INTERFAZ USB DE ALMACENAMIENTO DE DATOS EN MEMORIAS FLASH PORTÁTILES, PARA SISTEMAS EMBEBIDOS HOST USB-DSP

ANDRÉS BADILLO RODRÍGUEZ

ROBERTO DURÁN CASTELLANOS

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

BUCARAMANGA

2006

INTERFAZ USB DE ALMACENAMIENTO DE DATOS EN MEMORIAS FLASH PORTÁTILES, PARA SISTEMAS EMBEBIDOS HOST USB-DSP

ANDRÉS BADILLO RODRÍGUEZ
ROBERTO DURÁN CASTELLANOS

Trabajo Final desarrollado como requisito para optar por el titulo de
Ingeniero Electrónico

Director. Ing. JAIME GUILLERMO BARRERO PÉREZ, Mpe.
Codirector. Físico & ing. DAVID ALEJANDRO MIRANDA MERCADO, Msc.

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELAS DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
BUCARAMANGA
2006

A Dios Padre amoroso en quien pongo en sus manos este logro en acción de gracias por sus bendiciones y por iluminar siempre mi camino.
A la memoria de mi padre Roberto José; su ejemplo de dedicación, trabajo y solidaridad serán siempre mi motivación para dar de todo corazón lo mejor de mí.
A mi madre Rosalba y a mis hermanitas Diana y Yenny por su gran amor, su apoyo e inconmensurable preocupación; por enseñarme que nada es imposible y que el amor y la fortaleza de una familia unida puede más que cualquier obstáculo por difícil que parezca superar.
A mi Silvis porque su amor incondicional ha sido una bella luz que ha llenado de resplandor y de aliento mi vida, y ha hecho de mí una mejor persona.
A aquellos amigos que han sido ejemplo vivo de Jesús y han amado a su prójimo como a sí mismos: Marco, Carlos, Andrés Badillo, Claudia, Pedro, Daniel, Gladys, Christian y Jaime.

Roberto Durán Castellanos

Al amigo y compañero paciente que esta en el momento adecuado para alegrar, proteger y escuchar, al que lo da todo y no quita nada, gracias Dios Padre, Hijo y Espíritu Santo por este regalo en las necesidades temporales y por permitirme escribir estas palabras.
A La Virgen Maria por la misericordia y por sus "palancas", las intercesiones.
A mi familia, en especial a mi mamá Análida, a mi papá Aubanel Aníbal y a mi hermano Alejandro, por ayudarme a descubrir que las decisiones y los caminos que se siguen hasta el final no se hacen por compromiso sino por amor y que la pasión es el amor extremo, donde se soportan los sacrificios y el dolor con el fin de obtener el bienestar de los que se aprecian.

"Dar ejemplo no es la principal manera de influir sobre los demás; es la única."
Albert Einstein
A mis amigos y compañeros de estudio Roberto y Silvia por compartir enseñanzas de vida y valores humanos; por mostrarme que un equipo no es un grupo de personas reunidas; sino guerreros que luchan para lograr metas y valoran la victoria del equipo sobre la victoria individual.

Andrés Badillo Rodríguez

AGRADECIMIENTOS

Los autores de este trabajo agradecemos en primera medida a Dios todo poderoso por permitirnos llevar a cabo satisfactoriamente el proyecto; a nuestros familiares y amigos por el apoyo incondicional que recibimos de su parte; a nuestro orientador y guía de este proyecto, Msc. David Miranda porque nos mostró el camino a recorrer y nos dio las herramientas para hacerlo, a nuestro director Msc. Jaime Barrero por sus valiosos aportes y comentarios.

También agradecemos a los grupos de investigación CEMOS y CIMBIOS por el respaldo y colaboración.

Al grupo estudiantil Rama IEEE y a todas las personas que nos ofrecieron su apoyo en la elaboración de este trabajo.

RESUMEN

TÍTULO:
INTERFAZ USB DE ALMACENAMIENTO DE DATOS EN MEMORIAS FLASH PORTÁTILES PARA SISTEMAS EMBEBIDOS HOST USB-DSP[*].

AUTORES:
BADILLO RODRÍGUEZ, Andrés y DURÁN CASTELLANOS, Roberto [**].

PALABRAS CLAVE:
Clase USB de almacenamiento masivo (MSC), Controlador *Host*, FAT32, *Host* USB embebido, Memorias Flash USB, USB.

DESCRIPCIÓN:
Se presenta el desarrollo de un sistema basado en un *Host* USB embebido que, sin la asistencia de un PC, controla memorias Flash USB portátiles. Con ello se busca adaptar un medio de almacenamiento de alta capacidad y portátil al diseño del medidor de bioimpedancia eléctrica para detección temprana de cáncer de cuello uterino desarrollado en la Universidad Industrial de Santander.

El sistema posee en hardware una arquitectura distribuida conformada por dos controladores: un controlador Híbrido DSP-µC (56F805 de Freescale) encargado del control principal del sistema, y un controlador secundario ASIC Controlador *Host* (EZ-Host de Cypress), encargado de dar soporte en hardware para el manejo y control de dispositivos USB

Para la ejecución de las diversas tareas del sistema que son realizadas por el Software, se diseñó una arquitectura de Software modular, en donde las diferentes entidades o capas se encargan respectivamente del manejo del sistema de archivos FAT32; del control de dispositivos de la clase de almacenamiento masivo; de la configuración y control de dispositivos USB; y de la administración de la comunicación entre el DSP y el EZ-Host. Adicionalmente se desarrolló un módulo de aplicación que le permite al usuario del sistema verificar la escritura de datos en las memorias USB.

El sistema cuenta con la capacidad de configurar y controlar dispositivos USB de almacenamiento Masivo, actuando como un *Host* USB embebido. Fue probado con diferentes dispositivos de almacenamiento, donde se logró la exitosa escritura de archivos en memorias Flash USB y en discos duros USB con formato FAT32 y con partición primaria de hasta 31GB. El sistema cumple con los requerimientos y recomendaciones del *USB Implementers Forum* (USB-IF) para el desarrollo de *Hosts* Embebidos.

---

[*] Trabajo de grado

[**] Facultad de ingenierías físico-mecánicas. Escuela de ingenierías eléctrica, electrónica y telecomunicaciones. Director: Ing. Jaime Guillermo Barrero, Mpe. Codirector: Ing & Físico David Alejandro Miranda, Msc

ABSTRACT

TITLE:
USB INTERFACE OF DATA STORAGE IN PORTABLE FLASH MEMORIES FOR HOST-USB-DSP EMBEDDED SYSTEMS[*]

AUTHORS:
BADILLO RODRÍGUEZ, Andrés and DURÁN CASTELLANOS, Roberto [**]

KEYWORDS:
FAT32, Host Controller, USB, USB Flash memories, USB Embedded Host, USB Mass Storage Class.

DESCRIPTION:
It is presented the development of a system for data storage in portable USB Flash memories, based on a USB Embedded Host. The goal is to adapt a high-capacity and portable storage device to the design of the electric bioimpedance measurement system for early cervix cancer detection developed in the Industrial University of Santander.

The system has, in hardware, a distributed architecture conformed by two controllers: a Hybrid controller DSP-μC (56F805 of Freescale) that is in charge of the system main control, and a secondary controller, an ASIC Host Controller (EZ-Host of Cypress), in charge of giving support in hardware, for the handling and control of USB devices

For the execution of diverse duties of the system, done by the software, it was designed a modular software architecture, in where the different entities or layers are in charge respectively of the FAT32 files system's management, of the control of mass storage class kind devices, of the configuration and control of USB devices; and the administration of the communication between the DSP and the EZ–Host. Additionally, it was developed an application module that allows the user of the system verify the data writing in USB memories.

The system has the capacity to configure and to control USB mass storage devices, acting as a USB embedded Host. It was proved with different storage devices, where was achieved successfully the writing of files in USB Flash memories and in USB hard disks with FAT32 format and primary partition of up to 31GB. The system fulfills the requirements and recommendations of the USB Implementers Forum (USB-IF) for the implementation of embedded Hosts.

---

[*] Degree project

[**] School of electrical, electronic and telecommunications engineering. Director: Ing. Jaime Guillermo Barrero, Mpe. Codirector: Ing & Físico David Alejandro Miranda, Msc

TABLA DE CONTENIDO

LISTA DE FIGURAS

Pag.

LISTA DE TABLAS

Pag.

GLOSARIO

Controlador Host	*Host Controller*. Es la entidad de hardware que gestiona el protocolo USB y le permite al Host USB la ejecución de transacciones.
EP	*Endpoint*, es un espacio de memoria inmerso en un dispositivo USB que sirve como buffer de datos de entrada o salida.
H-ARD	Sistema Embebido para el Almacenamiento de Datos en memorias Flash portátiles USB. Sistema desarrollado en este trabajo.
HC-USB	Tarjeta de hardware integrada en el sistema H-ARD, diseñada para el funcionamiento del controlador Host y de los puertos USB.
LBA	Dirección del sector o bloque lógico en la unidad de almacenamiento.
Periférico USB	Es un dispositivo USB que se conecta al bus para ofrecer una función específica al *Host*.
Sector	Es el bloque lógico básico en una unidad con formato FAT. Típicamente para las memorias portátiles es de 512bytes.
Sistema USB	Es el sistema de comunicación que comprende el protocolo USB y todas las funciones y procesos que este sistema lleva a cabo
Tdesc	*Transaction Descriptor*. Es una estructura de datos mediante la cual el *Driver* del Controlador *Host* le indica al Controlador cómo debe ejecutar una determinada transacción.

INTRODUCCIÓN

A medida que el avance de la electrónica permite el desarrollo de aplicaciones y sistemas embebidos más complejos, la necesidad de medios de almacenamiento de datos de alta capacidad y con facilidad de expansión es cada vez más significativa. Esta necesidad se acentúa aún más con la actual cultura de lo portátil, donde el interés se centra en que los nuevos dispositivos ofrezcan mayores servicios en un menor tamaño, con menor consumo de potencia y sin comprometer el precio.

Un caso específico de esta situación se puede observar en la Universidad Industrial de Santander donde se está llevando a cabo el diseño de un Bioimpedanciometro para caracterizar tejido humano [MIRANDA]. Una de las limitaciones de los diseños realizados hasta el momento es su dependencia con un PC para el almacenamiento de los datos. Esta limitación ha llevado a estudiar tecnologías tales como memorias Flash, DSP y Host USB embebidos con el fin de hacer un Bioimpedanciometro completamente autónomo y con una gran capacidad de almacenamiento de datos.

Por otra parte, debido a la necesidad de simplificar el manejo del dispositivo y después de evaluar diferentes estrategias con el equipo médico que da soporte al proyecto, encabezado por el doctor Jorge Humberto Echeverry, se optó por una forma de almacenamiento que fuera de bajo costo, comercial, removible y reemplazable, y de fácil uso para el cuerpo médico. Con base a esto y las necesidades técnicas se seleccionaron memorias flash USB portátiles.

Por tal motivo, en el presente texto se expone el desarrollo de un sistema embebido que, sin la asistencia de un PC, controla dispositivos USB portátiles de almacenamiento masivo de datos. Con ello se busca adaptar un medio de almacenamiento de alta capacidad y portátil al diseño del medidor de bioimpedancia eléctrica para detección

temprana de cáncer de cuello uterino (entre otras aplicaciones) desarrollado en el grupo de investigación CEMOS de la Universidad Industrial de Santander. Así mismo, se espera que el sistema implementado sea de fácil incorporación a nuevos diseños de sistemas embebidos y pueda ser aprovechado en aplicaciones que requieran almacenamiento masivo de datos.

En este orden de ideas, el proyecto presentado en este texto es una propuesta que pretende ofrecer tanto a los dispositivos que se desarrollen en la investigación como a los sistemas embebidos en general, capacidades de almacenamiento mayores, con los beneficios de las memorias portátiles.

El presente texto expone en su primer capitulo los fundamentos teóricos en los cuales se sustenta el trabajo desarrollado. Específicamente, estos fundamentos exponen las ideas fundamentales para el desarrollo de un sistema *Host* USB embebido, y permiten adquirir un concepto propio sobre la incidencia de la tecnología USB y de las aplicaciones que se pueden desarrollar con base en esta. Adicionalmente, se exponen los antecedentes del proyecto y se hace una revisión del estado del arte con el fin de evidenciar el punto de partida y las herramientas tecnológicas disponibles.

Los capítulos 2 y 3 presentan con más detalle el trabajo desarrollado en este proyecto; se muestran respectivamente, el diseño de un hardware y el diseño del software para un *Host* USB embebido, y en general del sistema de almacenamiento desarrollado.

En el capitulo 4 se resumen las pruebas realizadas al sistema, diseñadas para evaluar sus características y analizar la versatilidad del dispositivo desarrollado.

El capitulo 5 presenta las conclusiones y observaciones que surgieron con la realización del proyecto; se exponen las ideas relevantes, las recomendaciones y nuevas perspectivas que pueden servir como punto de partida para nuevos proyectos, de tal

forma que se plantean las ventajas que USB puede ofrecer en aplicaciones basados en *Host* USB embebidos.

1. FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se resumen los conceptos básicos necesarios para poder entrar en detalle en el diseño del sistema de almacenamiento de datos basado en un *Host* USB embebido desarrollado en este trabajo. Como parte de la metodología, se presenta una descripción general del proyecto, una recopilación de los antecedentes de trabajos anteriores realizados en la Universidad Industrial de Santander relacionados con el tema de USB y una revisión del estado del arte de las tecnologías en las cuales el sistema diseñado se sustenta.

1.1 ANTECEDENTES DEL PROYECTO

1.1.1 TRABAJOS PREVIOS EN LA E3T*

La comunicación por el puerto USB ha sido en los últimos años el camino para la estandarización de las diversas interfaces que permiten la conexión de dispositivos periféricos con el computador. Así mismo, es la tecnología que se impone a nivel mundial en la conectividad de todo dispositivo periférico y/o portátil.

Este fenómeno no es ajeno a la Universidad Industrial de Santander, donde la Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones en su interés por fortalecer la academia, se preocupa por estar avante en los procesos que conllevan a la implementación de tecnologías de punta.

Los trabajos que se mencionan a continuación son proyectos de grado desarrollados en la Escuela de Ingeniería Eléctrica, Electrónica y Telecomunicaciones de la Universidad Industrial de Santander, en los cuales se aborda en cierta medida el tema de la comunicación USB.

* E3T entiéndase como Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones de la Universidad Industrial de Santander

La conectividad USB se presenta como tema central de proyecto, en trabajos como el de Nathalie Hernández e Ivonne Mantilla [HERNANDEZ-MANTILLA], así como el de Fredy Ascencio [ASCENCIO], con el propósito de estudiar las características que provee el protocolo USB.

Hernández y Mantilla [HERNANDEZ-MANTILLA], dos de las pioneras en la E3T en la utilización del puerto USB, exponen la implementación de una tarjeta de adquisición de datos basada en el microcontrolador MC68HC908JB8JP, que actúa como un dispositivo USB Periférico* de aplicación específica. Dicho dispositivo trabaja la interfaz USB a 1.5Mbps (denominado *Low Speed* de acuerdo a las especificaciones USB 1.1) y utiliza transferencias† de tipo Interrupción (utilizadas por los Dispositivos de Interfaz Humana *HID*) para la transmisión de datos por el bus. Las autoras recomiendan trabajar la interfaz USB a velocidades mayores (*Full Speed* a 12Mbps ó *High Speed* a 480Mbps, de acuerdo con USB 2.0) para dispositivos de adquisición de datos que requieran altas tasas de transferencia de datos, dado que la velocidad de transferencia de datos efectiva obtenida fue más baja en comparación con otras interfaces seriales de comunicación, como el RS232.

Por otra parte, Ascencio [ASCENCIO] en su trabajo expone el diseño de una tarjeta de adquisición de datos por medio del puerto USB, basado en una versión más reciente de USB (USB 2.0). Presenta algunas modificaciones con relación al trabajo anterior respecto a la implementación de transferencias de volumen de datos (*Bulk*) además de las de tipo Interrupción. Adicionalmente, el dispositivo final trabaja la interfaz USB a 12Mbps y logra una tasa de transferencia efectiva de datos de 960Kbps con un DSP a una frecuencia interna de bus de 60Mhz y 124.8Kbps con un microcontrolador a una frecuencia interna de bus de 7.9872Mhz.

---

* Dispositivo USB Periférico se entiende como un dispositivo que está conectado al Bus y es parte pasiva de la comunicación USB, donde el administrador de la comunicación es el *HOST*.

† Véase apartado 1.3.1.3, Flujo de la comunicación, Pipes y Transferencias.

Así mismo, en la E3T se han desarrollado trabajos de grado que han abordado, de forma menos profunda, el tema de la comunicación USB, como es el caso de los trabajos de Monroy y Zabala [MONROY-ZABALA], y de Conde y Santos [CONDE–SANTOS] en donde se utiliza el puerto USB como una interfaz serial de alta velocidad.

De manera global se ha observado la utilización de la comunicación USB en dispositivos catalogados como Periféricos (de acuerdo a las especificaciones USB 2.0), donde la parte activa o administrador del Bus ha sido el PC (*Host)*.

1.1.2 MEMORIAS FLASH PORTÁTILES

El mercado de dispositivos de uso personal y doméstico (como es el caso de los celulares, cámaras digitales y reproductores de audio) ha generado una cultura que demanda productos portátiles, de tamaño reducido y de bajo consumo de potencia. Este comportamiento se ha extendido hacia todos los campos de la electrónica, donde se requiera de *hardware* especializado para la realización de tareas explícitas.

Con el avance de la electrónica, la reducción en el tamaño de los dispositivos y disminución en el consumo de potencia de los mismos, el desarrollo de sistemas embebidos portátiles dejó de ser una opción a ser una necesidad. En consecuencia ha surgido una gama de herramientas y nuevas tecnologías que proporcionan un valor agregado al diseño, la implementación y la utilización de estos sistemas.

En el campo de las memorias o dispositivos de almacenamiento de datos se observa el mismo comportamiento. Desde el surgimiento de las memorias RAM y ROM para computadores personales, pasando por la EPROM y la EEPROM, hasta las más recientes memorias Flash (cuya tecnología de fabricación ha permitido mayor capacidad de almacenamiento en un menor espacio, con características de escritura y borrado con muy bajo consumo de potencia), el campo de las memorias ha tenido un gran avance, entre otros factores impulsado por la misma necesidad de diseñar sistemas de menor tamaño y con mayores prestaciones.

Existen dos tipos de memoria flash: las que están basadas en compuertas NOR (tipo NOR) y las que están basadas en compuertas lógicas NAND (tipo NAND). Cada tipo tiene sus ventajas y desventajas con respecto a la otra. No obstante, pese a que las memorias tipo NOR manejan una mayor velocidad de escritura, las memorias NAND son actualmente las más utilizadas en dispositivos portátiles por ser más económicas y permitir una mayor densidad de almacenamiento [WIKIPEDIA1] (lo que implica mayor capacidad a un mismo tamaño). Además, de acuerdo con Kaplan [KAPLAN], la tecnología de memorias NAND ha evolucionado tan drásticamente que ha superado la ley de Moore[*], doblando la capacidad cada 12 meses, en comparación con los 18 meses que dicta dicha ley.

La necesidad latente de darle el carácter de portátil a un dispositivo, junto con las grandes ventajas que presentan las memorias Flash, han propiciado el surgimiento de lo que se conoce hoy en día como memorias portátiles. Actualmente existe una gran variedad de formatos de memorias portátiles, la mayoría basadas en tecnología NAND Flash.

Un aspecto importante de una memoria portátil es su interfaz de comunicación. Con el auge de USB y el fenómeno de estandarización de interfaces de comunicación, los fabricantes de diversos dispositivos que hacen uso de memorias portátiles (las cámaras fotográficas, los PDA, los reproductores de MP3 y las mismas memorias USB) han optado por la utilización de USB como interfaz de comunicación con el PC para la transferencia de archivos y datos en general.

La proyección de la utilización de memorias portátiles en nuevos diseños de dispositivos es grande. Con la tecnología de memorias Flash y el protocolo USB de la mano, los

[*] La ley de Moore es una predicción hecha por el co-fundador de Intel, Gordon Moore, quien afirmó que el número de transistores en un chip se duplicaría cada 18 meses.

recursos disponibles para el diseñador de sistemas embebidos serán cada vez mayores y las aplicaciones más completas y robustas.

1.1.3 SISTEMAS *HOST* USB

USB es un sistema de comunicación cuyo protocolo tiene una arquitectura tipo Maestro-Esclavo, donde se ha definido como un Bus de un sólo *Host* (quien es el que coordina toda la comunicación) y muchos Periféricos. La definición de dispositivo USB Periférico implica la ejecución de tareas muy específicas y dependientes de un *Host* (que generalmente es el PC). Desde el punto de vista de un sistema portátil, esta definición limita su funcionalidad en el sentido de que no se puede comunicar con otros dispositivos USB. Además, está forzado a depender de un computador para hacer uso del Bus.

La tendencia de la estandarización de protocolos ha llevado a que las interfaces de conexión de los diversos dispositivos periféricos tiendan a unificarse. Como consecuencia de ello, interfaces de comunicación como el RS232, PS/2 y demás, están siendo cada vez menos implementadas. Muestra de ello lo podemos ver en los PCs más actuales, en los cuales no se encuentran puertos como estos. En cambio, se pueden encontrar cinco o más puertos USB y en algunos casos puertos IEEE 1394 (*Firewire*).

Lo anterior ha generado que la gran mayoría de nuevos diseños de dispositivos incluyan estándares de conectividad vigentes como USB, para dar solución a la comunicación con el PC. Desde dispositivos Periféricos (como teclados, ratones e impresoras) hasta dispositivos de aplicación específica (como tarjetas de adquisición de Datos y equipos médicos), ya incluyen puertos USB como interfaz de comunicación.

¿Pero que sucede con la conexión con los dispositivos entre sí? Dentro de la concepción original de USB, los dispositivos Periféricos no poseen la capacidad de comunicarse entre si, debido principalmente a que, como se mencionó anteriormente,

este protocolo se definió con una arquitectura Maestro-Esclavo, donde el Maestro es quien controla la comunicación (que usualmente es el PC). Con el avance de los sistemas portátiles, el requerimiento de conectividad entre dispositivos se hace cada vez más notorio.

El concepto de *Host* USB se aplica a los dispositivos que administran la comunicación USB y pueden acceder a los servicios que ofrecen los dispositivos Periféricos. Estos sistemas hasta el momento siempre se han relacionado con el PC. Sin embargo, dentro del concepto de *Host* USB, ha surgido una clasificación para identificar a dos grupos representativos de *Host* USB: de propósito general y *Host* embebidos.

Un *Host* de propósito general es un sistema que está en capacidad de soportar una gran variedad de dispositivos USB, siempre y cuando cuente con los *Drivers* apropiados. En la comunicación USB, el concepto de *Host* de propósito general se aplica al PC.

Un **sistema *Host* USB embebido** es un concepto reciente que surge en respuesta a la necesidad de los sistemas embebidos de poseer no sólo características propias de un dispositivo USB Periférico sino también la capacidad de desempeñar las tareas de un *Host* USB, para poder comunicarse con otros dispositivos sin la asistencia de un PC. Esta clase de dispositivos interactúan con un conjunto específico de periféricos y su complejidad o soporte de dispositivos periféricos va acorde con la capacidad del sistema embebido en el que se implemente.

El desarrollo de nuevos diseños basados en *Host* USB amplía los horizontes en materia de recursos. Cualquier dispositivo USB Periférico puede potencialmente proporcionar servicios y funciones adicionales a un sistema *Host* USB. En el caso específico del presente proyecto, un diseño que fue concebido inicialmente para medición de

impedancia eléctrica de tejido de cuello uterino puede acceder a los servicios de almacenamiento de una memoria USB, gracias a la incorporación de un *Host* USB[*].

Cabe resaltar que el desafío de adicionar a un sistema embebido la capacidad de *Host* USB no es sencillo. El *Host,* como administrador del Bus, tiene que lidiar con la complejidad del protocolo. Hasta ahora era tarea del PC cumplir con las labores del *Host*. Ahora, están siendo necesarios nuevos mecanismos para incorporar un *Host* USB a sistemas embebidos (como el propuesto en este trabajo), para ampliar la cobertura de conexión de dispositivos.

Existe un campo en la tecnología USB que aún queda por explorar, el OTG [PHILIPS] (*On The Go* por sus siglas en inglés). OTG es un suplemento de las especificaciones USB 2.0, el cual habilita la posibilidad que los dispositivos USB puedan cumplir con cualquiera de los dos roles, el de *Host* y el de Periférico. Se diferencia con la tecnología de los sistemas *Host* USB embebidos en que estos últimos por lo general comprenden puertos dedicado para *Host* y puertos dedicados para Periféricos. En cambio OTG permite que un dispositivo tenga en un mismo puerto la posibilidad de conectarse como *Host* o como Periférico. OTG, así como todo el concepto de USB, simplifica para el usuario el manejo del dispositivo a costa de aumentar la complejidad de su implementación.

1.1.4 NUESTRO PROYECTO

Con el presente proyecto se buscó desarrollar un sistema embebido que, sin la asistencia de un PC, controle memorias *Flash* USB portátiles. Con ello se pretende ofrecer a nuevos diseños de dispositivos mayor capacidad de almacenamiento, con las ventajas de las memorias portátiles de facilidad para el transporte de datos y flexibilidad de expansión o reemplazo del medio de almacenamiento.

[*]Véase apartado 1.1.4, Nuestro Proyecto.

El proyecto se encuentra dentro del marco de una investigación adelantada por el físico e ing. David Alejandro Miranda, Msc. y el ing. Jaime Guillermo Barrero, Mpe., en el grupo CEMOS [MIRANDA]. La investigación busca, como objetivo principal, proponer una técnica para la detección precoz de cáncer de cuello uterino basada en espectro de impedancia eléctrica y para ello se requiere diseñar un dispositivo de medición del espectro de impedancia eléctrica para estudiar el tejido de cuello uterino.

La aplicación de este proyecto está orientada al almacenamiento de los datos obtenidos en la medición del espectro de impedancia eléctrica de las muestras del tejido de cérvix, en una memoria USB y autónomo del PC, de tal forma que se facilite el transporte de la información y se aumente significativamente la capacidad de almacenamiento de datos.

En resumen, el presente trabajo propone, por un lado, el desarrollo de un sistema de almacenamiento de datos con memorias Flash portátiles, y por otro lado, la implementación de un sistema embebido capaz de controlar dispositivos USB de almacenamiento de datos.[*]

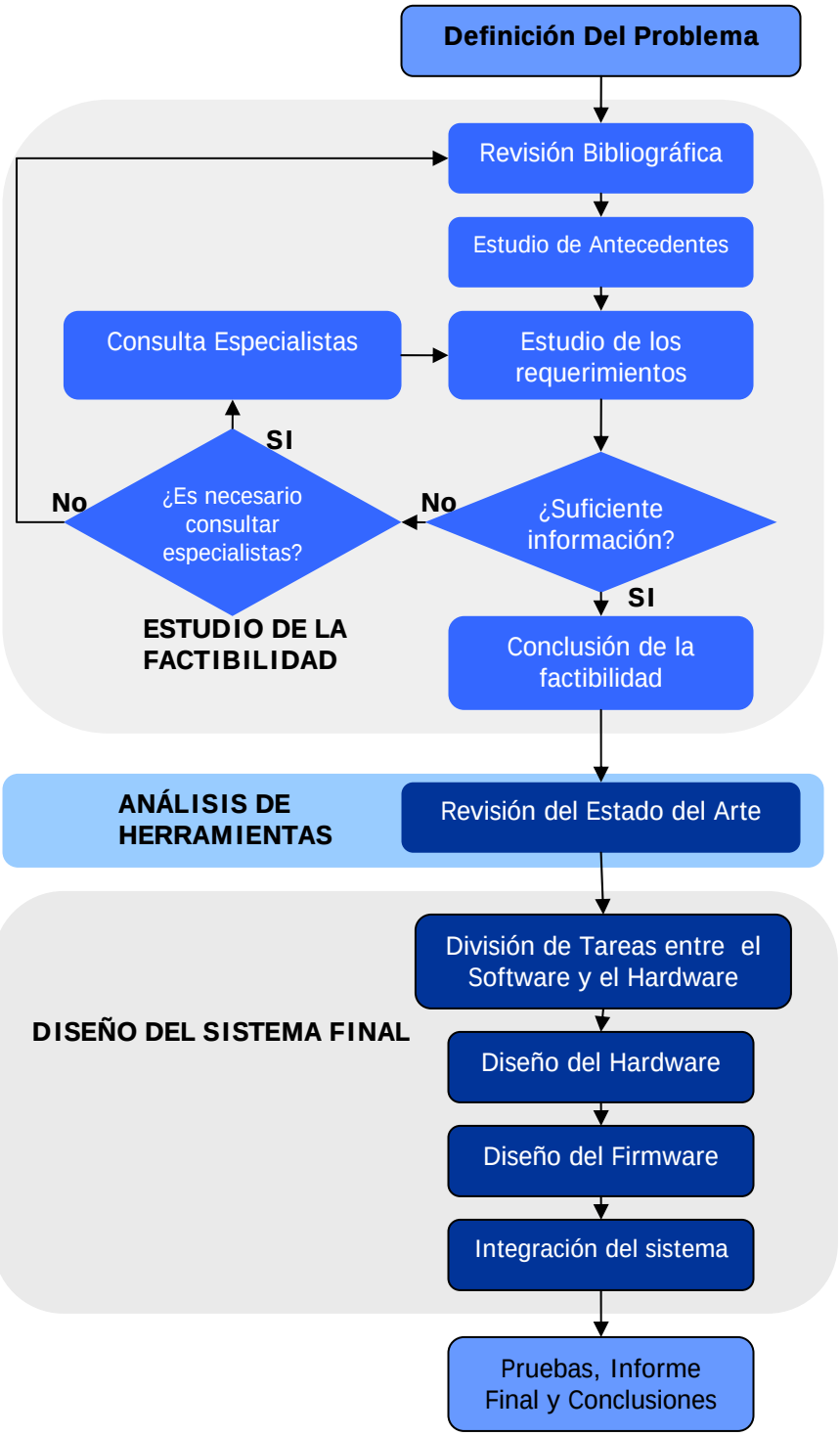
1.1.5 METODOLOGÍA

En el proyecto se utilizó una metodología de adaptación de tecnología, en la cual se siguieron parámetros generales para resolver un problema de ingeniería, tales como la definición del problema, la recopilación bibliográfica, el estudio de los antecedentes, la revisión del estado del arte y el diseño del sistema final, entre otros.

La figura 1 ilustra el proceso metodológico que se implementó para el desarrollo del proyecto, donde se resaltan tres grandes fases: el estudio de la factibilidad, el análisis de herramientas y el diseño del sistema final.

[*] Refierase al apartado 2.1, Planteamiento del Problema y Requerimientos del Sistema.

Figura 1. Metodología para el desarrollo del proyecto



Fuente: Los Autores

Del estudio de factibilidad vale la pena resaltar que, debido a la escasa información técnica disponible y a la falta de antecedentes estrechamente relacionados con el problema, fue necesario la consulta de especialistas en el tema, de otros países, para obtener una orientación acerca de cómo llevar a cabo el sistema *Host* USB para el almacenamiento de datos propuesto en este proyecto.

Una vez definida la factibilidad, se inició la segunda fase con el proceso de selección de las herramientas útiles para el proyecto. Se hizo énfasis en la revisión del estado del arte con el fin de evaluar las herramientas tecnológicas disponibles y el impacto técnico del proyecto.

En la tercera fase llevó a cabo el desarrollo del sistema final. Se identificaron las limitaciones y las tareas que podía ejecutar el hardware, para así determinar las funciones que se tendrían que implementarse en software. El diseño tanto del hardware como del software se enfocó en el aprovechamiento de los recursos; en software se implementaron técnicas de programación para mejorar la velocidad de ejecución y el espacio en memoria de la aplicación[*], mientras que en hardware se buscó un bajo consumo de potencia[†] y una reducción de efectos electromagnéticos negativos, entre otras estrategias.

1.2 MODELO DE CAPAS DE USB

En el análisis de un protocolo es conveniente determinar o definir un modelo de capas que permite esclarecer las entidades que participan en el sistema y evaluar la complejidad del protocolo

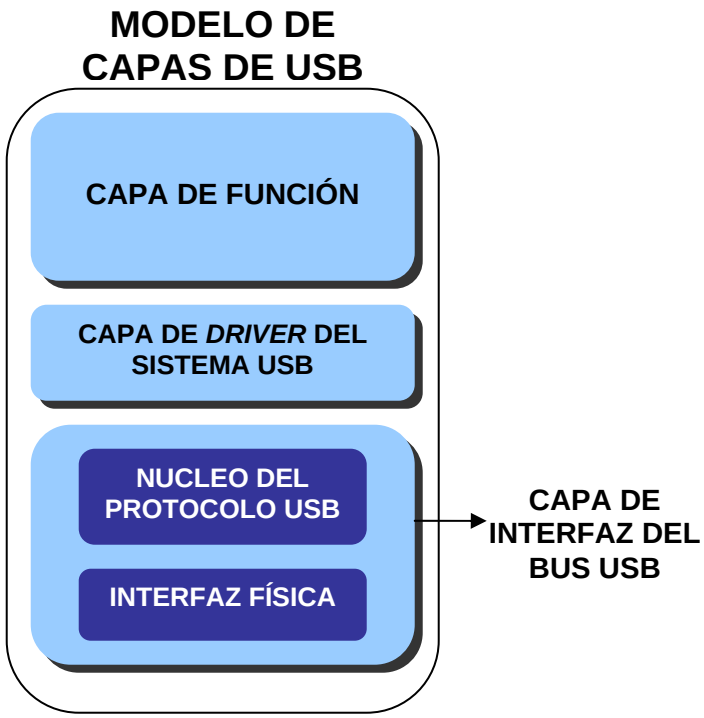
Por tal motivo, en este apartado se presenta el modelo de capas del sistema USB y se describe cada una de estas entidades. De esta forma se pretenden esclarecer los

[*] Véase Capítulo 3 de este texto.
[†] Véase Capítulo 2 de este texto.

servicios que ofrece USB como sistema de comunicación y a su vez, las funciones de cada capa del protocolo. El modelo USB consta básicamente de tres capas: La capa de Interfaz del Bus USB, la capa del *Driver* del sistema USB y la capa de Función, tal como ilustra la figura 2.

La capa de Interfaz del Bus USB se divide a su vez en dos subcapas: la subcapa de Interfaz Física y la subcapa de Protocolo USB[*]. La subcapa de Interfaz Física permite la conexión eléctrica y física en el bus. Así mismo, se encarga de la señalización y codificación de los bits que transmiten. La subcapa de Protocolo se encarga de la elaboración y envío de los Paquetes USB, los cuales son la unidad de datos del protocolo o PDU y la esencia de la comunicación por el Bus (Refiérase al apartado 1.3.4, Protocolo).

Figura 2. Modelo de capas de USB



Fuente: Los Autores

[*] Esta división es una abstracción hecha por los autores y no se describe explícitamente en las especificaciones USB.

La capa de *Driver* del sistema USB es quien se encarga del manejo de todo el sistema USB y de que la comunicación por el bus sea exitosa. Para ello realiza funciones, tales como: la administración del acceso al Bus y de las transacciones de datos que tienen lugar en el mismo; la configuración del Sistema USB (que corresponde a la configuración de un dispositivo recién conectado y la asignación de un *driver* para este), y la ejecución de las peticiones de la capa superior, entre otras. Esta capa hace uso de los servicios de la subcapa de protocolo para llevar a cabo la comunicación entre el *Host* y el Periférico.

La capa de Función comprende las aplicaciones que hacen uso de todo el sistema USB para entablar una comunicación con otra entidad. En el caso del *Host*, un dispositivo conectado al bus puede proporcionarle uno o varios servicios adicionales. Cada clase de servicio que le proporciona es visto como un ente independiente catalogado como **Función** (refiérase al apartado 1.3.2. Arquitectura). Así, la capa de Función comprende el software de la aplicación en el lado del *Host* y la o las funciones en el lado del dispositivo Periférico. En el diseño del sistema *Host* USB embebido del presente proyecto, se utiliza una memoria USB que le provee la función de almacenamiento de datos al *Host*. Así mismo, en el lado del *Host*, es la aplicación quien hace uso de este servicio.

1.3 BUS SERIAL UNIVERSAL USB

USB es un sistema de comunicación que se ha concebido para unificar las interfaces eléctricas de comunicación con el PC así como para facilitar el uso y la conexión de los dispositivos. Esta última característica es posible gracias a complejos procesos que lleva a cabo el *Host* o administrador de la comunicación.

Con el fin de sentar las bases para el diseño de un *Host* USB, en la presente sección se exponen los conceptos básicos de USB así como una abstracción del funcionamiento

de este sistema y una breve descripción de los procesos que un *Host* USB realiza para controlar los dispositivos USB conectados al Bus.

1.3.1 GENERALIDADES

USB ha sido en los últimos años la interfaz de comunicación que ha unificado la forma de conexión de los dispositivos Periféricos con un *Host* (PC). Utiliza un bus serial para la transmisión de información y soporta la conexión y direccionamiento de hasta 127 dispositivos. USB gestiona el acceso al medio mediante el envío de Testigos o *Tokens.*
Algunas de las características principales de USB son:

- Unificación de la interfaz de comunicación de los dispositivos Periféricos, eléctrica y mecánicamente.
- Soporte para conexión y desconexión en caliente así como la detección y configuración automática del dispositivo (*Plug and Play*).
- Soporte para la conexión de nuevos diseños de dispositivos.

La versión más actual de USB (2.0) define tres velocidades diferentes de transmisión, las cuales permiten soportar y aprovechar de la mejor forma los recursos de un dispositivo y al mismo tiempo mejorar el rendimiento del sistema. Las velocidades son:

- *Low Speed*: 1.5Mbps
- Full Speed: 12Mbps
- High Speed: 480Mbs

USB es un protocolo tipo Maestro-Esclavo. Ha sido definido como un bus de un sólo ***Host*** o administrador y muchos Esclavos o **Periféricos**. El *Host* es el encargado de controlar la comunicación, además de proporcionar la alimentación a los dispositivos que requieren una fuente externa, mientras que el Periférico es quien le provee servicios adicionales al *Host*. Toda la complejidad del sistema USB se recargó sobre el *Host*, por tanto se delegó al PC esta tarea, permitiendo la implementación del protocolo en los dispositivos de forma sencilla y por ende de bajo costo.

Los nuevos desarrollos con interfaz USB se están orientando al concepto de dispositivos OTG (dispositivos que usan la tecnología emergente *On The Go*)[*] los cuales son dispositivos periféricos que además de ofrecer una función especifica, pueden interactuar con otros Periféricos USB.

1.3.2 ARQUITECTURA

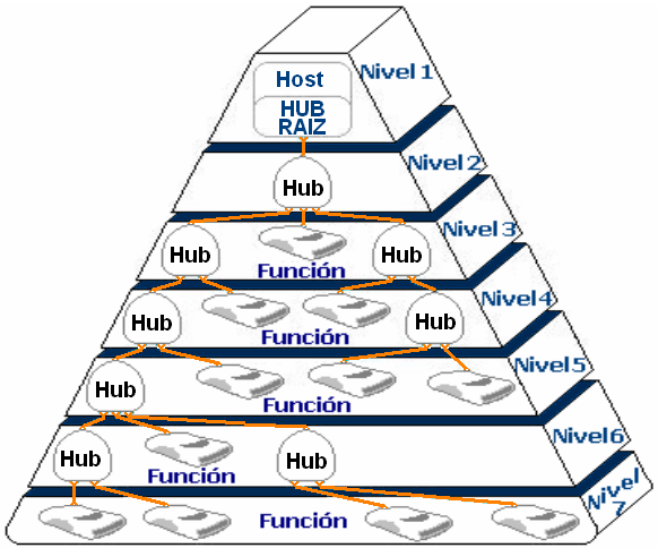
El sistema USB tiene una arquitectura que permite la interconexión simultanea de diversos dispositivos con el administrador (*Host*) de la comunicación, aun así debido a las características físicas de esta tecnología, la interconexión antes mencionada requiere la presencia de nodos que expandan el BUS, los cuales hacen que la conexión física entre el *Host* y los Periféricos difiera de la conexión lógica. Por lo anterior en USB existe una Topología física y una Topología lógica, las cuales se presentan a continuación.

1.3.2.1 Topología Física. En una interconexión común de USB básicamente se hacen presentes el administrador de la comunicación (*Host)*, y los Periféricos que están conectados a él por medio del Bus USB. En esta interconexión pueden existir *nodos* que permiten la expansión del Bus para que más dispositivos se conecten a este. Estos nodos son entidades de USB definidas como ***Hubs***.

La interconexión de todos los dispositivos Periféricos al *Host* por el bus USB presenta una topología en forma de estrella y se organiza en escalones de acuerdo a los niveles de expansión del bus. El centro de la comunicación es el *Host* y los Periféricos se conectan a él directamente o a través de los *Hubs.* Cada nivel se genera por la extensión de un *Hub*. La figura 3 ilustra la posible conexión entre diferentes dispositivos y el *Host*.

[*] Referirse al apartado 1.1.3.

Figura 3. Configuración de escalones sistema USB



Fuente: Los Autores

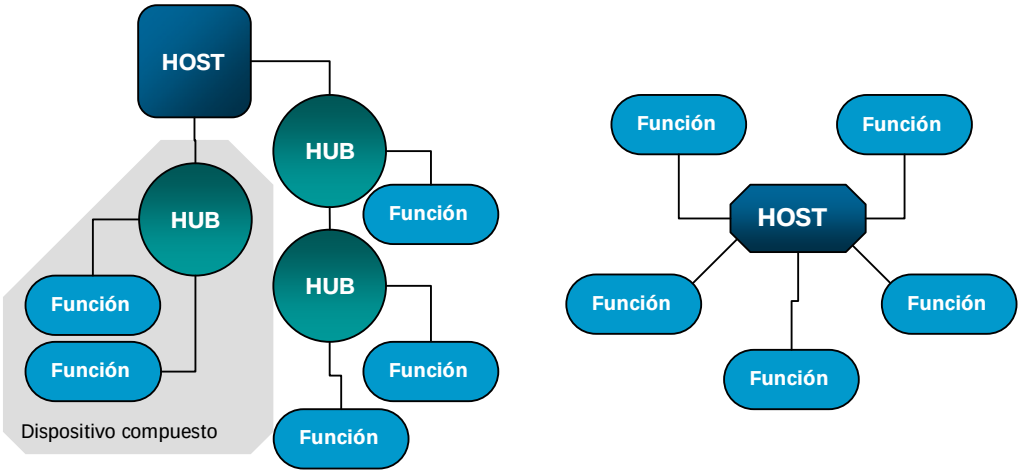
Un dispositivo es identificado por el *Host* de acuerdo a la clase de función que ofrece. Pueden existir dispositivos que ofrecen más de una función conocidos como dispositivos *multifuncionales* o compuestos. Estos dispositivos son representados como un *Hub* el cual reúne las diferentes funciones que son vistas por el *Host* como entidades lógicas independientes, como lo muestra la figura 4.

El *Host* puede soportar una conexión de 127 dispositivos y es posible implementar máximo 7 niveles de propagación incluyendo el *Hub* raíz (el *Hub* que esta contenido dentro del *Host*).

1.3.2.2 Topología Lógica. La topología lógica de USB difiere un poco de la configuración física ya que la comunicación USB es una interacción punto a punto entre el *Host* y un Periférico, es decir, el *Host* ve a las funciones de todos los dispositivos como entidades independientes que están conectadas directamente a él (figura 4).

Analizando las dos topologías de sistema, se puede ver como en USB la inserción de un *Hub* resulta transparente en la comunicación entre el *Host* y un dispositivo Periférico.

Figura 4. Topología del bus USB



a) Topología física b) Topología lógica

Fuente: Los Autores

1.3.2.3 Flujo de la comunicación, *Pipes* y Transferencias. USB, como un protocolo estructurado, maneja un flujo de datos entre entidades lógicas paritarias (las entidades entre el *Host* y el Periférico de la misma capa del protocolo), y un flujo de datos entre una capa superior y su capa de servicio.

En el sistema USB el *Host* <<ve>> al dispositivo Periférico como una colección de ***Endpoints***. Los *Endpoints* se pueden definir como espacios de memoria en los dispositivos donde finalmente se envían o reciben los datos en una transmisión; son configuraciones de hardware que el *Host* puede direccionar.

El dispositivo mantiene información relevante sobre la naturaleza de los *Endpoint*, que comprende tanto el sentido de la transmisión como la capacidad máxima de datos que puede manejar. El *Host* debe solicitar esta información y con ella controlar adecuadamente el intercambio de datos con el Periférico.

Aunque los Periféricos pueden tener diferentes tipos de *Endpoints,* como mínimo deben tener el *Endpoint* de control o <<*Endpoint* cero>>, el cual se usa en la configuración y el

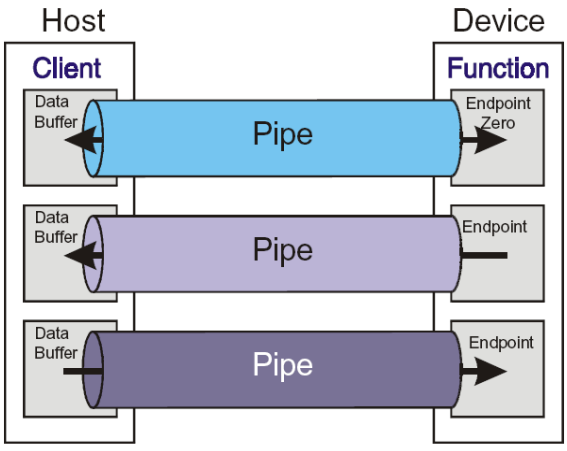
control del dispositivo Periférico, por ejemplo para el proceso de enumeración (refiérase al apartado 1.3.3, Configuración de Dispositivos USB). El *Endpoint* cero utiliza únicamente transferencias de control.

En un sistema USB, la comunicación entre el *Host* y los *Endpoints* en el dispositivo se maneja a través de caminos o interconexiones lógicas identificadas como ***Pipes.*** La figura 5 ilustra una representación de las *Pipes* y la forma como el *Host* ve a un Periférico como un conjunto de *Enpoints* a través de las *Pipes*.

Por otro lado, USB soporta distintos tipos de **transferencias** que presentan características específicas dependiendo de las necesidades de transmisión del dispositivo. Esto, en cierta medida, permite evidenciar la flexibilidad y versatilidad que ofrece el bus USB para trabajar con distintos tipos de Periféricos con requerimientos de comunicación diferentes.

Específicamente, en el protocolo USB existen 4 tipos de transferencias posibles para la comunicación, las cuales son: la transferencia de control, por volumen de datos (BULK), isócronas y por interrupción.

Figura 5. Comunicación USB a través de las *Pipes*



Fuente: ANDERSON, Don. USB System Architecture. Menlo Park, CA : ADDISON-WESLEY. 2001. ISBN: 0-201-46137-4

Las transferencias de control son usadas en el proceso de configuración de un dispositivo. El *Host* implementa las transferencias de control para transportar peticiones propias de USB que sirven para configurar, enumerar y controlar un dispositivo. Todos los dispositivos USB soportan las transferencias de control y un *Host* USB debe tener implementado mínimo las transferencias de control.

Las transferencias por volumen de datos (BULK) son transferencias utilizadas, como su nombre lo indica, en casos de manejo de grandes cantidades de datos y donde no es critico el tiempo de transmisión de estos. Estas transferencias están soportadas en *full-speed* y *high-speed*. Son utilizadas en los dispositivos de almacenamiento de datos como discos duros y memorias *Flash* portátiles.

Las transferencias isócronas son transferencias de datos periódicas que se usan cuando el tiempo es un factor relevante. En este tipo de transferencia existe una comunicación continua entre el *Host* y el dispositivo. Los dispositivos de audio y video y aplicaciones de manejo de datos en tiempo real que requieren una tasa constante de transmisión, soportan este tipo de transferencia

Las transferencias por interrupción se manejan en aplicaciones donde el ancho de banda no es relevante para la transmisión de datos. Estas transferencias se manejan en dispositivos de interfaz humana[*] como teclados y *mouses*.

La capacidad del bus USB de manejar diferentes tipos de transferencias permite al sistema USB aprovechar las ventajas de comunicación que cada dispositivo ofrece. Un caso que ilustra este hecho son los dispositivos que trabajan con transferencias de interrupción y no necesitan el establecimiento de una comunicación continua con el *Host*.

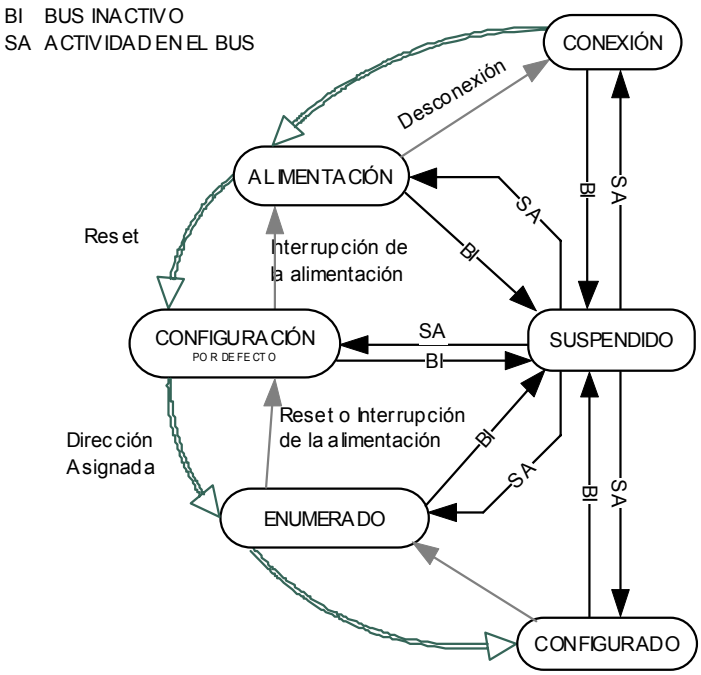
[*] Referirse al apartado 1.3.6. Clases USB

1.3.3 CONFIGURACIÓN DE DISPOSITIVOS PERIFÉRICOS USB

Se mencionó anteriormente que una de las características de USB es la detección y configuración automática de un dispositivo. Como USB soporta la conexión de dispositivos con características diferentes, el administrador de la comunicación (el *Host*) debe obtener información específica de éstos para identificarlos y asignarles un *Driver* adecuado. Este procedimiento, que permite la configuración automática de un dispositivo USB, es denominado **Enumeración** y consta de las fases mostradas en la figura 6.

En las fases de conexión y alimentación, un Periférico USB es conectado físicamente al Bus y alimentado por el *Host* (en el caso de que el dispositivo Periférico no sea autoalimentado). Posteriormente, el dispositivo es <<reinicializado>> por el *Host* y pasa a un estado en el cual presenta una Configuración por Defecto. Allí, el *Host* le asigna una dirección al dispositivo, lo que le permite a este último entrar en la fase de enumerado. En esta fase el *Host* solicita los Descriptores del dispositivo, los cuales

Figura 6. Pasos en la enumeración de un dispositivo Periférico



Fuente: Los Autores

contienen información útil que le permiten al *Host* asignarle un *Driver* al dispositivo para que sea controlado adecuadamente.

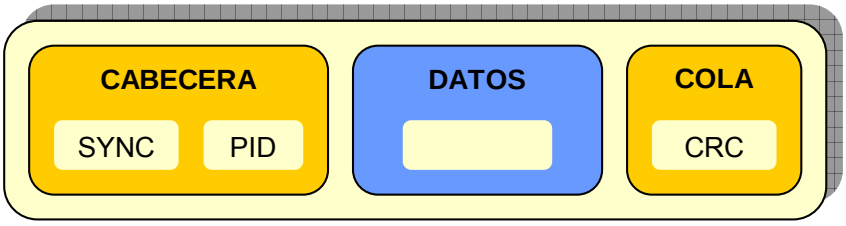
Un dispositivo físico, como se mencionó en el apartado 1.3.2, puede estar compuesto por una o varias funciones, que son vistas por el *Host* como dispositivos lógicos diferentes (como por ejemplo un teclado que tenga la capacidad de *Hub)*. Cada función es interpretada por el *Host* como una posible configuración del dispositivo. A su vez, cada configuración puede estar conformada por una o varias interfaces de comunicación lógica con la aplicación y cada interfaz maneja sus propios *buffers* o *Endpoints*.

Es importante para el *Host* conocer las características generales del dispositivo, para saber sus posibles configuraciones. De cada configuración es necesario conocer las interfaces que maneja y los *EndPoints* que abarca cada interfaz. Por ello existen descriptores de dispositivo, de configuración, de interfaz y de *EndPoint,* los cuales suministran esta información y permiten la adecuada comunicación entre el *Host* y el dispositivo Periférico (Cada descriptor se expone en el anexo A).

1.3.4 PROTOCOLO

La unidad de datos del protocolo o PDU se conoce como **Paquete**. USB coordina toda la comunicación en el bus mediante el envío y recepción de paquetes. Un paquete USB está usualmente conformado por tres partes: una cabecera, un campo de datos y una cola, tal y como lo ilustra la figura 7.

Figura 7. Estructura de un paquete



Fuente: Los Autores

La cabecera consiste en un *byte* de sincronización, que permiten que el reloj del receptor se sincronice a la misma velocidad que se hizo la transmisión, y un identificador de paquete ó PID (Package ID), que permite reconocer el tipo o la función del paquete. El campo de datos del paquete contiene la información que se quiere enviar en este y la cola comprende un campo de código de redundancia cíclica (CRC) para detección de errores.

Dado que USB maneja un protocolo complejo, existen varios tipos de paquetes con funciones específicas que permiten el intercambio de información por el bus. Estos paquetes son: *Token*, Datos, Validación y Preámbulo[*].

El paquete *Token* contiene la información necesaria para establecer la comunicación en el bus. Estos paquetes especifican la dirección del dispositivo, el *Endpoint* y el sentido o el objetivo de los datos a transmitir. Para lo último utiliza cuatro clases de identificadores de paquete diferentes: *IN* (datos hacia el *Host*), *OUT* (datos desde el *Host*), *SETUP* (datos de control) y *SOF* (inicio del Frame). El *Host* es quien siempre hace uso de los paquetes *Token* puesto que son la herramienta principal para administrar el acceso al medio.

El paquete de Datos contiene la información que el *Host* desea enviar o recibir. La longitud generalmente está determinada por el tamaño del *EndPoint* de destino, sin embargo, la longitud puede estar entre 0 y1023 *bytes*. Existen además dos clases de paquetes de Datos: Data0 y Data1. USB hace uso de ellos alternadamente, como medida para detectar la pérdida o la corrupción de paquetes.

El paquete de Validación es enviado por el Periférico receptor de los datos y permite reportar al *Host* la correcta entrega de estos. Existen tres tipos de paquetes de Validación: Ack que indica una transacción exitosa, Nak que indica que el dispositivo está ocupado y no puede atender los datos que llegan y Stall que indica que el

[*] El paquete preámbulo solo se maneja para dispositivos *Low Speed* y no se contempla en el presente texto.

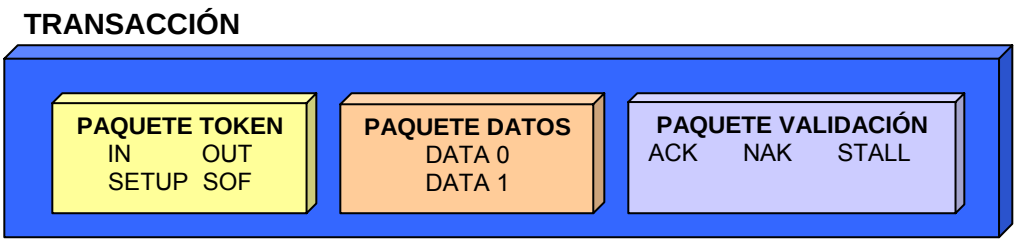
dispositivo no está en condiciones de recibir datos. De esta forma USB permite un control de flujo, evitando la saturación del dispositivo Periférico.

USB organiza el intercambio de datos con los diferentes dispositivos mediante la transmisión de una secuencia de paquetes denominada **Transacción**. En una Transacción se lleva a cabo la transmisión de datos del tamaño permitido por el *EndPoint* y consta generalmente de un paquete *Token*, un paquete de Datos y un paquete de Validación. La figura 8 muestra la secuencia de paquetes que conforman una transacción.

De manera global, se puede decir que el protocolo USB procede de la siguiente manera: el *Host* envía un paquete *Token* cuando desea establecer la comunicación con un dispositivo, especificando la dirección, el *EndPoint* y la naturaleza de la transacción a realizar. Seguidamente, quien va a hacer uso del Bus, ya sea el Periférico o el *Host*, envía un paquete de datos con la información correspondiente. Si el *Host* fue quien envió los datos, el Periférico le confirma la entrega enviando un paquete de Validación tipo Ack, completando de esta manera una transacción USB. La figura 9 ilustra todo el procedimiento de una transacción.

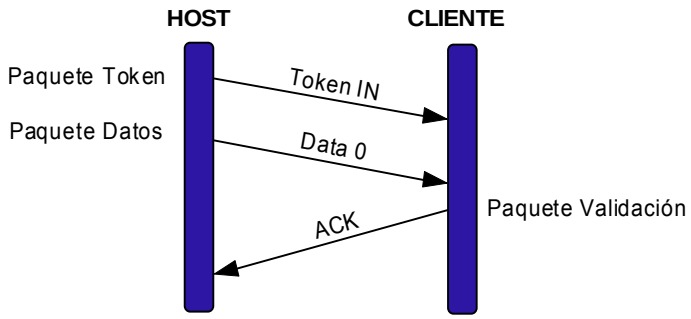
USB divide el tiempo de transmisión de datos en intervalos de 1ms denominados tramas o *frames* para velocidades de *Low Speed* y *Full Speed,* ó en intervalos de 125us denominados *microframes* para *High Speed K* [USB-IF5].

Figura 8. Componentes de una Transacción



Fuente: Los Autores

Figura 9. Envío de datos del Host a un Periférico, mediante una Transacción



Fuente: Autores

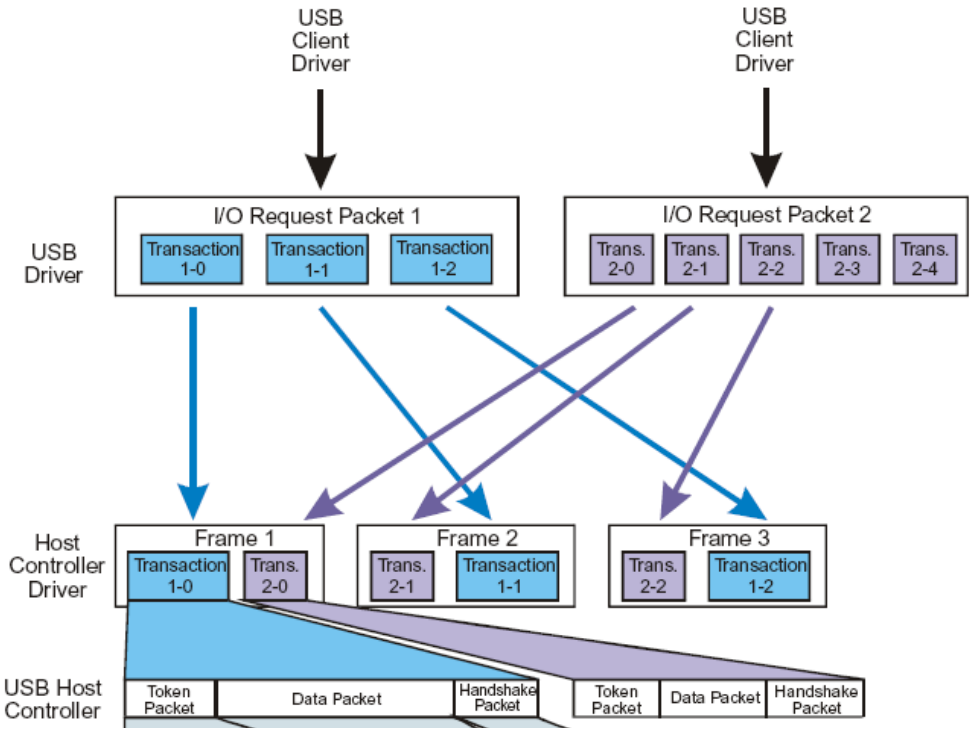
El *Host* USB controla la comunicación con los diversos dispositivos conectados al Bus estableciendo transacciones y organizándolas en un *Frame*, mediante la técnica de multiplexación por división de Tiempo. De igual manera, las transacciones son conformadas de acuerdo con la petición de algún tipo de transferencia específica, hecha por la aplicación. Así, una transferencia se lleva a cabo mediante varias transacciones y una transacción mediante el intercambio de paquetes. La figura 10 ilustra este proceso.

1.3.5 INTERFAZ FÍSICA

1.3.5.1 Interfaz Eléctrica. USB tiene una configuración de 4 cables por los cuales se transmiten, datos y señales de alimentación. El Bus provee la alimentación de los dispositivos en el mismo puerto de conexión. Esto se analizará en el apartado de consumo de potencia donde se verán algunas características que los dispositivos USB tienen respecto a la alimentación.

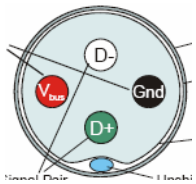
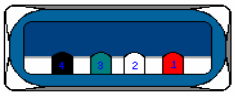
La tabla 1 muestra los cables que están presentes en el puerto USB y su propósito.

Figura 10. Conformación de las unidades de datos en las diversas capas de USB



Fuente: ANDERSON, Don. USB System Architecture. Menlo Park, CA : ADDISON-WESLEY. 2001. ISBN: 0-201-46137-4

Tabla 1. Descripción de pines del bus USB

	Pin de conexión	Señal	Descripción
	1	VBUS	Alimentación USB 5V
	2	D-	Datos
	3	D+	Datos
	4	GND	Tierra de la alimentación

1.3.5.2 Interfaz Mecánica. En USB una conexión desde el *Host* a un dispositivo recibe el nombre de *downstream* mientras que un puerto orientado de un dispositivo al *Host* recibe el nombre de conexión *upstream*.

USB tiene diversos tipos de conectores con características mecánicas determinadas que facilitan su identificación. Según el tipo de conector se puede identificar la función del dispositivo en USB, es decir si se trata de un dispositivo periférico o un *Host*. Esta característica de la interfaz mecánica permite un uso sencillo de los dispositivos USB sin lugar a errores en la conexión entre estos. A continuación se listan los tipos de conectores existentes actualmente:

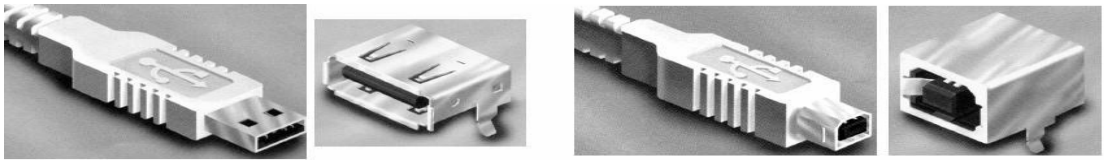
- Conectores tipo A: estos conectores están asociados a los dispositivos *Host* ya sea un PC o un *Host* USB embebido. Los conectores *plug* se implementan para las conexiones *upstream* (hacia el *Host o* hacia un *Hub*) y los receptores Tipo A se usan en los dispositivos *Host* para las conexiones *downstream* (desde el *Host* o un Hub hacia los periféricos).
- Conectores tipo B: tiene relación con los dispositivos periféricos. Los conectores *plug* están implementados para las conexiones downstream hacia los dispositvos periféricos. Los receptores B se implementan en los periféricos o en los Hub.

Actualmente existen otros tipos de conectores USB que se están introduciendo al mercado con los dispositivos portátiles con tecnología OTG [PHILIPS]. Estos conectores son de dimensiones reducidas y se conocen como conectores tipo MiniA, MiniB y MiniAB.

Los conectores miniA y miniB están diseñados para la conexión de *Hosts* y periféricos respectivamente y tienen dimensiones muy reducidas respecto a los conectores tipo A y B. Los receptores miniAB están implementados en los dispositivos que usan la tecnología OTG. A pesar de que los conectores miniB se implementaron antes de la aparición de la tecnología OTG, ésta ya estaba prevista, por lo cual estos puertos son compatibles para conectarse con los dispositivos OTG.

Los conectores USB se presentan en la figura 11 donde se pueden identificar algunas diferencias entre los conectores estándar y los nuevos tipos de conectores (los puertos presentados en la figura no tienen la misma escala).

Figura 11. Conectores USB.

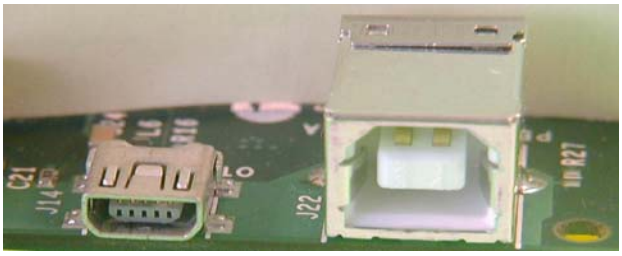


a. Conector y receptor tipo A

b. Conector y receptor tipo B

c. Conector miniA y receptor MiniAB

d. Conector y receptor MiniB



e. Conector *MiniB* vs conector estándar B

Fuente: Los autores.

1.3.5.3 Consumo de Potencia. En USB los dispositivos se identifican de forma distinta según el tipo de alimentación que usan. Existen dispositivos que se alimentan directamente al bus USB (***bus-powered devices***) y dispositivos que tienen su propios terminales de alimentación, denominados dispositivos autoalimentados (***self-powered devices***).

Cualquier dispositivo que se conecta al bus (ya sea autoalimentado o alimentado por el bus) no puede consumir mas de 100mA antes de ser configurado. En el momento de la enumeración, un dispositivo o un *Hub* proveen la información respectiva de los requerimientos de demanda de corriente al *Host* y este determinará si está en capacidad de cumplir con dichos requerimientos. Sin embargo, un dispositivo alimentado por el bus no podrá consumir más de 500mA.

1.3.6 CLASES USB

En la tecnología USB se han definido conjuntos de dispositivos que poseen características y servicios similares. Estos conjuntos se denominan **Clases USB** y permiten identificar las características principales del grupo, como la forma de transferencias de datos y los servicios que pueden prestar, para que todos los dispositivos que conforman una clase puedan ser controlados por un mismo *Driver* que se acomode a las exigencias de una clase o conjunto de dispositivos y no de cada dispositivo individualmente.

En USB, las características principales de las clases están detalladas en las **especificaciones de clase,** las cuales determinan la forma como un dispositivo se comunican con el *Host* y los servicios que pueden ofrecer. Con base en estas, se configuran *Drivers* que puedan controlar las diversas clases de dispositivos. También existen *drivers* adaptativos o genéricos, que el *Host* implementa para dispositivos que no pertenecen a una clase definida y *Drivers* propietarios, que son diseñados por los fabricantes de dispositivos de aplicación específica.

Debido a la gran variedad de dispositivos con conectividad USB, actualmente se encuentran establecidas varias clases USB. Dentro de las clases también pueden existir subclases, donde se manejan protocolos especiales desarrollados por los fabricantes. La tabla 2 resume algunas de las clases USB que están actualmente definidas.

Algunas de las clases más comunes y de mayor uso son: la clase de interfaz humana HID, la clase de dispositivos de comunicación, los *Hubs*, la clase de almacenamiento masivo **MSC** y la clase de impresoras entre otras.

1.4 HOST USB

Para tener una idea más clara de la participación del *Host* en el sistema USB, de las tareas que este lleva a cabo y de las entidades que lo conforman, a continuación se

Tabla 2. Clases USB

CLASE	Características o ejemplos
Audio	Dispositivos de audio y música, sistemas de sonido
Chip/Smart Card interface Devices (CCID)	Dispositivos con Tarjetas inteligentes (*Smart Card*)
Common Class (CCS)	Dispositivos genéricos
Communications Device	Módems, teléfonos e interfaces de redes
HID	Dispositivos de interfaz humana como el *mouse* o el teclado.
Hub	Hub o concentradores de USB
IrDA	Dispositivos de infrarojo
Mass Storage	Discos duros, CD-ROMs, DVD-ROMs y memorias Flash portátiles
Monitor	Monitores de PCs y dispositivos de despliegue
Physical Interface Devices	*Joystics* y controles de juegos
POS Terminals	Dispositivos de Puntos de salida como registradoras
Power	Dispositivos con control de alimentación
Printer Class	Impresoras
Imaging Class	*Scaners* y cámaras

exponen con más detalle las principales diferencias entre un dispositivo Periférico y un *Host* USB y se presentan las funciones que el *Host* esta encargado de realizar como administrador de la comunicación.

1.4.1 HOST Y PERIFÉRICO

En el camino de estudiar la tecnología USB, para el desarrollo de un sistema basado en *Host* USB, es necesario analizar los actores que interactúan en una comunicación por el Bus. Estos entes son el *Host* y el Periférico.

Como se mencionó anteriormente, USB es un protocolo tipo Maestro-Esclavo, donde el Periférico se encarga de proveerle servicios adicionales al *Host* y este último administra la complejidad del sistema USB a su cargo.

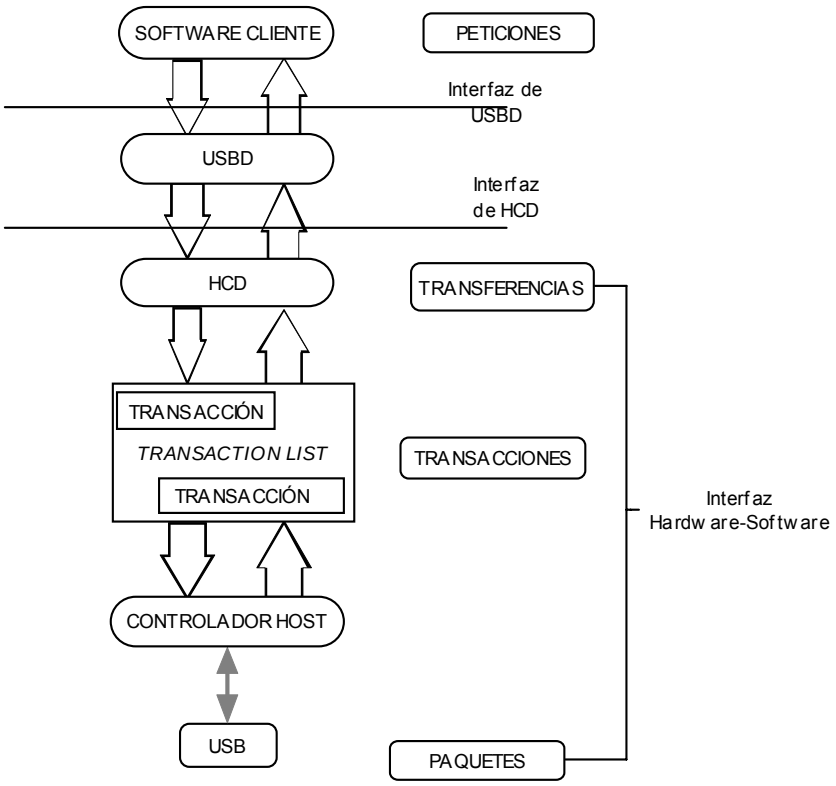
Las características principales de un dispositivo Periférico son:

- Provee servicios adicionales al *Host*, de acuerdo a funciones o aplicaciones implementadas en el dispositivo.
- Es un ente pasivo en la comunicación USB. Está atento a las indicaciones que sean enviadas por el *Host.*
- Hace uso del protocolo USB para establecer una comunicación lógica entre la función que ofrece y el software de aplicación en el *Host*.
- Puede soportar uno o más tipos de transferencias, dependiendo de la naturaleza del Hardware y de la clase a la cual pertenece (como mínimo soporta transferencias de control).
- Mantiene información con características relevantes de si mismo, que le proporciona al *Host* para que realice el proceso de configuración.

Entre tanto, USB es un sistema que proporciona una interfaz de comunicación flexible y robusta donde no sólo maneja un protocolo de comunicación sino también proporciona, mediante procesos adicionales, servicios que hacen fácil para el usuario la conexión de dispositivos USB, como la configuración automática y la capacidad de soportar diversidad de dispositivos, entre otras (véase apartado 1.3, Generalidades de USB). El *Host* como administrador de la comunicación USB es el encargado de llevar a cabo este compendio de tareas, además de las funciones de controlar el acceso de los dispositivos al Bus, el control de errores, el control del flujo de la comunicación y de configurar (de acuerdo a la naturaleza del dispositivo Periférico) los mecanismos necesarios para establecer una comunicación con un Periférico.

Con base en lo anterior, se evidencia que el desarrollo de un *Host* USB es de mayor complejidad que el de un dispositivo Periférico USB, y por ello generalmente es el PC quien tiene inmerso el *Host*. Sin embargo, nuevos mecanismos para incorporar un *Host* USB a sistemas embebidos (como el propuesto en este trabajo) están siendo desarrollados para ampliar la cobertura de conexión de dispositivos.

Figura 12. Esquema de las entidades presentes en el Host



Fuente: Los Autores

1.4.2 EL HOST USB COMO ADMINISTRADOR DEL BUS

El *Host,* como el encargado del funcionamiento del sistema USB, distribuye todos los procesos en cuatro entidades principales: El Software Cliente, el *Driver* de USB (USB *Driver*), el *Driver* del Controlador *Host* (*Host Controller Driver)* y el Controlador *Host* (*Host Controller*). Así mismo, entre cada entidad existen interfaces que permiten el flujo de datos entre ellas, tal y como lo muestra la figura 12.

El **software Cliente** es la entidad donde se encuentra inmersa la aplicación. Esta entidad se encarga de determinar el tipo de transferencias necesarias para comunicarse con los *Endpoints* de la Función en el dispositivo y enviar el flujo de datos por medio de las *Pipes*.

El ***Driver* de USB** se encarga de realizar tareas propias del sistema USB como el control del acceso al Bus, la configuración de los dispositivos y la administración de recursos (ancho de banda y consumo de potencia). Esta entidad configura automáticamente un dispositivo tan pronto detecta su conexión al bus. Posteriormente se habilita para atender las peticiones del software Cliente. Se encarga también de organizar las diferentes transacciones, necesarias para llevar a cabo una transferencia [ANDERSON].

El ***Driver* del Controlador *Host*** (*Host Controller Driver)* se encarga de convertir la información de las transacciones en un formato especial que entiende el controlador *Host,* denominado ***Transaction Descriptor***, el cual contiene parámetros como el tamaño de los datos en *bytes*, la dirección del dispositivo y el número del *Enpoint* destinatario. El conjunto de transacciones que tendrán lugar en un *Frame* es denominado ***Transaction List.***

El **Controlador *Host*** traduce la información contenida en el *Transaction List* a paquetes USB, para ser enviados por el Bus. Esta entidad posee mecanismos para reportar el estado de la transacción y asegura que los requerimientos del protocolo y de la capa física se cumplan.

1.5 CLASE USB DE ALMACENAMIENTO MASIVO MSC

La clase de almacenamiento masivo (**MSC** por sus siglas en inglés) agrupa a los dispositivos que requieren grandes transferencias de datos. Soporta un número diverso de dispositivos como discos duros USB externos y memorias Flash portátiles.

De acuerdo a las especificaciones de clase de almacenamiento masivo [USB-IF4], los dispositivos de esta clase soportan un determinado juego de comandos estándar que le indican las acciones a ejecutar (como lectura o escritura). Estos comandos son

transferidos entre el *Host* y el dispositivo mediante protocolos de transporte de comandos definidos por *MSC*. Entre los juegos de comandos estándar más utilizados se encuentran:

- Juego de comandos primarios SCSI [NCITS1], [NCITS2]
- Juego de comando ATAPI

Las especificaciones de clase de MSC definen dos protocolos para el transporte de comandos: el *Bulk only transport* **BOT** y el *Command Block Interrupt* **CBI**. Cada uno de estos modos de transporte de datos maneja diferentes tipos de transferencias. El BOT maneja transferencias tipo *Bulk* y de control mientras que el modo de transmisión CBI maneja transferencias de control, de tipo *Bulk* y de interrupción. El BOT es el modo de transporte más usado actualmente por los dispositivos de almacenamiento. Generalmente los sistemas operativos soportan ambos tipos de transporte.

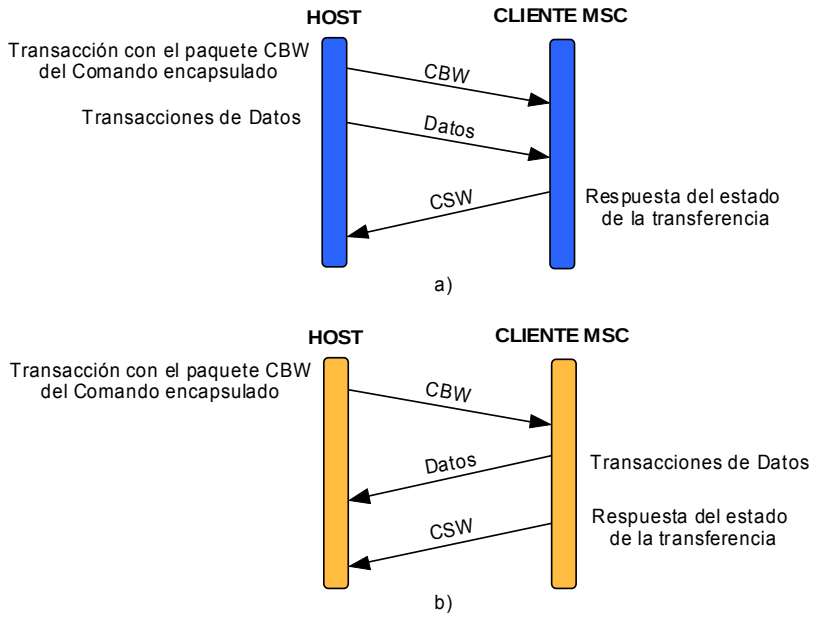
El *Host,* mediante la petición de descriptores, extrae información específica de los dispositivos de la clase MSC, donde identifica protocolo de transporte, el juego de comandos que soporta el dispositivo y la capacidad de los *buffers* de transmisión y recepción de datos.

1.5.1 PROTOCOLO DE TRANSPORTE BOT

En el protocolo de transporte **BOT**, el *Host* se comunica con el dispositivo mediante el intercambio de tres transacciones, con el objeto de enviar: el comando de la acción a tomar por el dispositivo de almacenamiento, los datos que se desean transmitir y el acuse respectivo de la transmisión [USB-IF2].

Los juegos de comandos se encapsulan en paquetes de datos USB especiales, denominados **CBW** (*Command Block Wrapper*), los cuales tienen una longitud y una estructura definida (véase anexo B). Los datos se envían en paquetes de datos regulares. Existe también un tipo de paquete de datos especial denominado **CSW**

Figura 13. Intercambio de datos entre el Host y un dispositivo USB MSC



a) Envío de datos del *Host* al Periférico. b) Envío de datos del Periférico al *Host*

Fuente: Los Autores

(Command Status Wrapper) que es enviado por el dispositivo Periférico como acuse de la transmisión. La figura 13 ilustra el protocolo de transporte BOT.

De la misma forma, en el protocolo BOT existen peticiones de clase propias para los dispositivos de almacenamiento, como son: el *Reset* de MSC y la petición de número de unidades lógicas.

1.6 SISTEMA DE ARCHIVOS FAT32

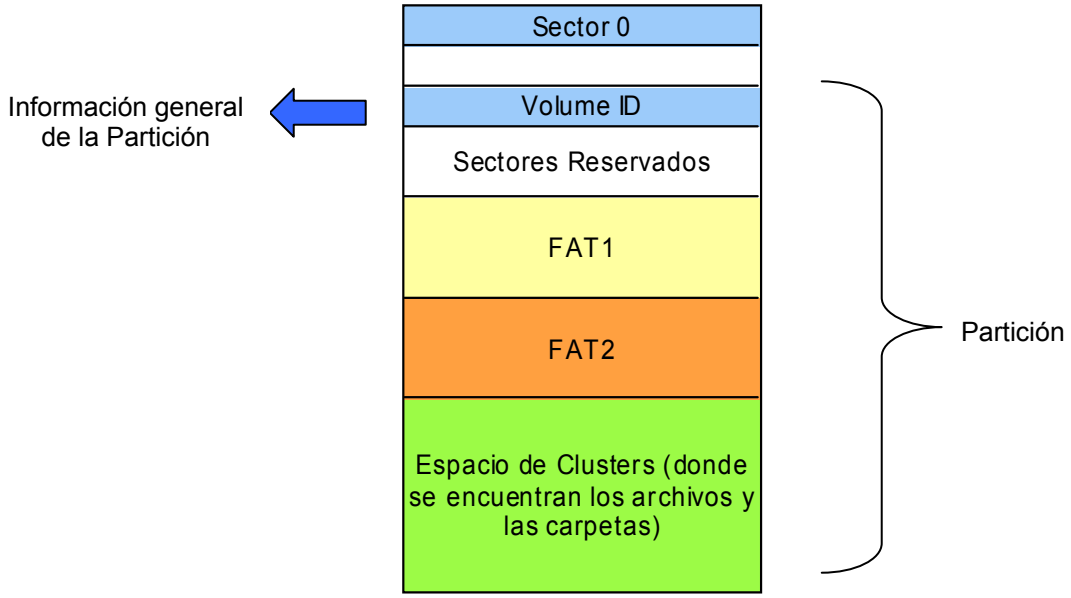
El sistema de archivos es el mecanismo principal que usan sistemas operativos como Windows para el manejo de archivos y carpetas. Los dispositivos de almacenamiento portátiles que interactúan con un PC deben soportar uno de los sistemas de almacenamiento compatibles con el sistema operativo, para que el intercambio de información sea exitoso. Por lo general, estos dispositivos tienen implementado el sistema FAT32 para manejo de datos.

El sistema de archivos FAT (más específicamente FAT32), posee unidades o bloques lógicos, denominados Sectores, generalmente de 512 bytes. La información de archivos y carpetas se almacena en conjuntos de Sectores llamados *Clusters*. Por otra parte, FAT permite la existencia de particiones (divisiones lógicas de la memoria física), siendo necesario que cada partición tenga: un Sector de información general de esta (llamado Volume ID), una tabla donde se encuentra la ubicación de los clusters de los diferentes archivos y carpetas, exceptuando el primero, llamada tabla de FAT y un espacio de *Clusters* donde se guardan los datos de los archivos. En este último se localiza el directorio raíz que es el directorio donde se encuentra almacenada la información de las características de los archivos y carpetas (como los nombres, las extensiones, la longitud y la ubicación del primer cluster) [MICROSOFT2]. Todo lo anterior hace ver a la memoria como una estructura lógica, tal y como lo muestra la figura 14.

Cabe resaltar que un archivo puede requerir varios clusters y no siempre se encuentran todos consecutivamente, por lo que en la tabla de FAT suele ser necesario seguir la <<cadena>> de Clusters para dar con el paradero del archivo.

Si se desea profundizar en los conceptos del sistema de archivos FAT32, las referencias [MICROSOFT1], [MICROSOFT2] y [DOBIASH] que están disponibles en la pagina web de Microsoft detallan los conceptos presentados en este apartado y describen específicamente las estructuras de los campos de información que se usan en este tipo de formato.

Figura 14. Estructura lógica de una unidad de almacenamiento con formato FAT32



Fuente: Los Autores

2. DISEÑO DEL HARDWARE Y REVISIÓN DEL SISTEMA EMBEBIDO

La descripción de los conceptos expuestos hasta el momento permiten una comprensión general de los requerimientos presentes en el desarrollo de un sistema que pueda controlar dispositivos USB y específicamente un sistema capaz de soportar memorias portátiles Flash USB.

El presente capítulo detalla el proceso de diseño de hardware de un sistema embebido con las anteriores características. Previamente a ello se contextualiza el problema que este sistema pretende solucionar y se presentan los requerimientos que orientaron su diseño.

2.1 PLANTEAMIENTO DEL PROBLEMA Y REQUERIMIENTOS DEL SISTEMA

El desarrollo de sistemas embebidos con mayores funciones y prestaciones, ha acentuado la necesidad de medios de almacenamiento con alta capacidad, flexibles para su expansión o reemplazo y que ofrezcan facilidad para el transporte de los datos. Con el objetivo de ofrecer a nuevos diseños de sistemas embebidos las características antes mencionadas, se propone la utilización de una memoria Flash USB portátil gracias a su alto volumen de almacenamiento (ligado al avance de la tecnología de memorias Flash), facilidad de conexión con el PC por el puerto USB (facilitando el manejo y el acceso de los datos al usuario), posibilidad de reemplazo y expansión sin afectar el hardware, reducido costo y creciente incidencia en el mercado.

Debido a que esta clase de dispositivos interactúan únicamente con un *Host* USB (que generalmente es un PC), consecuencia de la característica Maestro-Esclavo del protocolo USB, se propone el desarrollo de un sistema embebido que, sin la asistencia

de un PC, controle memorias *Flash* USB portátiles. La puesta en marcha de éste sistema requiere básicamente:

- Una etapa de Hardware para el manejo de potencia de las entidades presentes en el sistema total, incluyendo los dispositivos USB conectados al Bus.
- Un controlador, núcleo del sistema, responsable de la ejecución del programa y de la administración de los recursos del sistema (Hardware y Software).
- La implementación de un *Host* USB embebido para la configuración y soporte de dispositivos Clientes USB, dada la característica Maestro-Esclavo del protocolo USB. El *Host* USB embebido implica el desarrollo de una plataforma de hardware basada en un controlador *Host* que se encargue de las capas inferiores del protocolo USB, un *Driver* para dicho controlador y un *Driver* para el control del sistema USB y la gestión de la comunicación por el bus.
- Un *Driver* para controlar dispositivos USB de la clase de almacenamiento masivo de datos (MSC), dado que es la clase a la que pertenecen las memorias Flash USB.
- La implementación de un sistema de archivos basado en FAT32 para la escritura de archivos dentro de la memoria portátil USB, debido a que este es un formato ampliamente usado por estos dispositivos para poder interactuar con un sistema operativo como *Windows* y *Linux*.

Como se mencionó en el apartado 1.1.4, la aplicación específica[*] que hará uso del sistema a desarrollar está orientada al almacenamiento de los datos procesados del espectro de impedancia eléctrica de tejido de cuello uterino [MIRANDA]. Por tanto, los requerimientos que tienen relación con dicha aplicación son:

- El sistema debe desarrollarse basado en un Controlador Híbrido-Procesador de Señales Digitales (DSP) de la familia 56800 de *Freescale*, debido a que este el controlador que ha sido seleccionado para llevar a cabo la medición del espectro de impedancia eléctrica,[GARCIA-VARGAS].

[*] Cabe resaltar que el diseño del sistema H-ARD servirá para dar soporte a diversas aplicaciones

- Es necesario que se puedan almacenar datos en una memoria USB.
- Se debe contar con una interfaz en hardware amigable para el usuario.
- Se debe desarrollar un software adicional para PC con el cual se puedan visualizar los datos del espectro de impedancia almacenados en la memoria portátil.

2.2 HARDWARE DEL SISTEMA H-ARD

El **Sistema Embebido para el almacenamiento de datos en Memorias Flash USB, H-ARD** diseñado e implementado en este trabajo, es un sistema que controla dispositivos USB, pertenecientes a la clase USB de almacenamiento masivo *MSC.* Este sistema requiere un *Host* USB embebido[*] que le permita interactuar con una memoria conectada al puerto USB. En esta sección se describe el diseño, desde el punto de vista de hardware, del *Host* USB embebido en el sistema **H-ARD**. Se muestran los requerimientos tomados en cuenta para la selección de dispositivos y las características de diseño del circuito impreso de la tarjeta. El diseño de este módulo de hardware junto con los requisitos iniciales del proyecto permitirán sentar las bases para el diseño de software del sistema final.

2.2.1 SISTEMA H-ARD Y TARJETA HC-USB

Como ya se mencionó, los requerimientos de la aplicación plantean que el proyecto este basado en un controlador híbrido-DSP de la familia 56800 de *Freescale*™ y dado que estos no tienen una interfaz física de comunicación que permita la implementación de un *Host* USB, se hace necesario la utilización de un **Controlador *Host*** independiente que trabaje en conjunto con el controlador principal del sistema. De esta forma se propone un sistema que, con una arquitectura distribuida, utilice un controlador híbrido-DSP de la familia 56800 de *Freescale* encargado del control principal del

[*] *Host* USB embebido se denomina a los dispositivos, diferentes a un PC que pueden interactuar con periféricos de USB los cuales tienen una aplicación específica.

sistema y un ASIC Controlador *Host* como co-procesador, el cual proporciona las herramientas de hardware necesarias para el diseño de un *Host* USB embebido.

Cabe resaltar que el diseño y la implementación del hardware relacionado con el controlador central (el controlador híbrido-DSP) pertenecen a otros trabajos relacionados con la investigación de la cual este proyecto hace parte [MIRANDA]. Específicamente se trabajó con el dispositivo diseñado por Juan Carlos Vargas y Cristihan García [GARCIA-VARGAS], el cual utiliza un Controlador Híbrido DSP56F805 de *Freescale*™ (este sistema será denominado **Tarjeta DSP805** para efectos de referencia en este texto)[*]. El Controlador Híbrido-DSP ejecuta las tareas principales del sistema de almacenamiento, controlando a su vez, la tarjeta HC-USB y el módulo de interfaz con el usuario (representada por los controles de decisión y la pantalla LCD).

En consecuencia, el desarrollo de hardware para este proyecto se centra en la implementación de la tarjeta del controlador *Host* (denominada **Tarjeta HC-USB**), que junto con un *firmware* embebido ofrece soporte de las capas bajas del protocolo USB, permitiendo al sistema H-ARD el acceso a los dispositivos USB, tal como un *Host USB embebido*.

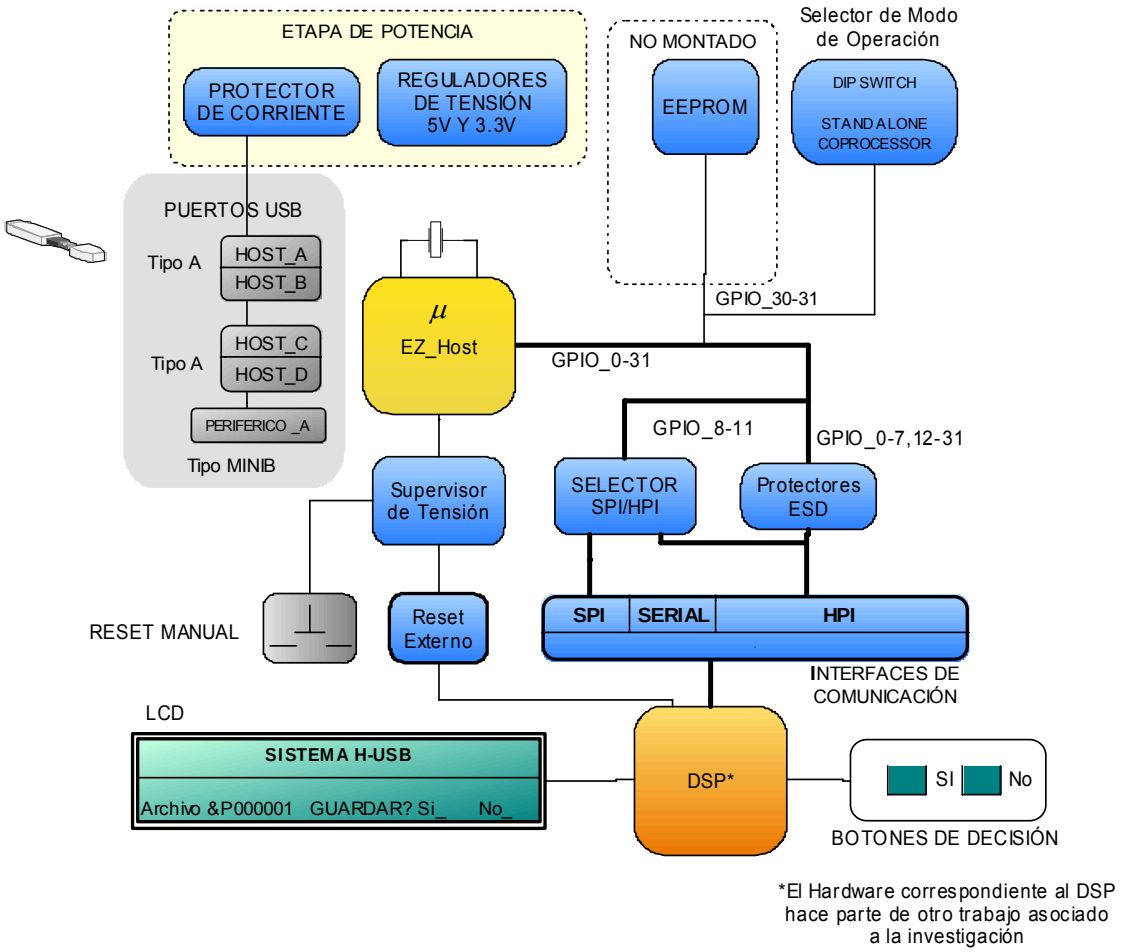
El diseño de la tarjeta HC-USB se orientó bajo los criterios de reducción en consumo de potencia, tamaño de implementación, independencia del PC, inmunización del dispositivo frente a los efectos eléctricos de sobrecargas e interferencias y a la selección de dispositivos construidos con tecnología actual, entre otros. La figura 15 ilustra los módulos funcionales presentes en el hardware del sistema **H-ARD** los cuales se describen a continuación.

2.2.2 CONTROLADOR HOST

El controlador *Host* es la entidad en *Hardware* más importante para la implementación de un sistema *Host* USB embebido debido a que este módulo tiene a su cargo la

[*] Véase 2.2.8 Controlador Central DSP

Figura 15. Diagrama de Bloques del Sistema H-ARD



Fuente: Los Autores

ejecución de las tareas de la Capa de Interfaz del Bus USB*, las cuales permiten entablar una comunicación con un cliente USB.

Como ya se ha descrito, la característica maestro-esclavo de la comunicación USB facilita la implementación de dispositivos periféricos aumentando por otro lado la complejidad en el *Host*, con base en el hecho de que toda la carga de control del Bus recae sobre este último. Por lo tanto, el desarrollo de un *Host* USB requiere entidades de hardware y software especializados y con un mayor grado de complejidad para la administración de la comunicación USB.

* La capa de interfaz de BUS se refiere a las capas física y al núcleo del protocolo USB.

En el sistema H-ARD específicamente, el Controlador *Host* trabaja como un coprocesador, bajo el mando del controlador principal (Controlador Híbrido-DSP). La tabla 3 resume las principales características de diversos controladores *Host* actualmente disponibles en el mercado, capaces de ofrecer el soporte para desarrollar un *Host* USB.

Tabla 3. Controladores *Host* Disponibles en el Mercado

Referencia	Fabricante	Empaquetado	Puertos USB	Pines	No. min. conexiones con el DSP	Precio USD
	Característica					
CY7C67300 EZ Host	**Cypress**	**TQFP**	**4**	**100**	**4**	**6.7**
	Interfaces: UART, HPI, IDE, PWM, HSS, SPI, I^2C EEPROM. Soporte: OTG, Host y periférico; 32 GPIO; memoria interna de datos RAM 15Kbytes; DMA; Procesador de 16 bits; soporte para memorias Ram y/o ROM					
CY7C67200 EZ-OTG	Cypress	FBGA	2	48	3	6.0
	Interfaces: UART, HPI 16 bits, HSS, SPI, I2C EEPROM. Soporte OTG, Host y periférico; 25 GPIO; soporte para memorias externas; DMA; memoria interna RAM 8kx16 bits; Procesador de 16 bits					
SL811HS Host-Peripheral	Cypress	PLCC	1	28	14	8.56
	Bus de datos de 8 bits; DMA (como esclavo); SRAM 256 bytes; reloj de 12 o 48 MHz.					
SL811HST-AC Host-Peripheral	Cypress	TQFP	1	48	14	8.36
	Bus de datos de 8 bits; DMA (como esclavo); buffer SRAM 256 bytes, reloj de 12 o 48 MHz					
ISP1362BD OTG controller	Philips	LQFP	2	64	21 min.	7.10
	DMA; interfaz paralela de 16 bits; protección contra sobrecarga; reloj de 12 MHz; buffer de 4096 bytes (*Host*) y 2462 bytes (*device*).					
ISP1160BD Host controller	Philips	LQFP	2	64	21 min	5.5
	interfaz paralela de 16 bits; reloj externo de 6MHz (interno de 48MHz); DMA					
ISP1161A1 Host / Device Controller	Philips	LQFP	3	64	21 min	7.0
	2 puertos de host (Transferencia de datos máxima 15 Mbyte/s); 1 puerto de dispositivo (Transferencia de datos máxima 11.1 Mbyte/s); reloj de 6MHz (48MHz interno); reloj de salida programable: 3 a 45MHz; DMA; interfaz paralela de 16 bits					
ISP1561BM Host Controller	Philips	LQFP	4	128	--	6.50
	Soporta High speed (480Mbps), interfaz PCI 32-bit 33 Mhz; reloj de 12MHz (48MHz interno), soporta conexión de mouse y teclado (PS/2).					
CSC1206 OTG Flash Disk Controller	Chesen Electronics	LQFP	1	128	--	--
	Actúa como una memoria flash, soporta conexión con memorias USB.					
82C871 OTG Controller	OPTI Technologies	CSP	1	81	22 min	--
		QFP		64		
	Bus de datos de 16 bits, bus de direcciones de 8 bits, reloj de 12 MHz, DMA,					

Entre otras características adicionales, el sistema H-ARD debe contar con un espacio de memoria para manejar los datos tanto de la aplicación (teniendo en cuenta que se va a implementar un sistema de archivos) como de las diferentes capas que lo conforman. Debido a que el espacio de memoria del procesador central (el Controlador Híbrido-DSP) está destinado al proceso de la aplicación[*] y en el proceso de control del sistema total, se hace necesario que en el diseño de la tarjeta HC-USB se incremente la capacidad de memoria RAM del sistema total.

Entre las posibles soluciones para extender la memoria volátil se puede recurrir a la implementación de memorias RAM externas; otra posibilidad es aprovechar las herramientas que ofrecen los Controladores *Host* que se analizan y seleccionar un dispositivo con capacidad en memoria interna que permita implementar *buffers* de datos con volúmenes apreciables. La última solución permite aprovechar las herramientas que el controlador ofrece sin adicionar elementos externos al hardware e incurrir en el uso de más conexiones con el Controlador Híbrido-DSP.

El controlador *Host* debe proveer un soporte robusto en la comunicación USB y en las funciones de *Host*, de tal forma que permita que el *Driver* del Controlador *Host* que se desarrolle en software tenga una complejidad reducida y pueda tener disponible la mayor cantidad de recursos para controlar un dispositivo USB. A su vez, debe ofrecer interfaces de comunicación que permitan minimizar la cantidad de conexiones necesarias con el Controlador Híbrido-DSP.

Con base en los parámetros mostrados anteriormente, se seleccionó el controlador **EZ-HOST™** de Cypress como controlador *Host*, el cual ofrece diversidad de interfaces de comunicación como SPI, HPI y HSS, que le permiten interactuar con otros

[*] Esta aplicación se refiere al proceso de calculo del espectro de impedancia eléctrica o a otro proceso que se este manejando dentro del controlador principal.

procesadores, y ofrece la interfaz IDE para la comunicación con discos duros. Posee 32 pines de propósito general compartidos con dichas interfaces y tiene un espacio en memoria de datos de aproximadamente 15Kbytes, el cual permite aumentar la capacidad de memoria RAM del sistema H-ARD.

El EZ-Host es un controlador que cuenta con un procesador RISC de 16 Bits que opera a 48 MHz. Puede actuar como un controlador independiente (*Stand-alone*) o como un controlador alterno (*Coprocessor*). Es un dispositivo *multifunción*[*] que puede trabajar como *Host* USB o como dispositivo Cliente USB, orientado a diversas aplicaciones donde se use la comunicación USB. Cuenta con 4 puertos de USB controlados por dos SIE[†] configurables (*Host/Peripherial*) y tiene soporte para direccionar 127 dispositivos USB por cada SIE, comportándose como un Hub raíz. En cada puerto es capaz de manejar las velocidades *Low Speed* y *Full speed* (1.5Mbps y 12Mbps respectivamente), compatibles con la versión más actual de USB (USB 2.0) y soporta en uno de sus puertos la tecnología emergente OTG (*On The Go*). Tiene 8Kbytes en una memoria estática *Masked* ROM[‡] que contiene un *Firmware* BIOS que controla las funciones básicas de las diferentes interfaces de comunicación, incluyendo los puertos USB. Soporta memorias externas SRAM y/o ROM; tiene dos módulos de *Timers* y un módulo *Watch Dog* configurables. Adicionalmente tiene un módulo PWM.

Otras de las ventajas que el EZ-Host ofrece como dispositivo controlador *Host*, es el conjunto de configuraciones de sus puertos USB, que se pueden implementar. Con estas configuraciones se pueden desarrollar diversas aplicaciones ya sea para *Host* embebidos o para dispositivos periféricos. La figura 16 ilustra las configuraciones posibles para los puertos USB mediante los SIEs del EZ-Host.

[*] Dispositivo Multifunción (*Multirole* Device): término usado para dispositivos que tienen soporte para actuar como *Host* y como dispositivo periférico USB.
[†] *Serial Interface Engine* por sus siglas en inglés.
[‡] Una memoria Masked ROM es un tipo de memorias programables que pueden ser borradas para re-programarse.

Figura 16. Configuraciones de los puertos USB en el EZ-Host.

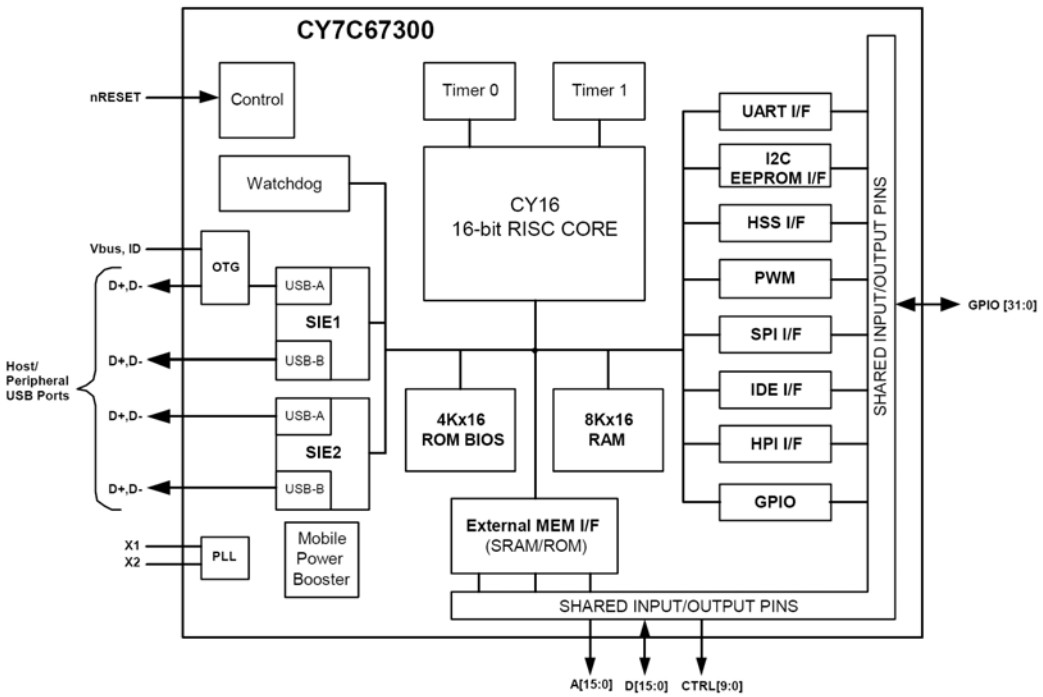
Port Configurations	Port 1A	Port 1B	Port 2A	Port 2B
OTG	OTG	–	–	–
OTG + 2 Hosts	OTG	–	Host	Host
OTG + 1 Host	OTG	–	Host	–
OTG + 1 Host	OTG	–	–	Host
OTG + 1 Peripheral	OTG	–	Peripheral	–
OTG + 1 Peripheral	OTG	–	–	Peripheral
4 Hosts	Host	Host	Host	Host
3 Hosts	Any Combination of Ports			
2 Hosts	Any Combination of Ports			
1 Host	Any Port			
2 Hosts + 1 Peripheral	Host	Host	Peripheral	–
2 Hosts + 1 Peripheral	Host	Host	–	Peripheral
2 Hosts + 1 Peripheral	Peripheral	–	Host	Host
2 Hosts + 1 Peripheral	–	Peripheral	Host	Host
1 Host + 1 Peripheral	Host	–	Peripheral	–
1 Host + 1 Peripheral	Host	–	–	Peripheral
1 Host + 1 Peripheral	–	Host	–	Peripheral
1 Host + 1 Peripheral	–	Host	Peripheral	–
1 Host + 1 Peripheral	Peripheral	–	Host	–
1 Host + 1 Peripheral	Peripheral	–	–	Host
1 Host + 1 Peripheral	–	Peripheral	–	Host
1 Host + 1 Peripheral	–	Peripheral	Host	–
2 Peripherals	Peripheral	–	Peripheral	–
2 Peripherals	Peripheral	–	–	Peripheral
2 Peripherals	–	Peripheral	–	Peripheral
2 Peripherals	–	Peripheral	Peripheral	–
1 Peripheral	Any Port			

Fuente: CYPRESS SEMICONDUCTOR. EZ-Host™, Programmable Embedded USB Host/Peripheral Controller : Data sheet. San Jose, CA : CYPRESS, 2003. 120p. : il. [online]. Disponible en: www.cypress.com.

El EZ-Host ofrece la posibilidad de cargar varios *Transaction Descriptors* al mismo tiempo (que en conjunto se denominan *Tlist*) en un espacio de memoria de tal forma que puedan llevarse a cabo varias transacciones con el traspaso de un sólo *Tlist* (una interacción entre el DSP y el controlador *Host*). Esta característica se presenta como una ventaja frente a otros controladores *Host* que manejan los *Tdesc* a través de registros de memoria pues, en estos casos, cada ejecución de un *TDescriptor* requeriría una interacción entre el *Host Controller Driver*[*] (implementado en el Controlador Híbrido-DSP) y el controlador *Host*. La figura 17 muestra el diagrama de bloques del

[*]Véase 1.4, Host USB.

Figura 17. Diagrama de bloques del controlador EZ-Host



Fuente: CYPRESS SEMICONDUCTOR. EZ-Host™, Programmable Embedded USB Host/Peripheral Controller : Data sheet. San Jose, CA : CYPRESS, 2003. 120p. : il. [online]. Disponible en: www.cypress.com.

EZ-Host. Para mayor información acerca de este dispositivo, referirse a su hoja de datos [CYPRESS2].

2.2.3 INTERFACES DE COMUNICACIÓN

Gracias al soporte que ofrece el EZ-Host, la tarjeta HC-USB cuenta con diversas interfaces de comunicación que lo hacen versátil para aplicaciones con requerimientos de conexión y tasas de transferencias diferentes pues, además de manejar el protocolo de alta velocidad USB, puede comunicarse con sistemas que utilicen puertos SPI, UART, HPI y HSS.

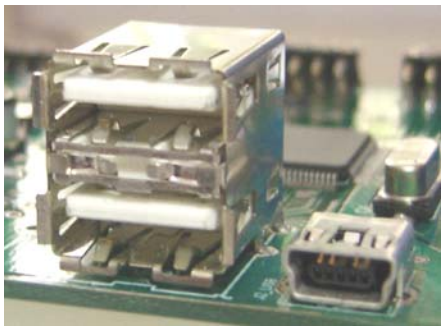
2.2.3.1 Puertos USB. Los puertos de conexión USB en la tarjeta HC-USB se implementan para tener 2 puertos *Host* (con posibilidad de expansión a cuatros puertos

Host) de acuerdo a las posibilidades de configuración de los puertos en el EZ-Host mencionada en el apartado 2.2.1. Los puertos seleccionados son receptores tipo A (definidos para los *Host* USB). Para la tarjeta HC-USB se seleccionaron puertos de doble receptor que permiten implementar 2 receptores USB en el mismo espacio de área de uno sólo.

Por otro lado, en la tarjeta HC-USB se implementó un puerto receptor miniB que permite a la tarjeta conectarse a un *Host* ya sea un PC u otro sistema Host USB embebido. Este puerto, de tamaño reducido, permite conectarse con dispositivos que utilicen la tecnología complementaria OTG*. En La figura 18 se ilustran los conectores seleccionados.

2.2.3.2 Otras Interfaces de Comunicación en la tarjeta HC-USB. En el diseño de la tarjeta HC-USB, se habilitaron todos los pines GPIO del EZ-Host (excepto los GPIO 30 y 31) mediante conectores tipo Pin *Header.* Estos GPIO, como se mencionó en el apartado 2.2.2 en las características del Controlador Host, son compartidos con las diferentes interfaces de comunicación que tiene el controlador. Por tal motivo la tarjeta HC-USB tiene soporte para todas las interfaces de comunicación que ofrece el EZ-Host.

Figura 18. Conectores USB. Receptores Tipo A y tipo MiniB



Fuente: los autores.

* Refiérase al apartado de 1.1.3.

Sin embargo, en la tarjeta HC-USB se implementaron como puertos de conexión independientes, las interfaces HPI y SPI para la comunicación con otro controlador. Estas interfaces de comunicación comparten pines del EZ-Host, por lo que se requirió adicionar un *buffer* para aislar y demultiplexar, en hardware, los dos puertos de conexión. El circuito esta encargado de habilitar una de las dos interfaces para evitar un posible corto circuito.

El dispositivo usado para esta etapa es el bus *switch* de diez pines **SN74CBTLV3384** de *Texas Instruments* el cual es un circuito integrado de tecnología CMOS diseñado para aplicaciones de bajo consumo de potencia. Es un arreglo de *switches* bidireccionales que provee las herramientas necesarias para demultiplexar los pines GPIO que comparten puertos de comunicación SPI y HPI. Debido a que se deben habilitar pines de entrada y salida, este bus *switch* se presenta como una mejor alternativa que un bus *transceiver* dado que este último actúa como un *switch* que habilita un camino en una sola dirección.

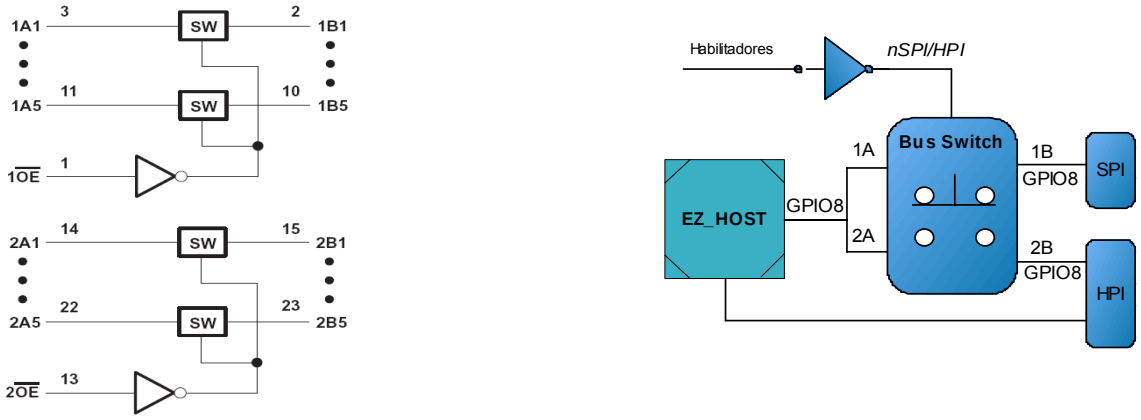
Este integrado del fabricante Texas, tiene dos buses de 5 pines cada uno de los cuales se pueden habilitar de forma independiente. Un circuito inversor de bajo consumo de potencia, el **74AC04** de FAIRCHILD (veáse [FAIRCHILD]) se implementa para habilitar uno de los dos buses. La figura 19 ilustra el circuito lógico del bus switch y su diagrama de conexión en la tarjeta HC-USB.

2.2.4 MANEJO DE POTENCIA

El manejo de potencia en el sistema embebido HC-USB esta conformado por la etapa de alimentación de los puertos *Host* USB[*] y la etapa de alimentación general de la tarjeta. La tarjeta se diseñó orientada al bajo consumo de potencia donde los niveles de tensión de alimentación para los dispositivos seleccionados son de 3.3V y con corrientes de operación reducidas, mientras que en la etapa de alimentación de USB, el

[*] Un puerto Host USB es definido por los autores como el receptor tipo A que utiliza el Host para conectarse con los Periféricos.

Figura 19. Interfaces de conexión en la tarjeta HC-USB.



a) b)

a) Diagrama lógico del bus implementado. b) Diagrama de conexión con el EZ-Host

Fuente: Los Autores

sistema debe manejar niveles de tensión de 5V y consumos de corrientes más elevados (hasta 500mA por puerto).

2.2.4.1 Etapa de Alimentación de los Puertos Host USB. Las especificaciones de USB definen al *Host* como la entidad que debe proporcionar la alimentación a los dispositivos que, al conectarse al bus, la requieran (*USB Bus Powered Devices)*[*]. La tensión de alimentación definida por USB es de 5V y la corriente consumida por un dispositivo no puede exceder los 500mA. En el diseño de la tarjeta HC-USB se presupuestó la conexión simultánea de dos memorias USB con posibilidad de expansión de cuatro, teniendo en cuenta que el EZ-Host tiene 4 puertos de conexión que pueden actuar como *Host.* Esto representa una posible corriente de alimentación de 2A (asumiendo un dispositivo conectado a cada puerto, con un consumo máximo de 500mA).

El manejo de potencia en USB presenta dos fases: la **fase de regulación** donde se ajustan los niveles de tensión a 5V y el presupuesto de corriente a un consumo superior

[*] Refiérase al apartado 1.3.5,Interfaz Física.

a 2A, y la **fase de protección** donde se controla el consumo de corriente de los dispositivos conectados al bus USB.

En la **fase de regulación** es necesario ajustar los niveles de tensión a un valor constante libre de alteraciones (rizado). El nivel de tensión se debe estabilizar mediante la implementación de un regulador de tensión que permita suministrar los 5V necesarios en la etapa de alimentación de los puertos *Host* USB y pueda soportar niveles de entrada de hasta 15V (basados en la variedad de fuentes de DC disponibles en el mercado). Dentro de los criterios más relevantes para la selección del regulador se analizó el consumo de corriente de operación del dispositivo (*Quiescent Current)* y la eficiencia de este.

La búsqueda de los reguladores, tanto para la etapa de alimentación de los puertos *Host* USB como para la etapa de alimentación general de la tarjeta, se realizó dentro de la clase de reguladores lineales, debido a sus características de baja distorsión en la regulación, sus reducidos requerimientos de hardware (configuración mínima de elementos pasivos) y a su costo promedio [CATSOULIS].

El regulador **LT1529CQ-5** del fabricante Linear Tech fue seleccionado para la etapa de alimentación de los puertos *Host* USB, el cual tiene capacidad de entregar 3A en la corriente de salida. Es un regulador que presenta muy bajo consumo de potencia (50μA) con relación a su corriente de salida y respecto a otros reguladores presentes en el mercado. Este regulador tiene la característica de *shutdown* donde, en este estado, no consume mas de 16uA. Presenta estabilidad con capacitancias de valores más bajos que otros reguladores con la misma salida de corriente (normalmente se usan capacitancias de carga superiores a 100uF mientras que el LT1529CQ-5 puede operar con capacitancias de 22uF). Se destaca entre otros reguladores por su eficiencia debido a sus reducida disipación de potencia frente a demandas de corriente; además este regulador se mantiene la tensión de salida estable (con cambios de unos pocos milivoltios) a altas temperaturas de operación, [LINEAR-TECH2].

La **fase de control** está representada por un protector de corriente que alimenta directamente a los dispositivos conectados al bus USB. Esta fase se debe implementar debido a que los dispositivos periféricos pueden demandar altos consumos de corriente que, si exceden el valor limite (en caso de dispositivos no soportados o que presenten fallos) establecido en las especificaciones de USB, afectarían de forma negativa a la tarjeta HC-USB. Cabe destacar que el sistema diseñado en hardware soporta un dispositivo USB en cada puerto, por lo cual se debe tener un presupuesto de corriente mayor al valor límite especificado por USB, el cual es de 500mA.

El protector del fabricante *Texas Instruments* con referencia **TPS2054B** fue seleccionado para la etapa de control en la alimentación de USB. El dispositivo tiene cuatro puertos de salida, implementados con tecnología CMOS; tiene protección de temperatura y de corto circuito [TEXAS1]. Ofrece señales de salida o banderas de aviso de sobrecorriente. Consume una corriente máxima de operación de 140uA; puede soportar una demanda continua de corriente de 500mA con un límite máximo de hasta 1.25A de corriente de salida. Por otro lado, el dispositivo fue diseñado para aplicaciones en sistemas que requieren múltiples puertos para distribución de alimentación. Entre las ventajas más representativas esta la implementación de *switches* de potencia con valores bajos de resistencias (70mΩ), gracias a su elaboración en tecnología CMOS.

2.2.4.2 Etapa de alimentación general de la Tarjeta HC-USB. El regulador para la alimentación del sistema debe manejar las demandas de corrientes máximas de operación de todos los dispositivos y los elementos pasivos presentes en la tarjeta HC-USB. La tabla 4 ilustra los valores máximos aproximados del consumo de corriente de los dispositivos de la tarjeta HC-USB.

El regulador **LT1129-3.3** fue el dispositivo seleccionado; soporta una corriente máxima de de 700mA con bajo consumo de potencia (corriente de consumo de 50uA); tiene la característica de trabajar con valores bajos de condensadores (desde 3.3uF), y

Tabla 4. Sistema HC-USB. Dispositivos conectados a 3.3V

DISPOSITIVO	NÚMERO DE DISPOSITIVOS EN EL SISTEMA	CORRIENTE MÁXIMA DE ALIMENTACIÓN [mA]	IMPLEMENTADO
EZ-Host	1	180	SI
Generador de reset	1	0.015	SI
Bus switch	1	0.3	SI
Inversor	1	0.020	SI
Protectores ESD	6	1u*6 = 0.006	SI
Memoria EEPROM	1	3	NO
Otros Elementos (resistencias, leds)	6	9.5	SI
Consumo aproximado de corriente [mA]		**192,841**	

presenta una salida de tensión regulada de 3.3V. Este dispositivo esta diseñado específicamente para aplicaciones de bajo consumo de potencia y sistemas alimentados por baterías [LINEAR-TECH1].

2.2.5 PROTECCIONES ELECTROMAGNÉTICAS

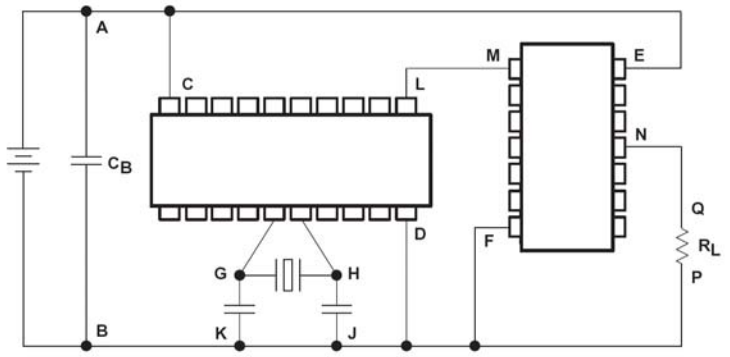
En el diseño de la tarjeta HC-USB se tuvo en cuenta la protección del sistema de interferencias producidas por el mismo circuito y por fuentes externas invasivas. Así mismo, se tuvo en cuenta la protección para posibles descargas Electro-Estáticas y la protección de corrientes elevadas, entre otros efectos electromagnéticos indeseables presentes en un circuito impreso. Entre los dispositivos adicionados se mencionan:

- Capacitores de desacople: Las actividades de *switcheo* de todo el circuito producen variaciones en la fuente de alimentación, lo que puede generar degradación en el rendimiento de los diferentes integrados y a su vez, degradación en el rendimiento del sistema. Por tal motivo se utilizaron capacitores de desacople en los pines

conectados a fuente de los circuitos integrados con el fin de reducir el rizado de la tensión de alimentación y garantizar un correcto funcionamiento. Así mismo, los capacitores ayudan a reducir el ruido presente en la alimentación de los dispositivos.

- Bobinas: Se adicionaron bobinas para reducir la interferencia electromagnética producida por las alteraciones en la corriente a través de los lazos virtuales de alimentación presentes en el circuito; alteraciones que los capacitores de desacople no pueden reducir y que son producidas por la actividad de *switcheo* de los dispositivos digitales del circuito [TEXAS2]. La figura 20 ilustra los posibles lazos que se forman en un circuito. Un ejemplo de un lazo virtual de alimentación es A-C-D-B.
- Dispositivos contra ESD[*]: Aún cuando la mayoría de dispositivos implementados en la tarjeta tienen protección de ESD, es importante mantener al controlador EZ-Host libre de estos efectos. Por lo cual, se protegieron los pines del controlador que conectan directamente con los puertos para conexiones externas tipo *pin Headers*, debido a que estas conexiones directas son posibles medios de riesgo para que el usuario produzca una descarga electrostática al EZ-Host. Los protectores seleccionados, **MSQA6V1W5T2** de ON Semiconductor, son de bajo consumo de potencia y tienen una capacitancia de carga baja (90pF) [ON-SEMICONDUCTOR]. La

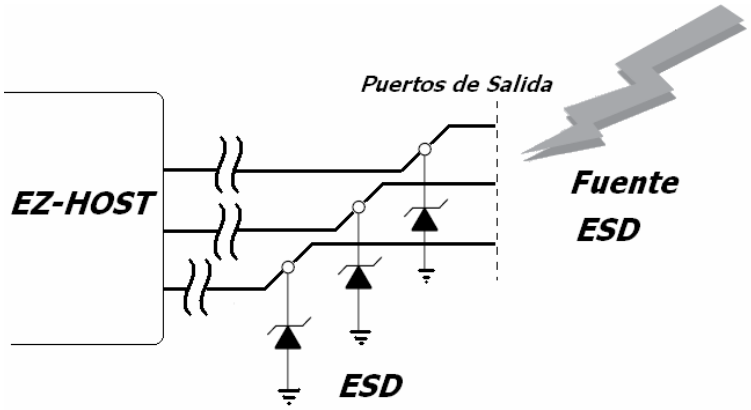
Figura 20. Lazos virtuales de alimentación en un circuito



Fuente: TEXAS INSTRUMENTS. Printed-Circuit-Board Layout for Improved Electromagnetic Compatibility [online].1996. 14p. Disponible en: www.ti.com.

[*] ESD por sus siglas en ingles *Electrostatic Discharge* o Descargas electrostáticas.

Figura 21. Modelo de conexión de los protectores ESD



Fuente: Los Autores

figura 21 ilustra el modelo de conexión para protección de los pines del Circuito integrado.

- Fusible: Se implementó un fusible de 3A de operación continua para proteger todo el circuito de corrientes excesivas o corto circuitos que se puedan producir por manejos inadecuado de la tarjeta o por incorrecto funcionamiento de dispositivos externos conectados a la misma (ya sea mediante USB, SPI o mediante cualquier otro puerto).

2.2.6 SERVICIOS A NUEVAS APLICACIONES

Además de diseñar la tarjeta HC-USB a partir de los requerimientos iniciales del problema, se buscó realizar una implementación de hardware que permitiera ofrecer servicios a nuevas aplicaciones, aprovechando las herramientas que el controlador EZ-Host ofrece. Por lo cual para en el diseño del PCB de la tarjeta HC-USB se tuvo en cuenta el soporte para posibles expansiones y servicios adicionales a los usados en este proyecto, entre los cuales están:

- **Puertos USB**: Tanto el presupuesto de consumo de corriente como el diseño de la tarjeta permiten la posibilidad de expansión de 2 a 4 puertos USB tipo A (*Host*) con el fin de usar todos los puertos *Host* disponibles en el EZ-Host.

- **Modo Stand *Alone***: Se habilitó un *switch* para seleccionar manualmente el modo de operación del controlador, ya sea independiente (*Stand Alone)* o como coprocesador, abriendo campo a nuevas aplicaciones donde se pretenda explorar el uso del EZ-Host como un controlador independiente.
- **Memoria externa EEPROM**: Con base en las especificaciones del EZ-Host, un programa (una aplicación) puede ser cargado en una memoria EEPROM para que el BIOS pueda descargarlo a la memoria RAM interna de programa y posteriormente pueda ser ejecutado, por lo cual en el diseño del circuito impreso se dejó la posibilidad para adicionar una memoria externa, con interfaz I2C, mayor a 2Kbits[*]. El dispositivo de referencia utilizado fue AT24C512 de Atmel®.
- **USB Periférico**: Como ya se ha mencionado, el EZ-Host es un dispositivo multifunción que se puede configurar como un *Host* USB o como un Cliente USB. El BIOS del controlador acepta por defecto la descarga de programas desde el PC directamente a la RAM interna o a una memoria EEPROM externa, a través de un puerto USB tipo B (cliente), razón por la cual en la tarjeta se implementó un puerto *minib* para accesar al EZ Host y descargar programas. Adicionalmente, este puerto permite que el controlador se pueda trabajar como un dispositivo Cliente para cualquier aplicación que así lo requiera. La ventaja de usar un conector tipo *minib* en contraste con uno tipo B es que permite el soporte a la tecnología emergente OTG:

2.2.7 DISEÑO DEL CIRCUITO IMPRESO DE LA TARJETA HC-USB

La tarjeta HC-USB se compone principalmente de dispositivos de montaje superficial y su implementación está basada en una placa de doble cara. El diseño del PCB de dicha tarjeta se planteó bajo los criterios básicos de diseño de Circuitos Impresos constituidos por dispositivos tipo *SMT(Surface Mount Technology* por sus siglas en inglés*)* y los criterios para el desarrollo de hardware con interfaz USB. Adicionalmente se tuvo en cuenta tanto las recomendaciones de profesionales en el diseño de PCB y la

[*] Este valor se define de acuerdo a los pines del Ez-Host.

experiencia de trabajos de grados anteriores* como las recomendaciones de los fabricantes de los dispositivos seleccionados.

2.2.7.1 Compatibilidad Electromagnética. Para el diseño del PCB la tarjeta se tuvo en cuenta las técnicas para el mejoramiento de compatibilidad electromagnética [TEXAS2], dentro de las cuales se atacó principalmente las interferencias electromagnéticas EMI y las descargas electrostáticas ESD.

La protección de EMI fue un factor importante de diseño debido a que las interferencias electromagnéticas afectan en gran medida los circuitos impresos. Además de los condensadores de desacople y de las bobinas para la reducción de interferencias por cambios en la corriente de alimentación, se protegió la tarjeta mediante la reducción de área de lazo de corriente, véase [CATSOULIS] para la reducción de flujos magnéticos, ubicando caminos de retorno de señal (tierras) tan cerca como fue posible. Otra técnica para la protección de la tarjeta respecto a las interferencias EMI fue la reducción del tamaño y complejidad de las pistas para evitar la generación de impedancias con componentes inductivas y capacitivas.

La tarjeta se protegió contra descargas electrostáticas conectando los protectores de ESD descritos anteriormente, en los pines de los dispositivos más sensibles a estos efectos.

2.2.7.2 Diseño de Rutas del PCB. Entre otras características se estableció el ancho de caminos en 0.4 mm para los caminos entre dispositivos (0.2mm de acuerdo a los caminos con el chip EZ-Host), 1mm para los caminos de alimentación y aislamiento entre caminos de 0.4mm. La disposición de caminos se hizo buscando linealidad con curvaturas suaves de 135° y se ubicaron vías de 0.6mm y 1mm de diámetro.

* Desarrollados en la escuela de ingeniería Eléctrica, Electrónica y Telecomunicaciones de la Universidad Industrial de Santander.

2.2.7.3 Planos de Tierra y Protección Contra Ruido. La **HC-USB** en su mayoría, excepto por la etapa de alimentación, es un sistema digital el cual maneja datos de alta velocidad, por lo cual, el circuito presenta principalmente tres planos de tierra que deben ubicarse independientes y unirse en puntos determinados con el objetivo de que los datos digitales no afecten el plano de alimentación. Los planos de tierra en el circuito son la tierra analógica, la tierra digital y el blindaje de USB.

2.2.7.4 Criterios para el Desarrollo de Hardware con Interfaz USB. Los sistemas que implementan la tecnología USB tienen recomendado el uso de planos de alimentación dedicados. Por esta razón y con el fin de ofrecer un blindaje adecuado, en el diseño del PCB se procuró dejar en una de las caras un plano de tierra tan grande como fuera posible [CYPRESS3].

Otra de las recomendaciones para el enrutamiento de los caminos en los puertos de USB fue mantener las pistas de datos paralelos entre sí, minimizando su longitud ubicando los puertos los más cerca posible al EZ-Host y adyacentes al plano de tierra.

En las recomendaciones de diseños de PCB con USB, al igual que en la mayoría de circuitos impresos, también se plantea la necesidad de proteger los pines de alimentación con capacitores de *Bypass* para reducir el rizado en la tensión. También se recomienda el uso de capacitores de carga en los pines de salida de alimentación de los puertos y en los reguladores de tensión.

Las recomendaciones principalmente advierten de la necesidad de aislar los conectores USB con <<penínsulas USB>> de tierra que están ubicadas en la cara posterior a los conectores USB y se ubican para reducir los efectos de EMI, así mismo recomiendan la protección de la señal del cristal que gobierna la operación del controlador *Host*.

2.2.7.5 Agrupación de los Dispositivos en Bloques Funcionales en la Tarjeta. El circuito está dividido en tres etapas: alimentación, BUS USB y Central o del EZ-Host,

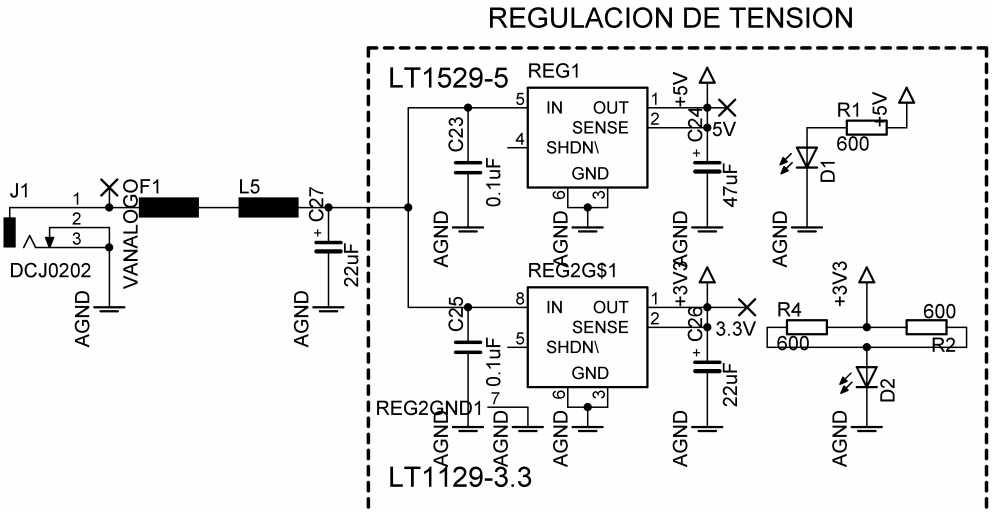
las cuales se diseñaron con el fin de agrupar en bloques funcionales los dispositivos con características similares. Dichas agrupaciones permiten reducir la distancia entre los dispositivos del mismo bloque funcional; reducir la complejidad en el trazado de los caminos de cobre y reducir los efectos indeseados que puede inducir de un bloque funcional en otro. En la etapa de alimentación están reunidas las señales analógicas de entrada del adaptador y las señales de los reguladores; en las dos etapas restantes se manejan datos digitales de alta velocidad.

2.2.7.5.1 **Etapa de Alimentación.** Esta etapa esta constituida por los reguladores de tensión y por el conector para el adaptador. Así mismo, contiene los protectores de tensión como el fusible y la bobina de alimentación, además del protector de corriente para los puertos USB. En esta etapa se encuentran dos leds que indican el estado de la alimentación tanto para USB como para el sistema embebido. Los leds seleccionados, LTST-C150KGKT del fabricante Lite-On, tienen características de alta intensidad luminosa con respecto a otros leds al mismo consumo de corriente (35mcd a 20 mA) y un amplio ángulo de visón (130°).

La etapa de alimentación esta blindada con un plano de tierra analógica aislado del plano de tierra digital y del blindaje de USB para evitar que el ruido digital interfiera en las señales de alimentación. La figura 22 ilustra los dispositivos que componen la etapa de alimentación.

2.2.7.5.2 **Etapa del BUS USB.** La etapa de USB tiene un blindaje de tierra recomendado para protección EMI [CYPRESS3]. Los puertos de USB se distribuyen tan cerca como sea posible del EZ-Host donde se busca reducir la resistencia que se adiciona al enrutar los caminos. Así mismo, las rutas de datos mantienen longitudes cercanas y una distancia constante entre sí. Todo lo anterior evita la degradación de las señales de datos.

Figura 22. Etapa de Alimentación de la tarjeta HC-USB



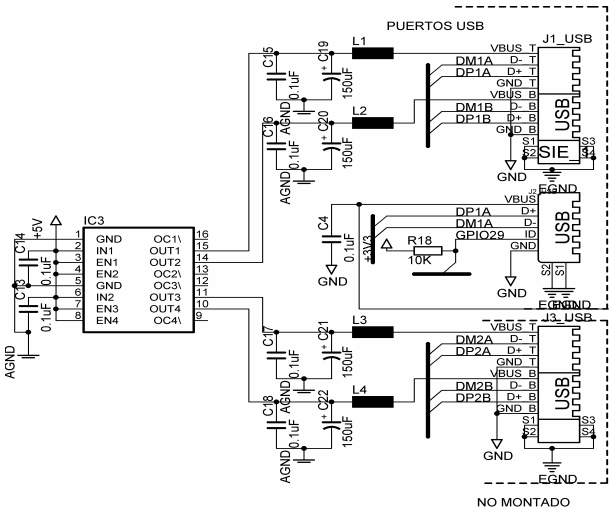
Fuente: Los Autores

En los puertos USB la península del plano de tierra se ubica en la cara posterior a los conectores y se hace un blindaje de estos con un plano que se une a la península por medio de una resistencia.

Con el fin de permitir una expansión de los puertos para nuevas aplicaciones, en la tarjeta HC-USB se dibujaron los caminos para los puertos del SIE2 que dejan la opción de adicionar otro receptor tipo A dual. El conector miniB se ubicó dentro de la península de USB procurando mantener los caminos de las señales de datos paralelas y de la mínima distancia posible. La figura 23 ilustra de los conectores USB con el protector de corriente.

2.2.7.5.3 **Etapa Central o del EZ-Host.** Esta etapa esta constituida por todos los dispositivos digitales del sistema HC-USB (El EZ-Host, el cristal y el *bus swtich,* entre otros). La disposición de los dispositivos en la tarjeta final, al igual que en la mayoría de los diseños de circuitos impresos, se diseñó para mantener todos los circuitos integrados muy cercanos entre sí, reduciendo la distancia y complejidad de los caminos enrutados.

Figura 23. Etapa de Alimentación los puertos USB



Fuente: Los Autores

El cristal se ubicó de tal forma que los caminos al EZ-Host fueran de la mínima longitud posible; se protegió con planos de tierra alrededor y en la cara posterior a este y se aisló de otras señales del circuito. Los caminos del cristal se mantuvieron en una sola cara de la tarjeta para evitar cambios de impedancia.

2.2.8 CONTROLADOR CENTRAL DSP

El controlador central del sistema H-USB es el encargado de supervisar y manejar todas las tareas del sistema. Este controlador ejecuta el software desarrollado para el almacenamiento de datos en memorias USB y trabaja en conjunto con el controlador EZ-Host para la comunicación con dispositivos USB. Como se mencionó en el apartado 2.2.1, el hardware de referencia utilizado como controlador central del sistema H-ARD es la tarjeta desarrollada por los ingenieros Juan Carlos Vargas y Cristihan García en su trabajo de grado [GARCIA-VARGAS], la cual está basada en el Controlador Híbrido DSP56F805 de *Freescale*™. De las prestaciones que ofrece esta tarjeta, sólo fueron utilizadas las estrictamente necesarias para el desarrollo del sistema H-ARD; las demás

se dejan a disposición para el desarrollo de la aplicación principal en la medición del espectro de impedancia eléctrica de tejido de cuello uterino.

Es importante para el desarrollo del sistema y para la conexión con la tarjeta HC-USB verificar los puertos de conexión de la Tarjeta DSP805. Por tal motivo, la tabla 5 muestra algunos de los conectores que esta última ofrece.

Tabla 5. Puertos de comunicación de la Tarjeta DSP805

Conector Tarjeta DSP805	Descripción
J2	Conector *pin Header* para la interfaz SPI del DSP
J3	Conector *pin Header* que abarca: los pines GPIOB0-7 y GPIOD0-1 del DSP; señales de tensión 5V y GND.
P1	Conector puerto paralelo DB25M para la interfaz JTAG
P2	Conector serial DE9F para la interfaz RS-232

Con base en la tabla 5, se seleccionó SPI como interfaz de comunicación entre la tarjeta DSP805 y la tarjeta HC-USB. A su vez, el puerto J3 es utilizado para el control de la interfaz de hardware con el usuario (LCD y botones de selección) y para las demás señales de control necesarias para el sistema H-ARD.

Dentro de las características del DSP 56F805, que son influyentes en el desarrollo del software, se destacan:

- Controlador híbrido de 16 bits de la familia 56800, con doble arquitectura Harvard. Incorpora las características de procesamiento de un DSP con la funcionalidad de un microcontrolador con un amplio conjunto de periféricos.
- Trabaja a una frecuencia de 80MHz y 40MIPS
- En memoria interna posee: 63Kbytes de memoria Flash de programa; 8Kbytes de memoria RAM de datos.
- Una interfaz SPI
- Dos interfaces SCI

- Compilador eficiente de lenguaje C.
- Tecnología CMOS con 3.3V de alimentación y compatible con tecnología TTL.

Para más información acerca de las características de la Tarjeta DSP805 y del DSP, referirse al trabajo de los ingenieros [GARCÍA-VARGAS].

2.2.9 INTERFAZ DE HARDWARE CON EL USUARIO

Para facilitar la interacción entre el usuario y el sistema HUSB se implementó un módulo de salida de datos mediante una pantalla LCD (*Liquid Crystal Display* por sus siglas en inglés) y un módulo de entrada o selector de acciones mediante botones pulsadores. El control de dichas entidades de hardware se llevó a cabo gracias al desarrollo de módulos dedicados de software (véase capítulo 3, Software) y su soporte en hardware se efectuó mediante la tarjeta de adaptación TA1, la cual es una extensión diseñada por los autores (véase numeral 2.2.10) a la Tarjeta DSP805, y que conecta a esta con la LCD y los botones de selección.

La LCD incorporada al sistema HUSB utiliza caracteres de matriz de puntos de 5x8 (incluyendo el cursor); utiliza dos líneas de 20 caracteres por línea. Su funcionamiento e interfaz con controladores externos se basa en controlador LCD de matriz de puntos HD44780A del fabricante Hitachi, el cual se puede configurar para un bus de datos de 8 o de 4 bits[HITACHI]. Con motivo de disminuir la cantidad de pines GPIO necesarios, la LCD es trabajada con una interfaz de bus de 4 bits.

La tabla 6 describe los pines o terminales del dispositivo LCD y su respectiva conexión con la tarjeta DSP805.

Por otra parte, debido a la limitada disposición de pines GPIO en el DSP, se utilizaron dos botones pulsadores normalmente abiertos para que el usuario pueda seleccionar una de dos opciones específicas en situaciones que requieran decisión e intervención por parte de este; las opciones son mostradas mediante mensajes textuales en la LCD.

Tabla 6. Descripción de pines de la pantalla LCD y su respectiva conexión

No. Pin	Señal	Función	Conexión
1-4	DB7 a DB4	Cuatro bits más significativos del bus bidireccional de datos. Son los que se usan cuando la interfaz es de 4 bits	GPIO B3 a GPIO B0 del DSP
5-8	DB3 a DB0	Cuatro bits menos significativos del bus bidireccional de datos.	No conectados
9	E	Con el flanco de bajada, esta señal inicia la lectura o escritura	GPIO B4 del DSP
10	R/nW	Selecciona lectura o escritura	Este pin es conectado a GND para realizar operaciones de solo escritura.
11	RS	Seleccionador de Registro: 0: Registro de instrucción (escritura) 1: Registro de datos	GPIO B5 del DSP
12	V_{EE}	Tensión para ajustar el contraste	
13	V_{CC}	Tensión de alimentación (2.7V a 5.5V)	5V
14	GND	Tierra de alimentación	0V
15	NC	No conectado	No conectado
16	NC	No conectado	No conectado

La implementación de los botones no requirió la conexión de resistencias de amarre de tensión (*pull up* o *pull down*) gracias a que el DSP cuenta con resistencias internas de *pull-up* que permiten que la señal de entrada esté normalmente en alto o uno lógico (véase siguiente sección). La tabla 7 muestra la conexión de los botones pulsadores con la tarjeta DSP.

Tabla 7. Conexión entre los botones pulsadores y el DSP

Botón	Pin DSP
Selector 1	GPIO D0
Selector 2	GPIO D1

2.2.10 TARJETA DE ADAPTACIÓN DE CONEXIONES

La tarjeta de adaptación TA1 permite interconectar los diferentes módulos de hardware que conforman el sistema general HUSB (La tarjeta DSP, el módulo HCUSB, la LCD y los botones de control y selección), mediante la adecuación de las conexiones entre los diferentes puertos e interfaces de comunicación. Se compone básicamente de conectores tipo *pin header,* conectores receptores de pines (*female header*) y botones pulsadores. El diagrama esquemático de la tarjeta se presenta en la figura 24. Cabe resaltar que para no introducir más componentes de hardware, el control del antirebote de los pulsadores se realiza por software.

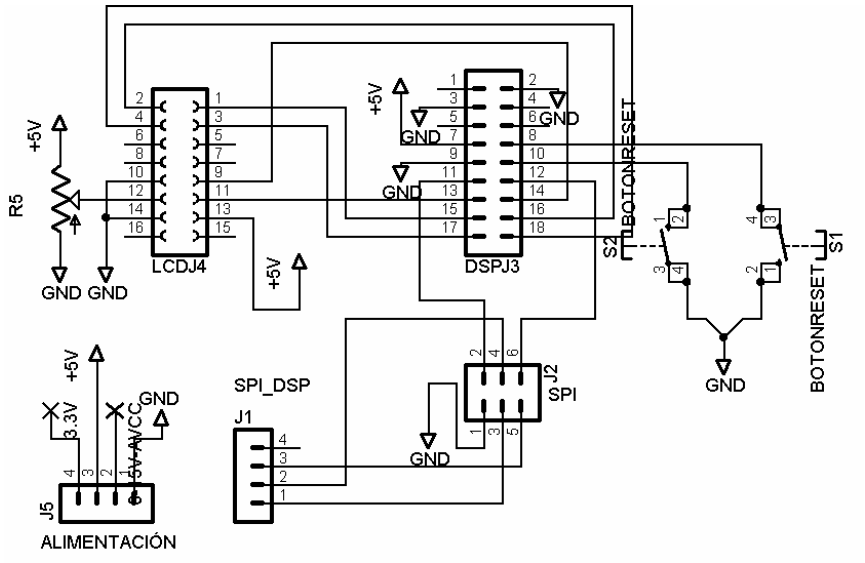
Las especificaciones de los diferentes puertos son:

- El puerto J1 de TA1 va conectado al puerto J2 de la tarjeta DSP (interfaz SPI del DSP). El pin 1 del primero corresponde al pin 1 del segundo.
- El puerto J3 va conectado al puerto J3 de la tarjeta DSP [GARCIA-VARGAS]. El pin 18 del primero debe conectar con el pin 1 del segundo
- Las conexiones realizadas al puerto J4 (que corresponden al puerto hembra que conecta con la LCD) se presentan en la tabla 4 del apartado anterior, al igual que las conexiones de los botones pulsadores con el DSP.
- Las señales del puerto J2 se presentan en la tabla 8. Este puerto está acondicionado para conectar directamente con el puerto SPI de la tarjeta HC-USB, donde el pin 1 de J2 de TA1 va conectado al pin 1 de el puerto SPI en HC-USB

Tabla 8. Señales presentes en el puerto J2 de TA1

Pin No.	1	2	3	4	5	6
Señal	GND	GPIOB7/RESET	CLK	nSS	MISO	MOSI

Figura 24. Diagrama esquemático de la tarjeta TA1

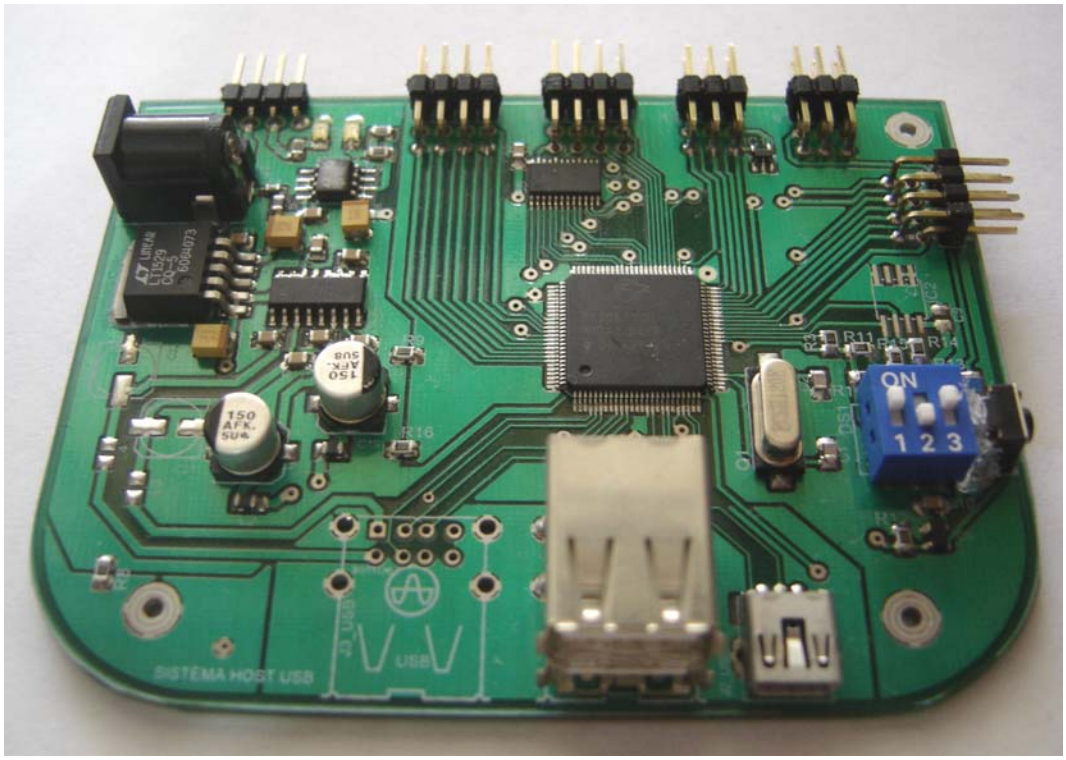


Fuente: Los Autores

2.3 IMPLEMENTACIÓN FINAL TARJETA HC-USB

La figura 25 muestra la implementación final de la tarjeta HC-USB. La tarjeta tiene dimensiones de 72mmx95mm.

Figura 25. Implementación final Tarjeta HC-USB



Fuente: Los Autores

3. SOFTWARE DEL SISTEMA EMBEBIDO

El H-ARD fue concebido como un sistema de comunicación estructurado por capas cuyas tareas fueron distribuidas a lo largo de diferentes entidades. Aunque el *Hardware* es un componente importante que cumple con tareas a nivel físico, la mayoría de las entidades que conforman el sistema fueron implementadas por *Software.*

En consecuencia de lo anterior, el presente capítulo hace énfasis en el diseño y desarrollo del software del sistema H-ARD. Se presenta la arquitectura diseñada para el *Software* junto con los diferentes algoritmos implementados para la ejecución de tareas. También se exponen las herramientas de programación y estrategias que fueron utilizadas para dar solución a criterios de diseño como velocidad de cómputo, tamaño del programa, tamaño de los datos y portabilidad, entre otras. Adicionalmente se muestra la distribución de memoria de datos del sistema, diseñada con base en los recursos de hardware disponibles.

3.1 HERRAMIENTAS

Como se mencionó en el capítulo 2 (Diseño de Hardware), el sistema comprende dos controladores trabajando en una arquitectura distribuida: un Controlador *Host (*EZ-Host de *CYPRESS*) actuando como esclavo y un Controlador Híbrido-DSP (de la familia 56800 de *Freescale*) actuando como maestro o procesador principal. A continuación se mencionan algunas de las herramientas utilizadas en el desarrollo del software del sistema H-ARD (tanto firmware disponible como herramientas de programación).

CYPRESS Semiconductor, el fabricante del EZ-Host, ofrece para éste controlador un firmware de sistema básico de entrada y salida (BIOS) [CYPRESS1], inmerso en la memoria ROM interna, que permite la comunicación con controladores externos y el control de dispositivos conectados al Bus.

Por otra parte, *Freescale* ofrece el entorno de desarrollo de software *CodeWarrior* para el DSP, el cual es una herramienta completa y sencilla, y permite tanto la implementación como el monitoreo de los algoritmos que se desarrollan para el DSP.

Con base en las herramientas disponibles para el desarrollo de software, se eligió trabajar el EZ-Host bajo su BIOS e implementar los diferentes módulos que componen la arquitectura del software del sistema H-ARD (ver apartado 3.3) en el DSP, principalmente por las siguientes razones:

- La aplicación a la que está orientado el sistema H-ARD, como se dijo en el apartado 2.1, se soporta en un DSP de la familia 56800 de *Freescale* [*]
- El DSP tiene una arquitectura Harvard y una frecuencia de Bus mayor, lo cual le permite trabajar a velocidades de procesamiento más altas.
- *Freescale* ofrece la herramienta de desarrollo de software CodeWarrior con un entorno de trabajo amigable y sencillo de usar. Dicha herramienta permite que el proceso comprendido por la compilación, el enlace de archivos (*linking),* la creación del archivo objeto y la descarga de este al DSP sea automática y transparente para el usuario.
- CodeWarrior adicionalmente ofrece herramientas de *debug* y monitoreo completas y sencillas de usar.
- En la Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones se cuenta con la experiencia de trabajos anteriores donde se han utilizado DSPs de la familia 56800 de *Freescale*.
- El *BIOS* del EZ-Host es un software completo que le permite a un controlador externo coordinar las transacciones que tienen lugar en el Bus. Además, se encarga del control de errores de la capa física del protocolo USB.

[*] Refiérase al apartado 2.1, Planteamiento del problema y requerimientos del sistema.

Basados en las especificaciones del BIOS del EZ-Host, se coordinó toda la comunicación y el trabajo en conjunto entre éste y el DSP.

Para la implementación de los diferentes algoritmos se eligió al lenguaje de programación **C** porque, además de ser ampliamente utilizado en sistemas embebidos, es un lenguaje de nivel medio que permite que el software desarrollado no dependa de una plataforma de hardware específica (como ocurre con el lenguaje *Assembler*), lo que facilita a su vez una mayor portabilidad del programa. Además, el lenguaje C facilita el entendimiento los algoritmos para una rápida expansión o modificación.

3.2 CRITERIOS DE DISEÑO

Para la adecuada programación de un sistema embebido, es fundamental conocer los recursos ofrecidos por el hardware y las herramientas que brinda el lenguaje de programación seleccionado con el fin de solucionar las limitaciones y criterios elementales de diseño como: velocidad de cómputo, tamaño de los datos (espacio en memoria RAM), tamaño del programa (espacio en memoria ROM), portabilidad y flexibilidad.

El software del sistema fue desarrollado buscando, entre otras características, un equilibrio entre la velocidad de cómputo y el tamaño del programa (pues como es sabido, en el desarrollo del software de un controlador programable existe un compromiso entre estas dos características [BARR]). Así mismo, toda la programación se orientó a maximizar la flexibilidad y portabilidad; en otras palabras, se buscó que el software del sistema permitiera una fácil utilización y modificación y a su vez una rápida adaptación a cualquier plataforma de hardware.

Lo anterior se llevó a cabo mediante el diseño de la arquitectura de software (véase apartado 3.3) en combinación con la explotación de las herramientas del lenguaje de

programación. Algunas de las herramientas de programación utilizadas para el desarrollo del software fueron:

- **Punteros**: se utilizaron punteros especialmente en la introducción de parámetros de funciones, para aumentar la eficiencia del programa, puesto que reducen la sobrecarga de datos a la pila y aumentan a su vez la velocidad con la que se llama una función.
- **Macros**: se implementaron macros para eliminar la necesidad de crear variables constantes y en algunos casos, de crear funciones, dado que funcionan como código insertado al momento de la compilación, permitiendo reducir el tamaño de datos y de programa. Además, ayudan a la comprensión de los algoritmos.
- **Estructuras y campos de bits**: a lo largo del desarrollo del sistema de comunicación H-ARD fue indispensable el manejo de campos de bits y/o bytes en las cabeceras de los diferentes protocolos. Así mismo, fue necesaria la creación de una base de datos para almacenar información propia de los dispositivos USB y de unidades de almacenamiento con formato FAT. Para todo lo anterior se utilizaron estructuras y campos de bits buscando facilitar la manipulación de dicha información en forma de variables y mantener los datos debidamente agrupados y organizados. A su vez, se trabajaron punteros a estructuras para reducir significativamente la carga computacional y de datos. Una característica importante de las estructuras es que estas permiten que los dispositivos USB sean manipulados como objetos (conjunto de características), facilitando el acercamiento a la programación o modelamiento de software, [MARWEDEL] orientado a objetos.

Entre tanto, ciertas técnicas fueron necesarias para mejorar la eficiencia del programa y el uso de memoria, tales como:

- **Decrementar el tamaño del *Heap***: el *Heap* es un espacio en RAM que permite asignación dinámica de memoria mediante funciones especiales de librerías estándar de C [BARR]. En principio se usó esta herramienta para ubicar *buffers* temporales de datos. Sin embargo, su utilización además de aumentar

significativamente el tamaño del programa, arrojó una serie de inconsistencias que obligaron a abandonar esta alternativa. Por ello se optó por reducir este espacio en memoria para aumentar el espacio para las variables y así poder ubicar los *buffers* como variables de arreglos de tipo temporal, en las funciones requeridas.

- **Equilibrar código insertado con creación de funciones**: en ciertas situaciones fue pertinente reemplazar las funciones por código insertado para reducir el tiempo de ejecución de una rutina, teniendo en cuenta de que ésta no afectara significativamente el tamaño del programa. No obstante, cuando se utilizaron funciones para rutinas que fueran llamadas constantemente, se procuró reducir al máximo la cantidad parámetros de la función para reducir la sobrecarga y aumentar la eficiencia.

Como último paso, pero no menos importante, se tuvo en cuenta aquellos factores negativos que afectan directamente el hardware y que pueden generar una inadecuada operación del sistema. Entre los factores más relevantes se encuentran:

- Interferencia y ruido en las interfaces de comunicación que puedan degradar la señal transmitida.
- Capacitancias presentes en las pistas del PCB y en general en todo conductor por donde se transmita una señal, generando retardos significativos y violaciones en los tiempos especificados por las interfaces de comunicación de los diversos dispositivos.

Para contrarrestar en el software estos factores negativos[*], se implementaron sistemas de corrección de errores basados en el reenvío de la información (para el primer caso) y ajustar adecuadamente las tolerancias de tiempos para alejarlas de las tolerancias marginales (para el segundo caso).

[*] La incidencia de estos y otros factores negativos que afectan el funcionamiento del sistema fueron tenidos en cuenta en el diseño del hardware (véase capítulo 2)

3.3 ARQUITECTURA DEL SOFTWARE

La arquitectura del software ha sido diseñada con base en los requerimientos del sistema y enfocada a una estructura por capas, como lo ilustra la figura 26, lo que proporciona beneficios como:

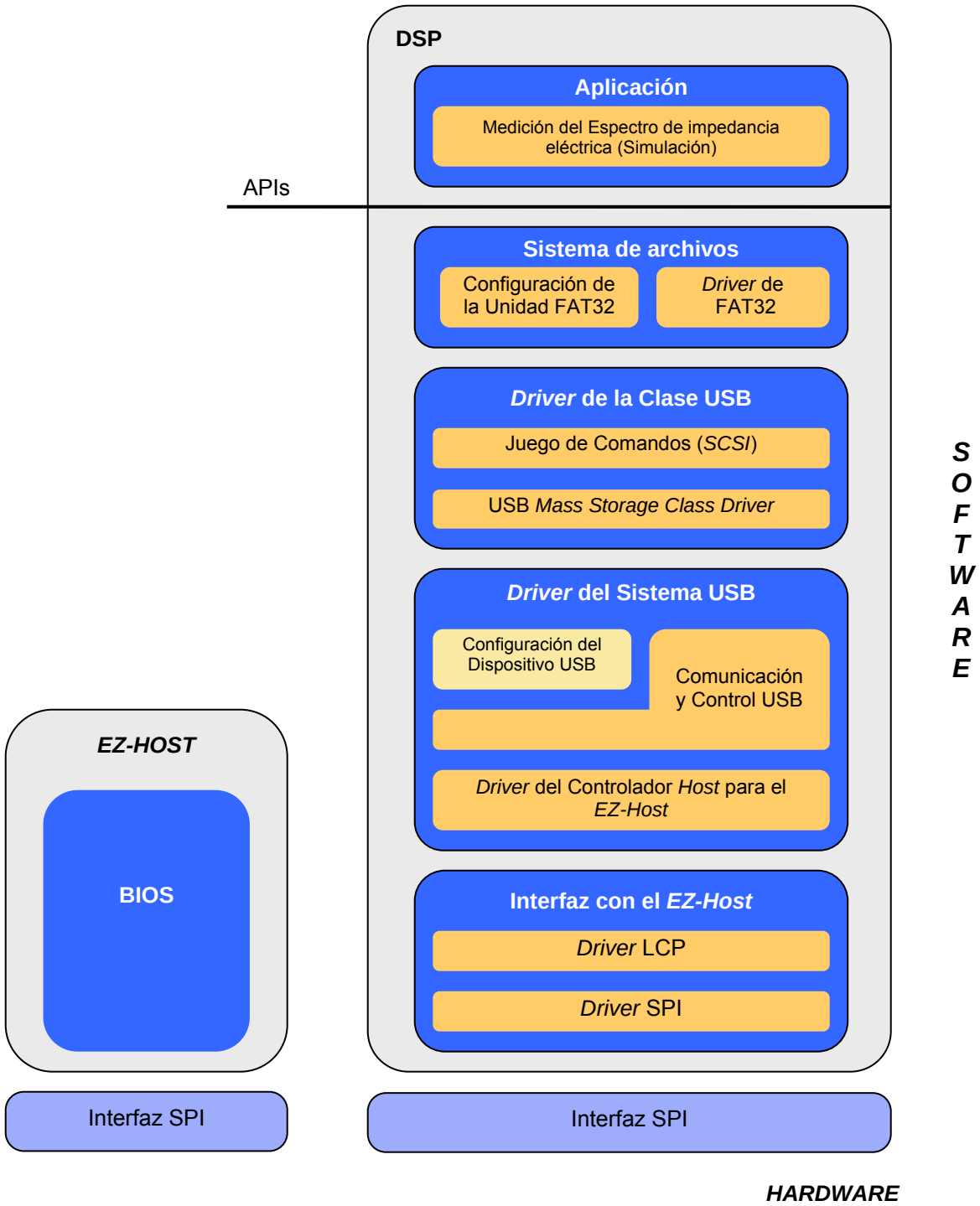
- Flexibilidad en soporte de hardware, aumentando la portabilidad del programa.
- Flexibilidad en modificaciones.
- Reducción en la complejidad de implementación, al dividir las tareas en varias entidades.
- Control de errores por entidades, aumentando su rigurosidad.
- Identificación de la jerarquía de cada capa y su dependencia con otras entidades.

Para la ejecución de las diversas tareas del sistema que son realizadas por el Software, se han diseñado 4 Módulos o capas fundamentales: el módulo de sistema de archivos FAT32, el módulo de *Driver* de la clase USB de almacenamiento masivo, el módulo de *Driver* del sistema USB y el módulo de interfaz con el controlador *Host* USB (EZ-HOST).

Es importante resaltar que el sistema base, compuesto por estas cuatro entidades, es independiente de la aplicación y por tanto está orientado a ser incorporado a cualquier otro sistema de aplicación específica que requiera el almacenamiento de altos volúmenes de datos. Para ello se ha diseñado una interfaz de *Software* de aplicación compuesta por funciones en C que hacen las veces de APIs (*Aplication Program Interface* por sus siglas en inglés) y que facilitan la interacción entre el sistema diseñado una aplicación (véase Anexo F).

Adicionalmente se desarrolló un módulo de aplicación con miras a adaptar el sistema H-ARD al dispositivo de medición que se desarrolla en la investigación que enmarca este proyecto. A su vez, se espera con este módulo verificar el funcionamiento de todo el sistema.

Figura 26. Arquitectura del software del sistema



Fuente: Los Autores

3.3.1 MÓDULO DE SISTEMA DE ARCHIVOS FAT32

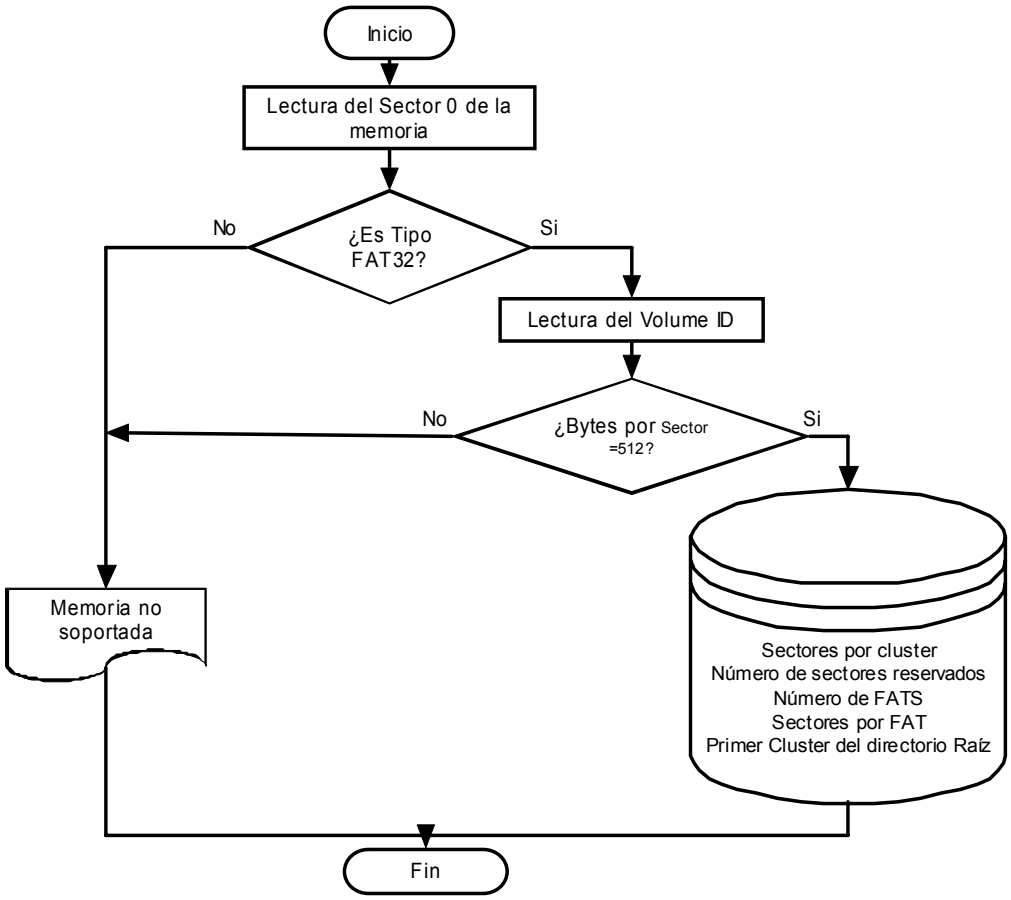
Como se mencionó en el apartado 1.6, el sistema de archivos FAT32 es ampliamente utilizado por las memorias Flash portátiles para el intercambio de archivos y carpetas con el PC. Por tal razón, este módulo es el encargado del soporte y manejo del sistema de archivos FAT32 y de la configuración, a nivel de FAT, de las memorias con este tipo de formato. Las tareas más relevantes a resolver son:

- Inicialización de la unidad de almacenamiento mediante la lectura de sus características y de su estructura lógica (tamaño en bytes de los Sectores, el número de particiones y el número de FATs que posee, entre otras).
- La escritura de archivos, que comprende la búsqueda en el directorio raíz de nombres de archivos, la búsqueda de clusters o espacios vacíos en la FAT, la escritura de archivos en clusters vacíos y el respectivo registro del nombre en el directorio raíz.

La fase de inicialización de una unidad de almacenamiento se ilustra en la figura 27. Con este algoritmo se busca extraer las características principales y la estructura lógica de la unidad. Del sector 0 de la memoria se obtiene la información del tipo de formato el LBA (*Logical Block Address* ó dirección del bloque lógico) y del inicio de la partición (por lo general las memorias portátiles sólo tienen una partición). Posteriormente, del primer sector de la partición (*Volume ID*) se obtienen los datos de bytes por sector, sectores por cluster, número de sectores reservados, número de FATs, sectores por FAT y el número del primer cluster del directorio Raíz. Con esta información es posible completar la estructura lógica de la memoria y así proceder a leer o escribir archivos en ella.

El *Driver* de sistema de archivos basado en FAT 32 implementado, se muestra en el diagrama de la figura 28. Para crear un archivo, primero se realiza una búsqueda en la tabla de FAT de Clusters libres, de acuerdo a la cantidad de datos que se deseen almacenar. Esto implica actualizar la tabla con la ubicación de los clusters del nuevo archivo (la cual es una tarea compleja y puede requerir un mayor tiempo de comunicación con la memoria debido a que, como se mencionó en apartados

Figura 27. Diagrama de flujo de la configuración de una unidad de almacenamiento con formato FAT32

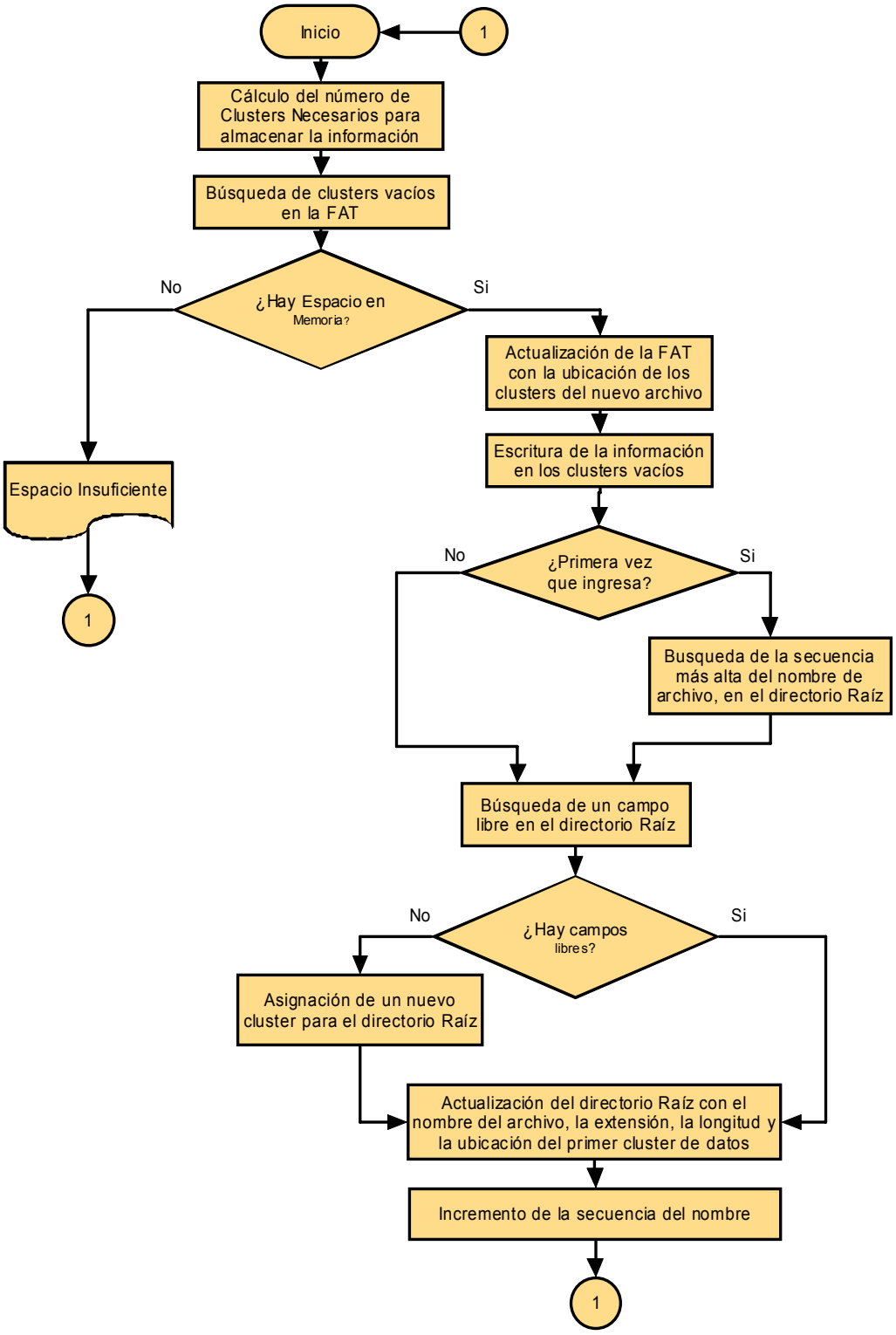


Fuente: Los Autores

anteriores, no siempre los clusters están ubicados consecutivamente). Los datos son almacenados en los clusters libres encontrados.

Posteriormente, para que el archivo aparezca creado en el directorio raíz y pueda ser visualizado por el sistema operativo, se ejecuta la actualización del campo de información del archivo (nombre, extensión, etc.) en dicho directorio. Para esta última tarea es necesaria la búsqueda, en el directorio Raíz, tanto de un campo libre como de un nombre que coincida con el archivo que se desea crear (puesto que no puede haber nombres repetidos).

Figura 28. Algoritmo del sistema de archivos implementado, basado en FAT32



Fuente: Los Autores

Con el objeto de evitar complicaciones en el hardware por el ingreso de los caracteres del nombre del archivo, se decidió dejar un nombre o cadena de caracteres constante con una secuencia o número variable generado por el sistema (por ejemplo Datos1, Datos2, etc) y con el formato de nombre 8.3*, de tal forma que el nuevo archivo almacenado continúe progresivamente con la sucesión. Para ello es necesario determinar el mayor valor de la secuencia de archivos almacenados en la memoria. El siguiente ejemplo ilustra lo mencionado:

Archivos localizados en la memoria	Datos0, Datos1, Datos2 y Datos3
Mayor valor de la secuencia	3
Nombre del archivo a crear	Datos4

De esta forma el sistema de archivos asegura que no se almacenen en la memoria archivos con nombre repetido. El nombre del archivo que genera el sistema es &P___, con un espacio de seis dígitos decimales para la numeración, lo que permite almacenar 1 millón de archivos con nombre diferente para una misma extensión. El caracter "&" es utilizado para facilitar la búsqueda de un nombre en la Unidad de almacenamiento, dada su baja probabilidad de uso. El caracter P, inicial de Paciente, también posee baja frecuencia de uso en el Castellano (2.77% de acuerdo con [WIKIPEDIA2]). Cabe resaltar que todos los archivos creados son almacenados en el directorio raíz.

Para ahorrar tiempo de procesamiento, la búsqueda de clusters libres en la FAT se inicia en los sectores finales de estos, debido a que la probabilidad de encontrar espacios libres es mayor.

* El formato de nombre 8.3 de FAT indica que el nombre del archivo tiene un máximo de 8 caracteres ASCII y la extensión comprende 3 caracteres ASCII

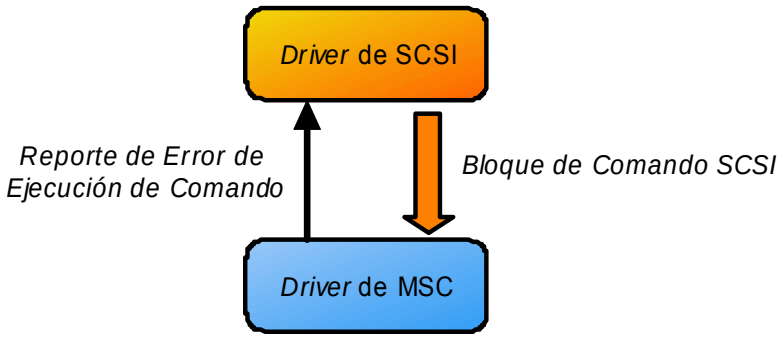
3.3.2 MÓDULO DE DRIVER DE LA CLASE USB DE ALMACENAMIENTO DE DATOS

En el apartado 1.5 se mencionó que los dispositivos pertenecientes a la clase USB de almacenamiento masivo de datos (MSC) hacen uso de un protocolo de transporte (compatible con las especificaciones de clase) para el intercambio de comandos de acción como lectura, escritura y verificación de errores. Para el caso de las memorias portátiles, el juego de comandos más ampliamente utilizado es el SCSI y el protocolo de transporte más utilizado para el envío de estos es el *Bulk Only Transport* (BOT).

Por tal motivo, el presente módulo está constituido por dos bloques funcionales, que se encargan tanto de la implementación del protocolo de transporte BOT como del juego de comandos SCSI, respectivamente. Estos bloques son: el *Driver* de SCSI y el *Driver* de la clase USB de almacenamiento masivo de datos (*Driver* de MSC). La interacción entre estas dos entidades se muestra en la figura 29.

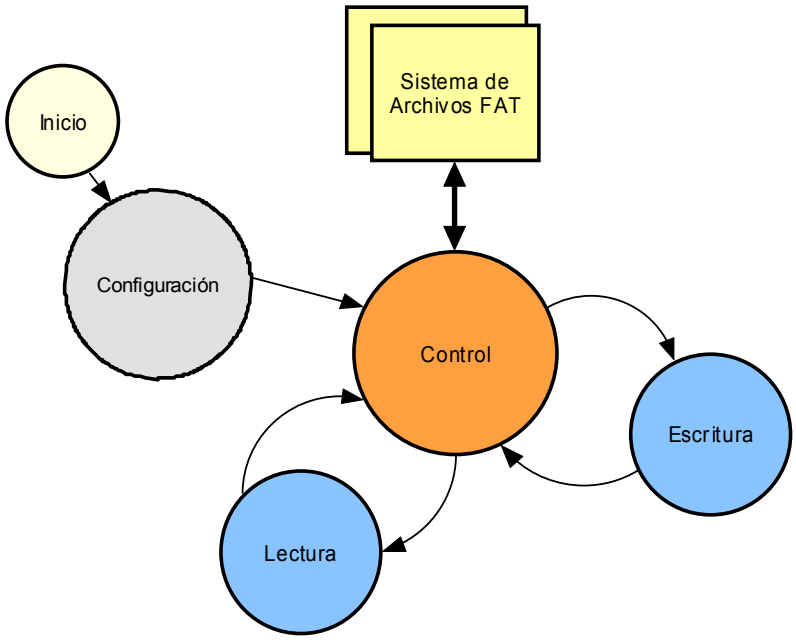
El *Driver* de juego de comandos SCSI implementado se encarga de traducir las peticiones del *Driver* del sistema de archivos FAT en comandos de SCSI como lectura y escritura, para que sean enviados a la unidad o dispositivo de almacenamiento y posteriormente ejecutados por esta; dicho *Driver* hace uso de los servicios que le proporciona el *Driver* de MSC para transferir tanto comandos como datos.

Figura 29. Interacción entre el *Driver* de SCSI y el *Driver* de MSC



Fuente: Los Autores

Figura 30. Diagrama de estados del *Driver* de SCSI



Fuente: Los Autores

Adicionalmente, lleva a cabo un proceso de configuración de SCSI que habilita a la memoria USB a su estado normal de trabajo. Este proceso de configuración sigue lo estipulado en [USB-IF3] y consiste en el envío del comando *Inquiry*, el envío reiterado del *Test Unit Ready* y el envío de *Read Capacity,* en secuencia. Los comandos implementados de acuerdo a las especificaciones de SCSI, [NCITS1] y [NCITS2], se presentan en la tabla 9. El diagrama de estado de la figura 30 muestra el funcionamiento del *Driver* de SCSI.

Por otra parte el *Driver* de Clase USB MSC se encarga de llevar a cabo el protocolo de transporte de comandos *Bulk Only Transport,* [USB-IF2] (BOT). Tiene como entrada el bloque de comando SCSI, obtenido del *Driver* de SCSI. Este bloque de comando es encapsulado en el paquete de datos CBW y enviado a la unidad de almacenamiento a través del bus USB. Posteriormente son enviados o recibidos los datos mediante una transferencia USB tipo *Bulk*. Por último se recibe el acuse de la transmisión, representado por el paquete de datos CSW, el cual informa al *Host* si el comando SCSI

Tabla 9. Comandos SCSI implementados

COMANDO SCSI	PROCESO
Inquiry	Configuración
Test Unit Ready	Configuración
Read Capacity	Configuración
Read (10)	Lectura
Write (10)	Escritura
Request Sense	Control de Errores

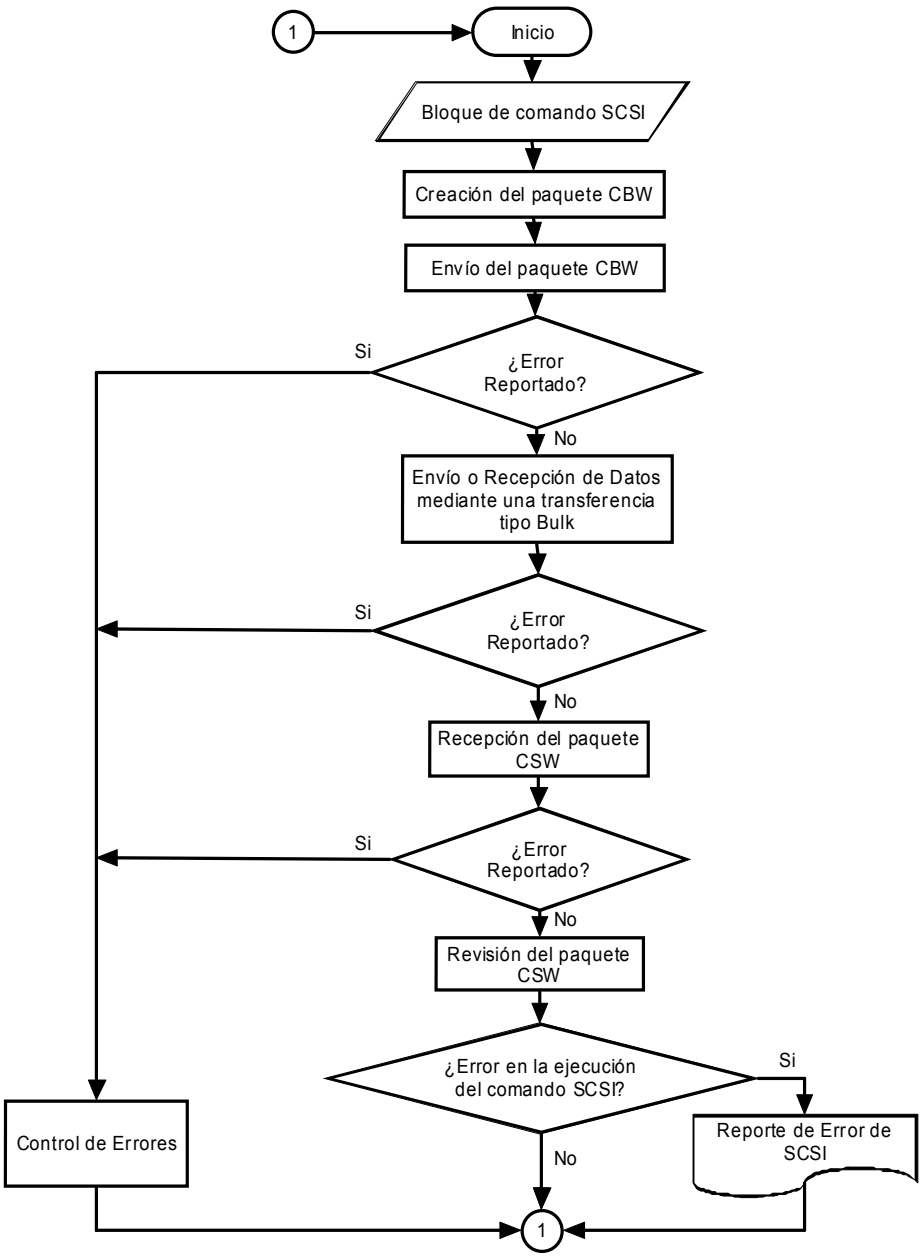
ha sido ejecutado con éxito. De ser negativa la ejecución, el *Driver* de USB MSC informa al *Driver* de SCSI el error para que tomen las medidas pertinentes.

Es relevante mencionar que cada fase de envío o recepción de datos a la unidad de almacenamiento tiene incorporado una rutina de control de errores, acorde a las especificaciones de la Clase USB MSC. El proceso que ejecuta el *Driver* de la Clase USB MSC implementado se expone en el diagrama de la figura 31.

3.3.3 MÓDULO DE *DRIVER* DEL SISTEMA USB

El módulo de *Driver* del sistema USB abarca la configuración y control del dispositivo USB, el manejo de las transferencias USB y el *Driver* del Controlador *Host*. La figura 32 muestra la estructura básica del *Driver* de USB implementado, donde se observan los diversos bloques funcionales que fueron desarrollados para llevar a cabo la comunicación por el bus. De las cuatro transferencias especificadas por USB, se implementaron las transferencias de control para el envío de peticiones USB (USB *Requests* definidas por las especificaciones USB) y las transferencias Bulk para la ejecución del protocolo de transporte BOT de MSC. Lo anterior favorece una reducción en tamaño de código y una reducción en la complejidad del sistema.

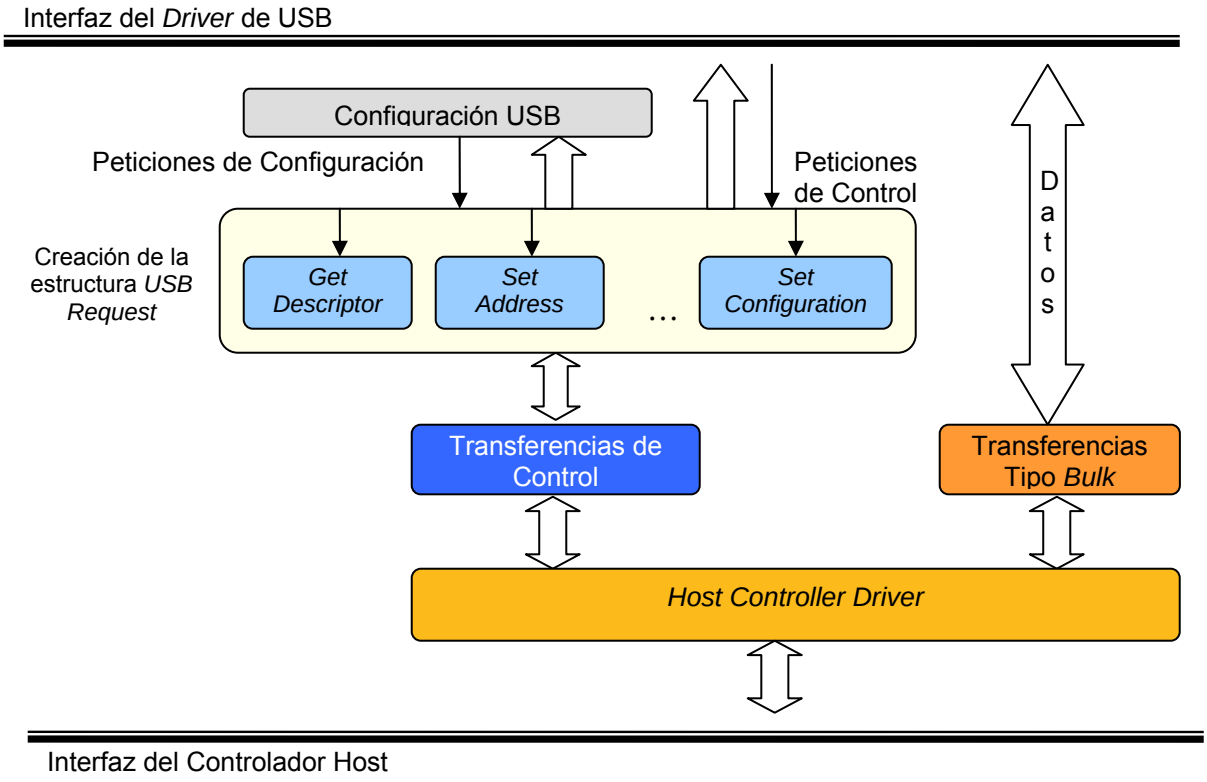
Figura 31. Algoritmo del *driver* de la clase USB *Mass Storage*



Fuente: Los autores

De los bloques funcionales que componen el *Driver* del sistema USB, cabe resaltar: el bloque de configuración USB, el bloque de transferencias de Control, el bloque de transferencias *Bulk* y el bloque de *Driver* del Controlador *Host.*

Figura 32. Estructura del *Driver* de USB



Fuente: Los autores

3.3.3.1 Bloque de configuración USB. El bloque de configuración USB lleva a cabo la identificación y la configuración del dispositivo USB en caso de detectarse su conexión al bus, mediante el proceso de Enumeración mencionado en el apartado 1.3.3 y que consiste básicamente en el reset del puerto, la asignación de una dirección, la petición de los descriptores y la asignación de una configuración. Estas acciones se llevan a cabo mediante peticiones de USB, [USB-IF5] (*USB requests*), que a su vez hacen uso exclusivamente de transferencias de Control para transmitir la información respectiva al *EndPoint* Control (o EP 0).

Cuando se obtienen los diferentes descriptores del dispositivo, específicamente el descriptor de interfaz, se configura todo dispositivo que cumpla con las siguientes especificaciones:

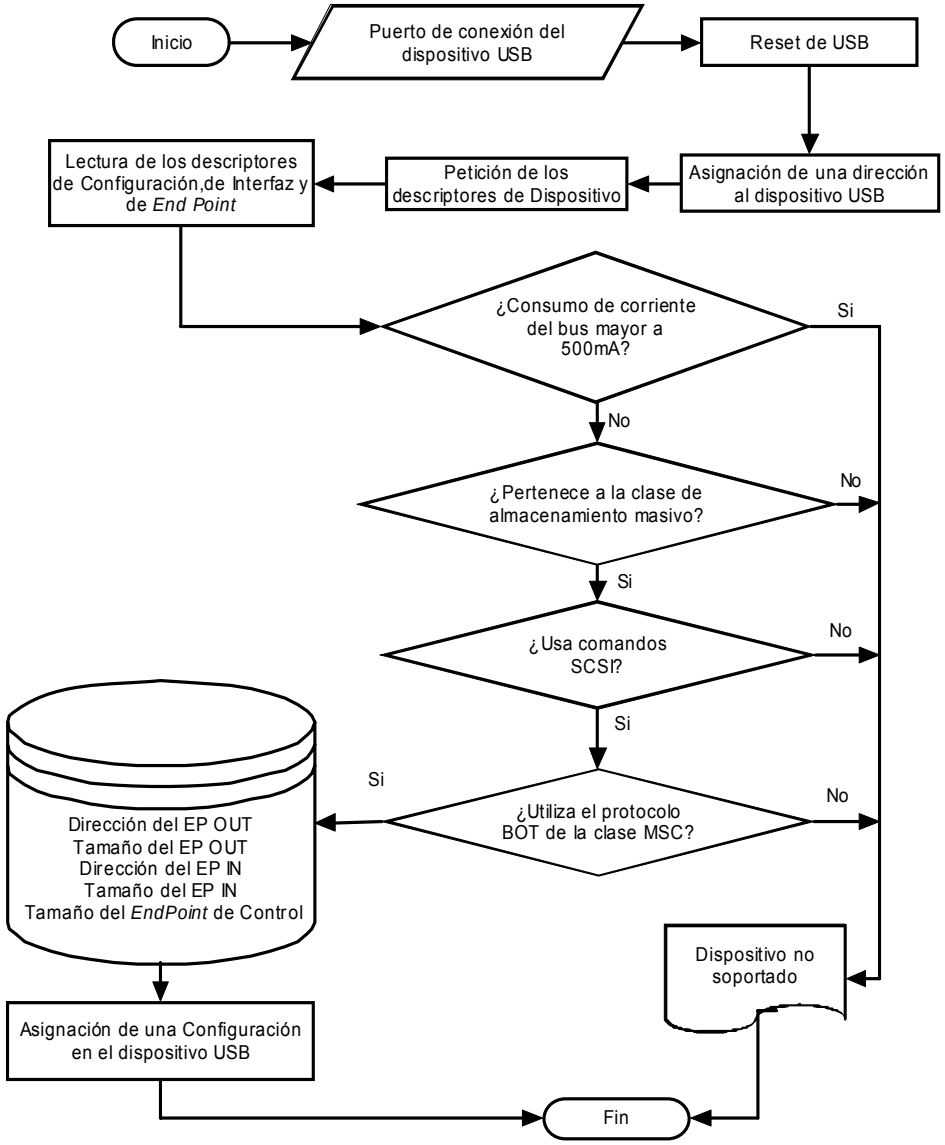
- Pertenezca a la clase USB de almacenamiento masivo
- Utilice el protocolo de clase BOT
- Utilice el juego de comandos SCSI

Entre tanto, gran parte de la información extraída de los descriptores es redundante para el sistema, por lo que sólo se almacenan los datos estrictamente necesarios, para no comprometer demasiado el espacio en memoria. El diagrama de la figura 33 muestra el proceso implementado para la configuración de un dispositivo USB.

Dado que el sistema soporta una conexión de hasta 2 dispositivos (con posibilidades de expansión de 4 de acuerdo al hardware) fue necesario crear una base de datos para almacenar la información respectiva del dispositivo, como el puerto de conexión en el Controlador *Host*, la dirección del dispositivo y los tamaños y las direcciones de los *EndPoints*. Para ello se recurrió a la utilización de estructuras, propias del lenguaje de programación C, para almacenar dicha información en los diferentes campos de las estructuras. Estas estructuras también se usaron para trabajar a los dispositivos como objetos, en el marco global de todo el software, puesto que facilitan el acceso a sus características y la manipulación de los mismos.

3.3.3.2 Bloque de Transferencias de Control. El bloque de transferencias de control se encarga de realizar las tres fases necesarias para este tipo de transferencia: *Setup*, *Data* y *Status* [USB-IF5]. En la fase de *Setup*, se crea una transacción en cuyo paquete de datos va encapsulado el bloque de información de la petición USB (USB *Request*) que se desea enviar. En la fase de *Data*, se crean tantas transacciones del tamaño del *EndPoint* de control como sean necesarias para recibir los datos solicitados en la fase de *Setup* (puesto que en esta fase, el *Host* no envía datos al dispositivo), para ello es necesario actualizar el tamaño del *EndPoint* de control del dispositivo (vale

Figura 33. Configuración de un dispositivo USB



Fuente: Los autores

la pena recordar que en una transacción, sólo se pueden enviar o recibir datos como máximo del tamaño del *EndPoint* en cuestión)[*]. En la fase de *Status* se crea una transacción con un paquete de datos nulo.

[*] Por ser un buffer de datos con espacio de memoria limitada.

Para la creación de las transacciones es fundamental tener especial cuidado en asignar bien los identificadores del paquete *Token* de cada transacción y llevar con exactitud la secuencia del paquete de datos.

Luego de creadas todas las transacciones que conforman la transferencia, estas son enviadas al dispositivo mediante el *Driver* del Controlador *Host*. La figura 34 muestra la operación de este bloque funcional.

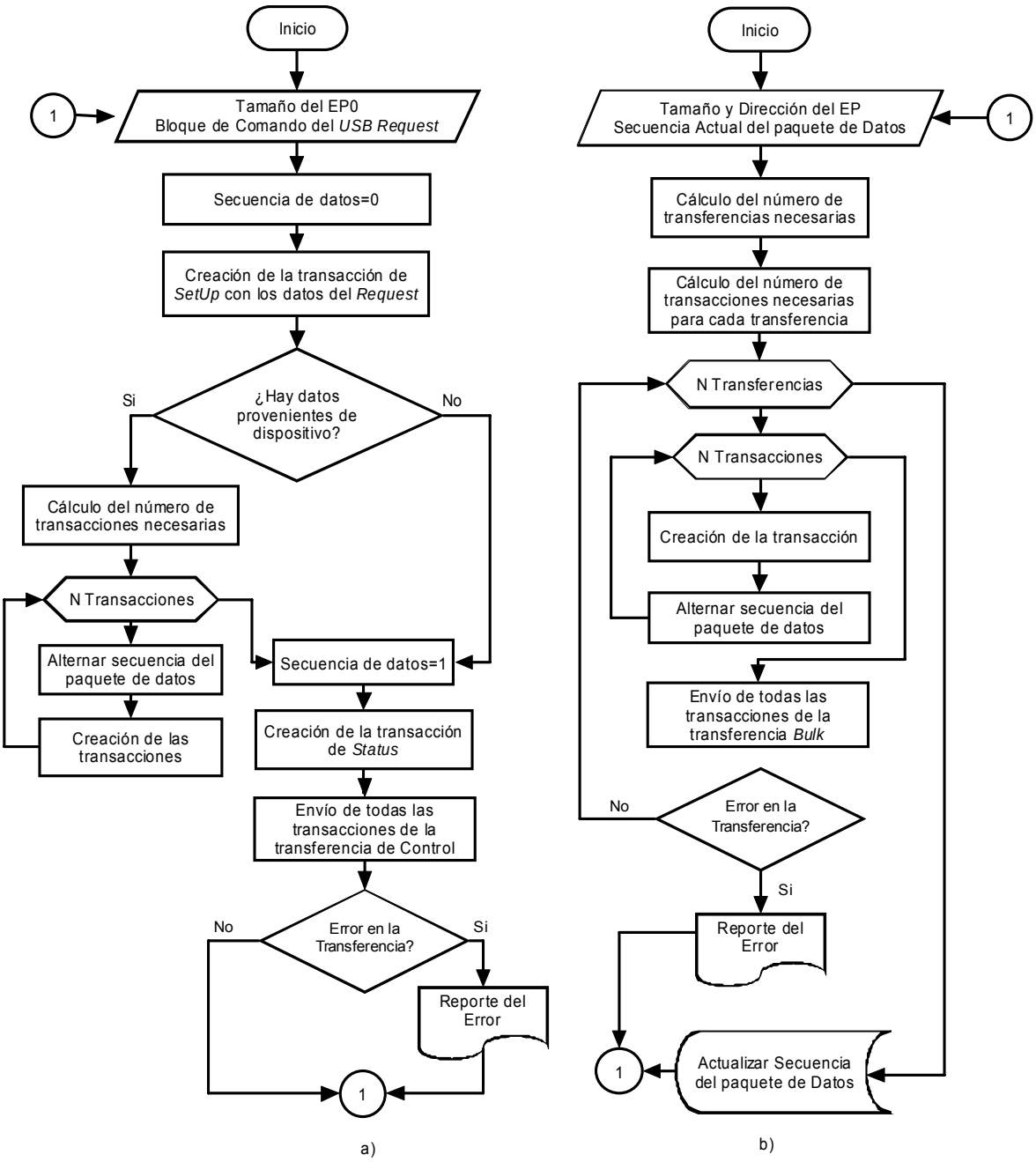
3.3.3.3 Bloque de Transferencias *Bulk.* El bloque de transferencia *Bulk* se encarga principalmente de realizar las transferencias tipo *Bulk* necesarias para enviar o recibir datos <<en volumen>>. Para ofrecer un mayor ancho de banda a la transferencia en curso y favorecer el intercambio de datos, este bloque asigna un *Frame* completo a la transferencia *Bulk* en curso.

Para asegurar un adecuado ancho de banda de transmisión, se ajustó la cantidad de datos a transmitir en un *Frame* teniendo en cuenta el compromiso entre el espacio en memoria del buffer de transmisión, la carga computacional y la tasa efectiva de datos (debido a que entre más reducido sea el buffer, menos requerimientos de memoria tendrá el sistema, pero menos datos podrán ser enviados en un *Frame*). Además, se tuvo en cuenta las limitantes en cuanto a número de transacciones que impone las especificaciones de USB2.0 para transferencias tipo *Bulk* [USB-IF5].

De acuerdo con las especificaciones USB 2.0 para transferencias Bulk en *Full Speed*[*]*(12Mbps)*, considerando un caso crítico[†] de una memoria USB con un tamaño de buffer o EP de 8bytes, en un *Frame* se pueden transmitir 568 bytes como máximo[‡].

[*] Refiérase a la tabla 5-9 de la página 54 de [USB-IF5]
[†] Se puede llamar crítico debido a que generalmente las memorias USB tienen *EndPoints* de 64bytes
[‡] La tasa efectiva de datos es menor a la velocidad de transferencia debido a la sobrecarga de datos que producen las cabeceras de los paquetes del protocolo USB

Figura 34. Transferencias en el bus USB.



a) Algoritmo para las Transferencias de Control. b) Algoritmo para las Transferencias Bulk (de volumen)

Fuente: Los Autores

Teniendo en cuenta que la unidad básica de datos del sistema de archivos implementado (FAT32) es de 512 *bytes* (cuyo espacio en memoria es asequible[*]), se puede establecer en 512bytes la longitud de los datos que tendrán lugar en un *Frame* a *Full Speed* (1ms), lo que permite tener una tasa de transferencia de datos de **500KBytes/s ó 3.9Mbps[†].** La figura 34.b muestra el proceso para llevar a cabo las transferencias tipo *Bulk.*

Al igual que con las transferencias de control, se debe asegurar que el *Driver* asigne correctamente los identificadores del paquete *Token* de cada transacción, para indicar el sentido de los datos (*IN* u *OUT*). Así mismo, debe llevarse estrictamente la secuencia del paquete de datos y un registro permanente de esta secuencia, para los *EP Bulk* puesto que el dispositivo no la reinicia a cero hasta tanto haya una actividad de configuración en los EP, [USB-IF5].

3.3.3.4 Bloque de *Driver* del Controlador *Host.* El bloque de *Driver* del controlador *Host* es el encargado de la configuración del Controlador *Host (*EZ-Host de *Cypress*) y del manejo de las herramientas concernientes al protocolo USB. Se encarga de crear los *Transaction Descriptors[‡](Tdesc),* de cargarlos en la memoria del EZ-Host, de enviar el comando necesario para que sean ejecutados y del control de errores relacionado con la ejecución de las transacciones. Todo en concordancia con las especificaciones del BIOS del EZ-Host.

Cada *Transaction Descriptor* es creado bajo petición específica de una entidad superior (por ejemplo el bloque de transferencias *Bulk*) que requiera organizar una transacción USB. Cada *Tdesc* creado es apilado en un espacio de memoria del DSP. Cuando toda una transferencia esté lista para enviar por el bus, el Driver del Controlador Host carga al EZ Host todos los *Tdesc* (que en conjunto se denominan *Tlist*) creados hasta el

[*] Véase apartado 3.4
[†] Esta tasa de transferencia se puede ver reducida por retardos en el flujo de datos del sistema y/o retardos en la respuesta del dispositivo USB
[‡] Véase Apartado 1.4.2

momento y da la orden de que sean ejecutados. Posteriormente se verifica mediante una señal de semáforo que el *TList* ha sido ejecutado y se procede a revisar el reporte del estado de cada *Tdesc* en busca de cualquier error presentado. Errores como Nak y Time Out son atendidos reenviando de nuevo la información. Errores más complejos como STALL son solucionados con la asistencia de entidades superiores.

La rutina de ejecución del *Driver* es presentada en la figura 35. La configuración del EZ-Host consiste en ajustar una o ambas SIE (Serial Interface Engine) como *Host*, borrar interrupciones pendientes, ejecutar *Tlist* pendientes y configurar el EOT [CYPRESS5].

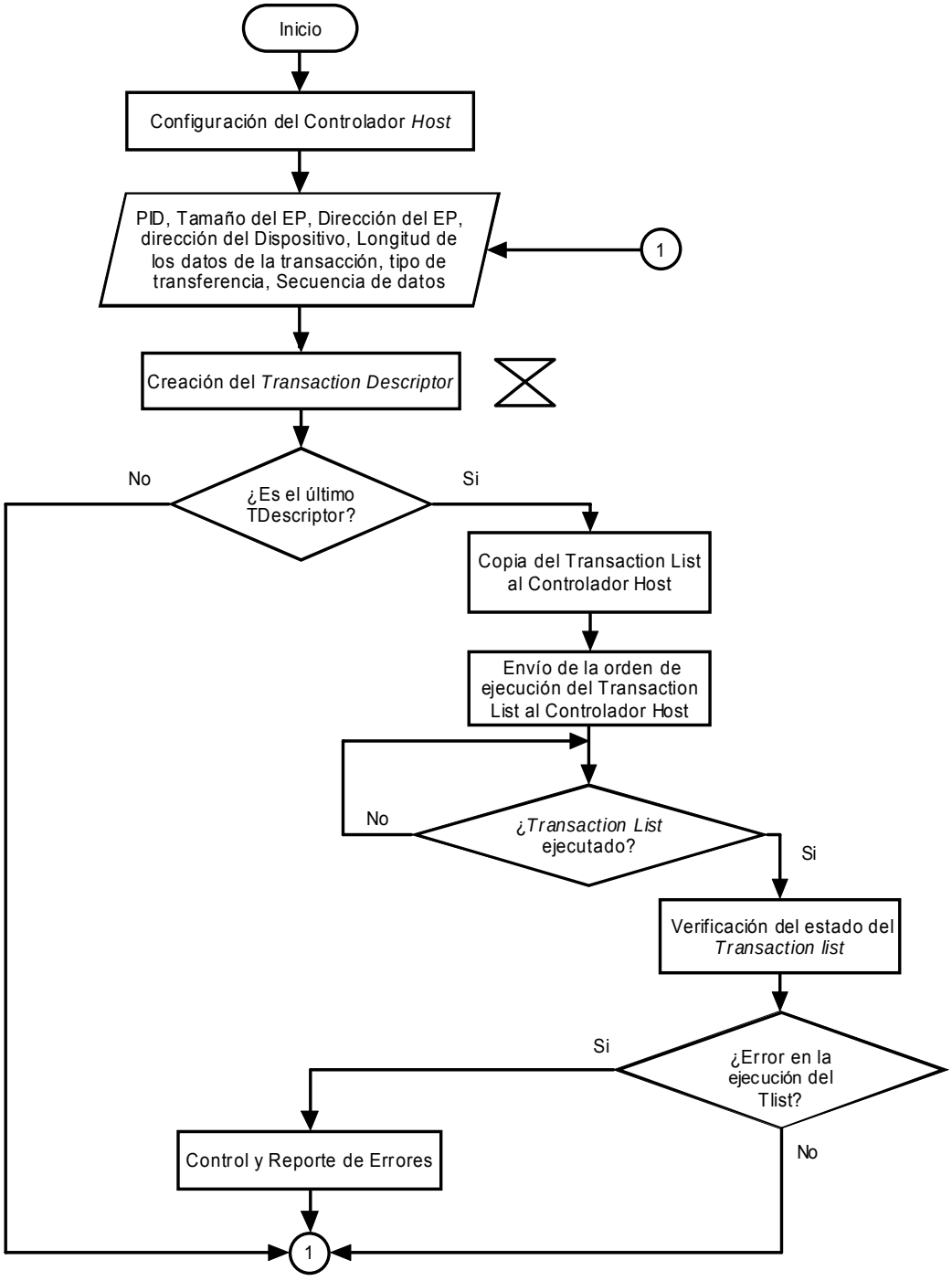
3.3.4 MÓDULO DE INTERFAZ CON EL CONTROLADOR HOST USB

El presente se encarga específicamente de coordinar la comunicación por hardware con el EZ-Host y se ha diseñado de tal forma que independice los demás módulos con el hardware sobre el que se soporta el sistema, haciendo al software desarrollado más portable y flexible.

El módulo está conformado por dos entidades: la primera, encargada de llevar a cabo el *Link Control Protocol (LCP),* [CYPRESS1]*,* requerido por el EZ-HOST para la comunicación con procesadores externos, y la segunda encargada de la comunicación por las interfaz eléctrica SPI.

3.3.4.1 Manejo del LCP. Haciendo uso del *LCP,* es posible comunicarse con el EZ-Host desde un procesador externo, mediante la interfaz SPI, HPI o HSS. Para el sistema H-ARD se utilizó SPI por las limitaciones de pines del hardware del DSP, mencionadas en el apartado 2.2.8. Por otro lado, el *LCP* utiliza el intercambio de comandos específicos que permiten la ejecución de diversas tareas, tales como escritura en memoria, lectura de memoria y ejecución de interrupciones, entre otras. Además, sólo permite, en el caso de SPI, el intercambio de datos en modo *Half Duplex.*

Figura 35. *Driver* del Controlador Host



Fuente: Los Autores

Aunque el sistema completo sólo requiere la utilización de 6 comandos LCP, esta entidad soporta un juego completo de 10 comandos que enriquecen las opciones para controlar y manejar el EZ-Host. La lista de comandos implementados se muestra en la tabla 10.

Tabla 10. Comandos LCP Implementados

Comando LCP	Comando Soportado	Comando Utilizado
COMM_RESET	SI	SI
COMM_JUMP2CODE	SI	NO
COMM_CALL_CODE	SI	NO
COMM_EXEC_INT	SI	SI
COMM_READ_CTRL_REG	SI	SI
COMM_WRITE_CTRL_REG	SI	SI
COMM_READ_MEM	SI	SI
COMM_WRITE_MEM	SI	SI
COMM_READ_XMEM	SI	NO
COMM_WRITE_XMEM	SI	NO

3.3.4.2 Manejo de la interfaz eléctrica. Para comunicar el EZ-Host con el DSP fue necesario poner en común acuerdo las características de funcionamiento de las interfaces de dichos controladores. En el EZ Host, la configuración por defecto del puerto SPI es quien determina las condiciones iniciales de trabajo. Por tanto, se configuró el módulo SPI del DSP para ajustarlo a las características del módulo SPI del EZ-Host. Cabe presentar los terminales que conforman el puerto SPI:

- **SS** o Slave Select, el cual se utiliza para habilitar el Esclavo.
- **MOSI**: es la salida de datos del Maestro y la entrada del Esclavo
- **MISO**: es la salida del Esclavo y la entrada del Maestro
- **SCLK:** es el reloj serial que se utiliza para sincronizar la comunicación

El módulo SPI del DSP fue ajustado de acuerdo a las siguientes características presentadas en la tabla 11:

Tabla 11. Configuración de los módulos de SPI en el DSP y en el EZ-Host

Característica	Configuración en el DSP	Configuración en el EZ-Host (por defecto)
Modo de operación	Maestro	Esclavo
Orden de transmisión de Bits	El más significativo primero	El más significativo primero
Frecuencia del reloj (SCLK)	1.25MHz	Soporta hasta 2 MHz
Fase del reloj (SCLK)	En atraso	En atraso
Polaridad del reloj (SCLK)	Negativa	Negativa
Modo de comunicación	*Full Duplex* (no se puede configurar de otra forma)	*Half Duplex*
Longitud de datos a transmitir	8 bits	8 bits

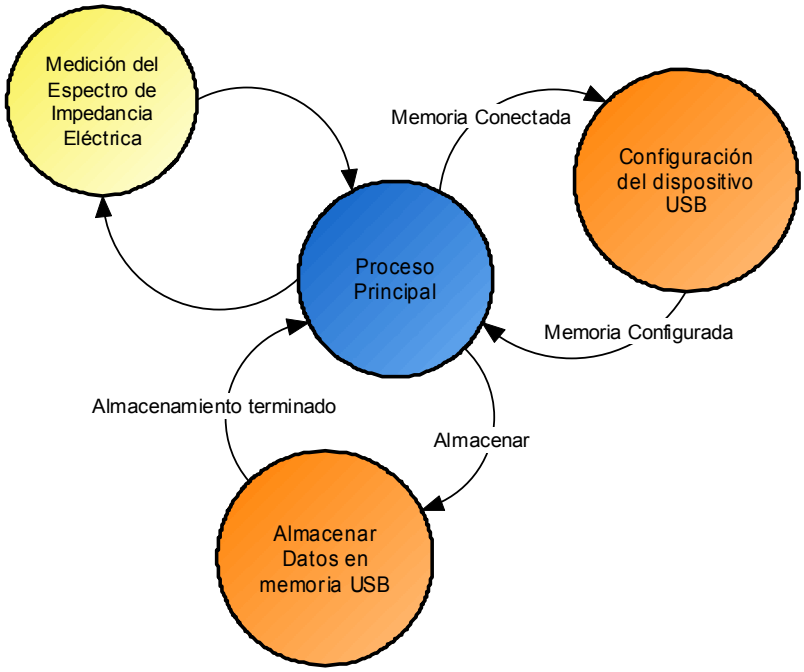
La transmisión y recepción de datos por el puerto SPI se lleva a cabo mediante la escritura del registro de transmisión y la lectura del registro de recepción, del módulo SPI en el DSP. Sin embargo, dado que el DSP trabaja en modo *Full Duplex* y el EZ-Host en modo *Half Duplex*, para la lectura de datos es necesario enviar un bloque de datos prueba de tal forma que active el reloj para que el esclavo retorne los datos respectivos.

Cabe resaltar que como el DSP ha sido designado para trabajar como Maestro, se destinó un terminal de propósito general del DSP para controlar la señal de selección de esclavo (terminal SS en el EZ-Host*).*

3.3.5 MÓDULO DE APLICACIÓN

El módulo de aplicación fue desarrollado con miras al acople del sistema H-ARD con la aplicación de la Investigación global [MIRANDA]. La estructura se muestra en la figura 36 y consta de un estado principal de donde se puede acceder a procesos como la medición del espectro de impedancia eléctrica y el almacenamiento de datos en memorias USB. La implementación de este módulo se basó en el funcionamiento de un sistema operativo, donde hay un estado principal, que es el escritorio, y de ahí se accede a los demás programas.

Figura 36. Diagrama de estados del módulo de aplicación



Fuente: Los Autores

Una vez el sistema se encuentre en el estado principal, el usuario puede ejecutar el proceso de medición del espectro de impedancia eléctrica del tejido de cerviz o el proceso de almacenamiento de datos en memoria USB. En este estado el sistema también habilita la detección de dispositivos conectados al puerto USB. En caso de haber una conexión, se da paso a la ejecución del proceso de configuración del dispositivo donde se retorna el reporte de memoria configurada si el dispositivo es una unidad de almacenamiento o el reporte de no soportado si es de otra clase.

Cabe resaltar que el proceso de medición del espectro de impedancia eléctrica es solamente simulado (dado que no corresponde a este proyecto [GARCIA-VALLE]). El resultado de esta simulación son datos que han sido almacenados previamente en la memoria del sistema y que corresponden a datos reales obtenidos de la investigación del Físico e Ingeniero David Miranda, en su trabajo de maestría, [MIRANDA]. Lo anterior permite el acercamiento a las condiciones reales de la medición. Los datos

corresponden a 8 mediciones diferentes por paciente, con 7 datos por medición, los cuales mantienen un formato de punto flotante de 4bytes; en total son 56 datos por paciente.

Para el proceso de la aplicación se desarrolló un sistema de archivos muy específico donde los datos obtenidos de la medición del espectro de impedancia son almacenados en la memoria interna del sistema como un archivo con una cabecera especial. Dicha cabecera corresponde a una cadena de caracteres ASCII específica (con una longitud de 48bytes), que sirve como identificador especial del sistema H-ARD, y a los parámetros del equipo de medición del espectro de impedancia eléctrica, tales como la ganancia y la corriente, con formato de punto flotante de 4bytes. De esta forma, los datos quedan preparados para ser almacenados en la memoria USB.

Gracias a la codificación en punto flotante de las mediciones, el sistema ofrece seguridad y privacidad en la lectura de los mismos debido a que los editores de texto solamente permiten ver la cabecera del archivo y no la información de las medidas. Los archivos creados y almacenados en la memoria USB podrán ser analizados con una herramienta especial de Matlab desarrollado por los autores (véase apartado 3.5). Igualmente para cuestiones de seguridad, una vez realizado el almacenamiento de datos en la memoria USB, el usuario no podrá realizar copias de los datos en la misma memoria ni en otras memorias diferentes. El total de datos a almacenar en la memoria USB son 280bytes que corresponden a 48bytes de la cabecera, 8bytes de los parámetros de medición y 224bytes de los datos medidos.

El proceso de medida del espectro de impedancia debe seguir el procedimiento documentado en el numeral 3.3 del trabajo de investigación, [MIRANDA]. Debe llenarse también el FORMULARIO PARA LA MEDICIÓN DE ESPECTRO DE IMPEDANCIA ELÉCTRICA EN CUELLO UTERINO, de la tabla 3.1 de la misma tesis (ver anexo D). Dado que el formulario requiere que se anote el nombre del archivo, el sistema despliega el nombre del archivo almacenado cada vez que son guardados los datos en

la memoria USB, resaltando que el sistema no permite que existan nombres repetidos en la memoria.

Para orientar al usuario en el uso de la aplicación, se ha elaborado una guía de usuario (documentada en el anexo C) donde se explican los pasos a seguir y se muestran los diferentes mensajes que el sistema muestra como información y para la toma de decisiones.

3.4 ARQUITECTURA DE LA MEMORIA DEL SISTEMA

Dadas las limitaciones en memoria RAM que presenta el DSP y al gran volumen de datos que maneja el sistema de almacenamiento H-ARD, se utilizó la memoria interna del EZ Host[*] como extensión de la memoria RAM del DSP. De esta forma, la memoria interna de datos del DSP junto con la memoria de datos del EZ-Host conforman la memoria general del sistema, como se muestra en figura 37.

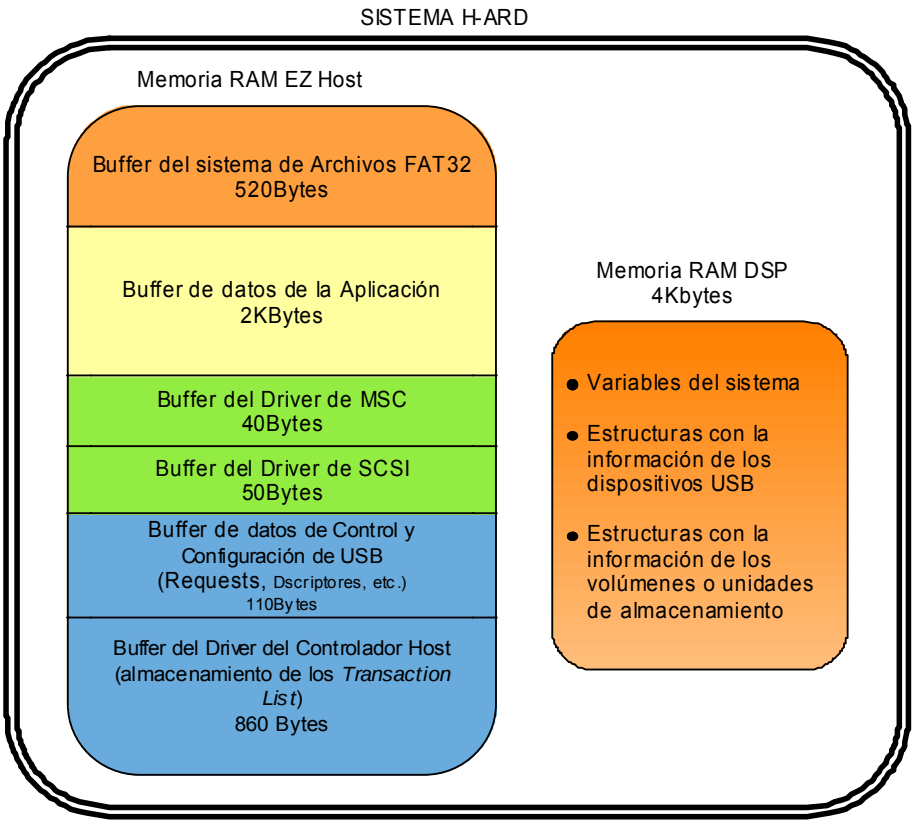
El espacio de memoria del DSP fue destinado para datos relevantes, de acceso constante y de tamaño reducido (por ejemplo las características de los dispositivos USB conectados al sistema y las características de las unidades de almacenamiento, en cuanto a su estructura lógica, incluidas en estos dispositivos[†]).

Entre tanto, en la memoria del EZ-Host fueron ubicados los *buffers* de las diferentes entidades de la arquitectura de software. El tamaño y la ubicación de estos Buffers son totalmente modificables por software pues es el DSP quien realmente administra esta memoria. La figura 37 expone la distribución de la memoria del EZ-Host.

Teniendo en cuenta que, por limitaciones de hardware, la interfaz SPI es la que permite el intercambio de datos entre el DSP y el EZ-Host y que el volumen de datos manejado

[*] El EZ Host ofrece cerca de 15kBytes de memoria RAM interna
[†] Una memoria USB ofrece dos tipos de información: la que es propia del dispositivo USB y la correspondiente a la estructura lógica relacionada con el formato de archivos de la unidad de almacenamiento.

Figura 37. Distribución de memoria del sistema H-ARD



Fuente: Los autores

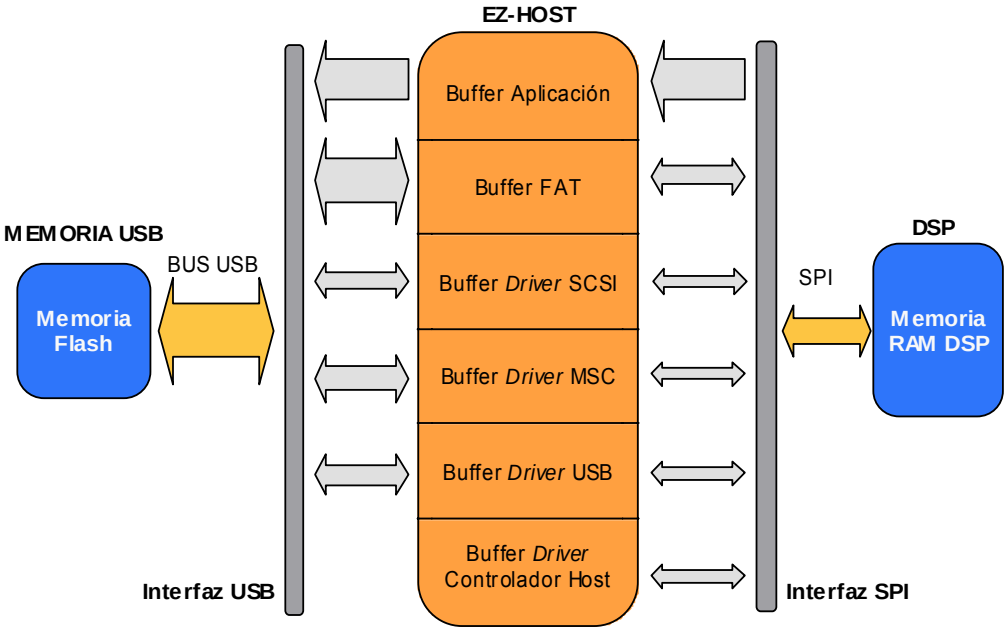
es elevado, es necesario buscar mecanismos que mejoren los tiempos de transferencia entre controladores.

Para ello, es necesario analizar los requerimientos de memoria y manejo de datos de cada entidad de software, de tal forma que se pueda obtener una noción del comportamiento del flujo de datos entre los controladores y en sí del manejo de memoria del sistema. El análisis es realizado en torno al módulo de sistema de archivos FAT desarrollado puesto que, aparte del módulo de aplicación, es la entidad con manejo de volúmenes de datos más crítico. Se asumirá un comportamiento similar para las demás capas que conforman el sistema.

El sistema de archivos FAT requiere, para aspectos de configuración y control, la lectura de sectores específicos del dispositivo de almacenamiento USB, los cuales tienen una longitud de 512 bytes. De estos sectores de datos usualmente se extraen o se actualiza información para luego enviarlos nuevamente al dispositivo. Este es un volumen de datos crítico para la memoria del DSP. En la mayoría de casos no todos los bytes del sector leído son relevantes.

Por tanto, se ha implementado un sistema de manejo de memoria donde los datos que se reciben del puerto USB permanezcan en *Buffers* dedicados del EZ-Host *(*cada entidad de hardware maneja su propio *buffer)* y únicamente los campos que sea menester extraer o modificar son los que se transportan por SPI al DSP. Así se disminuyen los tiempos de flujo de datos por SPI y a su vez se evitan excesivos volúmenes de datos en la memoria RAM del DSP. La figura 38 muestra el mecanismo implementado para el flujo de datos a través de todo el sistema.

Figura 38. Flujo lógico de datos del sistema H-ARD



Fuente: Los Autores

3.5 INTERFAZ GRÁFICA EN MATLAB PARA LECTURA DE ARCHIVOS

Con el objetivo de ofrecer un software para PC que realice la lectura de los archivos creados con en sistema H-ARD, se ha creado la herramienta gráfica **LectorA**, con el editor de interfaces gráficas *Guide,* del software Matlab, la cual permite de una forma amigable para el usuario, visualizar los datos de las diferentes mediciones del espectro de impedancia eléctrica de tejido de cuello uterino, tomadas para un paciente y almacenadas en una memoria USB. A continuación se explica de forma general el funcionamiento de la herramienta y se muestran los elementos que la componen y que hacen posible su uso.

Cuando un archivo es leído mediante **LectorA**, la herramienta verifica que la cabecera corresponda a la cadena de caracteres válida prealmacenada, para proseguir con la apertura del archivo, de lo contrario no se leen los datos. Posteriormente se extraen los parámetros de medición, corriente y ganancia, propios del equipo con que se tomaron los datos del espectro de impedancia de tejido. Por último, se extraen los datos, almacenados en punto flotante, correspondientes a las ocho mediciones[*] por paciente y se habilitan para que el usuario pueda graficar la medición que desee.

LectorA.m contiene el código de ejecución de la herramienta. La figura 39 muestra su entorno gráfico, donde se puede apreciar los diferentes elementos que la conforman. Los más representativos son:

1. **Abrir Archivo**: Seleccionando el botón *Abrir Archivo*, el usuario puede elegir la unidad de almacenamiento o la ubicación de donde desee abrir el archivo .BIN (dado que el sistema H-ARD crea archivos con esta extensión).
2. **Mediciones**: después de que la herramienta verifica que la cabecera del archivo sea la adecuada, se habilita el selector de opciones *Mediciones* el cual

[*] Refiérase al apartado 3.3.5

permite escoger para graficar, una de las ocho mediciones del espectro de impedancia.

3. **Nombre del Archivo:** Lectora permite mediante este cuadro de despliegue de texto, ver el nombre del archivo abierto para facilitarle al usuario la identificación del paciente y la correspondencia de los datos.

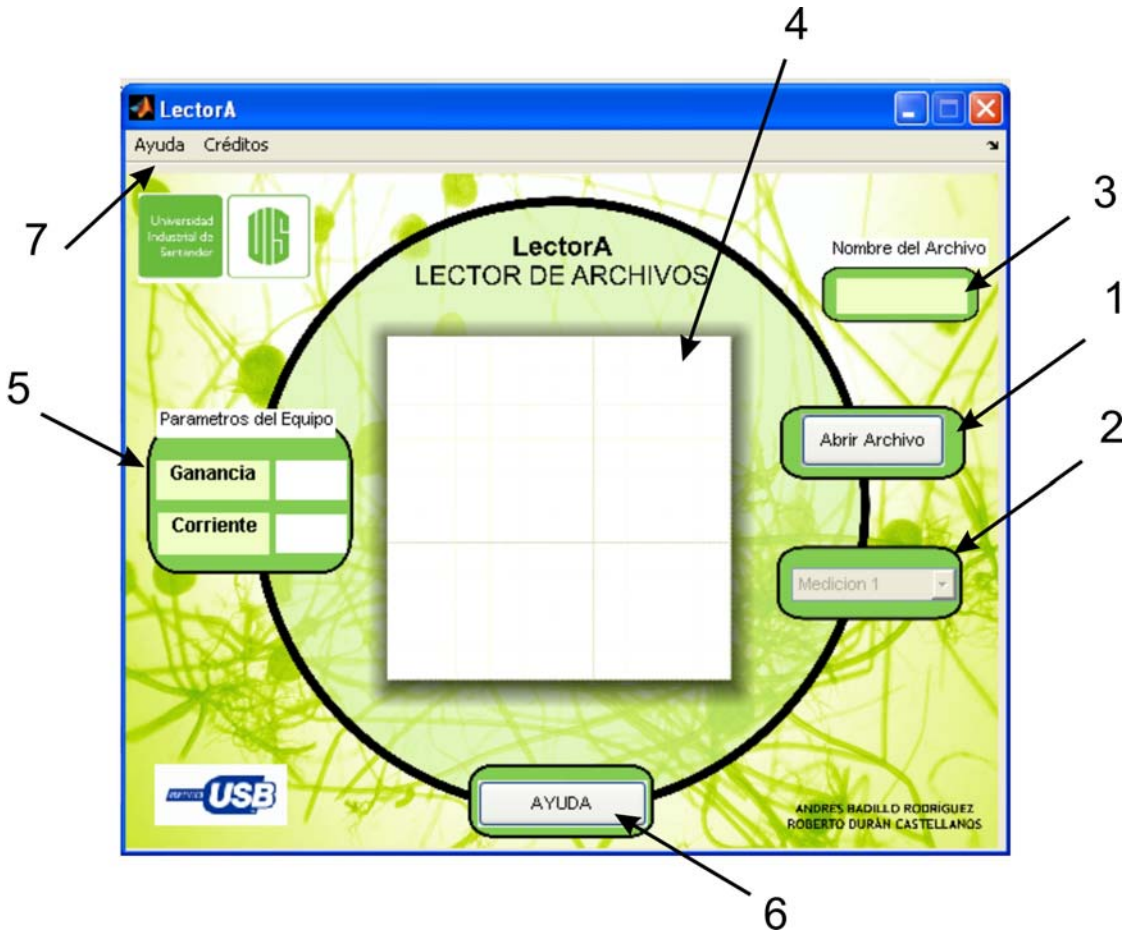
4. Cuadro de Gráficas: el cuadro de gráficas permite visualizar los datos de las mediciones. El eje X corresponde a las frecuencias utilizadas en la señal de medición [MIRANDA] y el eje Y corresponde a la parte real del espectro de impedancia del tejido de cuello uterino.

5. Parámetros del Equipo de Medición: **LectorA** despliega en los cuadros de los parámetros del equipo de medición, la ganancia y la corriente utilizadas en la toma de muestras.

6. Ayuda: mediante el botón de ayuda, **LectorA** permite desplegar una ventana adicional que contiene una breve información acerca del procedimiento para utilizar la herramienta.

7. Menú: El menú ubicado en la parte superior contiene las opciones Ayuda y Créditos, las cuales orientan al usuario en la función y la utilización de la herramienta, y le permite saber de los creadores de la herramienta, respectivamente.

Figura 39. Cuadro de controles de la herramienta LectorA



Fuente: Los Autores

4. PRUEBAS

Con el fin de establecer el grado de rendimiento del sistema H-ARD y de verificar y resaltar sus características, en este capitulo se exponen las pruebas realizadas al sistema, las cuales se realizaron desde la capa física o hardware hasta las capas superiores; teniendo en cuenta que el sistema H-ARD tiene una arquitectura estructurada en capas.

Finalmente, se evaluó el sistema H-ARD bajo los requerimientos y recomendaciones de la autoridad en materia de USB, el *USB Implementers Forum* (USB-IF), para el desarrollo de sistemas *Host* USB embebidos, [USB-IF1].

Para todas las pruebas se utilizó una memoria USB marca Flash Drive de 128MB, mientras no se especifique lo contrario.

4.1 PROCEDIMIENTO PRELIMINAR

Las diferentes pruebas del sistema requieren la ejecución preliminar de la configuración del sistema H-ARD, que incluye la configuración del puerto SPI y de los pines GPIO del DSP; la configuración de la LCD y la configuración del controlador EZ-Host desde el DSP.

4.2 PRUEBA 1. PROTOCOLO LCP

Tiene como objetivo verificar el protocolo LCP (*Link Control Protocol*)[CYPRESS1] usado en la comunicación entre el DSP y el EZ-Host. Para ello son enviados los diferentes comandos LCP usados por el sistema, desde el DSP hacia el EZ-Host. Mediante la recepción del ACK y la respectiva ejecución del comando, se verifica tanto

su correcta creación y transmisión por parte del DSP como su entendimiento del lado del EZ-Host.

La tabla 12 muestra los comandos utilizados para la prueba y el respectivo resultado.

Tabla 12. Comandos LCP probados

COMANDO LCP	RECEPCIÓN ACK	EJECUCIÓN	RESULTADO
COMM_RESET	SI	SI	Aprobado
COMM_EXEC_INT	SI	SI	Aprobado
COMM_READ_CTRL_REG	SI	SI	Aprobado
COMM_WRITE_CTRL_REG	SI	SI	Aprobado
COMM_READ_MEM	SI	SI	Aprobado
COMM_WRITE_MEM	SI	SI	Aprobado

4.3 PRUEBA 2. *DRIVER* DE USB

Esta prueba verifica el funcionamiento del *Driver* de USB en la configuración de dispositivos periféricos y a su vez en la ejecución de las transferencias de control, en cumplimiento de las funciones de un *Host* embebido para el control de los dispositivos USB conectados a los puertos tipo A.

De acuerdo con la figura 31 del apartado 3.3.3, en la estructura del *Driver* de USB se vislumbran dos flujos de datos: el primero abarca desde la entidad de Configuración de dispositivos USB, pasando por la entidad de creación de USB *request,* las transferencias de control y llegando al *Driver* del Controlador *Host;* el segundo comprende los datos provenientes de la capa superior (*Driver* de la Clase de Almacenamiento Masivo), pasando por las transferencias Bulk y finalizando con el *Driver* del Controlador *Host* nuevamente. Por tal motivo, la prueba del *Driver* de USB se centra en la comprobación del flujo de datos correspondiente a la Configuración USB; la comprobación de las transferencias Bulk se realiza junto con la prueba del *Driver* de MSC.

El desarrollo de la prueba consiste en ejecutar todo el proceso de enumeración de un dispositivo USB, llevado a cabo mediante *Requests* USB específicos [USB-IF5] y a su vez mediante transferencias de control. El dispositivo debe responder adecuadamente a los *Requests* que conforman el proceso de enumeración (sin retornar STALL). Así mismo, el sistema debe identificar y aceptar los dispositivos USB de almacenamiento masivo de datos (pertenecientes a la clase MSC) que utilicen el protocolo de transporte BOT y que hagan uso del juego de comandos SCSI. La tabla 13 muestra los resultados de la prueba.

Tabla 13. Resultado de la prueba de configuración de dispositivos USB.

DISPOSITIVO CONECTADO	RESULTADOS	OBSERVACIONES
Memorias Flash USB:		
Sony 512MB	Configurado	
Flash Drive 128MB	Configurado	
Kingston 512MB	Configurado	
Avixe 256MB	No Configurado	Pertenece a MSC* pero no cumple con SCSI
Otros Dispositivos:		
Optical Wheel Mouse USB, Dell	No configurado	No Pertenece a MSC*
Impresora USB, LexMark Z715	No configurado	No Pertenece a MSC*
Cámara Fotográfica Digital, Sony DSC-P93A	No configurado	Pertenece a MSC* pero no cumple con SCSI ni con BOT

Las pruebas siguientes requieren que el dispositivo USB se encuentre previamente configurado a nivel de USB (enumerado).

4.4 PRUEBA 3. *DRIVER* DE LA CLASE USB DE ALMACENAMIENTO MASIVO (MSC)

Esta prueba verifica el funcionamiento del protocolo de transporte BOT y de las transferencias *Bulk*. Adicionalmente son probados los comandos SCSI de configuración

* *Mass Storage Class*

únicamente (de acuerdo con la tabla 9 del apartado 3.3.2) debido a que los demás comandos necesitan una dirección de bloque lógico (LBA) o número de sector específico (cuyo manejo es propio del *Driver* de FAT). Por ello, comandos como lectura y escritura son probados junto con el sistema de archivos FAT32.

El procedimiento consiste en enviar un comando SCSI mediante el protocolo BOT y a su vez mediante las transferencias Bulk. La prueba se considera aprobada si el dispositivo responde correctamente ante el comando SCSI enviado.

Para esta prueba se utilizaron las memorias USB configuradas en el apartado anterior (ver tabla 13). En los resultados obtenidos, se observó la correcta respuesta de los dispositivos de almacenamiento ante los comandos SCSI utilizados para configuración (*Inquiry, Test Unit Ready* y *Read Capacity*).

4.5 PRUEBA 4. SISTEMA DE ARCHIVOS FAT 32

Las pruebas para el sistema de archivos FAT32 consisten en la verificación de procesos como la configuración de FAT (extracción de las características y estructura lógica de la unidad), la búsqueda y actualización de sectores libres en la tabla de FAT, la búsqueda y actualización de entradas libres y de nombres de archivos en el directorio raíz, y la escritura de datos en clusters. Igualmente son probados los comandos SCSI de lectura y escritura.

La prueba concluyó con el correcto funcionamiento de todos los procesos necesarios para la escritura de un archivo en las unidades de almacenamiento, de la tabla 13, configuradas.

4.6 PRUEBAS GENERALES DEL SISTEMA

Las pruebas generales del sistema consisten en el análisis, comprobación y obtención de diversas características y parámetros generales de funcionamiento.

4.6.1 NÚMERO DE ARCHIVOS

Esta prueba permite observar de manera general la cantidad de archivos (independiente de su tamaño) que el sistema es capaz de almacenar en una misma memoria USB, de acuerdo a lo especificado con el formato del nombre (ver apartado 3.3.1). A su vez, la prueba permite evaluar la capacidad del sistema H-ARD para mantener una interacción continua con una memoria USB.

El procedimiento que se siguió para la prueba fue el almacenamiento reiterado de un archivo de 2Kbytes en una memoria con suficientes espacio. La memoria Flash USB utilizada en esta prueba fue la Kingston, Data Traveler® de 512MB. Dentro del desarrollo de la prueba el sistema mantuvo intercomunicaciones con una memoria permenetemente escribiendo entre 140 y 180 archivos de forma continua y siendo detenido manualmente por el ususario (por cuestiones de tiempo de ejecución). Para verificar el número de archivos que el sistema es capaz de crear (de acuerdo al formato de nombre estipulado) se procedio a crear un archivo con el nombre &P999990.TXT y se observo la creación satisfactoria de los archivos continuos hasta el archivo &P999999.TXT

Aunque no se logró obtener el límite de capacidad de generación de nombres del sistema H-ARD (1millón de archivos) por cuestiones de tiempo de ejecución, la realización de esta prueba permitió corroborar que el sistema es capaz de crear hasta el nombre &P999999, resaltando su amplia capacidad de generación de archivos. Aún así, cabe anotar que las limitantes de este parámetro son: la capacidad de la unidad de almacenamiento y el formato del nombre utilizado por el sistema H-ARD, y que en caso

de habilitarse un teclado para el ingreso del nombre, la única limitante pasaría a ser la capacidad de almacenamiento de la memoria USB conectada.

4.6.2 TAMAÑO DE ARCHIVOS

El objetivo de esta prueba es analizar la capacidad del sistema para almacenar archivos de gran tamaño, por lo cual se procede a guardar un archivo tan grande como la memoria del H-ARD lo permita. De acuerdo con la capacidad de memoria del sistema, se asignó un total de 10kbytes de datos de prueba.

Como resultado se obtuvo que el sistema logró satisfactoriamente almacenar los 10kbytes de datos en la memoria, escritos como un único archivo con extensión .BIN, lo cual comparado con el tamaño de datos a almacenar por la aplicación (280bytes de acuerdo a lo mencionado en el apartado 3.3.5), es una característica que garantiza que los requerimientos de tamaño de archivos puedan ser cumplidos.

4.6.3 ESCRITURA EN DOS MEMORIAS CONECTADAS SIMULTÁNEAMENTE

El sistema está diseñado para el almacenamiento de datos en hasta dos memorias USB conectadas simultáneamente (con posibilidad de expansión a cuatro). Por tanto para corroborar esta característica se escribieron 280bytes de datos en dos unidades de almacenamiento diferentes (la memoria Flash Drive 128MB y la memoria Kingston Data Traveler® 512MB), de manera secuencial, las cuales se encontraban conectadas a la vez, obteniéndose una respuesta de escritura exitosa en las dos memorias.

4.6.4 DETECCIÓN DE CONEXIÓN Y DESCONEXIÓN

Cada vez que el sistema identifica la conexión y/o desconexión de un dispositivo USB, despliega un mensaje de la acción respectiva en la LCD. Adicionalmente, cuando una unidad de almacenamiento de la clase USB *Mass Storage* es identificada y configurada, el sistema despliega un mensaje de configuración exitosa.

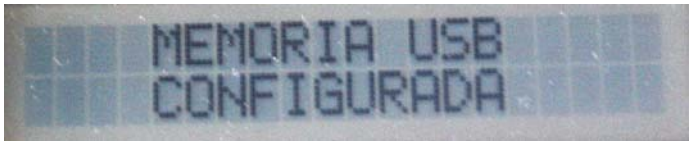
Para verificar la capacidad del sistema en identificar la conexión y desconexión de dispositivos USB de almacenamiento, se procedió a conectar memorias USB en cada uno de los puertos donde el sistema desplegó correctamente los mensajes de conexión y desconexión. Así mismo, se verificó la configuración y el respectivo despliegue del mensaje, cuando es conectada una memoria USB soportada por el sistema. La figura 40.a, 40.b y 40.c muestran los respectivos mensajes de la LCD.

4.7 REQUERIMIENTOS Y RECOMENDACIONES DEL USB IMPLEMENTERS FORUM

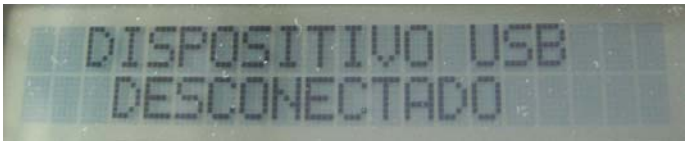
Para tener un punto de referencia acerca de la calidad y versatilidad del sistema H-ARD, se evaluaron los diferentes requerimientos y recomendaciones del *USB Implementers Forum Inc. (USB-IF)*, para sistemas con *Hosts* USB embebidos [USB-IF1]. Estos requerimientos permiten optar por la certificación del logo USB*

Figura 40. Mensajes de conexión, configuración y desconexión de un dispositivo USB

a.



b.



c.

a. Conexión de dispositivo USB b. Configuración exitosa de memoria USB
c. Desconexión de dispositivo USB

Fuente: Los Autores

* La certificación del logo USB es un aval de calidad proporcionado por el *USB Implementers Forum* a aquellos dispositivos que cumplan con el estándar y pasen las pruebas de verificación de desempeño, de acuerdo a las

4.7.5 Requerimientos para Puertos *Host* Embebidos

- **Pruebas eléctricas del Sistema Host**: El EZ-Host está certificado con el logo USB OTG, por tanto cumple con los requerimientos exigidos por el USB-IF tanto a nivel eléctrico como a nivel de característica OTG, [CYPRESS4].
- **Lista de periféricos objetivo**: Un *Host* Embebido debe especificar una Lista de periféricos objetivo (*Tageted Peripheral List* ó TPL) para indicar cuales dispositivos periféricos son soportados. La tabla 14 muestra la TPL con el formato de clases USB, del sistema H-ARD.

- **Alimentación**: El sistema es capaz de suministrar hasta 500mA de manera simultánea en cada puerto, soportando el valor máximo que puede consumir del bus cualquier dispositivo USB no autoalimentado, de acuerdo con [USB-IF5]. De esta manera se abarcan los dispositivos especificados en la TPL, cumpliendo con el requerimiento de alimentación de periféricos.
- **Velocidad:** El sistema H-ARD soporta dispositivos con velocidades de *Full Speed* únicamente.
- **Tipos de Transferencias:** El USB-IF determina que un *Host* USB embebido debe soportar al menos las transferencias de Control. Los otros tipos de transferencias (bulk, isócronas y de interrupción) son opcionales. El sistema H-ARD soporta las transferencias de Control y Bulk.
- **Soporte de Hub:** El USB-IF no exige soporte para HUB. En el caso del sistema H-ARD, no fue implementado soporte para HUB con el propósito de reducir la complejidad del sistema y los recursos necesarios.
- **Interoperabilidad:** El *Host* USB embebido debe demostrar interoperabilidad con los dispositivos de la TPL. El sistema H-ARD está en capacidad de interactuar con los dispositivos pertenecientes a la Clase *Mass Storage* y que cumplan con los requerimientos de la TPL del sistema (ver tabla 14).

---

características del dispositivo. Este aval implica el derecho de utilización del logo oficial de USB. Más información en www.usb.org

Tabla 14. Lista de periféricos objetivo del sistema H-ARD

Nombre de la Clase	Descripción	Código de Clase	Código de Sub Clase	Protocolo	Velocidades Soportadas
Mass Storage	Memorias Flash USB y/o unidades de almacenamiento. masivo	08h	06h	50h	*Full Speed*
Algunos Dispositivos Probados					
Fabricante	**Modelo**	**Vendor ID**	**Product ID**	**Descripción**	**Velocidad**
Flash Drive	OT_USB2.0	0x0EA0	0x2168	Memoria Flash USB de 128MB de capacidad	*Full Speed*
Kingston	Data Traveler®	0x08EC	0x0016	Memoria Flash USB de 512MB	*Full Speed*
Sony		0x054C	0x0243	Memoria Flash USB de 512MB	*Full Speed*
SanDisk		0x0781	0x7200	Memoria USB de 256Mb, reproductor de MP3	*Full Speed*
PNY		0x0D7D	0x1600	Memoria Flash USB de 256MB	*Full Speed*

- **Indicación al Usuario:** Con el propósito de cumplir con el requerimiento de indicarle al usuario si el dispositivo periférico conectado es soportado, el sistema despliega mensajes textuales indicativos en la pantalla LCD.

4.7.6 REQUERIMIENTOS PARA *HOST* USB EMBEBIDOS CON MÚLTIPLES CONECTORES RECEPTORES TIPO A.

El H-ARD posee dos puertos con conectores tipo A (con posibilidad de expansión a cuatro), característica que lo ubica en esta categoría de dispositivos. Aún así, el sistema

cumple con los requerimientos de certificación exigidos por el USB-IF para *Host* Embebidos con múltiples conectores receptores tipo A, de la siguiente forma:

- Cumple de los requerimientos para *Host* USB Embebidos, expuestos en el apartado anterior.
- Cada puerto está en capacidad de operar independiente de la actividad de los demás puertos, siempre y cuando la aplicación que se ejecute sobre el sistema H-ARD lo permita.
- El sistema es capaz de suministrar simultáneamente la corriente requerida en cada puerto (500mA en cada puerto que conlleva a un máximo de 1A en la configuración de dos puertos y 2A en la configuración de expansión).
- Todos los puertos soportan la misma velocidad (*Full Speed*).
- Todos los puertos soportan los mismos dispositivos, en concordancia con la TPL del sistema.

4.7.7 RESULTADOS DE LA EVALUACIÓN DEL H-ARD BAJO LOS REQUERIMIENTOS DEL USB-IF

Después de evaluar el H-ARD bajo los requerimientos y recomendaciones del USB-IF, se puede concluir que el sistema cumple con las condiciones necesarias para solicitar ante la autoridad en materia de USB, el USB-IF, la certificación del logo USB, lo que resalta las características del sistema y reitera su calidad bajo los estándares para la implementación de sistemas *Host* USB embebidos.

CONCLUSIONES Y OBSERVACIONES

- Se diseñó e implementó el sistema de almacenamiento de datos en memorias Flash USB portátiles, H-ARD, basado en un *Host* USB embebido, el cual se ha propuesto para suplir la necesidad tanto del grupo de investigación CEMOS[*] como de los sistemas embebidos actuales, de medios de almacenamiento con mayor capacidad. El sistema tiene completa autonomía del PC; puede controlar de manera independiente hasta dos memorias portátiles Flash USB[†] conectadas simultáneamente y puede guardar archivos con un formato compatible con un sistema operativo común. Sus prestaciones permiten aprovechar las características de las memorias portátiles con tecnología Flash y con conectividad USB, en cuanto a facilidad para el transporte de datos, alta densidad de almacenamiento, flexibilidad de expansión o reemplazo y cómodo acceso desde un PC a los datos almacenados en la memoria.

- El Sistema desarrollado cuenta con las características de un *Host* USB embebido multipuerto. Posee dos puertos *Host* USB (con posibilidad de expansión a cuatro) los cuales permiten la conexión de dispositivos periféricos USB.

- Para el control general de dispositivos USB, se implementó un *Driver* de USB que permite llevar a cabo las diversas tareas de un *Host*, tales como la configuración y control de dispositivos USB, el manejo de peticiones USB (*USB requests*) y la transmisión de datos por el Bus. Adicionalmente, el *Driver* de USB desarrollado permite el manejo de transferencias de Control y el manejo de Transferencias *Bulk*, para el intercambio de datos.

[*] Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones de la Universidad Industrial de Santander
[†] Que cumplan con los requerimientos expuestos en el apartado 4.7

- Para el control específico de dispositivos de almacenamiento, se implementó un *Driver* de clase que permite la interacción con dispositivos USB de la clase de almacenamiento masivo (MSC), que funcionen bajo el protocolo de clase *Bulk Only Transport* (BOT) y el juego de comandos SCSI para dispositivos de Acceso directo SBC. Por lo tanto, el sistema H-ARD está en capacidad de controlar dispositivos USB de almacenamiento (con las anteriores propiedades), sin la intervención de un PC; característica ideal para los sistemas embebidos portátiles.

- Dado que las memorias portátiles con tecnología Flash trabajan con el formato de archivos FAT32, se desarrolló un *Driver* para el manejo de archivos basado en el formato FAT32, el cual soporta dispositivos con sectores de longitud de 512bytes. Dicho *Driver* está en capacidad de escribir y leer sectores, escribir en clusters de datos, leer y escribir en la tabla FAT, buscar campos de nombre de archivos en el directorio raíz y crear archivos en este directorio.

- En la recopilación bibliográfica se analizó y concluyó que para el desarrollo de sistemas *Host* USB embebidos es necesario implementar un Controlador *Host* que facilite en hardware los medios para el control de dispositivos periféricos. Por tanto, se diseñó e implementó la tarjeta HC-USB basada en el controlador *Host* EZ-Host™ de Cypress, la cual se encarga de las tareas de la capa física y de protocolo del sistema USB, propias de un *Host* y permite el envío de transferencias y la administración del acceso al bus, entre otras características. Cabe resaltar que un Controlador *Host*, puede también ofrecer las herramientas para implementar un dispositivo periférico, como el caso del EZ-Host, sin embargo, un controlador de periférico (como el FT2232 de FTDI) no tiene soporte para *Host* USB.

- La tarjeta HC-USB fue desarrollada siguiendo los requerimientos generales para el desarrollo de un circuito impreso con tecnología de montaje superficial, así

como con las recomendaciones para la implementación de hardware con interfaz USB. Específicamente se logró mantener planos de tierra apreciables, los cuales protegen a la tarjeta de efectos negativos como los analizados en el apartado 2.2.7, dado que se manejan altas velocidades de transmisión. Además, la tarjeta se desarrolló con dispositivos de tecnología CMOS, de bajo consumo de potencia.

- Con el objetivo de interactuar con el controlador *Host* y de administrar sus funciones, se desarrolló un *Driver* que controla desde el DSP las funciones del EZ-Host. Este *Driver* permite también la creación de *Transaction Descriptors* a partir de las transacciones a llevar a cabo y tiene la capacidad de cargar varios *Transaction Descriptors* al mismo tiempo. Adicionalmente posee un control de errores del protocolo USB, complementario al control de errores del BIOS del EZ-Host.

- El hardware global del sistema H-ARD esta constituido por la tarjeta desarrollada por los ingenieros Juan Carlos Vargas y Cristihan García, [GARCIA-VARGAS] (proyecto relacionado con la investigación que enmarca este proyecto) y la tarjeta HC-USB desarrollada en este trabajo. La unión de estos dispositivos evidencia cómo se pueden aprovechar herramientas desarrolladas en trabajos anteriores a beneficio de nuevos proyectos; así mismo, el trabajo conjunto en la investigación [MIRANDA] permite argüir la validez de desarrollar investigaciones multidisciplinaras que permitan la integración de varios proyectos y la interacción de diferentes áreas de conocimiento.

- El sistema H-ARD cumple con los requerimientos y recomendaciones propuestas por la autoridad en materia de USB, el *USB Implementers Forum* (USB-IF), para la implementación de dispositivos *Host* USB embebidos, haciendo al sistema apto para solicitar la certificación de calidad del logo USB ante el USB-IF y a su vez la aprobación de conformidad con el estándar.

- Se diseñó una arquitectura de software por módulos que ofrece flexibilidad en cuanto al reemplazo y modificación de las diferentes entidades de software. Complementario a ello, la programación del software del sistema estuvo orientada a la portabilidad e independencia de la plataforma de hardware y al mismo tiempo se hizo énfasis en el aprovechamiento de las herramientas de programación del lenguaje C para resolver criterios de diseño de software como velocidad de cómputo y tamaño de programa, entre otros.

- Durante las pruebas del sistema se observó que cuando en una transmisión ocurre el error Nak (el cual indica que los *buffers* de datos del dispositivo se encuentran llenos y no están en condiciones de recibir datos) es preciso esperar mínimo 10ms (10 *frames* de datos)[*] para volver a intentar la transmisión, tiempo significativo que se ve reflejado en retardos de transmisión de datos. Sin embargo, el tiempo de respuesta del dispositivo o memoria USB depende de las características de este (como velocidad de procesamiento y tamaño del *buffer*). Esta clase de error se resuelve a nivel de *Driver* de Controlador *Host*, contrario a errores como *Stall* los cuales deben ser resueltos por capas superiores al *Driver* de USB, dado que indican problemas fuera del protocolo USB.

- Con este trabajo, y tomando como base los avances presentados en trabajos anteriores relacionados con USB, es posible argumentar que se ha logrado un amplio conocimiento de la interfaz USB, de su funcionamiento, su arquitectura y sus herramientas; logro que permite el desarrollo de nuevas aplicaciones y proyectos con el uso de tecnologías vigentes como lo es USB, así como la adquisición de nuevas competencias y conceptos en materia de dispositivos USB (periféricos y *Host* Embebidos).

[*] Este dato fue obtenido de acuerdo al promedio de mediciones tomadas para varias memorias USB

- Para facilitar la lectura desde un PC de los archivos almacenados con el sistema H-ARD, se desarrolló la herramienta gráfica LECTORA, mediante el editor de interfaces gráficas Guide de Matlab7.0, la cual facilita la apertura de archivos con extensión .BIN desde una memoria USB y permite graficar los datos de las mediciones del espectro de impedancia eléctrica del tejido de Cerviz de una paciente, de acuerdo con lo estipulado en los requerimientos preliminares presentados en el apartado 2.1.

- Se utilizaron los programas para PC WinHex v12.85 y USB Monitor v2.37, los cuales son un buen complemento para desarrollar dispositivos que se comuniquen con memorias Flash USB, dado que permiten el análisis de los diferentes bloques lógicos de una unidad de almacenamiento, y facilitan el monitoreo de datos entre el PC y la memoria, para hacer ingeniería inversa, respectivamente.

- El sistema diseñado en el presente proyecto es el primer paso en el desarrollo de sistemas *Host* USB embebidos, y un avance en la búsqueda de la conexión, de manera independiente del PC, entre dispositivos con interfaz USB. Con lo anterior se vislumbra la posibilidad de desarrollar sistemas embebidos con mayores prestaciones gracias a los servicios que pueden proporcionar periféricos como teclados, impresoras y memorias USB, así como los dispositivos de aplicación específica en instrumentación, comunicaciones y otros tantos campos de la electrotecnología y bioingeniería.

NUEVAS PERSPECTIVAS

- Con base en los *Drivers* de USB y de Controlador *Host* implementados para el sistema H-ARD, se pueden desarrollar diferentes *Drivers* de Clase (por ejemplo para dispositivos de interfaz Humana e impresoras) para ampliar el rango de dispositivos USB soportados y aumentar las opciones de servicios al sistema embebido que se diseñe. Sin embargo, para ello es necesario complementar el *Driver* de USB con las transferencias Isócronas y de interrupción para cubrir todos los tipos de transferencias y todas las clases de dispositivos periféricos.

- El desarrollo de un *Driver* de clase HID (dispositivos de interfaz humana) permitiría adicionarle al sistema conectividad con teclados USB, de tal forma que se pueda ingresar los nombres de los archivos que se crean en la memoria portátil, complementado de esta forma el sistema de archivos FAT32 y la aplicación misma del sistema H-ARD. Por otra parte, varios dispositivos de aplicación específica basan su interfaz USB en el funcionamiento de esta clase de dispositivos (HID), lo que indica que el *Driver* de HID serviría también para soportar dispositivos de aplicación específica como tarjetas de adquisición de datos y demás.

- La tarjeta HC-USB fue diseñada con miras a aprovechar las herramientas y recursos que ofrece el EZ-Host. Por tanto, dicha tarjeta ofrece para futuras aplicaciones la posibilidad de expansión de dos a cuatro puertos *Host* USB (receptores tipo A), la posibilidad de trabajar en modo independiente (*Stand-Alone*) y la posibilidad de utilizar interfaces de conexión, con procesadores externos, de alta velocidad (HPI). Así mismo, cuenta con un puerto miniB que le permite trabajar como un periférico USB. Vale la pena resaltar que la adición en hardware de estas herramientas requiere la respectiva adaptación y complementación en software.

- Dado que USB es una interfaz serial de alta velocidad, requiere de técnicas específicas para reducir el impacto de efectos electromagnéticos negativos en las señales que se transmiten. Las recomendaciones para la elaboración de PCB propuesta por [CYPRESS3], sugieren la utilización de placas de PCB de cuatro y más caras para obtener planos de tierra y de fuente dedicados. Aún cuando en este proyecto el diseño de PCB consiste en una tarjeta de dos caras con áreas apreciables de planos de tierra, se plantea la necesidad de incursionar con la tecnología de diseño y fabricación de PCB de cuatro caras, para cumplir con las recomendaciones de diseño de PCB para dispositivos con puertos USB, y a su vez lograr una reducción en el área total de la tarjeta. Este proceso de fabricación de PCB de cuatro caras permitiría también a la Escuela de Ingeniería Eléctrica, Electrónica y Telecomunicaciones de la Universidad Industrial de Santander estar a la vanguardia en los procesos tecnológicos mundiales de diseño de circuitos impresos.

- El presente trabajo permite vislumbrar la complejidad en software de los sistemas embebidos actuales. Por ello, nuevos lenguajes de programación y herramientas de modelamiento de software orientadas a objetos (como C++ y el reciente UML, respectivamente) están siendo necesarios para mejorar los procesos de desarrollo de software en un sistema embebido y reducir su complejidad de implementación [MARWEDEL].

- Una excelente alternativa para integrar el proceso de almacenamiento de datos en memorias portátiles (desarrollado en el presente trabajo) con otros procesos dentro de un sistema embebido, es el desarrollo y/o la implementación de sistemas operativos en tiempo real (RTOS o *Real Time Operating Systems*), para controlar diferentes y complejos procesos de forma rápida y eficiente. Sin embargo, dado que la carga computacional es elevada, para ello se recomienda

la utilización de procesadores de más de 16bits (como por ejemplo tipo ARM de 32bits o incluso mayores).

- Gracias al puerto OTG del EZ-Host y al conector miniB incorporado en la tarjeta HC-USB, el sistema está en capacidad de ofrecer conectividad con dispositivos que utilicen la tecnología complementaria OTG, la cual es el siguiente paso a seguir, después del estudio y desarrollo de sistemas *Host* USB embebidos, en la tarea de explorar e incorporar nuevas tecnologías a los diseños de sistemas embebidos.

REFERENCIAS BIBLIOGRÁFICAS

[ANDERSON] ANDERSON, Don. USB System Architecture. Menlo Park, California : ADDISON-WESLEY. 2001. 542p. ISBN: 0-201-46137-4

[ASCENCIO] ASCENCIO, Freddy. Diseño de Un Sistema de Adquisición de Datos por Bus Serial Universal (USB) y memoria SRAM externa. Bucaramanga, 2005. 140p. Trabajo de Grado (Ingeniería Electrónica). Universidad Industrial de Santander. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones.

[AXELSON]AXELSON, Jan. USB complete. 2ed. Madison : Lakeview Research, 2001. 514p.

[BARR] BARR, Michael. Programming Embedded Systems: in C and C++. O'Reilly, 1999. 191p. ISBN 1-56592-354-5

[CATSOULIS] CATSOULIS, John. Designing Embedded Hardware. Sebastopol, CA : O'Reilly, 2003. 318p. ISBN 0-596-00362-5

[CONDE-SANTOS] CONDE, Sergio y SANTOS, Andrea. Diseño, Construcción e Implementación de un Sistema para la Medición de Variables Fisiológicas en Pruebas Autonómicas. Bucaramanga, 2005. 207p. Trabajo de Grado (Ingeniería Electrónica). Universidad Industrial de Santander. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones.

[CYPRESS1] CYPRESS SEMICONDUCTOR. OTG-Host BIOS User's Manual EZ-Host. Version 1.1. San Jose, CA : CYPRESS, 2003. 172p. : il. [online]. Disponible en: www.cypress.com.

[CYPRESS2] _____. EZ-Host™, Programmable Embedded USB Host/Peripheral Controller : Data sheet. San Jose, CA : CYPRESS, 2003. 120p. : il. [online]. Disponible en: www.cypress.com.

[CYPRESS3] _____. High-Speed USB PCB Layout Recommendations. San Jose, CA : CYPRESS, 2002. 4p. [online]. Disponible en: www.cypress.com.

[CYPRESS4] _____. USB on-the-go (OTG) Basics. San Jose, CA : CYPRESS, 2002. 8p. [online]. Disponible en: www.cypress.com.

[DOBIASH] DOBIASH, Jack. Fat32 Structure Information. Abril 14, 1999. [online]. Disponible en: home.teleport.com/~brainy/fat16.htm.

[FAIRCHILD] FAIRCHILD SEMICONDUCTOR™. 74AC04-74ACT04 Hex Inverter : Data sheet. FAIRCHILD, 1999. 8p. : il. [online]. Disponible en www.fairchild.com.

[GARCIA-VALLE] GARCIA, Ludwing D. y VALLE, Francisco C. Cálculo del Espectro de Impedancia Eléctrica de Cuello Uterino por Transformada Rápida de Fourier Implementando la Teoría de Muestreo por Desfase en un DSP MOTOROLA de la Familia 56800. Bucaramanga, 2006. 98p. Trabajo de Grado (ingeniería Electrónica). Universidad Industrial de Santander. Escuela de ingenierías Eléctrica, Electrónica y de Telecomunicaciones.

[GARCIA-VARGAS] GARCIA, Cristihan y VARGAS, Juan. Diseño y Montaje de un Sistema de Adquisición de Señales de Voltaje para la Medida del Espectro de Impedancia Eléctrica en Tejido Humano. Bucaramanga, 2005. 119p. Trabajo de Grado (Ingeniería Electrónica). Universidad Industrial de Santander. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones.

[HERNANDEZ-MANTILLA] HERNÁNDEZ, Nathalie y MANTILLA, Ivonne. Diseño e Implementación de una Tarjeta de Adquisición de Datos por bus USB. Bucaramanga, 2004. 131p. Trabajo de Grado (Ingeniería Electrónica). Universidad Industrial de Santander. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones.

[HITACHI] HITACHI. HD44780U (LCD-II) Dot Matrix Liquid Crystal Display Controller/Driver. ADE-207-272(Z) : Data sheet. Tokio : HITACHI, 1998. 60p. : il. [online]. Disponible en semiconductor.hitachi.com

[INCITS1] INTERNATIONAL COMMITTEE ON INFORMATION TECHNOLOGY STANDARDS. SCSI Primary Commands – 4 (SPC-4) [online]. Revisión 1. Dallas: T10, 2005. 499p. il. Disponible en: www.t10.org/ftp/t10/drafts/spc4/spc4r06.pdf

[INCITS2] INTERNATIONAL COMMITTEE ON INFORMATION TECHNOLOGY STANDARDS. SCSI Block Commands – 3 (SBC-3) [online]. Revisión 6. Rochester : T10, 2006. 145p. il. Disponible en: www.t10.org/ftp/t10/drafts/sbc3/sbc3r06.pdf

[KAPLAN] KAPLAN, Francois. Embedded Flash Drivers: Standardizing NAND Flash for use in mobile handsets. M-Systems, 2005. 7p.

[LINEAR-TECH1] LINEAR TECH. LT1129-3.3, 700mA Low Dropout Regulators with Micropower Quiescent Current and Shutdown : Data sheet. LINEAR TECH. 16p. : il. [online] Disponible en: www.linear.com.

[LINEAR-TECH2] _____. LT1529-5, 3A Low Dropout Regulators with Micropower Quiescent Current and Shutdown : Data sheet. LINEAR TECH. 12p. : il. [online]. Disponible en: www.linear.com.

[MARWEDEL] MARWEDEL, Peter. Embedded Systems Design. Springer, 2006. 259p.

[MICROSOFT1] MICROSOFT CORPORATION. FAT: General Overview of On-Disk Format. Version 1.02. Mayo 5 de 1999. 25p. [online]. Disponible en: www.microsoft.com.

[MICROSOFT2] _____. FAT32 File System Specification. Version 1.03. Diciembre de 2000. 34p. [online]. Disponible en: www.microsoft.com.

[MIRANDA] MIRANDA, David. Detección Precoz de Cáncer de Cuello Uterino Basada en Espectro de Impedancia Eléctrica. Bucaramanga, 2005. 129p. Trabajo de Investigación (Maestría en Ingeniería). Universidad Industrial de Santander. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones.

[MONROY-ZABALA] MONROY, Diego y ZABALA, Sergio. Repotenciación y Actualización de un Potenciostato Galvanostato Princeton Modelo 363 para el Laboratorio de Corrosión de la Escuela de Ingeniería Metalúrgica y Ciencia de Materiales. Bucaramanga, 2005. 167p. Trabajo de Grado (Ingeniería Electrónica). Universidad Industrial de Santander. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones.

[ON-SEMICONDUCTOR] ON-SEMICONDUCTOR. MSQA6V1W5T2 Quad Array For ESD Protection : Data sheet. ON semiconductor, 2003. 4p. [online]. Disponible en: www.onsemi.com.

[PHILIPS] PHILIPS. USB On The Go: a Tutorial. PHILIPS, 2002. 9p. [online]. Disponible en: www.semiconductors.philips.com

[TEXAS1] TEXAS INSTRUMENTS. Current-limited, Power-Distribution Switches, TPS2054B : Data sheet. TEXAS : Dallas, 2006. 35p. [online]. Disponible en: www.ti.com.

[TEXAS2] _____. Printed-Circuit-Board Layout for Improved Electromagnetic Compatibility. TEXAS : Dallas,1996. 14p. [online]. Disponible en: www.ti.com.

[USB-IF1] USB IMPLEMENTERS FORUM. Requirements and Recommendations for USB Products with Embedded Host and/or Multiple Receptacles. Revisión 1.0. USB-IF, Julio 8 2004. 18p : il. [online]. Disponible: www.usb.org

[USB-IF2] _____. Universal Serial Bus Mass Storage Class Bulk-Only Transport. Revisión 1.0. USB-IF, 1999. 22p : il. [online]. Disponible en www.usb.org

[USB-IF3] _____. Universal Serial Bus Mass Storage Specification for Bootability. Revisión 1.0. USB-IF, Octubre de 2004. 19p : il. [online]. Disponible en: www.usb.org.

[USB-IF4] _____. Universal Serial Bus Mass Storage Class Specification Overview. Versión 1.2. USB-IF, 2003. 7p : il. [online]. Disponible en: www.usb.org.

[USB-IF5] _____. Universal Serial Bus Specifications : Version 2.0. USB-IF, Abril 27 de 2000. 650p : il. [online]. Disponible en: www.usb.org.

[WIKIPEDIA1] WIKIPEDIA. Memorias Flash. [Enciclopedia en línea]. Disponible en: www.wikipedia.com

[WIKIPEDIA2] _____. Ortografía del español. [Enciclopedia en línea]. Disponible en: www.wikipedia.com

ANEXOS

ANEXO A. DESCRIPTORES DE LOS DISPOSITIVOS DE LA CLASE MSC

Los descriptores de los periféricos USB son estructuras de datos que mantienen información relevante que el *Host* necesita para iniciar y controlar adecuadamente una comunicación con cada dispositivo USB.

A continuación se listan algunos descriptores estándares que tienen los dispositivos USB de la clase de almacenamiento masivo (como las memorias Flash USB) y se menciona una pequeña descripción de los campos más relevantes para el *Host*.

A.1 DESCRIPTOR DE DISPOSITIVO

Todos los Periféricos USB tienen el **Descriptor de Dispositivo (*Device Descriptor*)**, en el cual se presentan datos generales del periférico tales como el identificador del vendedor (*Vendor ID*) y el identificador de producto (*Product ID*) entre otros. Un campo importante del descriptor de dispositivo es el número de configuraciones que soporta el Periférico (refiérase al apartado 1.3.3 para ver la estructura lógica de un dispositivo). La figura A1 ilustra el descriptor de dispositivo de un periférico en general.

A.2 DESCRIPTOR DE CONFIGURACIÓN

Los descriptores de configuración contienen información sobre las características de alimentación del dispositivo, tales como si es autoalimentado o alimentado por el bus y el consumo de corriente que puede requerir del *Host*. Adicionalmente, este descriptor indica el número de interfaces que tiene el dispositivo. Los periféricos USB tienen mínimo un descriptor de configuración. La figura A2 ilustra el descriptor de configuración de un dispositivo de la clase de almacenamiento masivo.

Figura A1. Descriptor de Dispositivo.

Offset	Field	Size	Value	Description
0	*bLength*	Byte	12h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	01h	DEVICE descriptor type.
2	*bcdUSB*	Word	????h	*USB Specification* Release Number in Binary-Coded Decimal (i.e. 2.10 = 210h). This field identifies the release of the *USB Specification* with which the device and is descriptors are compliant.
4	*bDeviceClass*	Byte	00h	**Class is specified in the interface descriptor.**
5	*bDeviceSubClass*	Byte	00h	**Subclass is specified in the interface descriptor.**
6	*bDeviceProtocol*	Byte	00h	**Protocol is specified in the interface descriptor.**
7	*bMaxPacketSize0*	Byte	??h	Maximum packet size for endpoint zero. (only 8, 16, 32, or 64 are valid **(08h, 10h, 20h, 40h)**).
8	*idVendor*	Word	????h	Vendor ID (assigned by the USB-**IF**).
10	*idProduct*	Word	????h	Product ID (assigned by the manufacturer).
12	*bcdDevice*	Word	????h	Device release number in binary-coded decimal.
14	*iManufacturer*	Byte	??h	Index of string descriptor describing the manufacturer.
15	*iProduct*	Byte	??h	Index of string descriptor describing this product.
16	*iSerialNumber*	Byte	??h	Index of string descriptor describing the device's serial number. **(Details in 4.1.1 below)**
17	*bNumConfigurations*	Byte	??h	Number of possible configurations.

Fuente: [USB-IF2]

Figura A2.Descriptor de configuración.

Offset	Field	Size	Value	Description
0	*bLength*	Byte	09h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	02h	CONFIGURATION Descriptor Type.
2	*wTotalLength*	Word	????h	Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, endpoint, and class- or vendor-specific) returned for this configuration.
4	*bNumInterfaces*	Byte	??h	Number of interfaces supported by this configuration. **The device shall support at least the Bulk-Only Data Interface.**
5	*bConfigurationValue*	Byte	??h	Value to use as an argument to the *SetConfiguration()* request to select this configuration.
6	*iConfiguration*	Byte	??h	Index of string descriptor describing this configuration.
7	*bmAttributes*	Byte	?0h	Configuration characteristics: **Bit** **Description** 7 Reserved (set to one) 6 Self-powered 5 Remote Wakeup 4..0 Reserved (reset to zero) **Bit** 7 is reserved and must be set to one for historical reasons. **For a full description of this *bmAttributes* bitmap, see the *USB 1.1 Specification*.**
8	*MaxPower*	Byte	??h	Maximum power consumption of the USB device from the bus in this specific configuration when the device is fully operational. Expressed in 2mA units (i.e. 50 = 100mA)

Fuente: [USB-IF2]

A.3. DESCRIPTOR DE INTERFAZ

Los dispositivos USB tienen al menos una interfaz por defecto y mínimo una interfaz por cada configuración [AXELSON]. Dentro de la información más importante que se puede extraer del descriptor de interfaz está el tipo de transferencias que soporta el periférico, la clase USB y la subclase a las cuales pertenece el dispositivo, y el protocolo de interfaz o de clase utilizado. Esta información la usa el *Host* para evaluar si puede o no controlar al periférico y para asignarle el *Driver* de clase apropiado.

Para el caso de las memorias Flash USB, la clase es de Almacenamiento Masivo (*Mass Storage Class*), la subclase depende del juego de comandos utilizado (que por lo general es SCSI para dichas memorias) y el protocolo de clase es el BOT, tal como se muestra en la figura A3.

Figura A3. Descriptor de Interfaz Bulk-Only.

Offset	Field	Size	Value	Description
0	*bLength*	Byte	09h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	04h	INTERFACE Descriptor Type.
2	*bInterfaceNumber*	Byte	0?h	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
3	*bAlternateSetting*	Byte	??h	Value used to select alternate setting for the interface identified in the prior field.
4	*bNumEndpoints*	Byte	??h	Number of endpoints used by this interface (excluding endpoint zero). **This value shall be at least 2.**
5	*bInterfaceClass*	Byte	08h	**MASS STORAGE Class.**
6	*bInterfaceSubClass*	Byte	0?h	Subclass code (assigned by the USB-**IF**). **Indicates which industry standard command block definition to use. Does not specify a type of storage device such as a floppy disk or CD-ROM drive. (See *USB Mass Storage Overview Specification*)**
7	*bInterfaceProtocol*	Byte	50h	**BULK-ONLY TRANSPORT. (See *USB Mass Storage Overview Specification*)**
8	*iInterface*	Byte	??h	Index to string descriptor describing this interface.

Fuente: [USB-IF2]

A.4 DESCRIPTOR DE ENDPOINT

Contiene la información de los *Endpoint* que tiene el dispositivo periférico. Para los dispositivos de la clase MSC debe existir un *Endpoint* de entrada y uno de salida, ambos deben soportar las transferencias tipo *Bulk*. La figura A4 ilustra los diferentes campos que componen los *Endpoint* IN y OUT.

Figura A4. Descriptores de Endpoint.

Offset	Field	Size	Value	Description
0	*bLength*	Byte	07h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	05h	ENDPOINT Descriptor Type.
2	*bEndpointAddress*	Byte	8?h	The address of this endpoint on the USB device. The address is encoded as follows. **Bit** **Description** 3..0 The endpoint number 6..4 Reserved, set to 0 7 1 = In
3	*bmAttributes*	Byte	02h	This is a Bulk endpoint.
4	*wMaxPacketSize*	Word	00??h	Maximum packet size. Shall be 8, 16, 32 or 64 bytes (08h, 10h, 20h, 40h).
6	*bInterval*	Byte	00h	Does not apply to Bulk endpoints.

a) Descriptor de Endpoint de Entrada.

Offset	Field	Size	Value	Description
0	*bLength*	Byte	07h	Size of this descriptor in bytes.
1	*bDescriptorType*	Byte	05h	ENDPOINT descriptor type.
2	*bEndpointAddress*	Byte	0?h	The address of this endpoint on the USB device. This address is encoded as follows: **Bit** **Description** 3..0 Endpoint number 6..4 Reserved, set to 0 7 0 = Out
3	*bmAttributes*	Byte	02h	This is a Bulk endpoint.
4	*wMaxPacketSize*	Word	00??h	Maximum packet size. Shall be 8, 16, 32 or 64 bytes (08h, 10h, 20h, or 40h).
6	*bInterval*	Byte	00h	Does not apply to Bulk endpoints.

b) Descriptor de Endpoint de Salida.

Fuente: [USB-IF2]

ANEXO B. BLOQUES DE COMANDOS DEL PROTOCOLO BOT

El protocolo de transporte BOT (*Bulk Only Transport*) es el protocolo más utilizado por los dispositivos de la clase MSC para comunicarse con un *Host USB*. Este protocolo se lleva a cabo mediante transferencias *Bulk* donde se transmiten tres bloques o paquetes de datos: el bloque de cabecera que contiene las generalidades de la transferencia incluyendo el comando a ejecutar por el dispositivo; el bloque de datos a transmitir y el bloque de estado o acuse de la transmisión (véase apartado 1.5). La estructura del bloque de cabecera y de estado, se presentan a continuación.

B.1 COMMAND BLOCK WRAPPER CBW

El CBW es el paquete de datos donde se encapsulan las peticiones del juego de comandos que esté siendo usado por el periférico USB (ya sea SCSI u otro). Este comando lo envía el *Host* hacia el dispositivo de almacenamiento masivo para indicarle la acción respectiva a ejecutar. La figura B1 ilustra los campos de la estructura de información que se describen a continuación.

dCBWSignature: Este campo indica que el paquete de datos corresponde a un CBW, gracias al identificador 43425355h. El valor se interpreta en formato *Little Endian*[*].
dCBWTag: Es una firma que envía el *Host* para asociar el CBW con el respectivo bloque de estado (CSW).
dCBWDataTransferLength: Indica el tamaño de los datos a transmitir ya sea de entrada o de salida.

[*] En el formato *Little Endian*, el byte menos significativo se almacena en la posición menos significativa de memoria.

Figura B1. Estructura CBW

bit / Byte	7	6	5	4	3	2	1	0
0-3	*dCBWSignature*							
4-7	*dCBWTag*							
8-11 (08h-0Bh)	*dCBWDataTransferLength*							
12 (0Ch)	*bmCBWFlags*							
13 (0Dh)	Reserved (0)				*bCBWLUN*			
14 (0Eh)	Reserved (0)			*bCBWCBLength*				
15-30 (0Fh-1Eh)	*CBWCB*							

Fuente: [USB-IF2]

bmCBWFlags: Es un campo de bits que contiene indicadores de características de la transferencia. Los campos son los siguientes:

Bit	Descripción
7	Dirección: indica la dirección de los datos
6	Obsoleto (el *Host* lo envía en cero)
5-0	Bits reservados(se transmiten en cero)

bCBWLUN: Número de la unidad lógica a la cual se va a escribir en el dispositivo de almacenamiento.

bCBWCBLength: Indica el tamaño del comando (SCSI) que se envía encapsulada en el CBW.

CBWCB: se envía el comando que va a ser ejecutado. Este campo tiene una longitud máxima de 15 bytes pero puede ser menor y sujeto a lo especificado en el campo *bCBWCBLength*. En caso de no ser necesarios los 15 bytes se deben rellenar con ceros los bytes que no se usen.

Figura B2.Estructura CSW

bit / Byte	7	6	5	4	3	2	1	0
0-3	*dCSWSignature*							
4-7	*dCSWTag*							
8-11 (8-Bh)	*dCSWDataResidue*							
12 (Ch)	*bCSWStatus*							

Fuente: [USB-IF2]

B.2 COMMAND STATUS WRAPPER CSW

El paquete de estado es creado por el dispositivo para informar los sucesos acerca de la transferencia de los datos. La estructura de este bloque se muestra en la figura B2.

dCSWSignature: Es el identificador del paquete CSW 53425355h en formato *Little Endian*.

dCSWtag: Contiene la misma firma del paquete CBW que inició la transmisión y permite asociar el CSW con el correspondiente CBW

dCSWDataresidue: Indica el residuo de datos en la transmisión. Para datos de salida el dispositivo calcula la diferencia entre los datos esperados (datos del campo *dCBWDataTransferLength* y los procesados realmente. Para datos de entrada el dispositivo calcula la diferencia entre los datos requeridos definidos *dCBWDataTransferLength* y los datos enviados por el dispositivo.

dCSWStatus: Indica el estado de la ejecución del comando enviado en el CBW y sirve como reporte de error de la transmisión.

ANEXO C. SISTEMA PARA ALMACENAMIENTO DE DATOS EN MEMORIAS FLASH USB PORTÁTILES, H-ARD, GUÍA DE USUARIO DE LA APLICACIÓN

El H-ARD es un sistema para almacenamiento de datos basado en un HOST USB embebido, que puede soportar los dispositivos de memoria con interfaz USB, tales como las memorias Flash portátiles y las cámaras digitales. Adicionalmente, tal y como se explicó en el apartado 3.3.5, el sistema H-ARD esta acompañado por un software de aplicación, el cual permite al usuario interactuar con el dispositivo y permite acceder a los diferentes procesos que ofrece el sistema (como la escritura de archivos en memorias Flash USB).

En esta guía se presenta una descripción del software de aplicación que la lleva a cabo la interfaz con el usuario.

C.1. CONFIGURACIONES DE HARDWARE PARA EL SISTEMA H-ARD

El sistema H-ARD esta constituido por la tarjeta DSP y la tarjeta HC-USB las cuales interactúan entre sí y se conectan por medio del puerto SPI (el conector J5 de la tarjeta HC-USB con el puerto J2 de la tarjeta de Adaptación TA1). Debido a las características de interacción entre los controladores, la tarjeta HC-USB debe ser configurada en modo de operación de coprocesador y habilitar la comunicación por el puerto SPI mediante el arreglo de *switchs* implementado en esta. En la tabla C.1 se describen los modos de operación de acuerdo a la configuración del *DIP-SWITCH*.

Tabla C 1. Configuración del DIP-SWITCH para la tarjeta HC-USB

DIP	Modo de operación
000	Stand Alone
010	HSS
011	HPI
101	SPI

C.2. DESCRIPCIÓN GENERAL DEL SOFTWARE DE APLICACIÓN DEL SISTEMA H-ARD

El sistema H-ARD cuenta con una pantalla LCD que le permite al usuario verificar los diferentes procesos y visualizar mensajes informativos. El sistema tiene dos botones pulsadores de decisión con los cuales el usuario puede seleccionar la tarea a ejecutar (refiérase al apartado 2.2.9, Interfaz de Hardware con el Usuario). A continuación se resumen los estados que se despliegan en la pantalla de interfaz con el usuario.

C.2.1 Inicio del sistema

El proceso de aplicación del sistema no se inicia mientras la tarjeta HC-USB no se encuentre conectada, por lo cual la primera tarea que se ejecuta es la verificación de conexión con dicha tarjeta. La figura C.1 ilustra el mensaje en caso de que la tarjeta HC-USB no este conectada.

Figura C 1. Mensaje de tarjeta HC-USB desconectada

Fuente: los autores

C.2.2 Pantalla principal

El sistema H-ARD ofrece un estado principal desde el cual es posible ejecutar las tareas de aplicación (medición del espectro de impedancia) y de creación de archivos en una memoria conectada. En el estado principal se despliegan dos mensajes dependiendo de las tareas pendientes por realizar. Los mensajes son:

- **Estado de Medición sin ejecutar:** Se despliega el mensaje de Medición de Espectro para que se pueda realizar la simulación de la medición (ver apartado

3.3.5) y se deja una opción para que el usuario pueda ver los créditos de los autores del sistema. La figura C2 ilustra el estado de aplicación sin ejecutar.

Figura C2.estado principal de medición sin ejecutar

Fuente: los autores

- **Estado de datos para guardar:** Una vez simulada la medición, el sistema H-ARD habilita la opción de guardar para crear un archivo en una memoria Flash USB con los datos previamente medidos. Si el usuario desea realizar una nueva medición sin guardar los datos, el sistema elimina automáticamente los datos que no se hayan guardado en una memoria USB. La figura C3 ilustra el estado principal con datos pendientes para ser guardados. Cuando el sistema crea el archivo, regresa al estado principal de aplicación sin ejecutar.

Figura C3. Estado principal de medición ejecutada.

Fuente: los autores

C.2.3 Proceso de creación de un archivo

Es el proceso que se lleva a cabo cuando el usuario selecciona la opción <<GUARDAR>>. El sistema informa acerca de las memorias disponibles conectadas a la tarjeta HC-USB, desplegando un mensaje de advertencia en caso de no detectar ninguna memoria USB conectada. La figura C4 ilustra el mensaje de reporte de memorias USB conectadas. La figura C5 muestra la advertencia para cuando no hay memorias disponibles.

Figura C4. Mensaje de memorias disponibles

Fuente: los autores

Figura C5. Mensaje de memorias USB sin conectar.

Fuente: los autores

Aprovechando una de las características más importantes de USB (*Plug and Play*), el sistema H-ARD detecta la conexión de una memoria a la tarjeta HC-USB y la configura automáticamente. El usuario puede seleccionar la unidad de almacenamiento que desee siempre y cuando este conectada, tal como muestra la figura C5.

Figura C6. Mensaje de selección de memorias

Fuente: los autores

Una vez que el usuario seleccione la unidad de almacenamiento, el archivo será creado en dicha unidad y los datos serán eliminados del buffer temporal para volver al estado principal de medición sin ejecutar.

Si ocurre un error en el proceso de creación del archivo, el sistema no elimina los datos con el fin de que el usuario pueda intentar nuevamente crear el archivo de datos

Por otro lado cuando el sistema crea un archivo correctamente, despliega el nombre del archivo.

C.3. SOFTWARE ADICIONAL PARA PC, LECTOR DE ARCHIVOS

Junto con el sistema H-ARD, se ofrece la herramienta de software para PC, LectorA con la cual se pueden leer los archivos creados en la memoria USB, por el sistema H-ARD. LectorA es una interfaz gráfica implementada con la herramienta *Guide* de MATLAB, la cual ofrece un entorno amigable para el usuario.

La interfaz lee la cabecera del archivo y verifica si este fue creado con el sistema H-ARD. Una vez que el archivo es reconocido, la interfaz le permite ver al usuario las mediciones de forma independiente. El programa LectorA.m ejecuta la herramienta.

Una descripción detallada de las características de la interfaz desarrollada se presenta en el apartado 3.5 del capitulo SOFTWARE.

ANEXO D. FORMULARIO PARA LA MEDICIÓN DE ESPECTRO DE IMPEDANCIA ELÉCTRICA EN CUELLO UTERINO

En el formato, el campo *NOMBRE DEL ARCHIVO* se llena con el dato que el Sistema H-ARD despliega al finalizar el proceso de creación del archivo en la memoria USB. Formato tomado del texto [MIRANDA].

FORMULARIO PARA LA MEDICIÓN DE ESPECTRO DE IMPEDANCIA ELÉCTRICA EN CUELLO UTERINO
Universidad Industrial de Santander 2004- 2005

DATOS DE LA PACIENTE		**Fecha:**		
Nombre:		Historia:		
Edad:		Hora de la Cirugía:		
PARÁMETROS DEL EQUIPO DE MEDICIÓN				
Nombre del archivo:				
Ganancia:	1	2	4	8
Corriente	10 μA	20 μA	40 μA	
EVALUACIÓN CLÍNICA DE LA LESIÓN				
MEDICIONES				

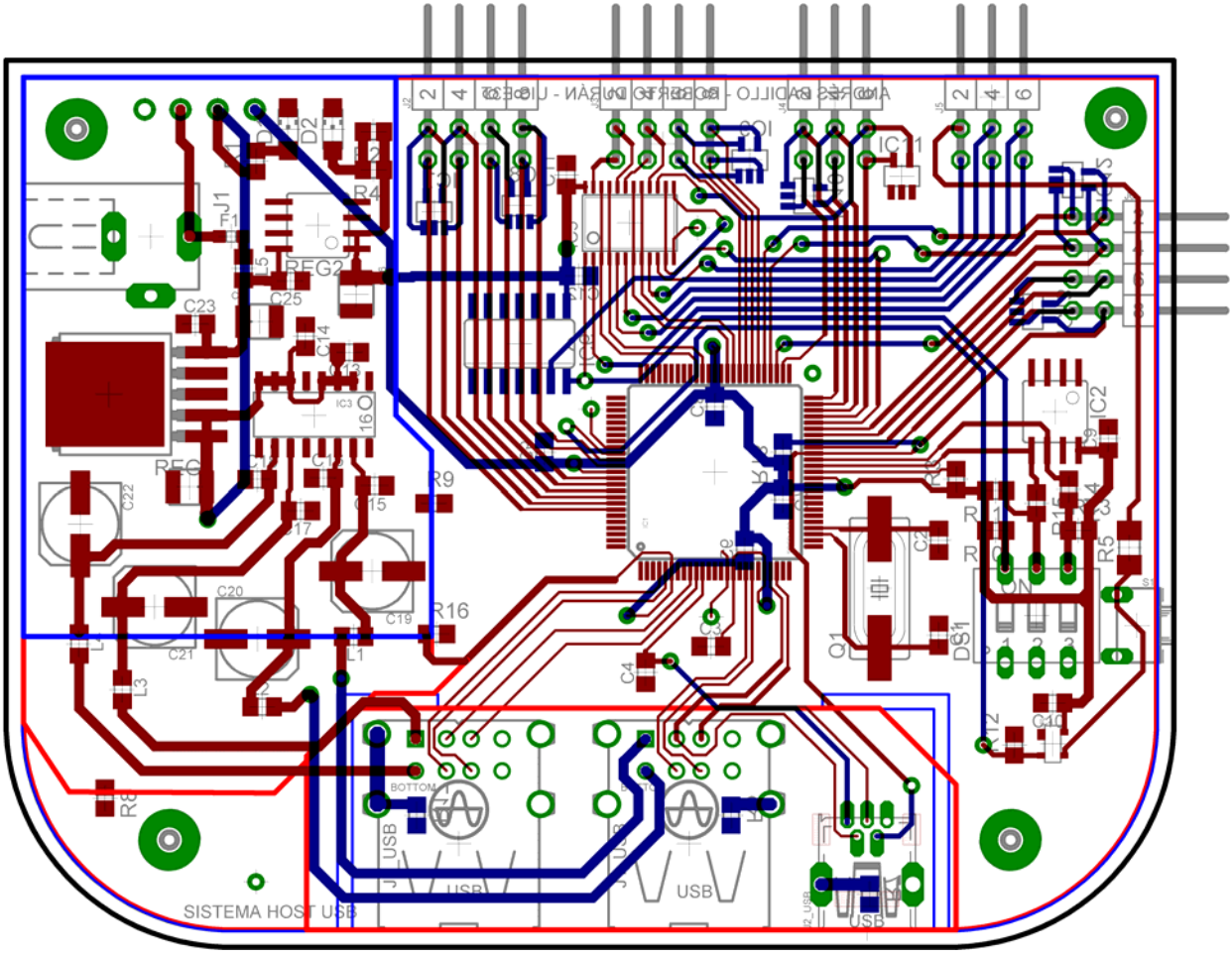
#	Hora	Observaciones

MÉDICO:
INGENIERO: David Alejandro Miranda Mercado

ANEXO E. ESQUEMATICO Y LAYOUT DE LA TARJETA HC-USB

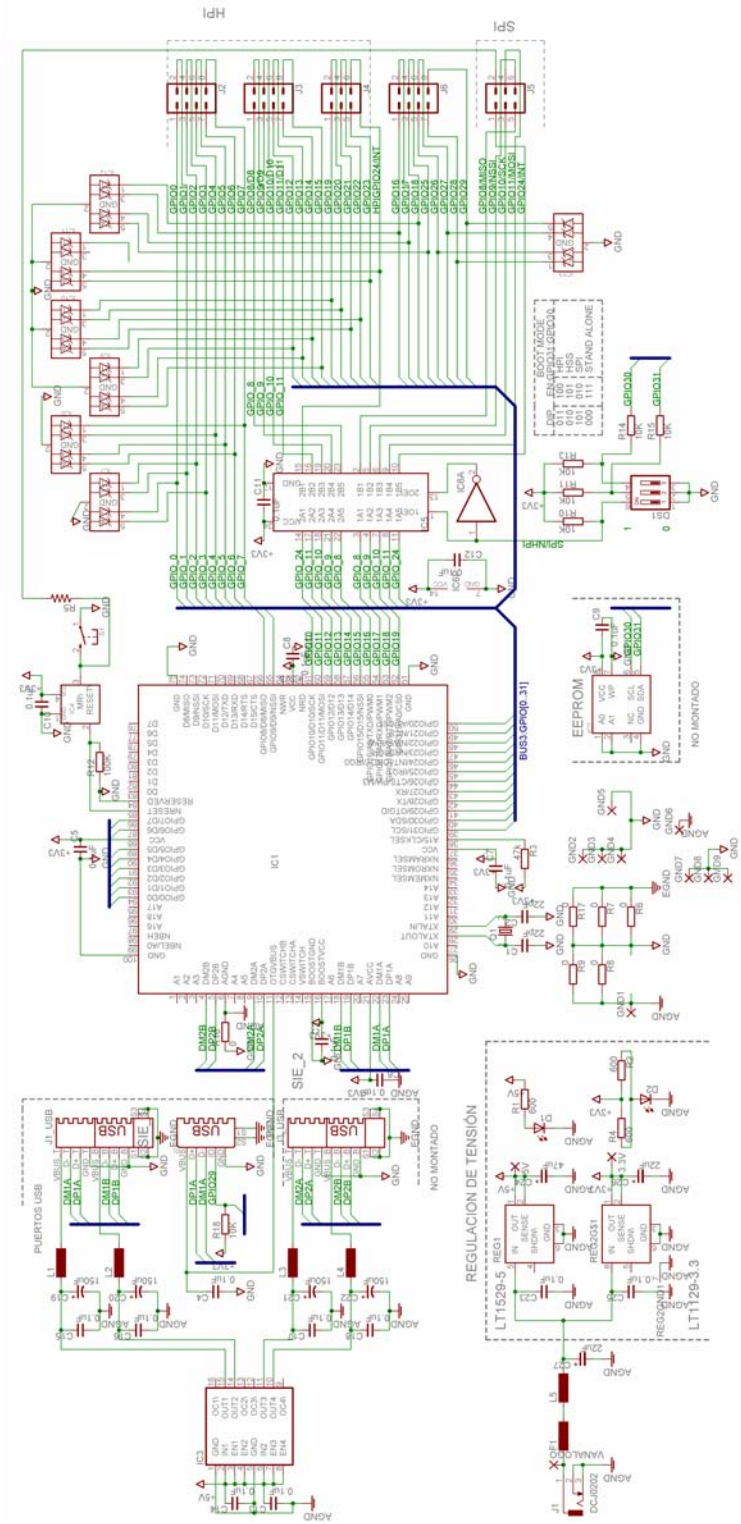
En el presente anexo se presentan los diagramas, esquemático y Layout de la tarjeta HC-USB donde se puede ver la ubicación de los bloques funcionales analizados en el capitulo 2 de este texto. El layout de la tarjeta HC-USB fue diseñado en la herramienta de software para diseño de circuitos impresos EAGLE.

Figura E1. Layout de la tarjeta HC-USB



Fuente: Los Autores.

Figura E2. Esquemático de la Tarjeta HC-USB



Fuente: Los Autores.

ANEXO F. DESCRIPCIÓN DE LAS FUNCIONES IMPLEMENTADAS EN EL SISTEMA H-ARD

A continuación se describen las funciones de *Software* implementadas en el Sistema H-ARD donde se mencionan las características de cada una.

F.1. LIBRERÍAS

DSP56F805.h: Es la librería que ofrece la herramienta *Codewarrior* con los diferentes macros definidos para el control del DSP.

ESTRGEN.h: En esta librería están declaradas todas las estructuras y macros que se usan en las diferentes funciones del sistema H-ARD. Se definen y declaran los macros para la capa de manejo de unidades de almacenamiento, para las capas de USB y para algunos registros del EZ-Host.

ESTRGLOBALES.h: Define las estructuras globales del sistema**.**

F.2. RUTINAS DE INICIO.

SPI_config() Se encarga de la configuración del puerto SPI en el DSP, deshabilita los GPIO y habilita los pines para SPI.

GPIO_config() Configura los GPIO del DSP que se usan en el control de la LCD, los Botones de decisión y la señal nSS de SPI de la tarjeta HC-USB.

LCD_config() Ejecuta los comandos de configuración inicial de la LCD.

EZHOST_config() Realiza la configuración general del Controlador *Host* de la Tarjeta HC-USB (Ez-Host) y configura el SIE1 como Host, ejecutando un Reset y actualizando la información de los registros del integrado.

F.3. FUNCIONES DE APLICACIÓN

void main(void**)** Rutina principal del Software que ejecuta las rutinas de inicio y de aplicación (en este caso App()).

void app() Ejecuta el proceso de aplicación en el cual se encuentran las tareas de interacción con el usuario, realiza la simulación de la medición del espectro de impedancia e invoca a las funciones necesarias para crear un archivo en una memoria USB.

F.4 FUNCIONES API (APPLICATION PROGRAM INTERFACE)

word API_escribir_mem(word data_buffer,word data_len**)** Función API que sirve como interfaz entre la aplicación y el sistema de almacenamiento H-ARD. La función se encarga de la administración de tareas para escribir un archivo en formato FAT32.

data_buffer Dirección del buffer en donde se encuentran los datos a almacenar.

data_len Longitud de los datos del archivo

F.5. FUNCIONES DEL SISTEMA DE ARCHIVOS FAT32

dword FAT_write_data_file(word data_buffer,word file_length_in_bytes**)** Función de la capa de sistema de archivos; se encarga de buscar los cluster vacíos para escribir un archivo; retorna el primer cluster.

data_buffer Dirección del buffer en donde se encuentran los datos.

file_length_in_bytes Longitud de los datos del archivo expresada en bytes.

word FAT_root_dir_search(word match_name**)** Función diseñada para buscar nombres y/o entradas libres en el directorio raíz, se apoya encadenando los cluster en la tabla de FAT.

match_name bandera que indica si se deben buscar nombres repetidos.

word FAT_write_sector_in_fats(dword lba_fat**)** Esta función escribe en los sectores de FAT1 y FAT2. Se ubica en el sector de la FAT indicado por el **lba_fat**.

word FAT_write_sector(word buffer_addr,dword lba**)** Esta función escribe en un sector de la unidad de almacenamiento.

buffer_addr Buffer en donde se encuentran ubicados los datos en el EZ-Host

dword lba LBA del sector donde se van a escribir los datos.

word FAT_read_sector(dword lba**)** Esta función lee el sector de la unidad de almacenamiento indicado en **lba**.

dword FAT_new_cluster_root_dir(dword last_cluster_root_dir**)** Función del sistema de archivos que se encarga de buscar un nuevo cluster para el directorio raíz; retorna el numero de cluster hallado. **last_cluster_root_dir** es el número del último cluster del directorio raíz.

word FAT_find_empty_clusters(word cantidad_ini,dword *clusters**)** Función del sistema de archivos se encarga de buscar un cluster vació desde la tabla de FAT; retorna el numero de cluster hallado.

dword FAT_next_cluster(dword cluster_actual**)** Función del Sistema de Archivos. Busca el siguiente campo de la FAT con el valor del número de cluster que apunta el campo actual de FAT; retorna el número del siguiente cluster

word FAT_config() Función de inicio y configuración del Sistema de Archivos. Crea la estructura para identificar un dispositivo de almacenamiento. Encadena la estructura que tiene la información referente a USB con la de FAT.

F.6. FUNCIONES DEL *DRIVER* DE CLASE USB MSC

word SCSI_inquiry() Función del juego de comandos que ejecuta el comando Inquiry, el cual extrae las características generales de la unidad.

word SCSI_read_capacity() Ejecuta el comando de READ_CAPACITY de SCSi.

word SCSI_test_unit_ready(void**)** función que ejecuta el comando que verifica si la unidad está disponible para transferir datos.

word SCSI(word command,dword lba,word data_len_lbas,word buff_addr**)** Función de SCSI con la que se ejecutan los comandos de SCSI como escritura y lectura.

word MSCDRV(word *SCSI_cmd_block,word data_len,word buffer_addr,byte data_dir**)** Función, Driver para MSC. Es la función encargada de crear los bloques para el protocolo BOT, con base en los comandos SCSI, y controlar el estado de las transferencias Bulk.

word MSC_reset_recovery() Esta función hace un reset de MSC y recupera los EP IN y OUT del STALL

F.7. FUNCIONES DEL DRIVER DE USB

word USB_dev_config() Pertenece a la capa de DRIVER USB se encarga de la enumeración y la configuración de un dispositivo; actualiza la estructura del dispositivo USB y si pertenece a la clase MSC le concatena una estructura de volumen.

word USBget_MSCmaxlun(word brequest_desctype,word desc_len**)** Función diseñada para ejecutar tanto la petición de unidades lógicas (Get_MAX_LUN) para el *driver* de MSC como lo requests de petición de UBS (GET-CONFIGURATION y GET DESCRIPTOR).

word USBset_MSCreset(word brequest,word wvalue_index**)** Ejecuta un reset de MSC o requests de control como SET-ADDRESS.

word USB_clear_feature(word windex**).** Borra el estado *Halt* de un Endpoint determinado.

word USB_bulk_transfer(word buffer_addr,word data_len,word data_dir) Función que crea las tranferencias USB tipo Bulk.

word USB_control_transfer(word data_dir,word data_len**)** Crea las transferencias tipo Control de USB.

word HCD_tlist(byte dtoggle_ddir,word pid,word buffer_addr,word data_len**)** Función Driver del controlador Host. Encargada de la creación de los tlist para las transacciones de USB ya sea en Control o en Bulk.

F.8. FUNCIONES DE INTERACCIÓN CON EL EZ-HOST

word write_bytes_buffer(word *DSP_data_ptr,word buff_addr,word data_len**)** Esta función escribe datos del DSP al EZHOST

word read_bytes_buffer(word buff_addr,word *DSP_sink_ptr,word data_len**)** Esta función lee datos de EZHOST hacia el DSP.

word exect_int(word *int_data,word int_len,word int_delay**)** Ejecuta una interrupción para el EZ-Host.

word enviar_cmd_lcp () envía el bloque del comando LCP por medio SPI

F.4. FUNCIONES GENERALES

word AP_zeros_buffer(word data_buffer,word data_len**)** Esta función rellena de ceros los datos para completar un sector de datos de 512bytes de longitud. Sus parámetros son:

data_buffer	Es el origen del buffer donde están ubicados los datos
data_len	Es la longitud de los datos reales

void retardo_base(unsigned long tiempo /*en us*/**)** Retardo cuya base es 1us

void dec2ASCII(dword numero,word digitos,word *destino**)** Esta función convierte números decimales de n dígitos a ASCII y los guarda en una cadena de caracteres.

numero	Es el numero decimal que va a ser convertido.
digitos	indica la cantidad de digitos del numero.
*destino	Es un puntero donde quedará ubicado el valor en ASCII.

dword ASCII2dec(word *numero,word digitos**)** Esta función convierte dígitos ASCII wchar a números decimales.

numero	Es el valor en ASCII (wchar).
digitos	numero decimal.

void bchar2wchar(char *origen,word *destino**)** Función que acomoda dos caracteres de 8 bytes en una palabra de 16bytes la cadena de caracteres puede ser de cualquier longitud.