

**UNIVERSIDAD NACIONAL DE SAN JUAN**  
**FACULTAD DE INGENIERIA**  
**Departamento de Electrónica y Automática**



**Universidad Nacional  
de San Juan**

**Trabajo Final**  
**COMUNICACIÓN USB 2.0 PARA SISTEMAS**  
**CIENTÍFICOS IMPLEMENTADOS EN FPGA**  
Informe

**Edwin Barragán**  
Autór

**Ing. Cristian Sistera**      **Mg. Martín Pérez**      **Dr. Marcelo Segura**  
Asesores



---

# Agradecimientos

Acá le agradezco a todos los miembros de la prestigiosa y gloriosa Comisión de Trabajo Final por sus incontables aportes a la causa. Si pongo punto y meto enter no se vé en el documento. Si escribo barra barra hago un salto de linea pero no cambio de párrafo.

Si doy doble enter, coloca sangría, pero no hace el salto de línea para el párrafo.

Este último sí que es un párrafo decente!



# Índice general

<b>1. Introducción</b>	<b>7</b>
1.1. Motivación . . . . .	7
1.2. El protocolo USB . . . . .	9
1.2.1. Capa física . . . . .	10
1.2.2. Capa de protocolo . . . . .	11
1.2.3. Capa lógica . . . . .	13
<b>2. Elección de las herramientas para la realización de la interfaz</b>	<b>15</b>
2.1. Elección de la FPGA . . . . .	16
2.2. Elección de la interfaz PC-FPGA . . . . .	18
2.3. Elección de la biblioteca libusb-1.0 . . . . .	19
<b>3. Programación y configuración del Controlador USB</b>	<b>21</b>
3.1. Arquitectura FX2LP EZ-USB . . . . .	21
3.1.1. Motor de Interfaz Serial . . . . .	22
3.1.2. Buffers de extremos . . . . .	23
3.1.3. Memorias FIFO esclavas . . . . .	24
3.1.4. Modos de entrada y salida automáticos . . . . .	26
3.2. Kit de desarrollo de Software de Cypress . . . . .	26
3.2.1. Framework Cypress . . . . .	27
3.2.2. Entorno de desarrollo y compilador . . . . .	28
3.2.3. Cypress USB Control Center . . . . .	28
3.3. Desarrollo del firmware . . . . .	28
3.3.1. Inicialización del dispositivo . . . . .	29
3.3.2. Encabezado y declaraciones importantes . . . . .	31

*ÍNDICE GENERAL*

---

# Capítulo 1

## Introducción

### 1.1. Motivación

Un carpintero desea medir la distancia de una barra de madera que luego será, tal vez, la altura de las patas de una futura mesa. Para ello, utiliza una cinta métrica, compuesta de una cinta metálica que posee una escala graduada. Sabe entonces que la barra mide la distancia que coincide con la distancia de la cinta graduada.

Un panadero desea medir cuanto pesa la harina que debe para poder amasar. Entonces, la coloca en una balanza y observa cuanto marca su indicador. Así conoce que la masa de la harina es equivalente a la fracción de medida que indica la balanza.

Un atleta desea conocer cuanto demora en correr un trayecto que posee 1.00 km. Por esto, registra el valor que indica su reloj al principio del recorrido y cuando alcanza el final observa nuevamente el artefacto. Luego de esto, calcula la diferencia entre el valor final y el inicial, conociendo cuanto tiempo le tomó realizar su travesía.

En los tres casos anteriores, tanto el carpintero, como el panadero y el atleta desconocen algo y necesitan cambiar su estado con respecto a esa incertidumbre. Por ello recurren a diferentes objetos, a fin de obtener conocimiento a partir de ellos. Sin embargo, estos objetos, por si mismos, no otorgan información, sino más bien otorgan un dato, que comparado y contrastado con otros datos, se traducen en conocimiento.

La información es el resultado de ordenar y procesar un conjunto de datos, de forma tal que permitan cambiar el estado de conocimiento sobre un asunto determinado. En el caso del carpintero, compara el tamaño de las patas de la mesa con una cinta metálica, que a su vez, posee registrada su distancia en función de algún patrón de metrología, establecido por convención. Esto quiere decir que el dato 1, la longitud del patron, junto al dato 2, escala graduada de la cinta, más el dato 3, la longitud de la cinta métrica, permiten al carpintero cambiar su estado de desconocido a conocido, con respecto a la longitud del trozo de madera, a través de la información proporcionada por el conjunto de datos.

Se puede realizar el mismo análisis con respecto a la balanza del panadero, considerando

## 1. Introducción

un peso patrón, un desplazamiento y una escala graduada o una señal eléctrica emitida por una celda de carga deformada un porcentaje de su capacidad, registrada previamente por su fabricante conforme a pesos patrones, y un circuito adaptador que transforma esa señal eléctrica en un valor numérico mostrado en un indicador.

El atleta compara las posiciones y los desplazamientos de las agujas de su reloj, previamente calibrado para que dé una vuelta por cada minuto en una aguja, otra aguja que dé una vuelta por hora y la tercera una vez cada 12 horas. Además, es probable que él haya ajustado la hora que indica el reloj para que otorgue un horario idéntico al de referencia, establecido por convención.

En todos los casos, se posee una gran cantidad de datos que, ordenados, procesados y comparados otorgan al usuario un valor útil, ya sea una longitud, una masa, un tiempo o cualquiera sea la variable física que se deseé conocer.

La ciencia es un conjunto de técnicas y procedimientos que, a través del método científico, busca adquirir, descubrir y/o desarrollar nuevo conocimiento. Se desprende entonces, que la ciencia produce, de forma fundamental, información que luego es transformada en conocimiento. Cuando hablamos de ciencia, hablamos de una gran gama de objetos de estudio, sujeto a través del cuál se clasifican, en la mayoría de los casos, las ciencias: las Ciencias Sociales estudian las relaciones humanas, las Ciencias Naturales estudian objetos que se encuentran en la naturaleza, las Ciencias de la Tierra se enfocan en una rama más particular de la naturaleza, como lo son los minerales, la superficie terrestre, etc; y siguiendo así se puede encontrar un sinúmero de ciencias. Sin embargo, toda ciencia necesita, para su correcta producción científica, adquirir una gran cantidad de datos que luego serán ordenados, procesados y transformados en información y conocimiento.

La incorporación de una herramienta especialmente diseñada para el procesamiento de datos, como lo es la computadora, permite manejar un número cada vez creciente de información. Es por eso que se encuentra en desarrollo un gran número de sensores y dispositivos que permitan obtener cada vez más datos.

En este sentido, uno de los desarrollos que se encuentran en boga es el sensores que adquieran imágenes. Como ejemplos podemos encontrar, entre muchos otros, el desarrollo de sensores de radiación[1], ultrasonografía[2], telescopía de objetos cercanos[3], imágenes de distancia[4].

La captura de imágenes, fundamentalmente en el desarrollo de sensores nuevos, requiere de sistemas digitales de alta velocidad que tengan la capacidad de acarrear los datos desde el lugar físico en donde se obtienen los datos, es decir, en el transductor mismo, hasta el circuito o el sistema destinado al proceso de los mismos. De esta forma, toma particular interés la utilización de FPGA's, circuitos integrados diseñados para que un diseñador pueda sintetizar un circuito digital de alta velocidad reprogramable en el cual se puede implementar, con ciertas restricciones, circuitos desarrollados para una tarea muy específica que resuelva la tarea que el diseñador necesite.

A diferencia de un microprocesador o un microcontrolador, también muy usados en la industria electrónica, en el cual una unidad lógica algorítmica ejecuta un programa cargado

## 1. Introducción

secuencial, es decir, linea por línea, un FPGA puede ser programado de forma tal que cada proceso se ejecute en forma independiente y paralela, dotando al sistema de una mayor velocidad en el procesamiento.

De esta forma, un diseñador puede manipular un volumen mucho mayor de datos, que a los efectos de la adquisición y medición de imágenes, resulta más adecuado.

Pero como ya se mencionó con anterioridad, la obtención de datos por si misma no le otorga al científico la información, y por ende, el conocimiento nuevo que desea. Para ello, es probable que este gran flujo de datos requiera de un procesamiento y análisis más exhaustivo de los mismos.

Es por esto que la PC *Personal Computer* se ha transformado en la herramienta indispensable en cualquier ámbito, pero en especial en los entornos en donde se requiere el manejo, calculo, procesamiento y análisis de grandes cantidades de informacion de diferentes índoles.

Desde la inclusión de la norma USB, en el año 1996 a la fecha, se ha convertido en el elemento que no falta en ningun equipo, al punto tal que ha desplazado a cualquier otro conector. Al punto tal es esto, que para requerir algún puerto adicional que no sea de esta norma, cualquier comprador debe especificar que así sea, mas o es necesario especificar que tiene USB como norma de conexión.

Este trabajo, pretende elaborar una interfaz entre los dos extremos, es decir, entre la FPGA y la PC, de forma tal que permita a un desarrollador, investigador o usuario en general, obtener una comunicación confiable y con un ancho de banda que permita mover el flujo de datos que genera una sensor que adquiera imágenes.

Es cierto que el protocolo puede ser totalmente implementado en una FPGA, sin embargo, esto requeriría un muy alto costo tanto económico como en recursos disponibles del chip programable para una tarea genérica que es mejor elaborar con un circuito integrado diseñado especialmente para tal fin. Es por esto que se utiliza como lazo de interfaz un chip comercial elaborado por Cypress Semiconductor.

## **1.2. El protocolo USB**

El protocolo USB es un sistema de comunicación diseñado durante los años 90 por los seis fabricantes de la industria de las computadoras, Compaq, Intel, Microsoft, Hewlett-Packard, Lucent, NEC y Philips, con la idea de proveer a su industria de un sistema que permita la conexión entre las PC's y los periféricos con un formato estandart, de forma tal que permita la compatibilidad entre los distintos fabricantes.

Hasta ese momento, el gran ecosistema de periféricos, sumado a los nuevos avances y desarrollos, hacia muy compleja la conectividad de todos ellos. Cada uno de los fabricantes desarrollaba componentes con fichas, niveles de tensión, velocidades, drivers y un sinnúmero de etc diferentes, lo cuál dificultaba al usuario estar al día y poder utilizar cada componente

que compraba. Lo más probable era encontrar que cada vez que uno comparaba una PC, debía cambiar el teclado, el mouse o algún periférico específico. Esto también complicaba a las mismas empresas productoras, por que la introducción de un nuevo sistema requería un mucho soporte extra para poder conectar todo lo ya existente.

Todo esto, quedó saldado con el standar USB, que debido a la gran cuota de mercado de sus desarrolladores, rápidamente fue introducido y se transformó en la norma a la hora de seleccionar un protocolo. Al punto tal esto se cumplió que hoy, más de 20 años después, es muy difícil encontrar PC's con otro tipo de puertos, salvo que en el momento de su compra uno requiera un puerto específicamente. Sin embargo, por norma, cualquier PC nueva dispónible en el mercado debe poseer puertos USB para la conexión de los periféricos.

Desde el punto de vista técnico, el protocolo USB es un sistema del tipo maestro-esclavo, donde el maestro, denominado HOST, debe ser necesariamente una PC y cualquier periférico a ella acoplada será un esclavo.

Para describirlo es conveniente tal vez separar el protocolo en tres partes. Una parte física, en donde se definen los componentes que intervienen, una capa de protocolo, en donde se define el formato y el marco en el que son enviados los paquetes, como se direcciona y como se comunican entre sí, y una parte lógica, en donde cada componente es visto solamente como un extremo y define como fluyen los datos desde un extremo hacia la PC y viceversa.

### **1.2.1. Capa física**

En esta sección no se describirán los detalles de las conexiones eléctricas ni mecánicas a las que se refieren las especificaciones de la norma USB debido a dos cuestiones fundamentales. Una de ellas es que toda esta sección de la norma está resuelta ya por los fabricantes del chip de Cypress. Este chip maneja todas las señales, arma y desarma los paquetes que salen hacia la PC y que llegan de ella respectivamente. Por otro lado, no es el objetivo de este trabajo adentrarse en esos detalles. Gracias a la extensión de este tipo de comunicación existen una gran cantidad de fabricantes en el mercado que fabrican cada uno de los componentes, ya sean, cables, conectores en todas sus versiones, adaptadores de un tipo de enchufe a otro, su costo es despreciable con respecto a cualquier tipo de desarrollo en ese sentido y son de una muy buena calidad, en el sentido que todos cumplen con lo que la norma establece.

Si se hará a continuación una descripción de los dispositivos físicos y su categoría, degun la norma, en función del rol que cumplen.

La comunicación USB posee una topología maestro-esclavo. Es decir, Existe un dispositivo que dirige todas las transferencias de datos y otros dispositivos que responden luego de que el maestro a emitido una directiva. Por esto, el elemento central de cualquier comunicación USB es el HOST (director o anfitrión, por su traducción de la voz inglesa). Este dispositivo que en la mayoria de los casos es la PC, aunquetambién puede ser algún dispositivo inteligente como un smartphone, es el que posee en Host USB Controller. El HOST se encarga de enviar los tokens a todos los periféricos, con la dirección del dispositivo, el sentido de la comunicación, el tipo de

transferencia que se espera y todas las acciones de control que el sistema requiera.

En el otro extremo de la comunicación, se encuentran los dispositivos. Los dispositivos son todos los periféricos que actúan como fuente o sumidero de información. Es decir, en caso de periféricos de entrada, serán una fuente de información hacia el Host. Si los periféricos son de salida, serán un sumidero de la información que proporciona la PC. Los casos de periféricos de entrada/salida, se denominarán periféricos compuestos.

Existe también, a los fines de la norma, un elemento intermedio, denominado HUB (concentrador o distribuidor, según la traducción del inglés). Este dispositivo se encarga de conectar dos o más dispositivos, ya sea de entrada o salida, de recibir y enviar las direcciones y de regenerar las señales que el host envía y deben ser recibidas por los dispositivos, o bien, los datos que fluyen por el sistema.

### **1.2.2. Capa de protocolo**

En la capa de protocolo, las especificaciones detallan como se compone el marco y como los paquetes deben ser armados para que sean efectivamente enviados a través del sistema.

Cada mensaje que se intercambia por el bus se denomina paquete. Cada paquete posee en su inicio lo que se denomina PID. El PID (del inglés packet ID o identificador del paquete) puede ser de 4 tipos, definidos por cada uno de los tipos de paquetes que puede haber:

- en primer lugar se encuentran los paquetes token, a través del cual el host envía las directivas a los distintos periféricos. Estas directivas pueden ser IN, cuando solicita datos a un periférico; OUT, cuando va a enviar datos a un dispositivo; SOF, que indica el inicio de cada cuadro, para que cada dispositivo se sincronice y SETUP, cuando va a enviar un paquete de configuración a algún dispositivo.
- el segundo tipo de paquetes es el paquete de datos. Este tipo de PID puede ser emitido por un dispositivo, si es que envía datos al host o bien por el mismo host cuando el flujo de datos es a la inversa.
- el tercer tipo de paquetes es un paquete de reconocimiento, denominado ACK, (por acknowledge o reconocimiento). Este paquete es enviado por los periféricos y le da idea al host de cuál es el estado del dispositivo, es decir, si se encuentra operativo o no, si se encuentra ocupado o si recibió la transferencia de forma correcta.
- finalmente existen paquetes especiales, a través de los cuales el protocolo se comunica con los hubs, emite directivas intermedias, envía señales de polling para conocer el estado del bus, entre otras.

Como se mencionó anteriormente, el host envía un token SOF que sirve para sincronizar los dispositivos al bus. En un sistema USB, el host provee la base de tiempo y envía cada 1 ms un SOF (Start of frame, o su traducción, inicio de cuadro) seguido de un número de 11 bits que sirve para contar cada uno de los marcos. Además, en sistemas de alta velocidad, cada cuadro

se divide en ocho microcuadros de 125 µs, que también son marcados por un SOF, sin embargo, el contador no se actualiza por cada microcuadro.

Luego de esto, el sistema puede comenzar con la transferencia de datos. USB dispone 4 tipos de posibles transferencias, que se detallan un poco más adelante, y que pueden ser usadas conforme a los diferentes requerimientos del sistema.

Cada transferencia de datos está compuesta por un primer paquete de token, emitido por el host, que posee el tipo de transferencia que se espera, sea de entrada, de salida, de control o especial; la dirección del dispositivo que debe responder o escuchar el mensaje enviado por el bus y un código de detección de errores del tipo CRC (código de checkeo redundante cíclico).

Luego, el siguiente paquete posee los datos transferir, precedido por un PID de datos, y otro código CRC para detectar errores. Este paquete es transmitido por el host o el dispositivo dependiendo del sentido de la transferencia.

Finalmente, el dispositivo devuelve un paquete de reconocimiento, indicandole al Host si el transferencia fue efectiva o no y por qué esta no fu efectiva, siendo ese el caso.

### **Transferencias por paquetes (Bulk transfers)**

Este tipo de transferencias puede ser dispuesta para trasmitir un gran flujo de datos. No posee perdida de datos gracias a un sistema de chequeo y retransmisión de datos. El inconveniente que presenta este tipo de transferencias es que en un nivel de prioridades se presenta en el final del sistema. Es decir, el bus solo va a ser usado para transferir este tipo de datos siempre que se encuentre desocupado, o bien, se le asignará una proporción ínfima de ancho de banda para poder trasnmitir con este modo. Es comunmente usado para trasmitir datos que no son críticos en tiempo, por ejemplo para scanners e impresoras.

### **Transferencias de interrupción**

Este tipo de trasnferencias sirve para enviar y recibir paquetes de datos que requieren un buen sistema de control de errores, pero que, son más restrictivos en tiempos. El sistema siempre destinará un intervalo fijo de tiempo para transmitir los datos que estén pendientes para trasnferencias de interrupción.

### **Transferencias Isocrónicas**

Este tipo de trasnferencias está destinado a datos que son realmente críticos en tiempo. Es usado, principalmente para enviar datos a chorro, como ser el caso de streaming de audio o video, en donde los datos producidos deben ser rápidamente llevados al usuario.

## 1. Introducción

No posee un control de errores muy sofisticado, más que un simple código CRC, pero no existe mecanismo de retransmisión de datos ni handshaking entre los dispositivos y el host.

Como el tiempo es el requerimiento crítico en este tipo de datos, el controlador le asigna una determinada cantidad de tiempo de bus, o en otras palabras, una determinada cuota de ancho de banda.

### **Transferencias de control**

Este tipo de transferencias solo las emite el host y el sistema las utiliza para configurar cada dispositivo. Debido a su criticidad, el controlador asigna cada cuadro de una fracción de ancho de banda para las transmisiones de control. Es el tipo de transferencias que posee el sistema de detección de errores más sofisticado, de forma tal de asegurar la integridad de los datos de control.

A cambio de esto, solo muy poca información puede ser transmitida por cada cuadro, de hasta 64 bytes en sistemas de alta velocidad.

### **1.2.3. Capa lógica**

Desde el punto de vista lógico, cada dispositivo es visto por el host como un extremo independiente, que posee un modo de comunicación, es decir, con ese dispositivo el protocolo se comunicara solo por un tipo de transferencia; y un solo sentido. En otras palabras, USB notará como separado un dispositivo de entrada y otro de salida, independientemente de si físicamente el dispositivo es un periférico de entrada y salida.

Esta independencia brinda la posibilidad de configurar cada extremo de forma diferente y obtener el ancho de banda necesario para la subida y bajada de datos, los tiempos de acceso al bus, la dirección y todo lo relacionado a los modos de comunicación conforme a los requerimientos.

El protocolo entiende que entre el host y cada uno de los extremos existe un tubo (la norma en inglés habla de *pipes*) en donde la información es colocada y transferida. Luego, cada tubo posee la configuración establecida por el controlador del host y se comunica con cada extremo por medio de estos tubos. A los fines del usuario, esto es lo importante, por cuanto uno solicita acceso al bus y define en qué buffer va a contener los datos a enviar o transmitir y el protocolo se encarga de el empaquetado, el armado de los cuadros, el acceso al bus y el posterior envío de datos.



## Capítulo 2

# Elección de las herramientas para la realización de la interfaz

El objetivo que persigue el presente trabajo es el desarrollo e implementación de una comunicación entre una PC y desarrollos científicos basados en FPGA mediante el protocolo USB 2.0 de alta velocidad.

Todo sistema USB puede ser dividido en, al menos, tres etapas fundamentales que cumplen funciones específicas a lo que el protocolo se refiere. Estas tres etapas se observan en la Figura 2.1.

De izquierda a derecha, la primera de ellas, el transceptor, se encarga de adecuar los valores de tensión, las frecuencias, impedancias y todo lo relacionado a las señales que envía el protocolo a través de sus conectores. El segundo, el Motor de Interfaz Serial (MIS), es el encargado de recibir los datos que se producen en el dispositivo, colocar el encabezado y la cola del mensaje, ordenar y armar los paquetes de forma tal que sean coherentes con el protocolo. También se encarga de recibir los paquetes que llegan por el bus, decodificarlos, corroborar que no presente errores y entregarlos al resto del dispositivo. Finalmente, la lógica de control es la encargada de enviar las órdenes de trabajo al MIS, producir y consumir los datos que van y vienen por el sistema.

Gracias al gran potencial que poseen los FPGA's, desde el punto de vista de la electrónica digital, es posible desarrollar en uno de estos dispositivos toda la lógica de control y el MIS. A los fines de la adecuación de las señales y todo lo relativo a la parte analógica del sistema, siempre será necesario un transceptor que se comunique con el bus. Esta implementación resulta minimalista desde el punto de vista de PCB y cantidad de CI necesarios.

Sin embargo, este enfoque requerirá un gran consumo de recursos de una FPGA, los cuales

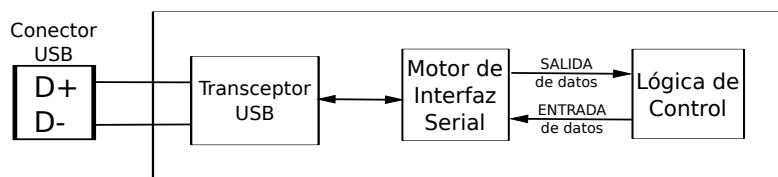


Figura 2.1: Etapas fundamentales de un dispositivo USB

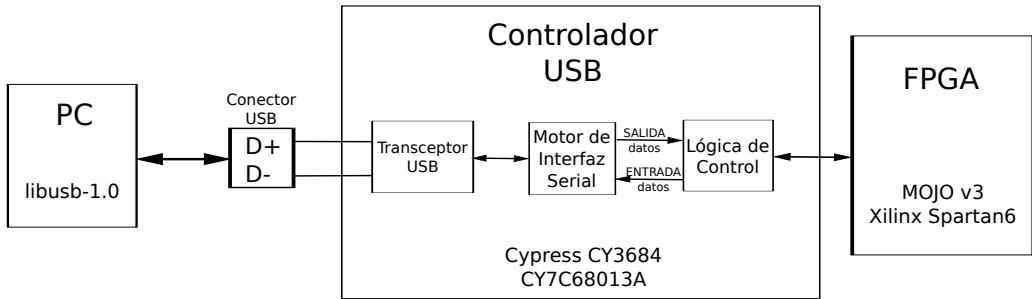


Figura 2.2: Vista general del sistema propuesto

son muy valiosos para realizar otro tipo de desarrollos y no para dedicarlos exclusivamente para la comunicación del sistema principal con una PC, presentando, en este sentido, ventajas mayores la utilización de otro tipo de trasnferencias.

Es por esto, que se propone en su lugar, ocupar un integrado específico, diseñado para tal fin, y disponer de la mayor parte de los *slices* y bloques del FPGA para desarrollos complejos y complementarios. La Figura 2.2 muestra el esquema planteado, en donde se realiza dentro de la FPGA una maquina de estados algorítmica (MEA) mínima, que realiza un control mínimo del sistema y posee una aplicación que produce y consume datos; un controlador USB, que hace las veces de puente o interfaz entre el FPGA y una PC. Éste dispositivo posee el MIS y su control, los que se conectan a un trascceptor que el fabricante incorpora para los propósitos de la comunicación y buffers incorporados en donde almacena los datos que fluyen entre ambos extremos; finalmente, una PC que sirve para enviar y recibir datos, con el objetivo de corroborar el correcto funcionamiento de la comunicación.

Como se explayará a continuación, para el bloque que corresponde a la FPGA, se utilizó un Spartan IV que viene integrado a una placa de desarrollo comercial que posee por nombre MOJOv3. La interfaz se implementa con un controlador USB CY7C68013A fabricado por Cypress Semiconductor, incorporado en una placa de desarrollo comercial CY3684 EZ-USB FX2LP Development Kit del mismo productor. En el lado de la PC, se utiliza la biblioteca de código abierto libusb-1.0, para elaborar una software escrito en lenguaje C que envie datos, los reciba y chequee los errores producidos.

## 2.1. Elección de la FPGA

Para la implementación de la comunicación de un desarrollo científico determinado, se requiere un nexo entre la síntesis del circuito y la memoria del controlador USB. Este vínculo se lleva a cabo mediante una pequeña MEA que ejecuta las señales de lectura y escritura. Esta MEA se desarrolla en lenguaje VHDL.

Se utiliza por esto la placa MOJO v3 desarrollada por la empresa Embedded Micro. Esta placa, la cual se observa en la Figura 2.3, posee un Spartan-VI de Xilinx. El FPGA brinda posibilidad de elaborar sistemas complejos de muy alta velocidad y permite al desarrollador

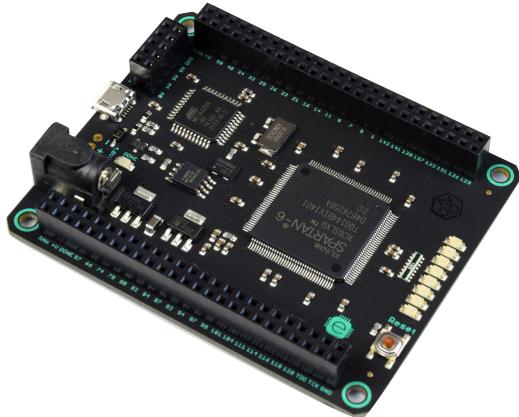


Figura 2.3: Placa de prototipado rápido MOJO v3, diseñada por Embedded Micro

de sensores y sistemas de adquisición de datos científicos la síntesis de circuitos que resuelvan problemas a la medida de los requerimientos. Dispone también de 84 puertos digitales configurables como entrada y/o salida, 8 entradas analógicas, 8 LED's de propósito general, un botón de tipo pulsador.

La placa MOJO es una placa inspirada en el concepto de prototipado rápido. Para ello los puertos se disponen en un arreglo de pines a través de los cuales es posible acoplar cualquier dispositivo que uno necesite. Se dispone en el mercado otros circuitos, que los fabricantes denominan shields (*escudo* traducido al castellano), que encajan a la perfección en todos los pines y que contiene un set particular de periféricos útiles para propósito general. Estos shields, también, con el fin de satisfacer requerimientos especiales, pueden ser diseñados por uno mismo, o bien es posible adaptar con algunos cables las entradas a un dispositivo particular.

Además de estos shields, los diseñadores pensaron en que no sea necesario ninguna herramienta extra a la hora de programar la FPGA. Para ello, dotaron al sistema de un microcontrolador ATmega32U4 de Atmel y cargaron un programa bootloader, que se encarga de transferir la configuración del FPGA cargada desde una memoria flash incorporada al sistema con ese propósito particular, o transmitida por el usuario desde una PC, a través de un transceptor USB que contiene el microcontrolador. Luego, este último es colocado en modo esclavo y se configura de forma tal que dota al sistema de una comunicación entre la FPGA y una PC, vía USB y se utiliza su ADC para leer los puertos analógicos.

Una vez llegado a este punto, el lector podría preguntar con toda razón ¿por qué es necesario realizar un sistema de comunicación USB extra, si ya cuenta con un microcontrolador que se encarga de dicho asunto? La respuesta se basa en el ancho de banda del sistema de comunicación que dispone la placa. La línea de controladores ATmega incorpora puertos USB 2.0 full-speed. Esto quiere decir que puede enviar datos a una tasa de 12 Mbps. Además, la comunicación entre ambos chips se realiza via SPI (*Serial Peripheral Interface*, o en español Interfaz Serie de Periféricos), comandada por un cristal de cuarzo de 50 MHz, ofreciendo una velocidad de salida que puede resultar lenta a los fines de este trabajo. Se pretende dotar al sistema del mayor ancho de banda posible, utilizando la capacidad de USB 2.0 High-Speed, de hasta 480 Mbps.



## **2.3. Elección de la biblioteca libusb-1.0**

libusb es una biblioteca de código abierto, muy bien documentada, escrita en C, que brinda acceso genérico a dispositivos USB. Las características de diseño que persigue el equipo de desarrollo que mantiene la biblioteca es que sea multiplataforma, modo usuario y agnótico de versión.

En el sitio web disponible se explica lo que significa cada uno de estos conceptos:

- Multiplataforma: Se apunta a que cualquier software que contenga esta biblioteca pueda ser compilado y ejecutado en la mayo cantidad de plataformas posibles, dotando al software de portabilidad, es decir, la biblioteca puede ser ejecutada en Windows, Linux, OS X, Android y otras plataformas sin necesidad de realizar cambios en el código.
- Modo usuario: No se requiere acceso privilegiado de ningún tipo para poder ejecutar programas escritos con esta biblioteca.
- Agnótico de versión: Sin importar la versión de la norma USB que se utilice, el programa se podrá comunicar siempre con el dispositivo USB que se requiera.

Otra ventaja que posee la biblioteca libusb es que, al ser de código abierto, posee una gran comunidad que contribuye al crecimiento del proyecto, como así también otros proyectos que utilizan esta biblioteca. Así, existe una gran variedad de ejemplos que facilitan el aprendizaje en su utilización y adaptaciones para diferentes lenguajes de programación, que se adapte a los conocimientos previos de la persona que desarrolla programas.



## Capítulo 3

# Programación y configuración del Controlador USB

### 3.1. Arquitectura FX2LP EZ-USB

El núcleo del Kit de Desarrollo FX2LP EZ-USB es un CY7C68013A. Dicho circuito integrado, cuya arquitectura se presenta en la Figura 3.1, pertenece a la serie FX2LP de la familia de integrados EZ-USB comercializado por Cypress Semiconductors.

Esta serie se caracteriza por brindar una conexión USB 2.0 de alta velocidad y bajo consumo energético, especialmente diseñados para productos con autonomía limitada. Se integra un controlador USB completo, incluyendo un transceptor USB, un MIS (Motor de Interfaz Serial), buffers de datos implementados con memorias tipo FIFO (*FirstInFirstOut*; Primero Entrado, Primero Salido), un microcontrolador 8051 mejorado y una interfaz programable hacia los periféricos. Además posee un PLL con divisor configurable a través de los cuales proveen las señales de reloj adecuadas para el correcto funcionamiento del sistema.

De esta forma, se permite al usuario trasmitir datos desde y hacia el anfitrión a través del mismo puerto USB, o bien vía RS-232. Para comunicarse con sistemas periféricos se puede

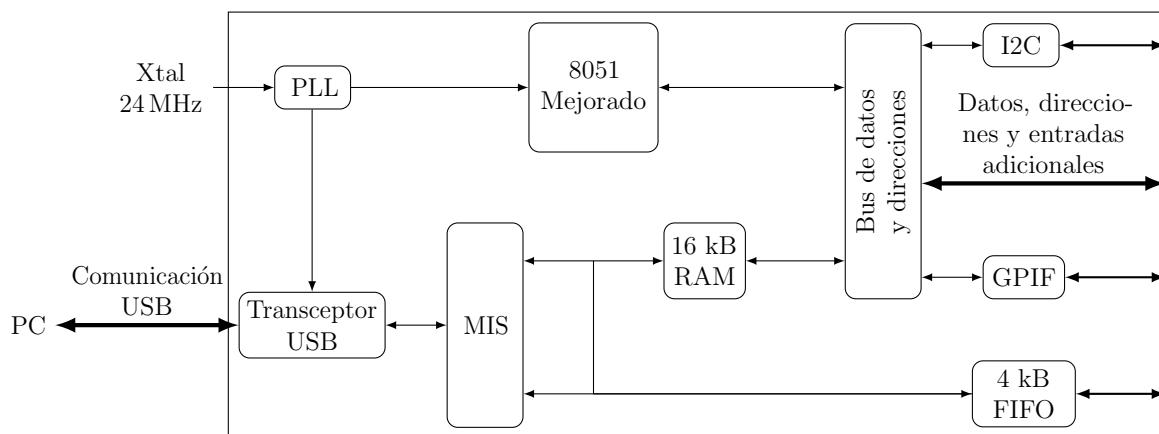


Figura 3.1: Arquitectura FX2LP

aprovechar el puerto  $I^2C$ , la interfaz de propósito general, que actúa como maestro y a la cual se le puede acoplar un periférico esclavo, y/o las memorias FIFO en modo esclavo que puede ser conectada a un sistema maestro. Esto brinda muchas alternativas, desde la conexión a puertos estandar, como ser ATA, PCMCIA, EPP, etc. o también la conexión de dispositivos tales como DSP's y FPGA's.

Este trabajo utiliza particularmente las memorias FIFO en modo esclavo, que responden a las diferentes señales que les proporciona un maestro externo implementado con un FPGA; por lo que a continuación se explicitan algunos detalles referidos a ellos, con lo que se busca aclarar el funcionamiento y que el lector comprenda los fundamentos de las configuraciones que se plasmarán en el código del firmware.

### 3.1.1. Motor de Interfaz Serial

La comunicación USB entre el controlador FX2LP y la PC se realiza a través del transceptor, unido al MIS. Como se observa en la Figura 3.2, el usuario, a fin de intercambiar datos, solo debe colocar o extraer los datos de registros destinados a tal fin y modificar las banderas de handshaking, que en la figura se observan como ACK (abreviación del inglés *acknowledge*, que significa reconocer, aceptar o agradecer), que indican si el sistema está disponible, si los datos fueron colocados o leídos, dependiendo el caso tratado. De forma automática, el MIS y el transceptor USB se encargan de empaquetar, enviar, recibir, desempaquetar toda la información, así como leer los tokens que emite el host, calcular y corroborar los códigos cíclicos de detección de errores y todo lo relacionado al protocolo propiamente dicho.

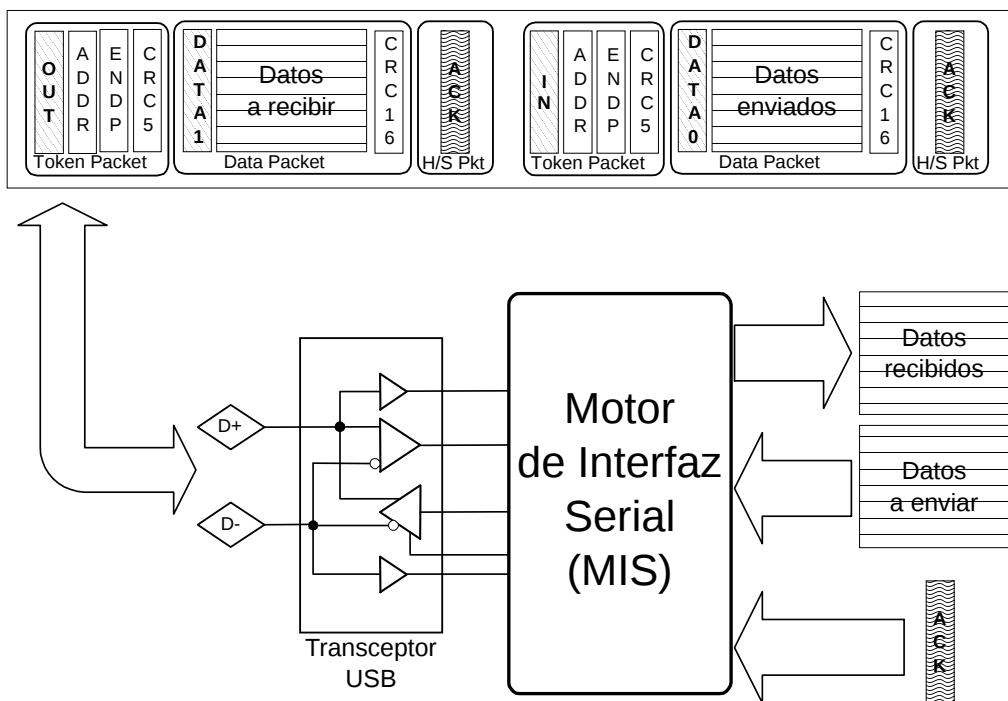


Figura 3.2: Implementación del enlace USB realizado por el EZ-USB

### 3.1.2. Buffers de extremos

El MIS guarda los datos que aún no han sido enviados y/o los que han sido recibidos pero no leídos por ningún periférico en una memoria RAM específica, denominada buffer de extremo.

La norma USB define a un dispositivo extremo como una porción exclusiva e identifiable de un dispositivo USB que es fuente o un sumidero de información. En otras palabras, USB ve a cada extremo como una memoria FIFO de donde surge o finalizan la información. En inglés, el término extremo se escribe endpoint, por lo que, en adelante, cuando se hable de ellos se abreviará como EP o EPx, siendo la x un número que indica la dirección del extremo.

La serie de controladores FX2LP dispone de hasta 7 EP's programables, los cuales deben poseer al menos dos buffers.

La norma USB indica que cualquier dispositivo USB debe poseer un EP con dirección 0 que se destina para control y configuración, por lo que el controlador está dotado de 64 B para este fin. Es el único EP que puede ser bidireccional en el sentido del flujo de datos. A través de él, anfitrión y dispositivo intercambiarán solo transferencias de control.

Luego, se incorpora un EP1, que posee dos buffers fijos, o sea no configurables, de 64 bytes, uno como entrada y el otro como salida.

Finalmente, 4 KiB de memoria debe ser configurada para los EP2, EP4, EP6 y EP8. La configuración de los EP la realiza el firmware en tiempo de ejecución. Las variables, conforme a los requerimientos de ancho de banda, acceso al bus son:

- Tamaño: Dependiendo del extremo a configurar puede ser de 512 o 1024 bytes.
- Tipo de acceso al bus: Definido según la norma USB, este tipo puede ser por bultos, isocrónico o de interrupción.
- Cantidad de buffers: Dependiendo del extremo, puede ser dos, tres o cuatro buffers por extremo.
- Habilitación: Se debe indicar al sistema si los extremos se usan o no. El EP no válido, no responderá a un pedido de entrada o salida.

La Figura 3.3 muestra solo dos las posibles configuraciones de los extremos. A la izquierda se observa la configuración por defecto del controlador FX2LP. Esto es, los cuatro EP's habilitados, con 512 bytes cada uno, buffers dobles y comunicación por bultos. A la derecha se muestra la configuración elegida para este trabajo, es decir, solo son válidos el EP2 y EP8. EP2 posee tres buffers de 1024 bytes y el EP8 dos buffers con 512 bytes de capacidad cada uno. Siempre hay que considerar no se dispone más que 4 KiB de memoria.

La característica de los buffers múltiples evita la congestión de datos. Con doble buffer, un periférico (o el microcontrolador) coloca o extrae datos de un buffer, mientras otro, del mismo EP, se encuentra enviando o recibiendo datos mediante el MIS. Cuando se configura un triple

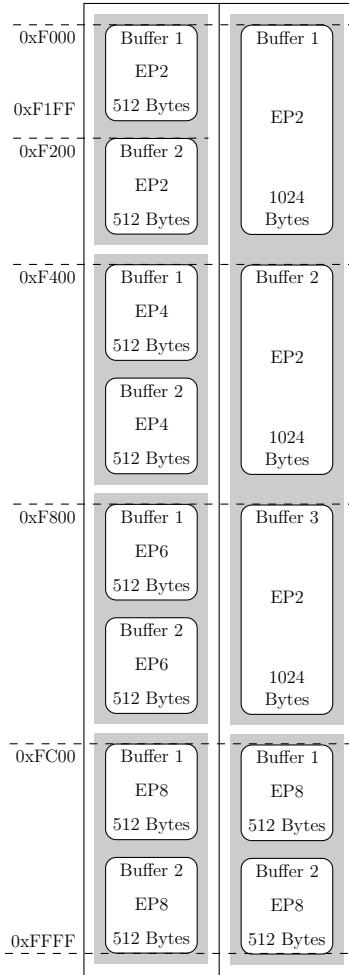


Figura 3.3: Buffers de extremos con sus direcciones de memoria. El cuadro de la izquierda muestra la configuración por defecto. El derecho, la implementada en este trabajo.

buffer, se agrega una porción mas de memoria a la reserva. De esta forma, se le otorga al sistema una gran capacidad de datos y ancho de banda.

Un detalle importante de los buffers múltiples es que, a la vista del controlador y/o de un periférico, el buffer posee una sola y única dirección y, es el sistema mismo quien se encarga de seleccionar el buffer en uso. Esto quiere decir que, por ejemplo, teniendo 4 buffers de 512 bytes cada uno, el 8051 verá solo uno de 512 bytes, sin necesidad de identificar con cuál de los cuatro está trabajando.

### 3.1.3. Memorias FIFO esclavas

Desde un punto de vista digital, el MIS recibe y envía datos desde y hacia el puerto USB. Para ello utiliza un cristal de 24 MHz. Por su parte, un sistema externo puede o no proveer una señal de reloj y manejo de datos propio. El controlador USB incorpora memorias FIFO que se encargan de proveer una interfaz entre el MIS y un dispositivo externo, salvando el problema de

poseer dos relojes diferentes e independientes.

Estas memorias funcionan en modo esclavo, es decir, se debe conectar, al controlador, un dispositivo capaz de proveer una lógica maestra externa que comande la entrada y salida de datos desde una memoria FIFO hacia o desde el exterior.

Para los fines del presente trabajo, este modo de funcionamiento es óptimo, ya que dotando al FPGA de una máquina de estados mínima, se logra la transferencia de datos en los tiempos requeridos.

El sistema de bus permite conectar a estas memorias hasta cuatro dispositivos diferentes. Por esto, existe una memoria FIFO para cada uno de los EP programables en el buffer de extremos.

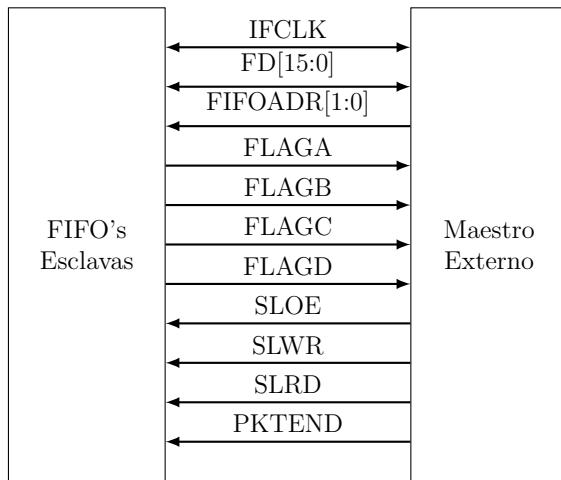


Figura 3.4: Configuración de puertos para la interfaz entre las FIFO's y un maestro externo

La Figura 3.4 muestra la interfaz entre las memorias FIFO's y un maestro esclavo. Estos son:

- IFCLK: señal de reloj. En caso de conectar las memorias en modo asincrónico, no es necesario. La señal de reloj puede ser establecida por el controlador o por el dispositivo de control en forma programable.
- FD[15:0]: constituye el bus de datos. Según se programe, este puede ser de 8 o 16 bits.
- FIFOADDR[1:0]: puerto de direcciones. A través de él se selecciona la memoria activa en el bus.
- FLAGx: Los cuatro puertos de flag son configurables e indican memoria llena, vacía o un nivel programable, en forma fija sobre una memoria en particular o que indiquen sobre la memoria activa.
- SLOE, SLWR, SLRD: son las señales de control. A través de ellas el maestro entrega las ordenes de lectura y escritura.
- PKTEND: a través de este puerto el maestro indica que terminó una transferencia de datos.

### 3.1.4. Modos de entrada y salida automáticos

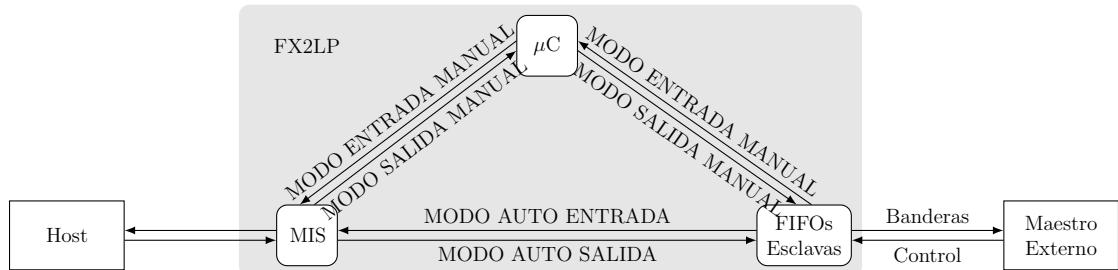


Figura 3.5: Modos de conexión de la memoria FIFO, el microntrolador y el MIS

Los datos que se reciben o envían a través del MIS. Pueden ser enviados en forma automática desde y hacia las memorias FIFO, o bien, pueden ser dirigidos a través del microcontrolador. Esto último permite leer, modificar, suprimir, agregar y/o generar nuevos datos antes de ser remitidos como paquete, es decir, todos juntos, a su respectivo EP. Estos caminos se pueden ver en la Figura 3.5.

Los fabricantes llaman a estos caminos "MODO MANUAL", en caso de enviar los datos a través del 8051, y "MODO AUTOMÁTICO", cuando la comunicación es directa entre el MIS y las FIFO. Además, se programan en forma independiente para cada extremo, sea este de salida o entrada.

Se debe notar en la Figura 3.5 que se refiere a paquetes de entrada cuando estos poseen una dirección que se inicia en un periférico y termina en el anfitrión y de salida cuando llevan el sentido contrario. Esto se debe al carácter *anfitrión-céntrico* de la comunicación USB, en donde el principal componente es el anfitrión y a él se acoplan los diferentes dispositivos.

## 3.2. Kit de desarrollo de Software de Cypress

Dentro del kit de desarrollo avanzado EZ-USB DVK, Cypress provee un amplio conjunto de herramientas que facilitan al desarrollador la implementación de todo lo necesario, desde lo que a software se refiere. Este conjunto de herramientas, Cypress lo denomina Kit de Desarrollo de Software (o SDK, acrónimo del habla inglesa, *Software Development Kit*).

El kit abarca una gran cantidad de aspectos, como ser la elaboración del firmware implementado en el 8051 del controlador, la elaboración de drivers, la conversión de archivos de programación y programas que grabar la información en los diferentes chip, ya sea en la RAM del microcontrolador o en EEPROM, para cargar programas no volátiles que posteriormente serán ejecutados por el 8051.

Debido a que no todo fue utilizado en este trabajo, a continuación se desarrollarán solo las herramientas utilizadas.

### 3.2.1. Framework Cypress

Con la intención dotar al desarrollador con una herramienta que le potencie la velocidad de diseño, Cypress provee una plantilla de código en lenguaje C para microcontroladores 8051.

Esta plantilla posee precargados todos los registros que posee la serie de controladores FX2LP con los mismos nombres que figuran en el manual de usuario. Además incorpora todas las funciones que el microcontrolador debe llevar a cabo para efectuar la comunicación USB. Los archivos que incorpora son:

- fw.c: es el código fuente principal. Contiene la función main() y lo necesario para manejar la comunicación USB.
- periph.c: contiene la implementación de las funciones invocadas por fw.c. Aquí se encuentran las funciones TD\_Init() y TD\_Poll() a través de las cuales el usuario implementa el programa que necesita. También contiene todas las funciones de interrupción vectorizadas.
- fx2.h: posee la definición de constantes, macros, tipos de datos y funciones prototipo de la biblioteca.
- fx2regs.h: declara registros y máscaras.
- dscr.a51: este archivo es el descriptor de dispositivo. Es la información que USB necesita para poder registrar el dispositivo en el anfitrión, asignarle dirección y establecer los parámetros.
- ezusb.lib: biblioteca que implementa las funciones provistas por el fabricante.
- syncdely.h: macro de sincronismo. Algunos accesos de registro requieren un tiempo de establecimiento específico que en el código se implementa deteniendo el microcontrolador ciertos ciclos de reloj.
- usbjmpbt.obj: especifica las direcciones en memoria de las interrupciones vectorizadas

La Figura 3.6 muestra una versión modificada del diagrama de flujo que sigue el código fuente provisto por Cypress. De ella se quitan funciones que no son necesarias para los objetivos del presente trabajo.

Cuando el programa es cargado al controlador, este se encarga de inicializar todas las variables de estado a su valor por defecto. También en este punto establece la comunicación con el anfitrión y le envía los descriptores provistos en el archivo **dscr.a51**. Acto seguido, ejecuta la función **TD\_Init()**, a través de la cual el usuario programa e inicia la configuración del sistema. Luego, es necesario habilitar todas las interrupciones necesarias y finalmente, se invoca repetidamente la función **TD\_Poll()**, en donde el usuario escribe las tareas que ejecutará el 8051.

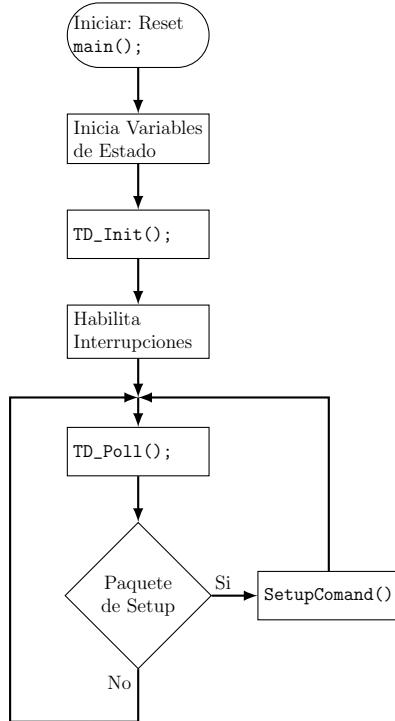


Figura 3.6: Diagrama en bloques simplificado

### 3.2.2. Entorno de desarrollo y compilador

Si bien Cypress no desarrolla entornos de desarrollo para escribir, compilar y depurar códigos, distribuye junto con el kit 3684 una versión de evaluación de Keil  $\mu$ Vision con el compilador C51 para programar microcontroladores basados en 8051. Aún la versión de limitada de este entorno, resulta suficiente para la programación del controlador USB.

### 3.2.3. Cypress USB Control Center

Dentro del kit de desarrollo de software que brinda Cypress se encuentra el programa USB Control Center. Este software se utiliza durante este trabajo para cargar un programa compilado en el controlador FX2LP, para transmitir y recibir mensajes.

La utilidad de este programa radica en la realimentación mínima de lo que está realizando el controlador, de forma que facilita las pruebas sobre el sistema en desarrollo.

## 3.3. Desarrollo del firmware

A continuación se desarrolla el aspecto más relevantes del código elaborado, que es la función de inicialización.

Al establecer el modo automático de funcionamiento, la función TD\_Poll() se encuentra vacía.

### 3.3.1. Inicialización del dispositivo

La inicialización del dispositivo se realiza a través de la función TD\_Init(). Ésta es invocada solo una vez en el código, antes de ejecutar el loop principal, donde el programa ejecuta tareas específicas una y otra vez.

En primer lugar, se debe configurar la frecuencia a la que corre el reloj principal. Esta puede ser de 12 MHz, 24 MHz o de 24 MHz. Para ello se deben colocar los registros CPUCS.4=1 y CPUCS.3=0. A través del framework de Cypress, esto puede ser escrito de la siguiente forma:

```
CPUCS = ((CPUCS & ~bmCLKSPD) | bmCLKSPD1); // 48 MHz
```

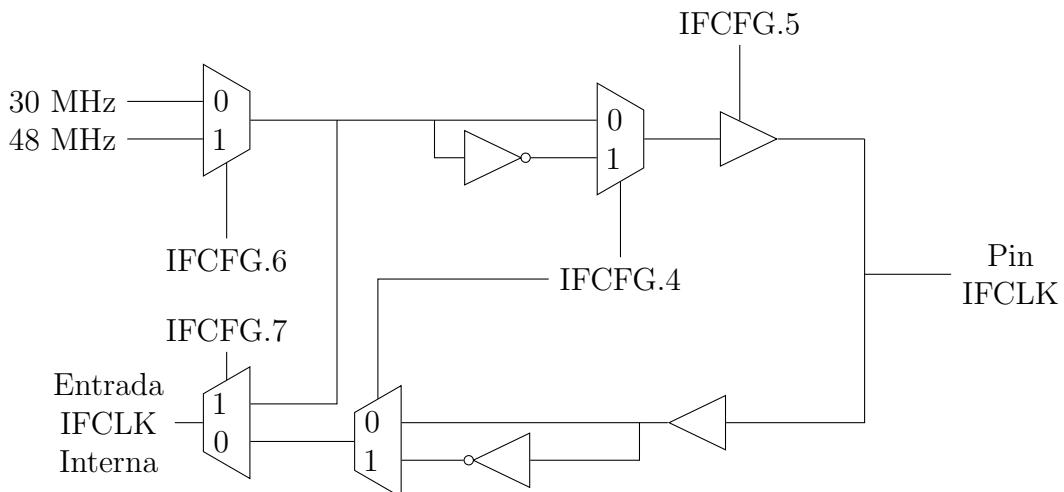


Figura 3.7: Esquema funcional para la entrada de reloj de la Interfaz

Luego se debe configurar el funcionamiento del reloj de la interfaz. El esquema de la Figura 3.7 muestra la configuración del hardware. Cómo se observa, La interfaz puede funcionar con frecuencias de 48 MHz o 30 MHz provistas por el FX2LP. Además, la señal puede ser dirigida al exterior, o bien, ser provista por un periférico. Los inversores se programan de forma tal que la interfaz sea activa en el flanco positivo o negativo del reloj fuente. En este trabajo, el registro se programa para tomar como reloj la señal de 48 MHz provisto por el mismo chip, la polaridad es de flanco ascendente y no se posee señal de reloj en el pin externo IFCKL. Además, se programa de modo asíncrono y en modo FIFO esclavo, a cuya configuración se accede por este puerto.

El autor entiende sobre la posibilidad y conveniencia de utilizar el modo sincrónico para conectar la interfaz. Más aún, es factible proveer la señal necesaria de reloj desde la FPGA y evitar así problemas de desajustes y fallas de sincronismo. Sin embargo, por error del alumno en el diseño del impresor de interconexión, la entrada de reloj no quedó conectada al chip de la FPGA. Durante la escritura de este informe, se encuentra en viaje el impresor con las correcciones

pertinentes y espera ser implementado en trabajos futuros.

La configuración, entonces, queda definida por la sentencia de código:

```
// colocar Interfaz FIFO esclava a 48MHz
// clk interno, no_salida, no_invertir clk, no_asincr
// fifo esclava (11) = 0xC3
// 0xCB; para asincr.
IFCONFIG = 0xCB;
SYNCDelay;
```

A continuación, se debe establecer el funcionamiento de los pines bandera. Estos avisan cuando un EP está vacío, completo o a un nivel programable. Para este trabajo, son necesarios solo una bandera que señale el nivel vacío del puerto por el que entran los datos a la FPGA y otra que señale el nivel completo del EP donde escribe los datos a enviar. Si bien no son leídos en ningún momento, para evitar problemas se configuran las banderas vacías y completas para ambos EP's:

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

Luego, se deben programar los EP's. La configuración por defecto define a todos los EP como transferencias por bultos con doble buffer de 512 B. El EP2 y EP4 son salidas (desde la PC). El EP6 y EP8 son entradas (hacia la PC).

La programación de este trabajo es con una entrada isocrónica de dos buffers con 512 B de capacidad, cada uno, definida en el EP2 y una salida por bultos de dos buffers de 512 B configurada en el EP8. Los otros EP's son deshabilitados. Sin embargo, EP1IN y EP1OUT se dejan configurados por defecto, ya que no interfieren en nada con el presente trabajo.

```
EP1OUTCFG = 0xA0;
SYNCDelay;
EP1INCFG = 0xA0;
SYNCDelay;
EP4CFG &= 0x7F;
SYNCDelay;
EP6CFG &= 0x7F;
SYNCDelay;
EP8CFG = 0xA0; //EP8 is DIR=OUT, TYPE=BULK, SIZE=512, BUF=2x
SYNCDelay;
EP2CFG = 0xD2; // EP2 is DIR=IN, TYPE=ISOC, SIZE=512, BUF=2x
SYNCDelay;
```

Como siguiente paso, se limpian las memorias FIFO de cualquier dato espúreo que contengan al momento del inicio del programa.

```
FIFORESET = 0x80 ;
SYNCDELAY;
FIFORESET = 0x02 ;
SYNCDELAY;
FIFORESET = 0x04 ;
SYNCDELAY;
FIFORESET = 0x06 ;
SYNCDELAY;
FIFORESET = 0x08 ;
SYNCDELAY;
FIFORESET = 0x00 ;
SYNCDELAY;
```

De esta forma, el controlador se encuentra listo para configurar las memorias asignadas a cada uno de los EP's. Se debe notar que en primer lugar se coloca 0x00 y luego el valor estipulado. Esto se basa en que el modo automático se prepara para transmitir ante un flanco ascendente del registro que lo habilita.

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

//setting on auto mode. rising edge is necessary
EP8FIFO CFG = 0x11; // & ~bmAUTOOUT; //at the end, auto mode is setted off
SYNCDELAY;
EP2FIFO CFG = 0x0D ;
SYNCDELAY;
```

Finalmente, se habilitan las interrupciones necesarias.

```
USBIE |= bmSOF;
```

Así, queda completa la inicialización y el dispositivo listo para enviar y recibir datos de forma automática.

### 3.3.2. Encabezado y declaraciones importantes

Para el correcto funcionamiento de este código, es necesario incorporar el encabezado que se observa a continuación.

```
#pragma noiv                                // No generar vectores de interrupción
#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    Rwuен;
extern BOOL    Selfpwr;

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;
```