Tal vez yo pondria que el objetivo es el diseño y la implementación de una interfaz USB para la transmisión de datos a alta velocidad en sistemas desarrollados en FPGA para aplicaciones cientificas.

Capítulo 1

Introducción

Yo uniría todos estos parrafos

El presente informe busca dar a conocer al lector las tareas y actividades desarrolladas por el autor, a fin de dar cuenta sobre lo realizado en el marco del Trabajo Final de la carrera Ingeniería Electrónica, dictada en la Facultad de Ingeniería de la Universidad Nacional de San Juan.

El objetivo principal de este trabajo es la implementación de un sistema de comunicación mediante la norma USB 2.0 para sistemas científicos y/o tecnológicos desarrollados en FPGA.

A lo largo de este trabajo, se busca que el lector comprenda el problema que se pretende resolver, la configuración del sistema propuesto, los fundamentos que dan base a la dicha configuración y el modo de uso del sistema.

En este...

Para ello, este Capítulo explica, en primer lugar, la motivación, es decir, se busca aclarar la temática que se busca resolver. Luego, se detallan los objetivos que persigue este trabajo. Seguido a esto, se otorga un esquema que describela solución planteada y se justifica el protocolo elegido. A continuación, el lector puede conocer la organización del presente informe. Finalmente, se repasan algunos conceptos importantes de la norma USB que luego se utilizan en el trabajo desarrollado.

1.1. Motivación

Yo sacaría estos parrafos

La información es el resultado de recopilar, ordenar y procesar un conjunto de datos, de forma tal que permitan adquirir mayor conocimiento sobre un asunto determinado y otorguen un significado mayor que cada uno de los datos por separado. Por ejemplo, Si un caprintero quiere medir la distancia de una barra de madera, utilizará una cinta métrica. Se denomina cinta métrica a una lámina metálica que posee marcas graduadas. Esta graduación, se realiza comparando la lamina metal con una barra patrón que indica que la distancia indicada se corresponde con la convención de distancia.

El carpintero compara el tamaño de las patas de la mesa con la cinta métrica, que a su vez, posee registrada su distancia en función del patrón. 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.

La Ciencia como fuente de datos

La Ciencia es un conjunto de técnicas y procedimientos que, a través de un método científico, busca adquirir, descubrir y/o desarrollar nuevo conocimiento. Se puede deducir, entonces, que la ciencia produce, de forma fundamental, datos, que luego de un procesado se transforman en información. A su vez, el estudio detallado de esta información genera conocimiento.

Cuando se nombra la palabra ciencia, se hace referencia a un conjunto conformado por diferentes objetos de estudio. El objeto de estudio es el sujeto a través del cuál se da la principal clasificación de las diferente ramas de la Ciencia: las Ciencias Sociales estudian las relaciones humanas y las Ciencias Naturales dedicar[illegible] naturaleza. Dentro del último grupo encon[illegible] rama más particular de la naturaleza, como [illegible] así se puede encontrar muchas más clasificaciones de ciencias, incluso como ramas englobadas por las anteriormente nombradas.

Un sensor en la industria es un objeto capaz de variar una propiedad ante magnitudes físicas o químicas, llamadas variables de instrumentación, y transformarlas con un transductor en variables eléctricas.[CITA]

Las herramientas que adquieren datos

El grado de avance

[illegible]ado de avance que ha experimentado la tecnología en general, y la electrónica en [illegible], gracias a la industria de los semiconductores, permite que la producción científica pueda adquirir una gran cantidad de datos.

Actualmente, se encuentra en desarrollo un gran número de sensores que permitan obtener flujos crecientes de datos. Se denomina sensor a los dispositivos que adquieren valores registrables en función de alguna propiedad física. Los sensores constan de, al menos, un transductor, es decir, un elemento que transforma una magnitud física en otra, y de un sistema que convierta esta magnitud en una señal eléctrica útil, si no la otorga el transductor mismo.

Uno de los desarrollos que se encuentra en boga es el de sensores y sistemas que adquieran imágenes de diferentes tipos. Desde el punto de vista digital, una imagen es un arreglo bidimensional de números, los cuales pueden ser exhibidos en una pantalla en forma de intensidad y colores de luz. Por esto, un sensor de imagen puede estar compuesto, bien por un arreglo bidimensional de transductores, por un transductor que es simultáneamente desplazado y leído y/o por una combinación de ambos métodos. En todos los casos, es de suma utilidad que la lectura de imágenes sea realizada en el menor tiempo posible ya que cada imagen conlleva una cantidad de datos que no es despreciable.

Sobre circuitos de alta velocidad

transmitir

Desde un punto de vista electrónico, para poder m[illegible] gr[illegible]des volumenes de datos en forma digital, se requiere de circuitos que sean capaces de manejar niveles de tensión que se modifican con alta velocidad. Esto no es trivial, ya que a frecuencias de funcionamiento elevadas

Tal vez yo pondria algo calarando que cuando las logitudes de onda de las señales a transmitir son comparables con las dimensiones de los circuitos comienzan a aparecer capacidades e inductancias parasitas no contempladas.....

parasitas, no contempladas en

se evidencian capacidades no contempladas en diseño que perjudican el desempeño de los sistemas.

En gran medida, la incorporación y evolución de microcontroladores permite obtener una captura y obtención cada vez superior de datos. Sin embargo, al poseer una estructura rígida y su capacidad de procesamiento está limitada a una instrucción por vez, no es del todo adecuada para el desarrollo de nuevos dispositivos y técnicas de medida especializadas. En ciertos casos, la posibilidad de realizar cientos de operaciones es excesivo para aplicaciones muy específicas. Una solución óptima, sin considerar los costos asociados a esto, sería el desarrollo de un circuito integrado de aplicación específica (ASIC) que resuelva en forma precisa el problema. Sin embargo, cuando sí se considera el costo asociado a este enfoque, se vuelve una solución totalmente ineficiente, del orden de las decenas de miles de dolares.

Otro enfoque,

El siguiente enfoque, óptimo al criterio de este trabajo, es la utilización de FPGAs para realizar el desarrollo. Al poseer, este tipo de herramientas, la posibilidad de conectar, adquirir y procesar en paralelo y a alta velocidad, incorpora una gran ventaja con respecto a un procesador. A su vez, al ser programable, se adapta mejor a una solución específica. En cuanto al costo de un FPGA, van desde decenas a centenas de dolar, es decir, dos ordenes de magnitud más económico que un ASIC.

dolares

Citar

Existen diversas publicaciones en donde se observa el uso de FPGAs para la implementación de sistemas que producen imágenes. Por ejemplo, el desarrollo de un sensor de radiación utilizando una cámara CMOS comercial. Para ello, los autores utilizaron un FPGA para configurar y transmitir imagenes a un grabador de video a través de puerto UART. Esto permitió adquirir una imagen accionando un disparador realizado con un pulsador[1].

reflexiones?

Se denomina ultrasonografía a la técnica de adquirir imágenes basandose en rebotes de ultrasonido. Sus aplicaciones son múltiple, en las que se destaca el diagnóstico médico, ya que es una técnica no invasiva y sin riesgos de radiación ionizante sobre el paciente. Un trabajo reciente desarrolló un sistema de ecografía médica con bajo costo utilizando un FPGA[2]. El autor también presentó un algoritmo realizado y probado en PC. Luego se implementó e en una FPGA.

Yanagisawa, entre otros, desarrolló un sistema con telescopios pequeños para explorar objetos de campo cercano con la finalidad de monitorear cuerpos celestes que puedan colisionar con el planeta[3]. Los autores utilizan la velocidad de los circuitos implementados en FPGA para minimizar el tiempo de adquisición.

El procesamiento de datos

La obtención de datos por si misma no otorga información. Para ello, es probable que un gran flujo de datos requiera de un procesamiento y análisis exhaustivo de los mismos. La invención y evolución de las computadoras, como así también el desarrollo de nuevos algoritmos, dan lugar a procesamiento de datos cada vez más complejos en tiempos mucho menores.

Las primeras ENIAC, computadora de propósito general desarrollada en el año 1946 para el cálculo de tablas balísticas de las fuerzas armadas estadounidenses, podía ejecutar 20 operación cada 10 µs, es decir, podía ejecutar instrucciones con una frecuencia máxima de 200 kHz. A su

vez, tuvo un costo aproximado de U$S 500.000, pesaba 5 t y consumía 175 kW.

En contraste con aquello, es posible conseguir en el mercado actual, computadoras que pesan unidades de kilogramos, pueden ejecutar instrucciones en unidades de nanosegundo, (5 ordenes de magnitud menos), consumen apenas centenas de watts y cuestan algunos cientos de U$S. A tal punto ha evolucionado esta tecnología que se encuentran presente computadoras muy potentes en casi cualquier laboratorio, oficina u hogar.

Esta potencia de cálculo, ayudado por el desarrollo de nuevos métodos y algoritmos de cálculo, permiten a los investigadores procesar miles de datos en tiempos muy reducidos, ayudando al análisis de los mismos y la obtención de información.

La comunicación entre los sistemas productores y los procesadores de datos

La generación de datos y el procesamiento de lo mismos, según el enfoque del presente trabajo, se da en sistemas diferentes. Estos sistemas requieren, de una conexión a través de la cual los datos puedan ser llevados de un lugar a otro. Se torna de suma utilidad, entonces, proveer una comunicación efectiva y robusta que permita mover grandes volúmenes de datos en poco tiempo, y de esta forma facilitar los tiempos de desarrollo, las pruebas y depuración de sistemas.

Implementar un sistema de comunicación en una FPGA, si bien no es trivial, puede ser resuelto de muchas maneras, quedando a criterio del desarrollador utilizar algún sistema de comunicación estándar, o bien, diseñar uno propio. Sin embargo, desde el punto de vista de una computadora, las comunicaciones se vuelven un poco más estrictas y acotadas a los puertos y señales que puede manejar el equipo, conforme el fabricante haya establecido.

Es por eso que este trabajo trata de implementar una comunicación entre una computadora personal y una FPGA, utilizando un protocolo estándar, que esté disponible en cualquier computadora comercial y que posea una tasa de bit suficiente para poder transmitir imagenes.

Objetivo

1.2. Propuesta del Trabajo Final

Dada la motivación, se puede decir a priori que el objetivo del presente trabajo es la implementación de una efectiva comunicación entre una Computadora Personal (PC) y una FPGA. Sin embargo, este objetivo tan amplio se plasmará de forma más concreta y detallada en la Sección 1.3 del presente Capítulo.

El dato de diseño más relevante es el poder transmitir imágenes por la comunicación a implementar. Pero, ¿Cuántos datos son suficientes para este propósito? Se toma como base de diseño el sensor que utiliza Pérez en su Tesis de Maestría [4], una cámara para adquirir imágenes monocromáticas de código MT9M001C12STM, comercializado por Actina Imaging [5]

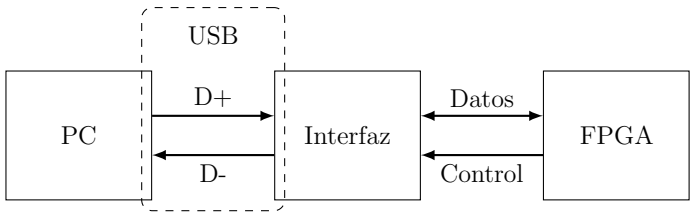

Figura 1.1: Esquema propuesto para implementar la comunicación

que transmite datos a una tasa de $48\,\mathrm{Mbit\,s^{-1}}$.

Además la implementación debe ser compatible con equipos convencionales que se consigan fácilmente en el mercado y no posean especificaciones fuera de las de uso doméstico. Esto se cumple hoy en día solo con dos tipos de puertos: Ethernet, dedicado principalmente a conexión de redes y USB, especializado en periféricos.

Si bien se puede implementar la comunicación vía Ethernet, cumpliendo con las especificaciones propuestas, es muy probable que el único puerto que posea la PC se encuentre conectado a la red y se necesitará una infraestructura mayor para lograr una efectiva comunicación. Por tanto, USB se observa como una solución óptima.

A su vez, USB presenta, al día de la fecha, con tres versiones de su norma, con diferentes velocidades: Citar

- La version 1 posee dos revisiones, 1.0 fue lanzada al mercado en el año 1996 y 1.1 que se presentó en Agosto de 1998. Ambas revisiones alcanzan una tasa máxima de $12\,\mathrm{Mbit\,s^{-1}}$.
- USB 2.0 fue presentado en Septiembre del 2000 y es capaz de transmitir $480\,\mathrm{Mbit\,s^{-1}}$.
- Dependiendo de la revisión, desde USB 3.0, lanzada al mercado en 2011, que transmite $5\,\mathrm{Gbit\,s^{-1}}$, hasta $20\,\mathrm{Gbit\,s^{-1}}$ de la revisión USB 3.2, que fue presentada en 2017.

Con estas velocidades, es suficiente para el propósito del siguiente trabajo, la implementación una comunicación USB 2.0, con $480\,\mathrm{Mbit\,s^{-1}}$.

El alumno sabe que es posible implementar una comunicación USB completa a través de una FPGA. Sin embargo, esto sería muy costoso en términos de tiempos de desarrollo y de recursos de FPGA disponibles para la implementación de otros sistemas, los cuales son el objetivo de la comunicación.

Se plantea, entonces, un esquema como el que se observa en la Figura 1.1 en la cual se utiliza una interfaz externa al FPGA. La comunicación USB propiamente dicha será efectuada entre la interfaz y la PC, mientras que se plantea una comunicación diferente entre la interfaz y el FPGA. Este último, por su parte, tendrá la tarea de realizar el control de esta comunicación.

1.3. Objetivos

1.3.1. Objetivo Principal

El objetivo del presente trabajo es obtener una comunicación USB 2.0 de alta velocidad entre una PC y un FPGA.

Esta comunicación debe realizarse y documentarse de forma tal que pueda ser usado posteriormente en aplicaciones científicas desarrolladas con FPGA's.

1.3.2. Objetivos Particulares

Para la consecución del objetivo general, se deben cumplir los siguientes objetivos particulares:

- Comprender el funcionamiento del protocolo USB.
- Seleccionar los componentes a utilizar.
- Configurar los componentes seleccionados.
- Desarrollar un núcleo en VHDL que sirva de interfaz.
- Diseñar e implementar la interconexión de los componentes seleccionados.
- Verificar el sistema desarrollado.
- Desarrollar un documento que explique el modo de uso del código VHDL utilizado.

1.4. Estructura del Informe

El presente informe se divide en 2 bloques principales: uno referido al desarrollo del sistema y el siguiente a su forma de uso y verificación.

Dentro del bloque referido al desarrollo del sistema, se encuentran los primeros 5 capítulos:

1. **Introducción:** En este capítulo se intenta exponer lo que motiva el presente trabajo, la propuesta que da solución a la motivación, el objetivo y alcance que el trabajo busca y la estructura del mismo. Se brindan, además, conceptos importantes de la norma USB que son significativos para los objetivos de este trabajo.
2. **Elección de las herramientas para la realización de la interfaz:** Se describe aquí todas las herramientas de las que se vale este trabajo para cumplir con os objetivos propuestos.
3. **Programación y configuración de la interfaz PC-FPGA:** Se presenta la arquitectura, configuración y código desarrollado para el presente trabajo, como así también las herramientas específicas provistas por el fabricante, que facilitan el desarrollo.

Esto lo moveria al final del capítulo

4. **Desarrollo del sistema maestro para la comunicación entre la FPGA y la interfaz Cypress:** Este capítulo detalla lo desarrollado para implementar la comunicación entre la FPGA y la interfaz. Se expone una maquina de estados descrita en VHDL y sintetizada en FPGA. También se describe un circuito impreso realizado para conectar ambas partes.

5. **Depuración y verificación del sistema:** Se desarrolla las tareas desarrolladas a fin de realizar las depuraciones del sistema y la verificación del cumplimiento de las especificaciones.

1.5. El protocolo USB

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

Hasta ese momento, el gran ecosistema de periféricos, sumado a los nuevos avances y desarrollos, hacia muy compleja la interoperatividad 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 cuando se comparaba una PC, se requería cambiar el teclado, el mouse y/o algún periférico específico. Esto también complicaba a las mismas empresas productoras, por que la introducción de un nuevo sistema requería de mucho soporte extra para poder conectar todo lo ya existente.

Todo esto, quedó saldado con el aparición de la norma USB, que debido a la gran cuota de mercado de sus desarrolladores, fue adoptado en forma rápida y se transformó en la especificación por defecto 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 solicite especialmente un puerto determinado. Así, cualquier PC nueva disponible 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 (o un dispositivo con software y hardware capaces de incorporar los drivers necesarios) y cualquier periférico a ella acoplada será un esclavo[6].

tres partes:

Para describirlo es conveniente diferenciar tres partes. Una capa física, en donde se definen los componentes que intervienen, una capa de protocolo, en donde se define el formato, el marco en el que son enviados los paquetes, como se direccionan 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.

Para describir el funcionamiento del portocolo es conveniente diferenciar tres partes:

Tal vez sería más claro si se explica a que se refiere con formato y marco

1.5.1. Capa física

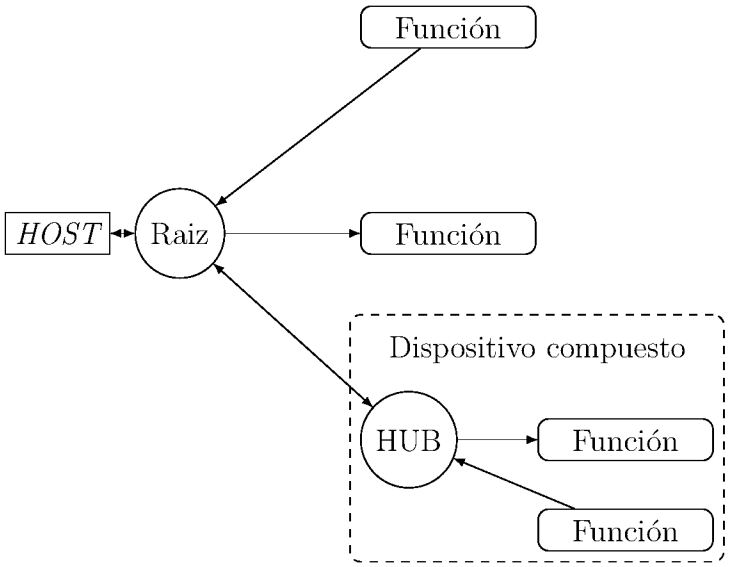

Figura 1.2: Topología de un sistema USB

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 de la interfaz que se utiliza en este trabajo. A su vez, 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 estos, su costo es despreciable con respecto a cualquier tipo de desarrollo en ese sentido y son de una muy buena calidad, es decir que todos cumplen con lo que la norma establece. Sí, se describen los dispositivos físicos y su categoría, según 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 que responden a sus directivas. Por esto, el elemento central de cualquier comunicación USB es el *HOST* (director o anfitrión, por su traducción de la voz inglesa). Él es el que posee el *Host USB Controller*[6]. Esto quiere decir que tiene la capacidad de registrar los dispositivos acoplados, asignarles dirección, colocar los paquetes de salida y/o llegada en sus respectivas listas y servilos a los procesos que utilizan esta comunicación. Además, 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 la mayoría de los casos, el *HOST* es una PC, auqnue también puede ser cualquier dispositivos "inteligente" como un smartphone.

En el otro extremo de la comunicación, se encuentran lo que la norma denomina *funciones*[6]. Las *funciones* son todos los periféricos que actúan como fuente o sumidero de información. Es decir, en caso de periféricos de entrada, serán una fuente de datos 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 denominan *dispositivos 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 *funciones*, 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 las *funciones*.

La Figura 1.2 muestra la topología típica de un sistema USB. En ella, se observa el *HOST* como un rectángulo, las *funciones* como rectángulos con los bordes redondeados y los distribuidores como círculos. Se puede notar que el *HOST* posee un distribuidor propio llamado *Raiz* en el cual se conectan todos las *funciones* y distribuidores. Cada *Función*posee una única dirección. Pueden existir dispositivos que posean funciones diversas con un mismo encapsulado, como por ejemplo un auricular que tenga micrófono incorporado. Este dispositivo, tendrá un *HUB* que concatena dos *funiones* diferentes.

1.5.2. Capa lógica

Desde el punto de vista lógico, cada dispositivo es visto por el *HOST* como un extremo (EP, del inglés, *endpoint*) independiente, que posee solo un modo de comunicación, es decir, el protocolo se comunicara solo por un tipo de transferencia y en un único sentido con cada *EP*. En otras palabras, USB registra un periférico de entrada/salida como un *EP* de entrada y otro de salida en forma independiente.

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 le *HOST* y cada uno de los extremos existe un tubo (la norma en ingles 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 *EP* por medio de estos tubos. A los fines del usuario, esto es lo importante, por cuanto se solicita acceso al bus y define en que 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 el bus y el posterior envío de datos.

1.5.3. Capa de protocolo

En la capa de protocolo, la especificación de la norma detalla cómo se compone un cuadro y cómo deben ser estructurados los paquetes para que sean efectivamente enviados a través del sistema. Cada mensaje que se intercambia por el bus se denomina paquete. Cada paquete puede poseer hasta cinco campos, dependiendo del tipo de paquete que sea enviado a través

del sistema y de quien sea el remitente. A su vez, cada paquete comienza con una señal de sincronismo (*SYNC*) y un Comienzo de Paquete (SOP de *Start-of-Packet*), y terminan con una señal de Fin de Paquete (EOP por *End-of-Packet*).

Por otra parte, los paquetes está temporalmente encapsulados en cuadros. Cada cuadro posee un Comienzo de Cuadro (SOF, *Start-of-Frame*) y posee una duración de 1 ms, hasta el próximo SOF. En las comunicaciones de alta velocidad, es decir, aquellas que poseen una tasa de bit de 480 $\mathrm{Mbit\,s^{-1}}$. Se subdivide un cuadro en 8 micro-cuadros de 125 µs cada uno.

Campos de Paquetes

Cada paquete contiene un campo denominado identificador de paquete (PID del inglés *Packet Identifier*). El PID indica el tipo de paquete que se está enviando y, como consecuencia, el formato de cada uno, es decir, que campos acarrea y que control de datos utiliza.

A su vez, cuando el host solicita algo al sistema, lo realiza a través del denominado campo de dirección. Este campo, se compone de dos partes, la primera es el campo de dirección de la función y el segundo es la dirección de extremo.

Los mensajes de datos, poseen un campo dedicado de forma específica a los datos. Puede poseer un numero entero de bytes, desde 0 a 1024.

Para corroborar el envío de datos, USB utiliza verificación de redundancia cíclica (CRC o *Cyclic Redundancy Checks*). Los paquetes especiales y los de token poseen un verificador CRC5, es decir, de 5 bits, cuyo polinomio generador es:

$$G(X) = X^5 + X^2 + 1$$

Por su parte, los paquetes de datos, poseen CRC16, ya que pueden llegar a ser bastante extensos. En su caso, el polinomio generador está dado por:

$$G(X) = X^{16} + X^{15} + X^2 + 1$$

Existe un campo relativo a los cuadros temporales, que se denomina campo de número de cuadro. Este es enviado por el *HOST* en cada SOF y es incrementado a cada cuadro. Los micro-cuadros también poseen un número de cuadro, sin embargo, este es aumentado solamente cada 8 micro-cuadros, es decir, el número se incrementa cada 1 ms y se repite durante los 7 micro-cuadros de 125 µs, en comunicaciones de alta velocidad.

Formato de paquetes

- **Paquetes Token:** A través de este tipo de paquetes el host envía las directivas a los distintos periféricos. Estas directivas pueden ser IN, cuando solicita datos de un periférico; OUT, cuando se dispone a enviar datos hacia una *función*; SOF, que indica el inicio de cada cuadro, para que cada función se sincronice y SETUP, cuando va a enviar un paquete de configuración a algún extremo.

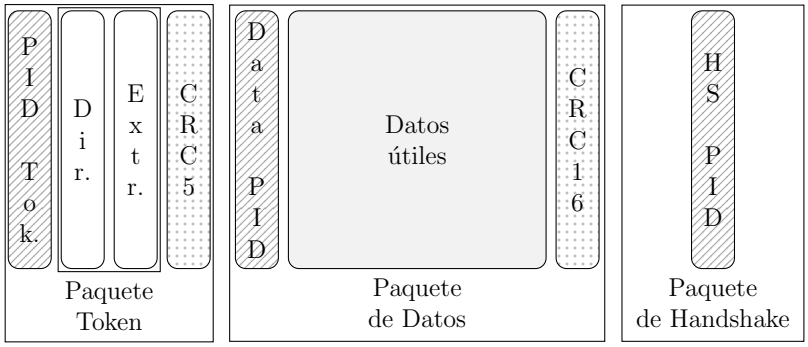

Figura 1.3: Distintos tipos de paquetes USB

- **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.
- **Paquete de *Handshake*:** Es enviado por el receptor del mensaje y le informa al emisor el estado de la transferencia. ACK significa que el paquete fue recibido sin errores; NAK, los datos poseen error o el emisor no puede enviar; la señal STALL quiere decir que la solicitud no es soportada o que el extremo está detenido; NYET implica que no hay respuesta aún por parte del receptor.
- **Paquetes Especiales:** Son paquetes con propositos específicos. Con ellos se señalan preambulos emitidos por el *HOST*, se informan errores, se solicitan mensajes divididos en diferentes paquetes y se intercambia señales de ping para conocer el estado de los componentes del sistema.

Cada uno de los tipos de paquetes posee un formato específico, tales como se muestran en la Figura 1.3. En ella se observa que los paquetes de token envían un PID, una dirección y un CRC5; los paquetes de datos, se componen de un PID, los datos transmitidos y un CRC16; en el caso de los paquetes de Hanshake, solo el PID indica que tipo de mensaje se envía. Los paquetes especiales no se detallan ya que el formato es muy variable, en función del paquete.

1.5.4. Flujo de datos

Como se menciono anteriormente, el host envía un toquen 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 numero 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 de la función que debe responder o recibir el mensaje enviado por el bus y los verificadores CRC5.

Luego, el siguiente paquete posee los datos que se transfieren, precedido por un PID de datos, y verificadores CRC16. Este paquete es transmitido por el emisor de los datos. Finalmente, el receptor devuelve un paquete de *handshake*, indicandole al emisor si el transferencia fue efectiva o no.

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 transferencias 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 transferencias está destinado a datos que son 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.

No posee un control de errores muy sofisticado, más que un simple código CRC, pero no existe mecanismo de retransmisión de datos ni handshake entre los *EP* 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 dispondra encada cuadro de una fracción de ancho de banda para las trasnmisiones 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 trasmitida por cada cuadro, de hasta 64 bytes en sistemas de alta velocidad.

1.6. Resumen del capítulo

En el presente capítulo se expuso la necesidad de la elaboración de un sistema de comunicación que permita la transferencia de datos entre una PC y un FPGA para ser utilizados por sistemas implementados con este último dispositivo. Se planteó una solución utilizando una interfaz comercial que sirve de intermediario entre estas herramientas y se brindó una justificación del empleo del protocolo USB 2.0 de alta velocidad como la implementación óptima del sistema. Se presentó también la estructura del presente informe y se dieron algunos detalles relevantes para este trabajo de la norma USB.

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 se 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 trasceptor, se encarga de adecuar los valores de tensión, las frecuencias, impedacias 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 ordenes 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

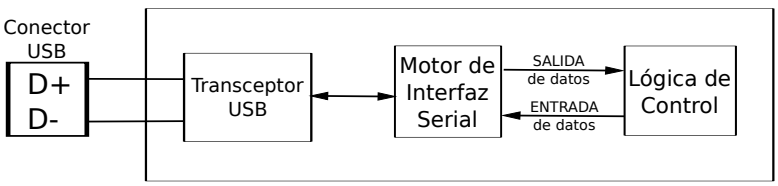

Figura 2.1: Etapas fundamentales de un dispositivo USB

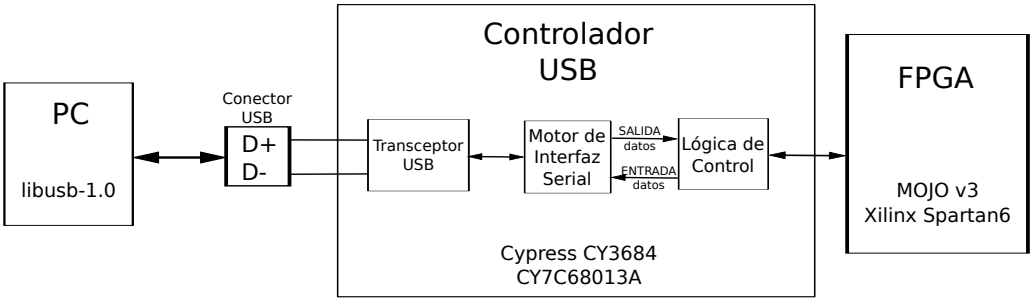

Figura 2.2: Vista general del sistema propuesto

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 trasceptor 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

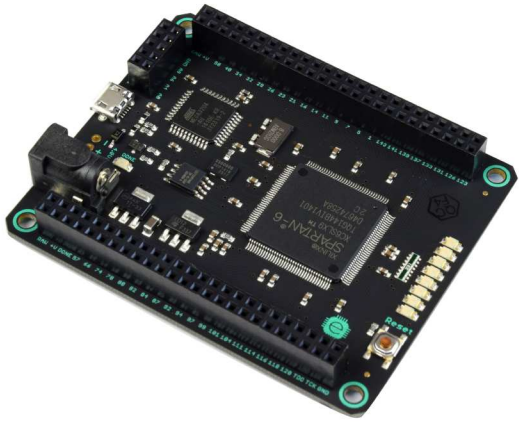

Figura 2.3: Placa de prototipado rápido MOJO v3, diseñada por Embedded Micro

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 perfiéricos 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 trasmitida 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 ($Serial Peripherical Interface$, 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.

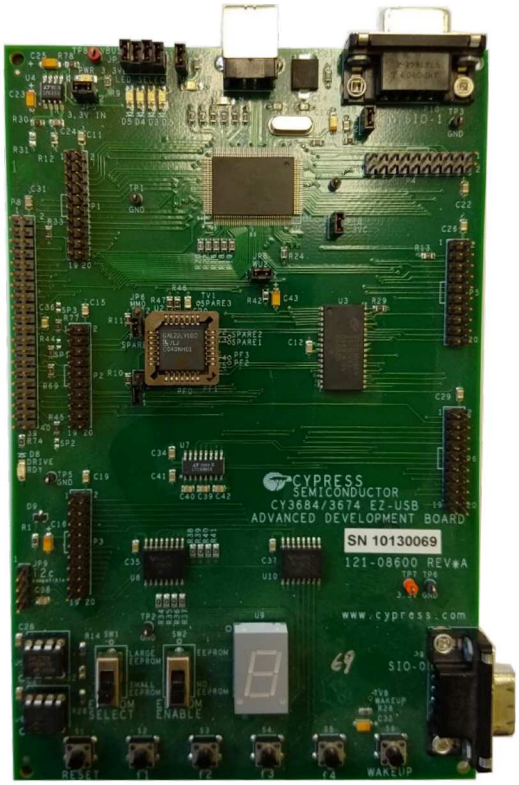

Figura 2.4: Circuito impreso principal del kit de desarrollo CY3684 FX2LP EZ-USB

2.2. Elección de la interfaz PC-FPGA

Como interfaz entre la FPGA y la PC se utilizó kit de de desarrollo CY3684 FX2LP EZ-USB Development Kit de Cypress Semiconductor,la que se observa en la Figura 2.4. Esta placa posee como núcleo el controlador USB CY7C68013A, circuito integrado que posee todas las herramientas necesarias para realizar la interfaz, como así también un buen número de periféricos que permiten al desarrollador realizar pruebas y depuración.

Entre estas, se destacan 6 pulsadores, de los cuales cuatro se utilizan para proposito general, uno para reestablecer los valores por defecto de la placa y uno para enviar señales de suspensión y reestablecimiento del programa actualmente cargado en el microcontrolador, lo que coloca al sistema en modo bajo consumo de energía. A su vez, posee dos memorias EEPROM que sirven para cargar firmware y archivos de configuración del sistema, un display de 8 segmentos, 4 leds de multiple propósito, dos puertos UART, una salida de pines compatible con puertos ATA y 6 puertos de 20 pines que se utilizan para la conexión hacia el chip núcleo. Como soporte para el firmware, posee también un bloque con 64 kB de memoria SRAM.

Se selecciona este controlador como interfaz con el objetivo de utilizar la menor cantidad de los recursos configurables de la FPGA, de forma tal que estos queden disponibles para el desarrollo de sistemas científicos complejos que sean necesarios a posteriori.

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, ala 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.
- Agnótisco 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 de la interfaz PC-FPGA

Un puerto USB posee una estructura fija: 4 contactos; dos de alimentación y dos datos. A su vez, el Protocolo USB es muy específico en cuanto a los niveles de tensión, la frecuencia de comunicación, las cadenas de bits clave que deben enviarse, entre muchas otras cosas que detalla la Norma USB 2.0[**?**].

Un FPGA, en contraste con lo anterior, es un dispositivo electrónico que posee en su interior una gran cantidad de elemento lógicos programables. Esto permite la síntesis de casi cualquier circuito lógico. Dicha síntesis puede poseer la cantidad y disposicion de puertos como pines posea el FPGA, exceptuando algunas entradas específicas para el funcionamiento del FPGA. Esto configura una verstilidad muy grande que permite ajustar un sistema desarrollado dentro del FPGA a la medida necesaria.

El nexo entre los datos que se producen en sistema en el que se desenvuelve un FPGA y un puerto USB, se lo denomina interface. Esta última, en el presente trabajo, está compuesta por un circuito integrado particular, cuya codificación comercial es CY7C68013A, fabricado por Cypress Semiconductor. Este componente electrónico pertenece a la familia de controladores FX2LP del catálogo de integrados EZ-USB. El chip CY7C68013A se encuentra incorporado el kit de desarrollo CY3684 EZ-USB Development Kit, provisto por el fabricante.

Un kit de desarrollo es un conjunto de herramientas que permiten elaborar soluciones electrónicas que requieren un componente en particular. Además, cuenta con algunos dispositivos genéricos que posibilitan emular el sistema que utilizará el componente central del kit. En el caso del kit que se utiliza en este trabajo se descompone en dos grandes grupos: hardware y software. En la parte de hardware se incluye de un circuito impreso (PCB) que posee soldado, el integrado que nos servira de interfaz y otros componentes que se describen en el Capítulo 2.

En lo que a software se refiere, Cypress incorpora en el kit de desarrollo un código marco que posee escritas todas las funciones y registros que ejecutan las tareas que el sistema necesita para llevar a cabo la comunicación USB. Además se cuenta con herramientas de software que permite escribri, compilar y cargar el programa que se ejecuta en el controlador FX2LP.

En este capítulo se desarrolla la arquitectura de los integrados FX2LP y se abordan los módulos más relevantes para este trabajo. También se explican las herramientas del kit que fueron utilizadas para la elaboración del firmware y, finalmente, los aspectos más sobresalientes del firmware que ejecuta el CY7C68013A, y establece el funcionamiento de la comunicación USB.

3.1. Arquitectura FX2LP EZ-USB

El núcleo del Kit de Desarrollo FX2LP EZ-USB es un CY7C68013A. Este circuito integrado es un controlador USB y pertenece a la serie FX2LP del catálogo de integrados EZ-USB comercializado por Cypress Semiconductors. Su arquitectura se presenta en la Figura 3.1.

La serie de controladores FX2LP se caracteriza por brindar una conexión USB 2.0 de alta velocidad y bajo consumo energético. Está diseñada, preferentemente más no exclusivamente, para periféricos con autonomía limitada. Se integra un controlador USB completo. Esto incluye un transceptor USB, un Motor de Interfaz Serie (MIS), buffers de datos configurables, un microcontrolador 8051 mejorado y una interfaz programable hacia los periféricos implementada con memoria tipo FIFO (*FirstInFirstOut*; Primero Entrado, Primero Salido). Además posee un PLL con divisor configurable a través de los cuales proveen las señales de reloj adecuadas para el correcto funcionamiento del sistema.

variante 1 El usuario puede trasmitir datos desde y hacia el anfitrion a traves del mismo puerto USB, o bien via RS-232. Para comunicarse con sistemas perifericos se puede aprovechar el puerto I^2C, la interfaz de proposito general, que actua como maestro y a la cual se le puede acoplar un periferico esclavo, y/o las memorias FIFO en modo esclavo que puede ser conectada a un sistema maestro. Esto brinda muchas alternativas de conexion, desde puertos estandar, como ser ATA, PCMCIA, EPP, etc. hasta dispositivos personalizables como DSP's y FPGA's.

variante 2
El flujo de datos posee dos puntas entre las cuales el controlador hace de nexo. Para ello

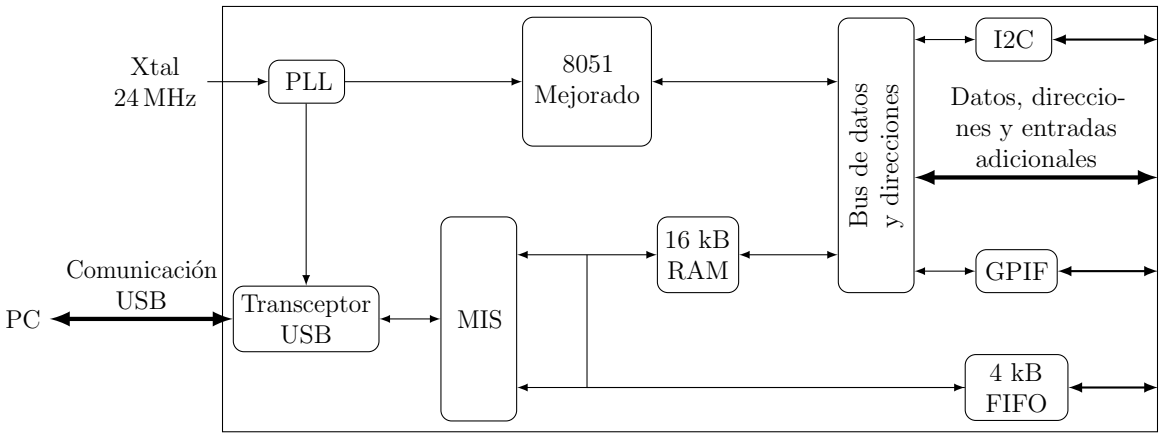

Figura 3.1: Arquitectura FX2LP

necesita poder comunicarse tanto con el HOST como con los perifericos.

El intercambio de informacion con el HOST se lleva a cabo a traves del mismo puerto USB, objetivo principal de este trabajo. Sin embargo, tambien posee dos puertos UART que facilitan en gran medida la tarea de depuracion del desarrollo.

En cuanto a la interfaz con uno o mas perifericos, el controlador posee un puerto I^2C, una interfaz de proposito general (GPIF), para sistemas que necesitan ser comandados en forma externa; y una interfaz con memorias FIFO esclavas, a traves de las cuales se puede conectar sistemas que cumplen un rol activo en el envio y recepcion de informacion.

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

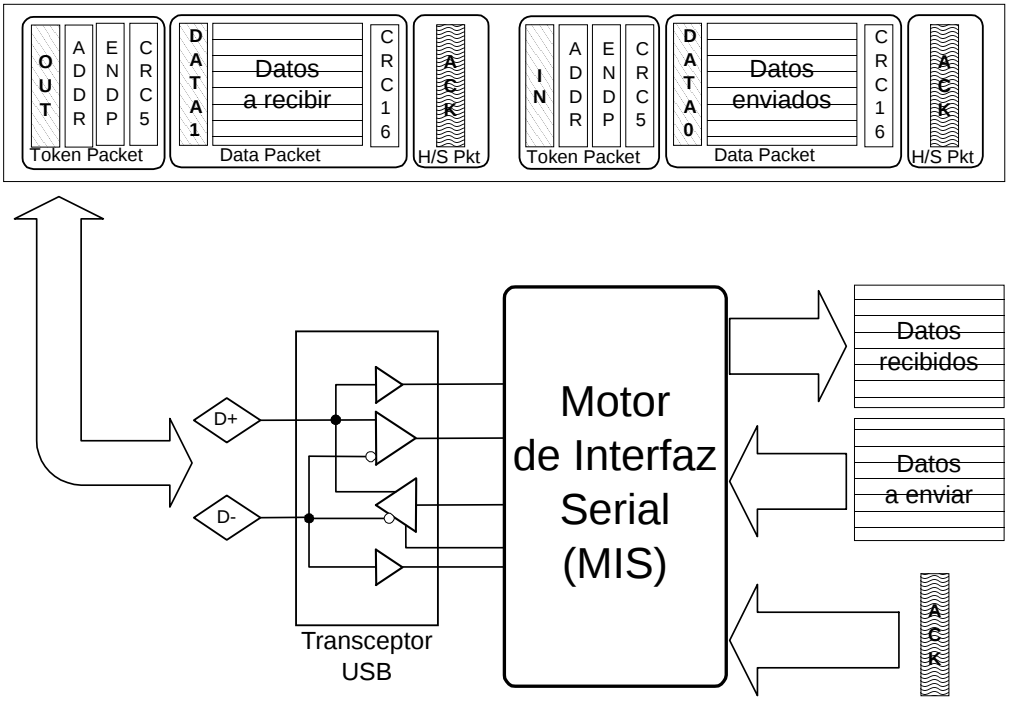

Figura 3.2: Implementación del enlace USB realizado por el EZ-USB (se copiaria con tikz para mejorar prolijidad)

La comunicación USB entre el controlador FX2LP y la PC se realiza a través del transceptor, unido al MIS. Con el objetivo de intercambiar datos, el firmware solo debe colocar o extraer los datos de buffers programables y modificar las banderas de handshaking. En forma automática,

el MIS se encargan de empaquetar, enviar, recibir y 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. El transceptor codifica y decodifica todo a nivel físico.

La Figura 3.2 muestra la función del MIS. Toma los datos colocados los buffers de extremos, agrega todo el encabezado y la cola y, finalmente, coloca el registro de handshaking. Esto último, se observan como ACK (abreviación del ingles *acknowledge*, que significa reconocer, aceptar o agradecer)en la Figura 3.2. En el extremo del controlador, estas banderas se colocan en un registro especial que indica si el sistema está disponible, si los datos fueron colocados o leídos, dependiendo el caso tratado.

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 identificable 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 finaliza la información. En ingles, el termino extremo se escribe endpoint, por lo que, en adelante, cuando se hable de ellos se abreviara 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 USBy 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 y 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. No se admiten en estos EPs paquetes de control.
- Cantidad de buffers: Dependiendo del extremo, puede ser dos, tres o cuatro buffers por extremo.

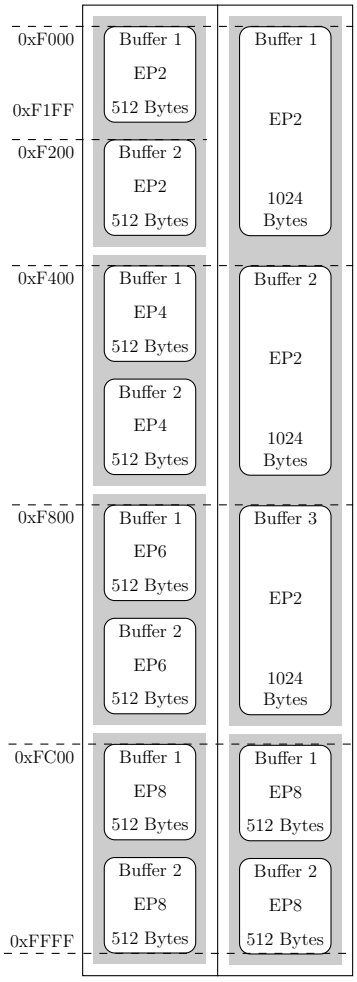

Figura 3.3: Buffers de extremos con sus direcciones de memoria. El cuadro de la izquierda muestra la configuración por defecto. El derecho, la implementada en este trabajo.

- Habilitación: Se debe indicar al sistema si los extremos se usan o no. El EP no valido, no responderá a un pedido de entrada o salida.

La Figura 3.3 muestra solo dos de las posibles configuraciones de los EPs. 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 que 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 o cuádruple buffer, se agrega una o dos porciones mas de memoria a la reserva, respectivamente. 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.

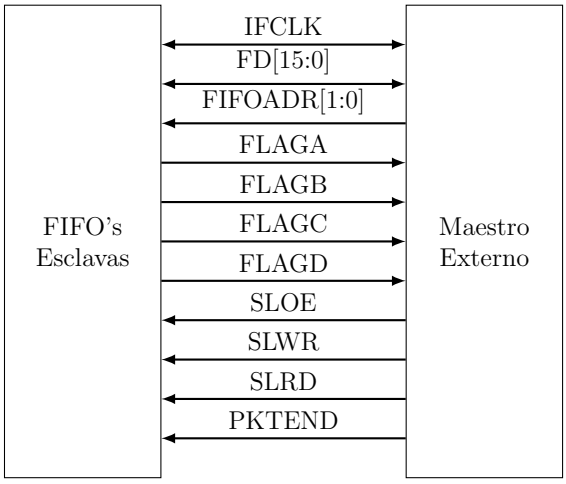

Figura 3.4: Puertos de interfaz entre las FIFO's y un maestro externo

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

- IFCLK: señal de reloj. No es necesario en caso de conectar la interfaz en modo asincrónico. La señal de reloj puede ser provista 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, en forma independiente para cada EP.

- FIFOADDR[1:0]: puerto de direcciones. A través de el 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. También pueden indicar el estado sobre una memoria en particular o sobre la 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

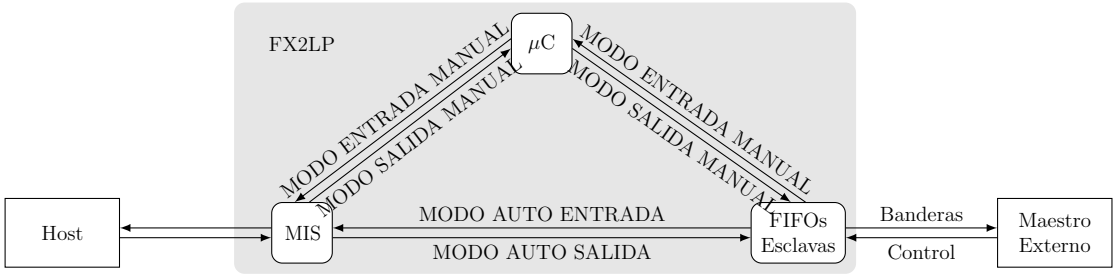

Figura 3.5: Modos de conexión de la memoria FIFO, el microntrolador y el MIS

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 *anfritión-céntrico* de la comunicación USB, en donde el principal componente es el anfitrión USB 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 la implementación del software a la persona que lo produce. Cypress denomina, a estas soluciones, 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 ejecutables que graban 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 se utiliza en este trabajo, a continuación se desarrollarán las herramientas más destacadas.

3.2.1. Framework Cypress

Con la intención de dotar al desarrollador con una herramienta que le potencie la velocidad de diseño, Cypress provee una plantilla de codigo 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 las funciones que el microcontrolador debe llevar a cabo para efectuar la comunicación USB y algunas que permiten interactuar con la placa de desarrollo CY3684. 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 TD_Init() y TD_Poll(), las funciones 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 USB, asignarle dirección y establecer los parámetros.
- ezusb.lib: biblioteca que implementa 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.

- usbjmptb.obj: especifica las direcciones en memoria de las interrupciones vectorizadas

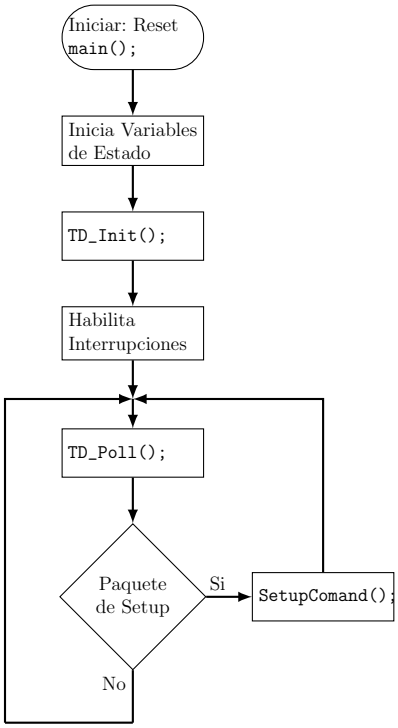

Figura 3.6: Diagrama en bloques simplificado

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 la 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 funcion `TD_Init()`, a través de la cual el usuario programa e inicia la configuración del sistema. Luego, es necesario habilitar todas las interrupcionnes necesarias y finalmente, se invoca repetidamente la función `TD_Poll()`, en donde el usuario escribe las tares que ejecutará el 8051.

3.2.2. Entorno de desarrollo y compilador

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

Keil μVision es un entorno de desarrollo integrado (IDE). Se entiende por IDE a un software que integra en un entorno gráfico las herramientas que permiten elaborar un programa que ejecutará un procesador, desde la escritura del algoritmo en uno o más lenguajes, las pruebas, el

depurado y el compilado del mismo.

El programa utilizado posee, entre otras cosas, editor de textos con atajos de teclado, comandos que aceleran la escritura de código y resaltador de palabras claves para diferentes lenguajes de programación, navegador de archivos. También ejecuta, con solo un click, el compilador con la sintaxis correcta, y posee un depurador que, a través de un intérprete, permite ir ejecutando el código línea por línea o por bloques.

Para realizar un programa en este entorno, Cypress provee, junto con su framework, un proyecto vacío que puede ser copiado y pegado. Sin embargo, se puede realizar la configuración manual. Las instrucciones de este procedimiento se ubican en el Apéndice **??**.

En cuanto al compilador se refiere, el utilizado es C51. Éste es un programa que otorga un archivo hexadecimal con un código que será ejecutado por microcontroladores que estén implementados con la misma estructura que un Intel 8051, cómo lo es el microcontrolador que posee el FX2LP.

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, aunque mínima, de la entrada y salida de datos, facilitando las pruebas sobre el sistema en desarrollo.

En el Apéndice **??** se explica su funcionamiento y manejo.

3.3. Desarrollo del firmware

A continuación se desarrollan los aspectos más relevantes del código final elaborado, compuestos por la función de inicialización y los descriptores del dispositivo USB.

Al establecer el modo automático de funcionamiento, la función `TD_Poll()` se encuentra vacía.

Luego, en el Capitulo 5 se abordará nuevamente algunos detalles realizados para la depuración del presente código, tales como la conexión del puerto serie y la utilización de algunas interrupciones específicas.

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 48 MHz. Para ello se deben colocar los registros CPUCS.4=1 y CPUCS.3=0. A través del framwork de Cypress, esto puede ser escrito de la siguiente forma:

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

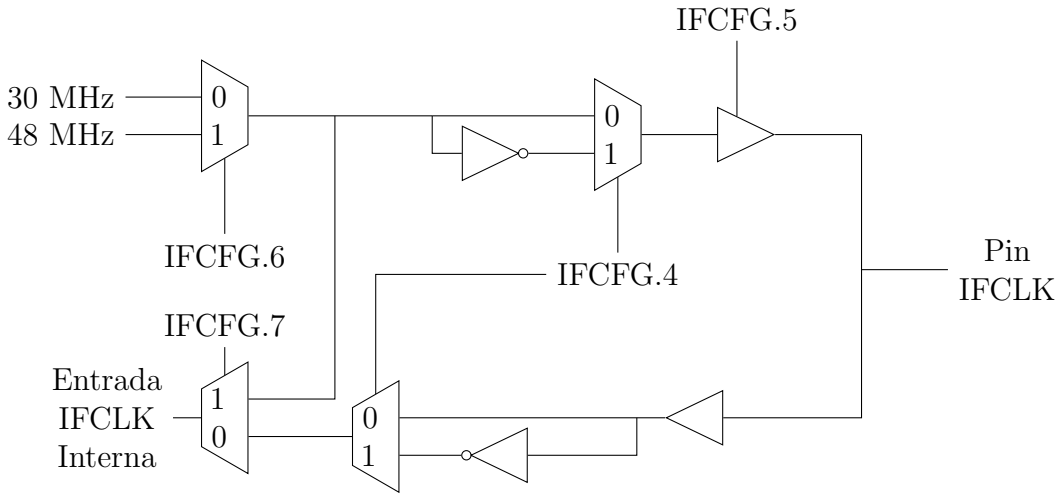

Figura 3.7: Esquema funcional para la entrada de reloj de la Interfaz

;

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 impreso 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 impreso 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
//fifoesclava(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 Configuratión
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.

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

//setting on auto mode. rising edge is necessary
EP8FIFOCFG = 0x11;
SYNCDELAY;
EP2FIFOCFG = 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 "leds.h"
```

Las primeras 4 líneas de encabezados son provistas por Cypress, a través de su framework. En ellas, la directiva de ensamblador noiv(identificada con `#pragma`), le indica al compilador que no debe habilitar las interrupciones vectorizadas. Estas, en cambio, serán manejadas y direccionadas a través del archivo objeto *usbjmptb.obj*.

El encabezado *leds.h* cuyo código se muestra a continuación, sirve para encender y apagar las luces de la placa de desarrollo. Estos leds se encuentran conectados a través de un decodificador y su funcionamiento se da con la sola lectura de direcciones específicas.

```
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;
```

Luego, el framework define algunas variables globales que utiliza en las funciones implementadas para el manejo de las tareas relacionadas al protocolo USB. Se listan estas variables a continuación.

```
extern BOOL   GotSUD;             // Received setup data flag
extern BOOL   Sleep;
extern BOOL   Rwuen;
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 blinktime = 0;
BYTE inblink = 0x00;
BYTE outblink = 0x00;
WORD blinkmask = 0;           // HS/FS blink rate
```

3.3.3. Descriptores USB

Los descriptores son una estructura definida de datos. A través de ellos, el dispositivo USB le comunica al anfitrión sus atributos, tales como velocidad de trabajo, cantidad de configuraciones e interfaces posibles, número de EPs, dirección de cada uno de ellos, tamaño máximo en bytes de paquetes que puede enviar en una comunicación, entre otros.

El framework de Cypress coloca toda la información sobre los descriptores de el dispositivo desarrollado en un archivo escrito en lenguaje de ensamblador, denominado *dscr.a51*.

Se observa a continuación los descriptores usados en el presente trabajo.

Capítulo 4

Desarrollo del sistema maestro para la comunicación entre la FPGA y la interfaz Cypress

4.1. Descripción del sistema desarrollado

Para el logro de los objetivos del presente trabajo, que es la realización de una comunicación USB para el envío y recepción de información originada en desarrollos científicos basados en un FPGA, es necesario no solo la elaboración de drivers, interfaz y periféricos, sino también un núcleo de FPGA capaz de comunicarse con ambos lados de la interfaz, es decir, con el controlador USB y con el desarrollo mismo, implementado dentro de la FPGA.

La Figura 4.1 muestra un esquema en el cual se observan, como extremos, un desarrollo científico particular y el controlador de Cypress. Entre ambos, se encuentra una Máquina de Estados Algorítmica (MEA) que hace de nexo. La MEA debe poder manejar correctamente las señales de control, así como envíar, recibir y canalizar los datos que circulan a través de ella.

La MAE se desarrolla en VHDL y se implementa en la placa MOJO que, cómo se menciona en el Capítulo, posee una FPGA Spartan VI de Xilinx.

Para poder elaborar la MAE, es necesario conocer y entender el funcionamiento y las señales que requiere la interfaz de las memorias FIFO del controlador FX2LP.

Por ello, en este capítulo se desarrolla, en primer lugar la estructura de la interfaz. Luego, se

Figura 4.1: alguna descripcion

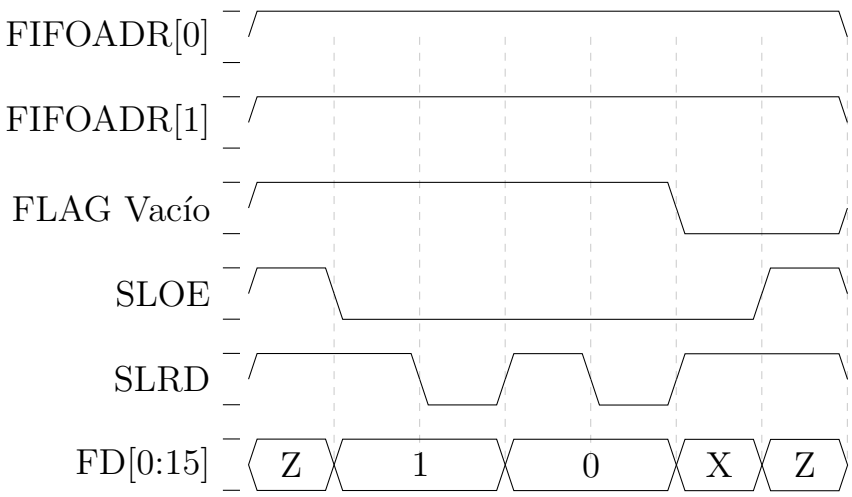

Figura 4.2: este texto

aborda el diseño de la MAE y finalmente, su implementación en VHDL.

Además de lo anterior, se detalla el diseño y la elaboración de un circuito impreso de interconexión, a través del cual las señales de las placas de desarrollo MOJO y CY3684 conectan en forma física sus puertos.

4.2. Señales de Control

En el Capitulo 3 del presente informe se describen los puertos de la interfaz entre un maestro esterno y las memorias FIFO del controlador de Cypress.

Al abordar, ahora, el control del mismo, es necesario desarrollar con mayor detalle cada una de estas señales. Así, se podrá comprender, con mayor detalle, las señales que debe enviar y recibir una MEA que controle esta interfaz.

Si bien el modo de funcionamiento de la interfaz puede ser síncorno o asíncorno, se detalla sólo este último, es decir, el implementado, ya que debido a errores de diseño de quien escribe, no es posible implementar, hasta el momento de la escritura, un funcionamiento sincrónico del sistema.

Por lo anterior, no se detalla en ningún momento la señal del puerto IFCLK.

4.2.1. Lectura de datos desde la interfaz

4.3. Máquina de Estados Algorítmica

4.4. Descripción en VHDL

4.5. Placa de Interconexión

Capítulo 5

Depuración y verificación del sistema

Bibliografía

[1] M. Perez, F. Alcalde, M. S. Haro, I. Sidelnik, J. J. Blostein, M. G. Berisso, and J. Lipovetzky, "Implementation of an ionizing radiation detector based on a FPGA-controlled COTS CMOS image sensor," in *2017 XVII Workshop on Information Processing and Control (RPIC)*, pp. 1–6, IEEE, sep 2017.

[2] R. Biswas, *An Embedded Solution for JPEG 2000 Image Compression Based Back-end for Ultrasonography System*. PhD thesis, IIT, Kharagpur, 2018.

[3] T. Yanagisawa, T. Ikenaga, Y. Sugimoto, K. Kawatsu, M. Yoshikawa, S.-i. Okumura, and T. Ito, "New NEO search technology using small telescopes and FPGA," in *2018 IEEE Aerospace Conference*, vol. 2018-March, pp. 1–7, IEEE, mar 2018.

[4] M. Perez, *DETECCIÓN DE RADIACIÓN IONIZANTE UTILIZANDO SENSORES DE IMAGEN CMOS COMERCIALES*. PhD thesis, 2018.

[5] I. Micron Technology, "1 / 2-Inch Megapixel CMOS Digital Image Sensor MT9M001C12STM (Monochrome)," pp. 1–35, 2004.

[6] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, and Philips, *Universal Seral Bus Specification*, vol. Revision 2.0. 2000.