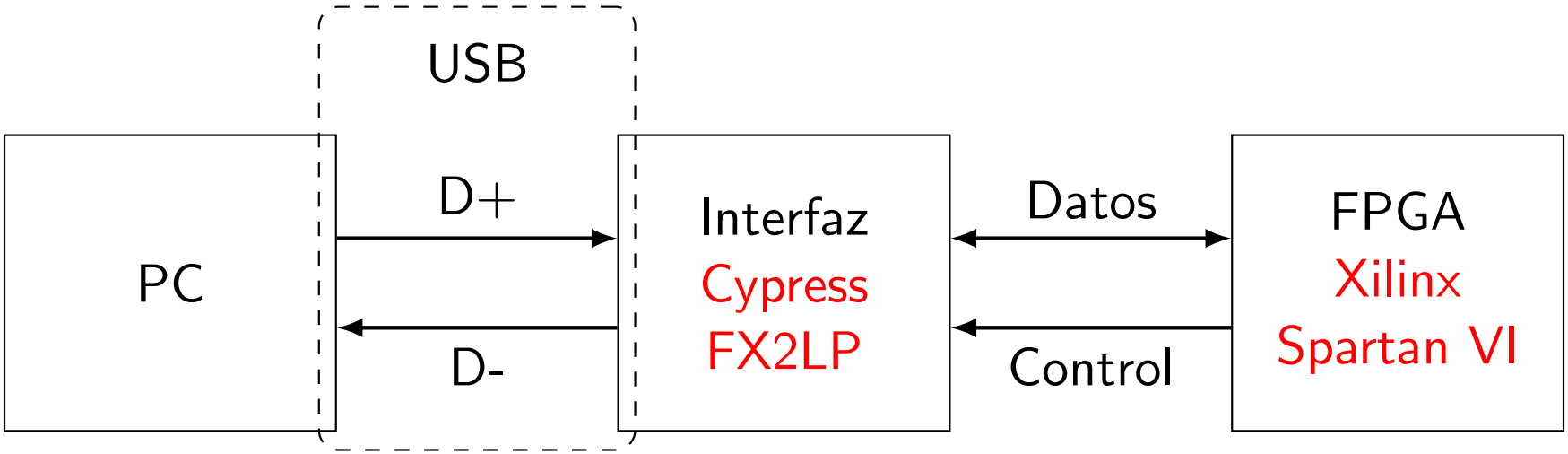
Arquitectura del sistema realizado



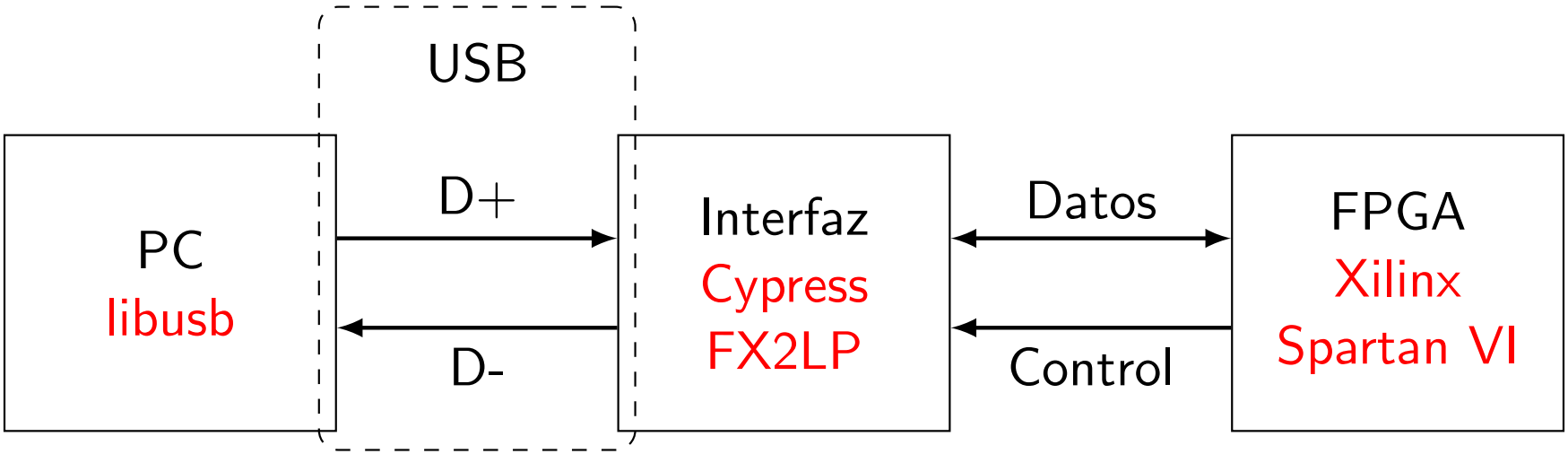
Arquitectura del sistema realizado



Arquitectura del sistema realizado

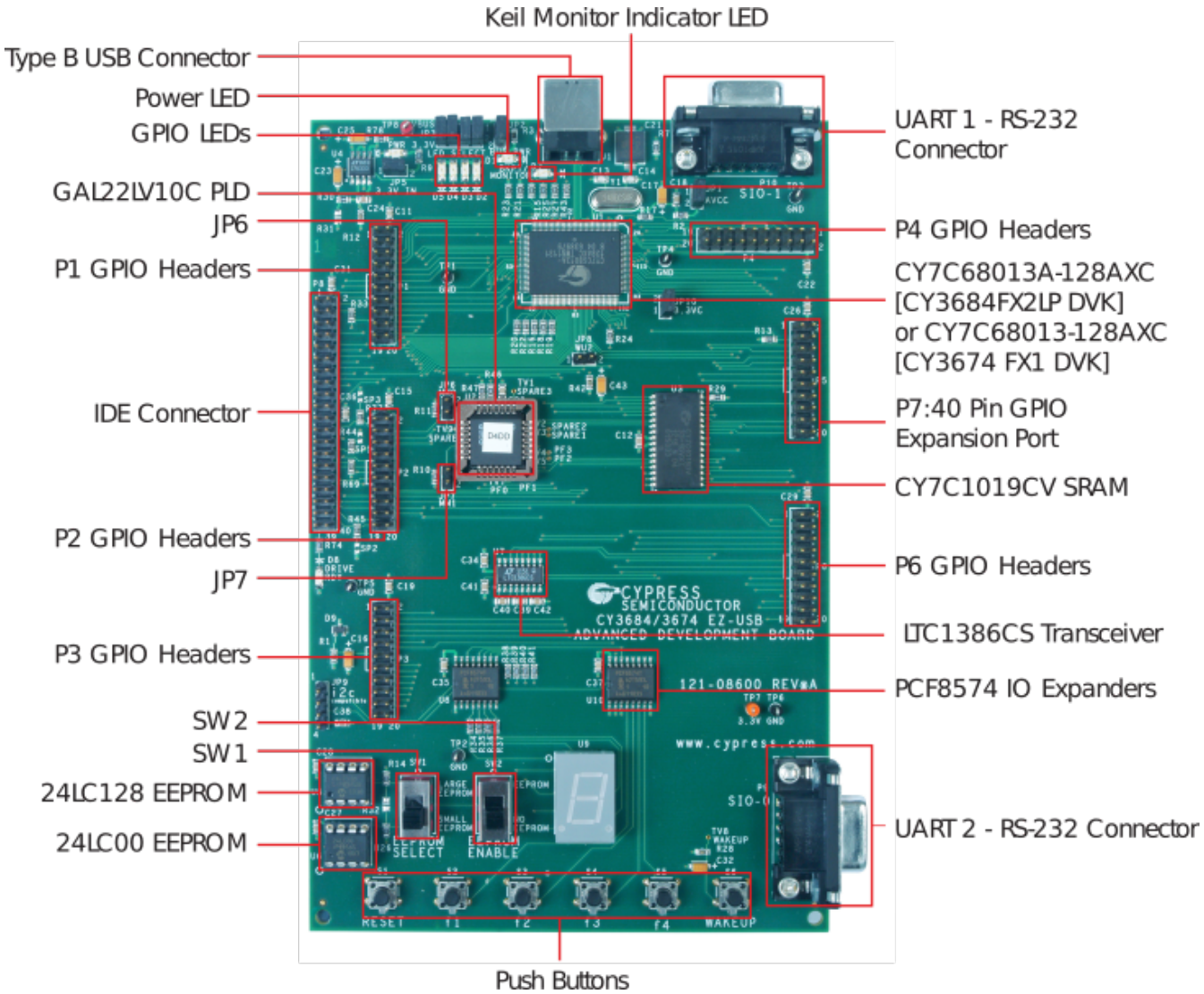


Arquitectura del sistema realizado

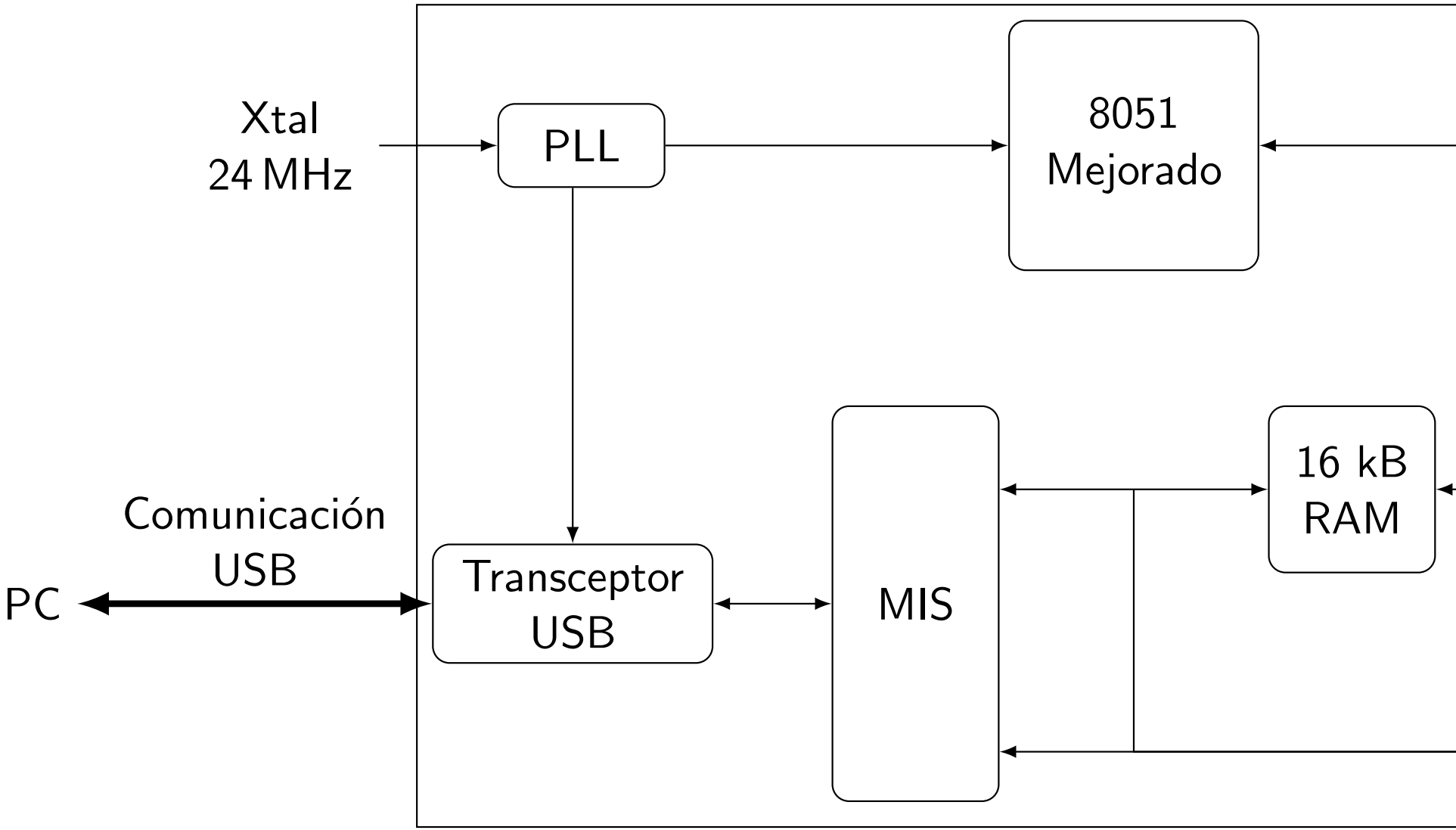


Agenda

Placa de desarrollo utilizada para la interfaz



El circuito integrado FX2LP



Configuración de la interfaz

Agenda

Interfaz - FPGA

Agenda

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

Circuito de interconexión

- Versión 1
- Versión 2
- Versión 3

Agenda

Test Bench

Agenda

Debug Cypress

Agenda

libusb-1.0

Agenda

Esquemas de prueba

Agenda

Flip-Flop para eco

ROM con patrón de repetición infinita

Agenda

Resultados de la prueba de robustez de la comunicación

Agenda

Resultados de la prueba de máxima transferencia de datos

TODO

Agenda

Lo que falta...

Consultas

Muchas gracias

Material Adicional

Respaldo y cosas que no entren

```
#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 //comented '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 diseno.
                //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 endopints
                    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("Cleraing 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;
```

```
        ((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 vacio\n");


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

```
}
```